Data-Efficient Probabilistic Time Series Style Transfer with Transformers

Master's Thesis

by JUSTUS WILL

supervised by
Prof. Dr. Sophie Fellenz,
Prof. Dr. Marius Kloft, and
Prof. Dr. Jörn Saß

November 4, 2022

Department of Mathematics,
Department of Computer Science

TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

Abstract

Style transfer is the task of recreating data in a different style while still preserving the original content. In applications to time series, this implies combining structural information of existing time series with new distinctive characteristics extracted from a dataset or style reference. If the style dataset contains experimental observations, style transfer techniques are used to enhance realism, for instance in simulation data. When used as data augmentation, results in downstream tasks such as time series forecasting can be greatly improved, especially when high-quality training data is limited. In contrast to image and text style transfer, where many efficient methods are available, research on time series style transfer has been limited to slow iterative methods. In this thesis, we show that model-based methods efficiently solve time series style transfer tasks in diverse settings. Specifically, we propose a transformer-based architecture with a latent representation that disentangles content and style information. The underlying generative model allows for fine control over the style of generated time series. After encoding the desired style, the style latent space can be fixed, resulting in fast and data-efficient stylized generation. Extensive experiments show the effectiveness of our approach both on synthetic data and in applications to finance and speech through comparison against several recent baselines.

Contents

1 Introduction and Overview **1**

1.1 Introduction . . . 1
1.2 Contribution . . . 3
1.3 Notation . . . 3
1.4 Outline . . . 4

2 Background **5**

2.1 Autoencoder . . . 5
2.2 VAE . . . 6
2.2.1 VAE Loss for Gaussians . . . 10
2.2.2 KL Divergence and Mutual Information . . . 13
2.3 LSTM . . . 14
2.4 Transformer . . . 16

3 Related Work **19**

3.1 Style Transfer . . . 19
3.1.1 Image and Text Style Transfer . . . 20
3.1.2 Time Series Style Transfer . . . 23
3.2 Transformer . . . 24

4 Method **25**

4.1 Generative Model . . . 25
4.2 Model Architecture . . . 28
4.3 Disentanglement and Inference . . . 31
4.4 Pretraining . . . 33

5 Experiments **35**

5.1 Baselines . . . 35
5.2 Metrics . . . 36
5.3 Data . . . 39
5.4 Training . . . 44
5.4.1 Hyperparameter Tuning . . . 45

5.5 Synthetic Data 47
5.6 Financial Data 50
5.7 Speech Data 51
5.8 Ablation 53
5.8.1 Feature Choice 53
5.8.2 Loss Choice 54
5.8.3 Content-Style Trade-Off 54
5.8.4 Number of Iterations 55

6 Conclusion **57**

7 Appendix **59**
7.1 Figures and Tables 59

References **80**

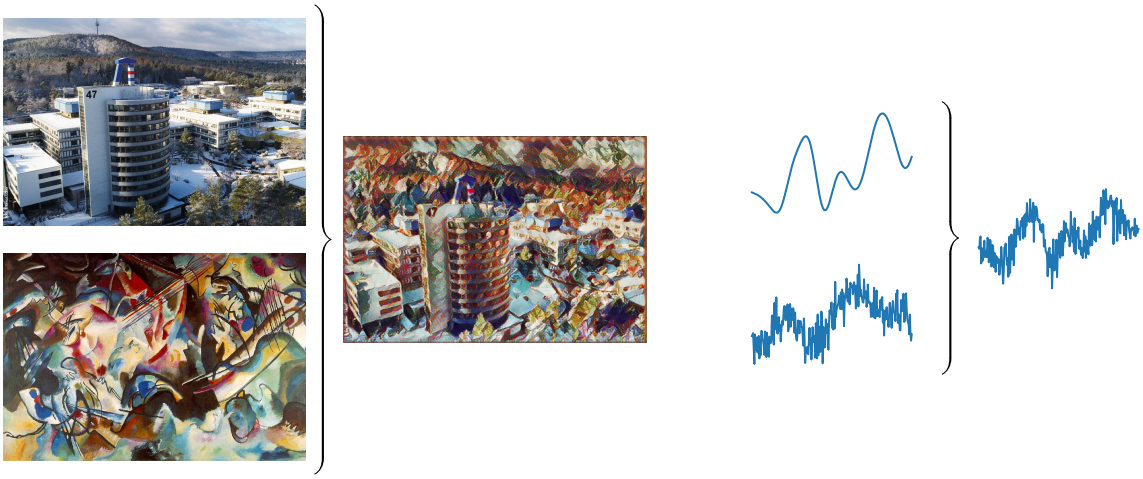
Chapter 1

Introduction and Overview

1.1 Introduction

Synthetic generation of realistic data is a key technique that has seen wide adoption in a variety of applications across many academic fields. Machine learning algorithms in particular benefit from additional labeled data, as it allows for models with improved generalization capabilities, especially when initial training data is limited or expensive to acquire. In this context, the quality and realism of the augmented data are crucial, as they directly impact the attainable performance. Good data augmentation allows for more accurate and more robust inference, for example in prediction tasks. Style transfer methodology enables realistic generation while still retaining high control over the generated data. Here, a careful balance between content preservation and stylization allows us to combine original high-level content with new characteristic style properties. We can, for example, take simulation data from an imperfect model and improve its utility by enhancing the realism of the data, essentially adding realistic noise. Simulation studies are one of the most important tools in mechanical engineering, natural sciences, and applied sciences like finance. Researchers greatly benefit from the ability to rapidly and inexpensively evaluate their hypothesis, before they are tested in practical experiments. The ability to improve simulation data is thus highly relevant, even beyond direct applications to machine learning algorithms.

The abstract notion of style in style transfer is inherently vague. We make the problem statement precise with the data-driven definition of style that defines style as the set of properties and characteristics that are inherent in a given dataset and set it apart from different data. These style features are, of course, highly specific to the exact task and application domain. In images style might be artistic style and medium, in text it might be word choice and grammar, and in time series it might be the level and kind of noise. Although it is sometimes possible to exactly define the desired task-specific style



(a) Image Style Transfer (b) Time Series Style Transfer

Figure 1. Examples of style transfer on different data. For images, we combine the contents of a photo (TU Kaiserslautern, Twitter @uni_kl, 19.12.2019) and the artistic style of a painting (Composition VI by Wassily Kandinsky, 1913). For time series, we combine the contents of a smooth Gaussian process sample with the style of a noisy sample of the same process.

characteristics, this becomes infeasible for complex styles such as the abstract style of realism. In these cases, Neural Style Transfer methods provide an elegant solution by defining style over a set of learned features. This allows us to automatically extract the style from a reference style sample or dataset, even for complex style transfer tasks.

Neural Style Transfer was first by proposed on images by Gatys et al. [1], based on convolutional features learned during an image classification task. They formulated style transfer as an optimization problem over the image pixel values and used an iterative optimization based on back-propagated gradients to obtain stylized images. Recently, this technique has been adapted for time series. Figure 1 illustrates style transfer on both kinds of data. While style transfer has been widely adopted in images, text, and audio applications, where recent model-based approaches perform effective real-time style transfer [2], the work on time series style transfer has been limited to basic iterative approaches.

1.2 Contribution

In this thesis, we attempt to close this performance gap by building on ideas from recent advances in text and image style transfer. We develop a model-based probabilistic framework that enables style transfer for time series tasks across a broad range of diverse domains. To this end, we propose a deep generative model that is specialized on stylized generation and disentangles the content and style information in its latent space. This allows for fast and straight-forward inference through latent swapping: We first encode the content and style samples into the model's latent space, and then use the generative model to decode a stylized time series from a recombination of the original content latent variables with the new style latent variables. Specifically, we use a variational autoencoder that is based on the modern transformer architecture. The desired disentanglement is obtained with a newly proposed technique that leverages the two losses specific to the style-transfer problem formulation.

1.3 Notation

We follow standard machine learning notation. When $x \sim P_x$ and $P_x \ll \lambda$, i.e. the distribution P_x of x is absolutely continuous w.r.t. a measure λ, we denote the Radon–Nikodym derivative of P_x w.r.t. λ with $p(x)$. Then, by definition,

$$P_x(A) = \int_A p(x)\, d\lambda(x).$$

If clear from the context, we use P_x and $p(x)$ interchangeably, for example, writing $x \sim p(x)$. If x is an (absolutely) continuous random variable (w.r.t. the Lebesgue measure), $p(x)$ is the probability density function. If x is a discrete random variable, then it is absolutely continuous w.r.t. the counting measure. In this case, $p(x)$ is the probability mass function. Throughout this thesis, we will mostly work with continuous random variables but we note that many arguments hold more generally. For example, Bayes' theorem

$$p(\theta|x) = \frac{p(x|\theta)p(\theta)}{p(x)} = \frac{p(x|\theta)p(\theta)}{\int p(x|\theta)p(\theta)d\lambda(x)}$$

holds as long as all Radon–Nikodym derivatives are given w.r.t. the same underlying measure λ [3]. Additionally, we assume that all of our datasets are sampled independently

from a (possibly unknown) data distribution, i.e. $x_i \overset{\text{iid}}{\sim} p(x)$. Finally, we denote the concatenation of elements $a^{(k)} \in \mathbb{R}^{Q_k}$ to a vector with square brackets, i.e. for each k and q there exists exactly one $i(k, q)$ with

$$[a^{(1)}, a^{(2)}, \ldots, a^{(n)}]_{i(k,q)} = a_q^{(k)}.$$

1.4 Outline

This thesis is structured as follows. In chapter 2 we lay the mathematical foundation for our method. We explore deep generative models, variational autoencoders, and the transformer architecture. Then, in chapter 3, we formulate the style transfer task as an optimization problem and discuss related work. Chapter 4 contains an in-depth description of our proposed method. Next, we show the effectiveness of our method. To this end, we design and evaluate extensive experiments in chapter 5. The thesis ends with a brief summary and an outlook on future work.

Chapter 2

Background

In this chapter we introduce the mathematical concepts that are relevant for our style transfer method. We formulate deep generative models and variational autoencoders, deriving them from first principles and giving clear intuitions. Finally, we discuss the several kinds of neural networks relevant to our method, including a detailed look at Long Short-Term Memory (LSTM) and transformers.

2.1 Autoencoder

Many complex datasets can be characterised sufficiently accurate by a small set of features. For example, a dataset of images containing faces might be described by, among others, eye and hair colors, size and relative position of facial features like mouth or nose, and pose. This description represents the contained information much more densely and meaningfully then the high-dimensional representation in terms of pixel values. Machine learning applications rely heavily on such sets of representative features as they are useful in a variety of tasks, e.g. for classification, clustering, regression or visualization. Instead of creating useful features by hand, neural networks can be used to learn good features. Using optimization techniques, for example based on gradients, they are trained directly to adapt to the desired task. The inverse problem, generating a detailed data sample from a limited description given by a set of features, also has many applications, e.g. for generation of realistic synthetic data or data compression. In the latter, our goal is to first encode data into a dense representation and then to decode it, matching the original as closely as possible. One particular approach to this task is to use a neural autoencoder (AE). Here, both feature extraction and generation are parameterized with a neural network, the encoder g_ψ and decoder f_θ, respectively. For a given data sample x, the goal is to minimize the reconstruction error between the original x and the reconstruction

$$x' = f_\theta(g_\psi(x)),$$

as measured by a reconstruction loss $\mathcal{L}(x, x')$. Over a dataset $X = \{x_1, x_2, \ldots, x_N\}$, we can, for example, use the Mean Squared Error (MSE), minimizing:

$$\begin{aligned} \mathcal{L}(X) &\coloneqq \frac{1}{N} \sum_{i=1}^{N} \|x_i - x_i'\|_2^2 \\ x_i' &= f_\theta(g_\psi(x_i)) \end{aligned} \tag{2.1}$$

Finding good parameters is straight forward: After specifying the architecture of the encoder and decoder, e.g. as a Convolutional Neural Network (CNN), we minimize the reconstruction loss over the given dataset, e.g. iteratively using back-propagated gradients.

2.2 VAE

Using the above autoencoder, in particular for data generation tasks, has several issues. The latent space, to which data is mapped to, lacks interpretability and structure and there is no straightforward way to generate new samples. The decoder is trained to work well only for data that is represented sufficiently by the training set. With increased distance to the points in the latent space representing seen data, the quality of decoded points deteriorates rapidly. This may suffice for simple compression but is unsuitable for generation. To deal with these issues we introduce the variational autoencoder (VAE), which was first described by Kingma et al. [4]. Based on a generative model of the data, it has two advantages: It allows for easy and interpretable generation and it regularizes the latent space, yielding more robust reconstructions.

First, we define a stochastic procedure to generate data. This (deep) generative model should be expressive enough to handle the complexity of data generation. Parameterized by θ, it needs to be flexible enough to allow us to find values for θ that are well adapted to the given training data. We model both the observed variable x and the latent variable z as random variables, with a joint distribution $p(x, z)$ specified by

$$\begin{aligned} z &\sim p_\theta(z) \\ x|z &\sim p_\theta(x|z). \end{aligned}$$

During generation, we first sample a point z in the latent space, yielding a low-dimensional representation. Without further information this is done by sampling from the prior $p_\theta(z)$.

This representation now informs the conditional distribution $p(x|z)$ over the data domain, from which a data point x can be sampled from. Usually, the prior is chosen to be simple and structured as to ensure the interpretability of the latent space. A common choice, for example, is a vector of independent standard normal random variables. Complexity arises from the conditional distribution, which can, for example, be defined using a neural network f_θ with parameters θ, yielding the deep generative model

$$\begin{aligned} z &\sim \mathcal{N}(0, \mathcal{I}) \\ x|z &\sim \mathcal{N}(f_\theta(z), \sigma^2 \mathcal{I}). \end{aligned}$$

Here, f_θ, similar to section 2.1, has the role of a decoder. We are, however, now able to additionally model the uncertainty, in this case, by an isometric Gaussian. Conversely, if we want to encode an observed x, we can compute the posterior $p(z|x)$. This again allows us to find the best latent encoding while also quantifying the prediction uncertainty. In a sense, the posterior thus has the role of an encoder. Unfortunately, for more complicated conditional distributions, the posterior is intractable due to the integral in

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)} = \frac{p(x|z)p(z)}{\int p(x|z)p(z)\, dx}.$$

To circumvent this problem, we use variational inference to approximate the posterior, selecting the best approximation from a family of simpler distributions Q. This family can, for example, be a Gaussian family that is parameterised by a neural network with parameters ψ, i.e. for each $q \in Q$ there exists ψ with

$$q_\psi(z) = \mathcal{N}(g_\psi(x), h_\psi(x)).$$

The exact details are left for later, but generally, we can first encode x into a meaningful feature vector and then apply several linear layers to extract the needed parameters. In the above case, if q_ψ is to be a vector of independent Gaussians, those are the elementwise mean and standard deviations. Now, we need to find an appropriate $q \in Q$ that best represents the posterior. The quality of this approximation has to be measured by a distance between two distributions. Here, it is very intuitive to use the Kullback–Leibler (KL) divergence [5], which is based on considerations from information theory detailed in

section 2.2.2. Defined through

$$\mathrm{KL}(q \,\|\, p) \coloneqq \mathrm{E}_{x\sim q(x)}[\log q(x) - \log p(x)] = \int q(x) \log \frac{q(x)}{p(x)}\, dx, \tag{2.2}$$

the KL divergence measures the efficiency of approximating p by q, essentially as the amount of redundant information that q introduces. Thus, a good posterior approximation is given by

$$q^* = \mathrm{argmin}_{q\in Q}\, \mathrm{KL}(q \,\|\, p_\theta(z|x)). \tag{2.3}$$

So far, we have described generation and inference but neglected to mention how to select the parameters θ both are based on. If we want to fit this model to a dataset $X = \{x_1, x_2, \ldots, x_N\}$ with $x_i \overset{\text{iid}}{\sim} p(x)$, we need to maximize the marginal (log-)likelihood

$$\log p_\theta(X) = \sum_{i=1}^{N} \log p_\theta(x_i) = \sum_{i=1}^{N} \int \log p_\theta(x_i, z_i)\, dz_i = \sum_{i=1}^{N} \int \log p_\theta(x_i|z_i) p(z_i)\, dz_i$$

Similarly to the posterior, for the complicated conditional distributions needed in practice, this marginal likelihood is intractable . To solve this problem, we introduce a lower bound that is easier to estimate and optimize: the evidence lower bound (ELBO), which for a posterior approximation q is defined as

$$\mathcal{L}_\theta(q) \coloneqq \mathrm{E}_{z\sim q}\left[\log \frac{p_\theta(x, z)}{q(z)}\right].$$

Note that

$$\log p_\theta(x) = \mathrm{E}_{z\sim q}[\log p_\theta(x)] = \mathrm{E}_{z\sim q}\left[\log \frac{p_\theta(x, z) q(z)}{q(z) p_\theta(z|x)}\right] = \mathcal{L}_\theta(q) + \mathrm{KL}(q \,\|\, p_\theta(z|x)).$$

Therefore, $\log p_\theta(x) \geq \mathcal{L}(q, \theta)$, as the KL divergence is always non-negative. Furthermore, $\mathrm{argmax}_{(q,\theta)}\, \mathcal{L}_\theta(q) = (p_{\theta^*}(z|x), \theta^*)$ with $\theta^* = \mathrm{argmax}_\theta \log p_\theta(x)$. Maximizing $\mathcal{L}_\theta(q)$ thus yields the same optimal parameters θ^*. Restricting the optimization to choices over approximations $q \in Q$, we still obtain $\mathrm{argmax}_{(q\in Q,\theta)}\, \mathcal{L}_\theta(q) = (\tilde{q}, \tilde{\theta})$ with

$$\log p_{\theta^*}(x) - \log p_{\tilde{\theta}}(x) \leq \mathrm{argmin}_{q\in Q}\, \mathrm{KL}(q \,\|\, p_\theta(z|x)). \tag{2.4}$$

This can be seen by first fixing θ^* and then selecting $q^* \in Q$ as in (2.3). By jointly

optimizing both the encoder and decoder, the approximation error can only get smaller. Generally, the error is small for a sufficiently expressive family Q. To find $\tilde{q}$ and $\tilde{\theta}$ consider, as a loss,

$$-\mathcal{L}_\theta(q) = \mathrm{E}_{z\sim q}\left[-\log \frac{p_\theta(x|z)p(z)}{q(z)}\right] = \mathrm{E}_{z\sim q}[-\log p_\theta(x|z)] + \mathrm{KL}(q \,\|\, p(z)). \tag{2.5}$$

Both terms on the right-hand side of (2.5) can be determined or estimated in a way that allows for efficient minimization of $-\mathcal{L}_\theta(q)$, e.g. through gradient descent. The second term includes a KL divergence, which can, in most cases, be computed analytically. For examples refer to section 2.2.1. The first term, containing an expectation, can be estimated effectively with Monte-Carlo simulation by drawing samples from q. In practice, taking only one sample per iteration suffices. To allow for back-propagation of gradients not only through $p_\theta(x|z)$ but also q, $q(z)$ has to be re-parameterized, separating stochastic and deterministic effects. For example, if $z \sim q_\psi(z) = \mathcal{N}(g_\psi(x), h_\psi(x))$, an equivalent formulation would be $z = g_\psi(x) + \sqrt{h_\psi(x)}\, w$, $w \sim \mathcal{N}(0, \mathcal{I})$, where $\sqrt{h_\psi(x)}$ is chosen to satisfy $\sqrt{h_\psi(x)}\sqrt{h_\psi(x)}^T = h_\psi(x)$. This is possible because the covariance matrix is symmetric and positive semidefinite. Equation (2.5) is similar to the loss used for the autoencoder in section 2.1. Indeed, the first term can be understood as a reconstruction loss between x and its reconstruction

$$\begin{aligned} x' &= f_\theta(g_\psi(x) + w_1) + w_2, \\ w_1 \sim \mathcal{N}(0, h_\psi(x)), &\, w_2 \sim \mathcal{N}(0, \sigma^2\mathcal{I}), \end{aligned}$$

while the second term has the function of a regularizer, enforcing structure in the latent space. This notion of structure is based on the prior used. For instance, an independent prior like $p(z) = \mathcal{N}(0, \mathcal{I})$ encourages more disentangled latent representations. The trade-off between reconstruction error and latent structure can be controlled by introducing a regularization parameter β, and then finding

$$\min_{q,\theta} \; \mathrm{E}_{z\sim q}[-\log p_\theta(x|z)] + \beta\, \mathrm{KL}(q \,\|\, p(z)),$$

which is the Lagrangian of, and therefore equivalent to,

$$\begin{aligned} \min_{q,\theta} \;& \mathrm{E}_{z\sim q}[-\log p_\theta(x|z)] \\ \text{s.t.} \;& \mathrm{KL}(q \,\|\, p(z)) \leq \epsilon_\beta. \end{aligned}$$

As β increases, ϵ_β decreases and allows for less deviations from the intended structure. This related formulation is called the β-VAE [6] and often arises naturally from the generative model, giving additional interpretation to β. For an example see section 2.2.1.

For larger datasets it is not feasible to optimize $\log p_\theta(X) = \sum_{i=1}^{N} \log p_\theta(x_i)$ directly because computing $\tilde{\theta}_i$ for every x_i is very time consuming. Instead, amortized variational inference can be used to find only one set of parameters $\bar{\theta}$ that works reasonably well for all x_i. The introduced amortization error is small when $\tilde{q}$ is expressive and flexible enough to be able to find good latent parameters z_i for every x_i. How we can further reduce the suboptimality in inference is still subject of ongoing research. Extensive analysis, for example by Cremer et al. [7] suggests that, generally, the approximation error is smaller than the amortization error. This can be attributed to the fact that, as q and θ are jointly optimized, the generator is able to adapt to the choice of approximation, producing true posteriors $p_\theta(z|x)$ that can be approximated with less error. In this case, the approximation error might be drastically lower than the upper bound in (2.4). Additionally, the approximation error can be improved by allowing for a more expressive class of approximations, for example, by using Normalizing Flows [8]. Cremer et al. showed that this increased capacity also reduces the amortization error. Another set of techniques that reduce the amortization error are iterative approaches, which during inference further improve the amortized parameters through individual optimization of each $q(x_i)$ [9, 10].

2.2.1 VAE Loss for Gaussians

We will now take a more detailed look at (2.5), the loss used to train a VAE. As stated above, it consists of two parts: the reconstruction loss $\mathcal{L}_{\text{rec}}(x) \coloneqq \mathrm{E}_{z\sim q}[-\log p_\theta(x|z)]$ and the KL loss $\mathcal{L}_{\text{KL}}(x) \coloneqq \mathrm{KL}(q \,\|\, p(z))$, which acts as a regularizer. Consider the case already discussed in section 2.2: a Gaussian model for generation with a Gaussian approximation to the posterior:

$$\begin{aligned} z &\sim \mathcal{N}(0, \mathcal{I}) \\ x|z &\sim \mathcal{N}(f_\theta(z), \sigma^2 \mathcal{I}) \\ q_\psi(z) &= \mathcal{N}(g_\psi(x), h_\psi(x)) \end{aligned} \tag{2.6}$$

We derive both loss terms for this common example, which is also highly relevant to our method. The reconstruction loss can be estimated with

$$\begin{aligned}\mathcal{L}_{\text{rec}}(x) &= \frac{1}{2\sigma^2}\,\mathrm{E}_{z\sim q}\left[\|x - f_\theta(z)\|_2^2\right] + \frac{d}{2}\log(2\pi\sigma) \\ &= \frac{1}{2\sigma^2}\,\mathrm{E}_{w\sim\mathcal{N}(0,\mathcal{I})}\left[\left\|x - f_\theta(g_\psi(x) + \sqrt{h_\psi(x)}\,w)\right\|_2^2\right] + \text{const} \\ &\approx \frac{1}{2\sigma^2}\sum_{j=1}^{m}\left\|x - f_\theta(g_\psi(x) + \sqrt{h_\psi(x)}\,w_j)\right\|_2^2 + \text{const},\end{aligned}$$

where in the last step we introduced samples $w_j \overset{\text{iid}}{\sim} \mathcal{N}(0, \mathcal{I})$. Together with an iterative optimization technique, setting $m = 1$ suffices and allows us to optimize instead, by dropping the constant,

$$\begin{aligned}\hat{\mathcal{L}}_{\text{rec}}(x) &\coloneqq \frac{1}{2\sigma^2}\|x - x'\|_2^2 \\ x' &= f_\theta(g_\psi(x) + w_1) \\ w_1 &\sim \mathcal{N}(0, h_\psi(x)).\end{aligned} \tag{2.7}$$

Comparing (2.7) with (2.1), this further illustrates how the VAE closely extends the ideas of the traditional autoencoder. Moreover, due to the multiplicative nature of σ^2, we can see that this VAE is equivalent to a β-VAE with $\sigma^2 = 1$ and $\beta = \sigma^2$. This contextualizes the reconstruction-structure trade-off inherent to the VAE formulation: Higher values of β encourage better latent structure but also decrease reconstruction quality, expressed here directly as increased generational uncertainty.

To find the regularizer $\mathcal{L}_{\text{KL}}$ we have to derive the KL divergence between two Gaussians.

Theorem 2.1
Let $p_1 = \mathcal{N}(\mu_1, \Sigma_1)$ and $p_2 = \mathcal{N}(\mu_2, \Sigma_2)$ be d-dimensional Gaussian distributions, then their KL divergence is given by

$$\mathrm{KL}(p_1 \,\|\, p_2) = \frac{1}{2}\left(\log\frac{|\Sigma_2|}{|\Sigma_1|} + \operatorname{trace}\{\Sigma_2^{-1}\Sigma_1\} + (\mu_2 - \mu_1)^T\Sigma_2^{-1}(\mu_2 - \mu_1) - d\right)$$

Proof.
The density of a multivariate Gaussian $X \sim \mathcal{N}(\mu, \Sigma)$ is given by

$$p(x) = \left((2\pi)^d |\Sigma|\right)^{-1/2} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right)$$

Thus,

$$\begin{aligned}
\mathrm{KL}(p_1 \,\|\, p_2) &= \mathrm{E}_{x\sim p_1}\left[\log \frac{p_1(x)}{p_2(x)}\right] \\
&= \frac{1}{2}\log\frac{|\Sigma_2|}{|\Sigma_1|} + \frac{1}{2}\,\mathrm{E}_{x\sim p_1}\left[(x-\mu_2)^T\Sigma_2^{-1}(x-\mu_2) - (x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1)\right] \\
&= \frac{1}{2}\log\frac{|\Sigma_2|}{|\Sigma_1|} + \frac{1}{2}\,\mathrm{E}_{x\sim p_1}\left[(x-\mu_2)^T\Sigma_2^{-1}(x-\mu_2)\right] \\
&\quad - \frac{1}{2}\,\mathrm{trace}\left\{\mathrm{E}_{x\sim p_1}\left[(x-\mu_1)(x-\mu_1)^T\right]\Sigma_1^{-1}\right\}.
\end{aligned}$$

The statement follows using the identities $\mathrm{E}[x^T A x] = \mathrm{E}[x]^T A\,\mathrm{E}[x] + \mathrm{trace}\{A\,\mathrm{E}[xx^T]\}$ and $\mathrm{E}[xx^T] = \mathrm{Cov}[x]$. □

Corollary 2.2
For $d = 1$, i.e. $p_1 = \mathcal{N}(\mu_1, \sigma_1^2)$ and $p_2 = \mathcal{N}(\mu_2, \sigma_2^2)$, we have

$$\mathrm{KL}(p_1 \,\|\, p_2) = \log\frac{\sigma_2}{\sigma_1} + \frac{\sigma_1^2 + (\mu_1-\mu_2)^2}{2\sigma_2^2} - \frac{1}{2}.$$

Thus, as the KL Loss, we obtain

$$\begin{aligned}
\mathcal{L}_{\mathrm{KL}}(x) &= \mathrm{KL}(\mathcal{N}(g_\psi(x), h_\psi(x)) \,\|\, \mathcal{N}(0, \mathcal{I})) \\
&= \frac{1}{2}\left(-\log|h_\psi(x)| + \mathrm{trace}\{h_\psi(x)\} + \|g_\psi(x)\|_2^2 - d\right).
\end{aligned}$$

If the elements of $X \sim \mathcal{N}(g_\psi(x), h_\psi(x))$ are constructed to be independent, with $X_i \sim \mathcal{N}(g_\psi(x)_i, h_\psi(x)_i^2)$, then $h_\psi(x) = \mathrm{Diag}(h_\psi(x)_1^2, \ldots, h_\psi(x)_d^2)$, and

$$\mathcal{L}_{\mathrm{KL}}(x) = \sum_{i=1}^{d} \mathcal{L}_{\mathrm{KL}}(x_i) = \sum_{i=1}^{d} -\log h_\psi(x)_i + \frac{h_\psi(x)_i^2 + g_\psi(x)_i^2}{2} - \frac{1}{2}.$$

2.2.2 KL Divergence and Mutual Information

As briefly mentioned above, the KL divergence $\mathrm{KL}(q \,\|\, p)$ measures the amount of redundant information when we approximate p with q. To make this statement more precise, we first quantify the information of a distribution by its entropy

$$\mathrm{H}(q) \coloneqq \mathrm{E}_{x\sim q(x)}[-\log q(x)] = \int -q(x) \log q(x)\, dx.$$

This definition goes back to Shannon [11] and is best understood in the context of optimal binary encodings: If we need to encode symbols whose occurrences can be modeled with a discrete distribution, than a symbol x which occurs with probability $q(x)$ should have an encoded length of roughly $-\log q(x)$ bits. More frequent symbols have shorter codes. Here, entropy is the average amount of bits we need to encode $x \sim q(x)$. In addition to contained information, entropy can also be understood as the uncertainty of data. Entropy is highest for uniform distributions where all possibilities are equally likely. If we know that some events are far more likely, the entropy is lower, and if only one outcome is possible, the entropy reaches the lowest possible value of 0. In the context of machine learning, we are often interested in how good a distribution represents an underlying true distribution. Mathematically this is made precise with the cross-entropy

$$\mathrm{H}_p(q) \coloneqq \mathrm{E}_{x\sim q(x)}[-\log p(x)] = \int -q(x) \log p(x)\, dx.$$

We again consider the context of binary encoding. Here, the cross-entropy is the average amount of bits we need to encode $x \sim q(x)$, using codes that are optimal for $p(x)$. If $q(x)$ is similar to $p(x)$ the encoding is still efficient, but if frequently occurring symbols in $q(x)$ are much less likely in $p(x)$ the encoding is far from optimal. Being able to quantify representation quality is extremely useful and many machine learning applications use cross-entropy minimization. The (binary) cross-entropy between two binomial distributions, for instance, is a very common loss in binary classification. Cross-entropy can also be used to quantify the quality of a distribution approximation. Indeed the KL divergence, defined in (2.2) for the same reason, is only a shifted cross-entropy with the property that $\mathrm{KL}(q \,||\, p) = 0$ iff p is equal to q:

$$\mathrm{KL}(q \,\|\, p) = \int -q(x)\Big(\log p(x) - \log q(x)\Big)\, dx = \mathrm{H}_p(q) - \mathrm{H}(q).$$

Having made precise the notion of information, we might also be interested in the information that is shared by two distributions. This will allow us to define a way to measure general correlation, beyond the simple linear correlation quantified by the Pearson correlation ρ. The common information of two distributions is defined by the joint entropy, the entropy of the joint distribution $m_{p,q}$, and never exceeds the marginal entropies:

$$\mathrm{H}(p,q) \coloneqq \mathrm{H}(m_{p,q}) \geq \max(\mathrm{H}(p), \mathrm{H}(q))$$

If p and q are independent, $\mathrm{H}(p,q) = \mathrm{H}(p) + \mathrm{H}(q)$, as the joint distribution is factorised: $m_{p,q}(x,y) = (p \otimes q)(x,y) = p(x)q(y)$. The conditional entropy

$$\mathrm{H}(p \,|\, q) \coloneqq \mathrm{H}(p,q) - \mathrm{H}(q) = \int -m_{p,q}(x,y) \log \frac{m_{p,q}(x,y)}{q(y)} \, dx \, dy$$

measures the information of p that cannot be explained through q. Subtracting the non-shared information, we can, finally, define the mutual information:

$$\mathrm{I}(p,q) = \mathrm{H}(p,q) - \mathrm{H}(p \,|\, q) - \mathrm{H}(q \,|\, p) = \int m_{p,q}(x,y) \log \frac{m_{p,q}(x,y)}{p(x)q(y)} \, dx \, dy.$$

As $\mathrm{H}(p) = \mathrm{I}(p,q) + \mathrm{H}(p \,|\, q)$, this also allows us to separate already explained and new information. Note that $\mathrm{I}(p,q) = \mathrm{KL}(m_{p,q} \,\|\, p \otimes q)$, illustrating that mutual information is the additional information that is not explained under the assumption of independence.

2.3 LSTM

There are several types of neural networks suitable for work with time series. Fully Connected (Feed-Forward) Neural Networks, Convolutional Neural Networks, and also Reccurent Neural Networks (RNN), a broad class of networks that are well suited to model the temporal and dynamic behaviour of time series. A particularly useful kind of RNN is Long Short-term Memory (LSTM) [12], which is capable of learning dependencies even over long distances. Generally, a RNN processes data sequentially, keeping an internal state, which allows the network to remember past inputs. After initializing the internal state to s_0 the output is obtained via

$$h_t, s_t = f_\theta(x_t, s_{t-1}),$$

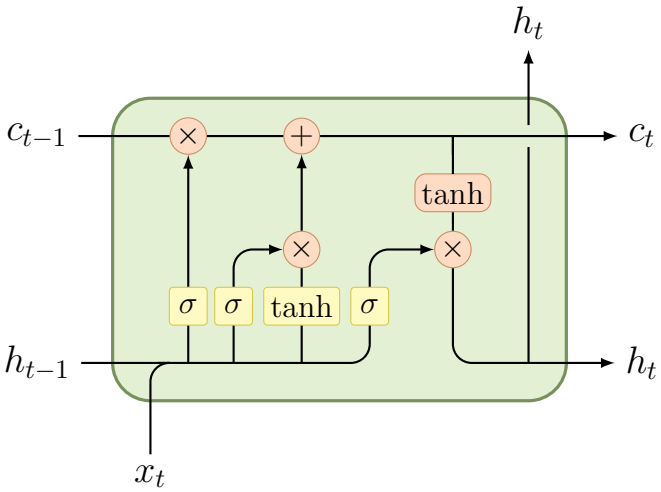


Figure 2. The LSTM architecture as described by Hochreiter et al. [12].

where h_t and s_t are the output and internal state at time t, respectively. In LSTMs there are two forms of recurrence. First, the most recent output is used as an input in the next time step, allowing for short-term memory; second, there is an internal state which can be written to and read from, allowing for long-term memory. With h_0 and c_0 initialized, e.g. to 0, the LSTM is defined by,

$$
\begin{aligned}
f_t &= \sigma(W_f[x_t, h_{t-1}]^T + b_f) \\
i_t &= \sigma(W_i[x_t, h_{t-1}]^T + b_i) \\
o_t &= \sigma(W_o[x_t, h_{t-1}]^T + b_o) \\
\tilde{c}_t &= \tanh(W_c[x_t, h_{t-1}]^T + b_c) \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \\
h_t &= o_t \odot \tanh(c_t),
\end{aligned}
$$

which is also illustrated in figure 2. One LSTM cell consist of 4 linear layers with sigmoid and hyperbolic tangent non-linearites. f_t controls how much of the internal state is forgotten and i_t how much of the proposed changes $\tilde{c}_t$ are remembered. Finally, o_t controls which parts of the memory are relevant in this time step and need to be output. At the time, LSTMs massively improved the state-of-the-art in many tasks, including speech recognition, machine translation, and time series prediction [13, 14]. BiLSTM [15], a bidirectional variant, combines the results of a forward and a backward pass by two different LSTMs and thus attends to the full context from both the past and the present.

2.4 Transformer

Another kind of neural network is the transformer. First described in 2017, in the seminal work by Vaswani et al. [16], this architecture has been proven to be both very powerful and flexible, advancing the state-of-the-art in many different tasks on text, images and beyond [17]. Based on the mechanism of self-attention, transformers avoid a lot of the inductive biases found in other neural networks, for example the locality inherent to CNNs. This generally leads to slower convergence but highly expressive, less biased models. Originally, transformers were designed to handle sequential data, similar to RNNs. Because transformers are permutation-invariant, they require an embedded representation of the data X, $X_{t\cdot} \in \mathbb{R}^d$ that also encodes position. This information can be included by adding a learned or designed positional encoding P, e.g.

$$P_{t,i} = \begin{cases} \sin(t/10^{\frac{3i}{d}}) \text{ if } i \bmod 2 = 0 \\ \cos(t/10^{\frac{3i}{d}}) \text{ else} \end{cases}. \tag{2.8}$$

The transformer architecture is mainly composed of two components: feed-forward layers and Multi-Head Attention. The latter is based on the (masked) Scaled Dot-Product Attention, which, given a set of keys K, values V and Queries Q, is defined as

$$\text{Attention}(K, V, Q) \coloneqq \text{softmax}\left(M \odot \frac{QK^T}{\sqrt{d_k}}\right) V,$$

where d_k is the dimension of keys and queries and M is a mask that can prevent the query $Q_{t\cdot}$ from accessing all values. This is essential in auto-regressive tasks like prediction and translation, where the query only considers the previous values $V_{t'\cdot}$, $t' < t$. Multi-Head Attention combines h attention heads that act on smaller learned subspaces with

$$\begin{aligned} \text{MultiHead}(K, V, Q) &\coloneqq [H_1^T, \ldots, H_h^T]^T W^O, \\ H_i &= \text{Attention}(KW_i^K, VW_i^V, QW_i^Q). \end{aligned}$$

Each feed-forward layer combines two linear layers with a ReLU non-linearity, i.e.

$$\text{FFN}(X) \coloneqq \max(0, XW_1 + b_1)W_2 + b_2.$$

Based on these blocks, the original transformer architecture [16] consists of a transformer encoder and a transformer decoder, each with N repeating transformer layers, as illus-

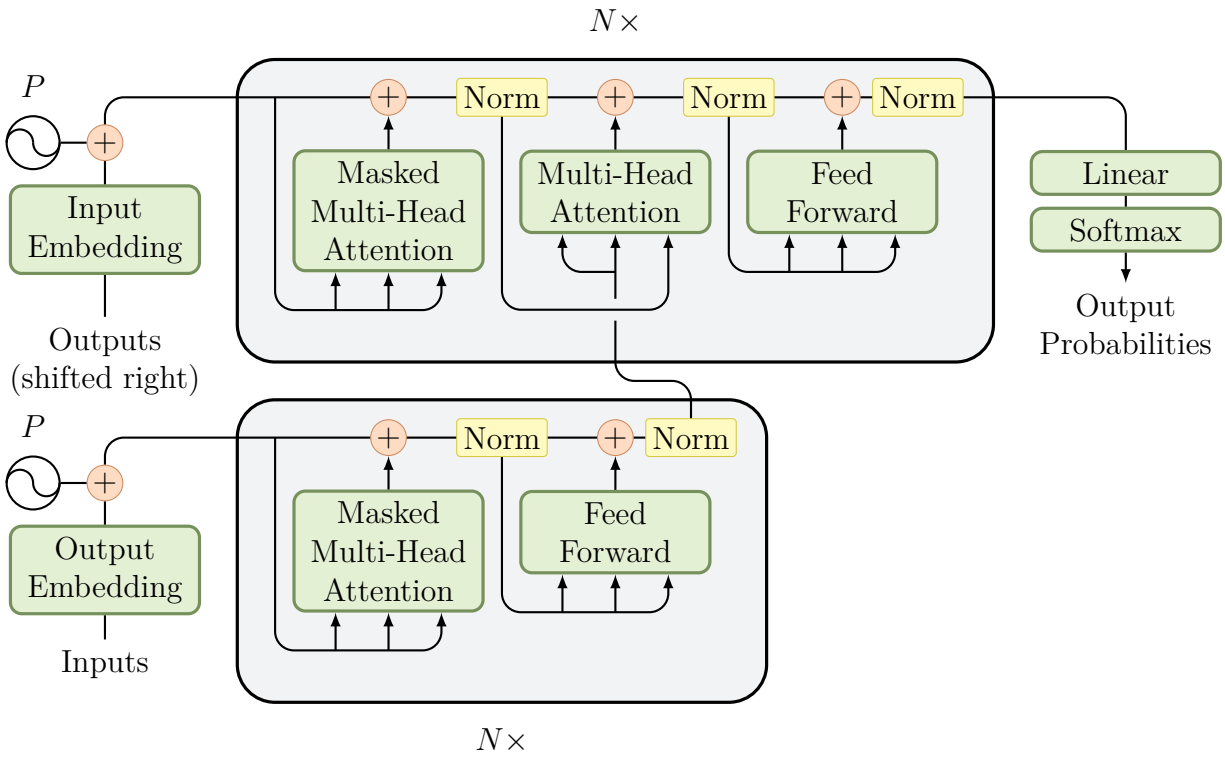


Figure 3. The transformer architecture as described by Vaswani et al. [16]. The model consists of two parts: the transformer encoder (below) and the transformer decoder (above).

trated in figure 3. The former encodes the inputs into a meaningful representation while the latter decodes this information and predicts the next-step output given all previous outputs. In many applications, the decoder is not needed and outputs are directly extracted from the learned representation. As can be seen, Multi-Head Attention is used in two ways: as self-attention with $K = V = Q$ and as encoder-decoder attention where both $K = V$ come from the decoder. To speed up training and avoid the problem of vanishing and exploding gradients, Layer Normalization [18] is used.

Chapter 3

Related Work

In this chapter we review relevant works from the field of (Neural) Style Transfer. We start by formulating style transfer as an optimization problem based on the content and style loss. Then follows a thorough review of image and text style transfer methods, including iterative and non-iterative approaches. Next, we discuss the available work on style transfer with time series. The chapter ends with a a brief introduction to the concept of disentangled representations and a survey of relevant transformer models.

3.1 Style Transfer

Style transfer is an inherently vague and ambiguous task. Before discussing different methods, we have to first formulate it mathematically by making precise the notion of style and content. This, of course, highly depends on the application and the kind of data. Informally, style can be defined as the distinctive characteristics inherent in (a set of) data, setting it apart from other data of the same kind. In text, this might be tone, choice of words, punctuation or sentiment; in images this might be composition, medium or color palette. There are two main ways of formalization: feature-based and distinction-based. The former, more commonly used approach, focuses on finding the characteristic features that define style in the desired application. Differences in style are then defined by differences in those features, and quality of stylized generation by the ability to match those features. The latter, on the other hand, focuses more broadly on the distinctiveness of style. Differences in style are then defined by the ability to distinguish between two styles, and the quality of stylized generation by its indistinguishability from real style samples. Many distinction-based methods use adversarial training, for example, Generative Adversarial Networks (GANs).

3.1.1 Image and Text Style Transfer

For now, we focus on style transfer with images and text, where style transfer is best understood and described. Many ideas, however, also generalize to different kinds of data, for example, to time series, as we discuss in section 3.1.2. Style transfer was first proposed on images. Here, some early feature-based methods were based on handcrafted features, e.g. hue and saturation [19], but their applications were limited. With the advent of learned features, which were first proposed by Gatys et al. [1] in 2016, stylized generation became feasible for more complex style transfer tasks. Furthermore, it allowed for indirect specification of the target style by providing style samples. Based on features from a CNN that was pre-trained on image classification, Gatys et al. define two losses

$$\begin{aligned} \mathcal{L}_c(x, y) &\propto \sum_i \|\phi_i(x) - \phi_i(y)\|_2^2 \\ \mathcal{L}_s(x, y) &\propto \sum_{i_1} \sum_{i_2} \left\| \phi_{i_1}(x)^{\mathrm{T}} \phi_{i_2}(x) - \phi_{i_1}(y)^{\mathrm{T}} \phi_{i_2}(y) \right\|_2^2, \end{aligned} \tag{3.1}$$

which measure similarity in content and style, respectively. Here, $\phi_i(x)_j$ is the activation of feature ϕ_i at position j of image x. If ϕ_i is a feature of a later convolutional layer it can detect high-level concepts, for example, doors or windows. Then, $\phi_i(x)$ captures where in the image these objects are located. Intuitively, in images that are close in content, the location of these objects is similar, which in turn, yields similar feature activations. Conversely, $\phi_{i_1}(x)^{\mathrm{T}} \phi_{i_2}(x)$, captures the interactions between pairs of features over the whole image. If its absolute value is high the concepts detected by ϕ_{i_1} and ϕ_{i_2} often occur together. This information reveals abstract properties that are not related to image content. If, for example, leaves and the color orange have a high co-occurrence, this likely means that the image was taken in fall. If we consider style in the artistic sense, it makes sense to compare the interactions of low-level features that are detected in earlier convolutional layers, since they might correspond to concepts like color, texture or brush stroke. Co-occurrences, however, are not the only way to formalize the notion of style. Any higher-order feature statistic could be a potential candidate. AdaIN [20] uses

$$\mathcal{L}_s \propto \sum_i \|\mu(\phi_i(x)) - \mu(\phi_i(y))\|_2^2 + \|\sigma(\phi_i(x)) - \sigma(\phi_i(y))\|_2^2 \tag{3.2}$$

arguing that style is best described by the presence and absence of features. If, for example, a painting mostly consists of straight black lines without any rounded corners, these concepts are detected more (or less) frequently. Thus the style information is represented

well by the corresponding feature means and standard deviations. Note that the losses defined above can be used for more than just images, as long as suitable features are designed or learned. After choosing appropriate features, content loss, and style loss, we can formulate style transfer as an optimization problem: Given a data sample x and a style distribution $p(s)$, find the stylized sample

$$y' = \min_y \lambda_c \mathcal{L}_c(x, y) + \lambda_s \, \mathrm{E}_{s \sim p(s)} \left[\mathcal{L}_s(s, y) \right],$$

which is close in style to $s \sim p(s)$ while still preserving the content of x. Here λ_c and λ_s are hyperparameters that allow for control over the trade-off between content preservation and stylization. The above expectation can be approximated by

$$y' = \min_y \lambda_c \mathcal{L}_c(x, y) + \lambda_s \sum_{i=1}^{N_s} \mathcal{L}_s(s_i, y) \tag{3.3}$$

if a set $S = \{s_1, \ldots, s_{N_s}\}$ of $N_s \geq 1$ style samples $s_i \overset{\text{iid}}{\sim} p(s)$ is given. Usually, only one sample suffices. There are many different approaches to find solutions to (3.3) or similar optimization problems. A detailed review of relevant style transfer methods is given by Jing et al. [21]. Generally, the approaches fall into two categories: data-based and model-based. Data-based methods optimize (3.3) directly. Gatys et al. [1], for example, initialize y randomly and use gradient descent with back-propagated gradients. This is slow, as it needs to be repeated for every new generated sample. Model-based approaches, on the other hand, generate stylized samples directly, training an end-to-end model with data samples $x \sim p(x)$. Either, with a fixed style, through $y' = f_\theta(x)$ and via

$$\min_\theta \mathrm{E}_{x \sim p(x)} \left[\lambda_c \mathcal{L}_c(x, f_\theta(x)) + \lambda_s \sum_{i=1}^{N_s} \mathcal{L}_s(s_i, f_\theta(x)) \right]. \tag{3.4}$$

or, with arbitrary style, through $y' = f_\theta(x, S)$ and via

$$\min_\theta \mathrm{E}_{x \sim p(x)} \left[\lambda_c \mathcal{L}_c(x, f_\theta(x, S)) + \lambda_s \sum_{i=1}^{N_s} \mathcal{L}_s(s_i, f_\theta(x, S)) \right]. \tag{3.5}$$

After initial training this allows for fast, real-time style transfer. On images, arbitrary style transfer has been implemented, for example, with CNNs [2, 20] and with transformers [22]. Inspired in part by the promising results of image style transfer, many methods have been proposed on other kinds of data. For text data, available methods are reviewed

thoroughly, for example, by Jin et al. [23]. Because style in text is less distinct than in images, it is usually extracted from a larger a corpus of texts. Thus, most methods tackle fixed style transfer as in (3.4). When we have parallel data, i.e. pairs (x_i, y_i) an end-to-end model can be trained directly via

$$\min_{\theta} \sum_{i=1}^{N} \mathcal{L}(y_i, f_\theta(x_i)),$$

where $\mathcal{L}$ is a task-specific reconstruction loss, for instance, a weighted sum of the above content and style loss. While parallel datasets are rare for images, in text they are available for a few transfer tasks between, among others, informal and formal [24], complicated and simple [25], or biased and neutral [26] language. Nonetheless, there are many tasks where only non-parallel data is available. In this setting, a recent approach [27] cascades two transformers in an encoder-decoder setup, first pre-training with a cloze task and then finetuning on the target style corpus. Through this training procedure, the decoder learns to generate stylized text based on the abstract representation encoded by the encoder, even when we input new text samples of a different style.

More directly and also without the need to define features, distinction-based style transfer methods try to generate stylized data that is indistinguishable from real data. For example, we might have access to a discriminator D that predicts the probability of data belonging to the style distribution $p(s)$ and want to train our generator, a neural network G_θ, to produce stylized versions of its input $x \sim p(x)$, i.e. $y' = G_\theta(x)$. In this case, we can train G_θ, for example, by finding

$$\min_{\theta} \mathrm{E}_{x \sim p(x)}[\log(1 - D(G_\theta(x)))].$$

Generally, such a discriminator is not available or easy to break in unintended ways. GANs therefore use adversarial training, jointly training a discriminator D_ψ with

$$\min_{\theta} \max_{\psi} \mathrm{E}_{s \sim p(s)}[\log(D_\psi(s)] + \mathrm{E}_{x \sim p(x)}[\log(1 - D_\psi(G_\theta(x)))]$$

until an equilibrium is reached. Ideally, D_ψ should then no longer be able to distinguish between $s \sim p(s)$ and $y' = G_\theta(x), x \sim p(x)$. One prominent example of a GAN that performs image style transfer on non-parallel data is CycleGAN [28], which also trains an inverse mapping F_θ and enforces the cycle consistencies $F_\theta(G_\theta(x)) = x$ and $G_\theta(F_\theta(y)) = y$.

3.1.2 Time Series Style Transfer

Thus far, the application of style transfer methods to time series has been very limited. Most approaches adopt feature-based methods from image style transfer, for example by directly transforming time series to images [29]. This is not very promising in general, as features trained on image classification are not well-suited for most time series applications. When using a perceptual loss such as (3.1) the choice of features is crucial. El-Lahman et al. [30] propose to use hand-crafted features, specifically designed to their application in finance. They define content by a running average that captures the trend and formalize style over the autocorrelation of the log-returns, volatility, and the power spectral density. Da Silva et al. [31] use learned features that are obtained via a denoising autoencoder trained on the source dataset $x \sim p(x)$. Both methods are data-based, iteratively improving a random initialization with back-propagated gradients.

When samples of the target style are abundant, stylized time series may be generated directly, for example with VAEs [31, 32] or GANs [33, 34]. This, however, sacrifices control over the content. Another way to transfer style is to use an autoencoder with a disentangled representation. Broadly speaking, if the latent space fully separates content and style with

$$x' = f_\theta(g^c_\psi(x), g^s_\psi(x)),$$

the style latent variable $z_s = g^s_\psi(x)$ can be substituted, during inference, by an encoded style sample $s \sim p(s)$ with

$$y' = f_\theta(g^c_\psi(x), g^s_\psi(s)),$$

which changes the style of the generated data. For high-dimensional time series such as video and speech, disentanglement can be obtained with a Disentangled Sequential Autoencoder (DSAE) [35], a VAE that splits the time series into global information (style) and time-dependent dynamics (content). If the dimension of the latent space encoding the dynamics is small enough, a good reconstruction is only possible by storing the time-invariant information in the latent space that encodes the global information. After careful calibration this enforces a disentanglement that allows for inference-time style transfer. Disentanglement can also be obtained more directly by introducing a regularizer that penalizes entanglement, for example, quantified using the mutual information [36].

3.2 Transformer

Transformer models have advanced the state-of-the-art in many applications [17] and achieve performance unparalleled by other architectures. In particular on text-based natural language processing (NLP) tasks, where they have been first proposed, they have become the de-facto standard. Here, GPT [37] and BERT [38] are powerful and flexible models that allow for rapid development to new applications by pretraining a transformer encoder on an enormous corpus of text. This encoder provides a task-independent representation of the data which can be leveraged by subsequently training smaller task-specific heads. GPT-3 [39], a more robust and more powerful successor to GPT has a staggering 175 billion parameters and excells in many NLP tasks, including but not limited to, question answering, essays writing, text summarization, translation, and guided generation of specific text like computer code [40]. Transformers have also been adapted for many computer vision tasks. In lieu of learned word embeddings, embedding techniques based on CNNs are used. Han et al. [41] and Khan et al. [42] provide detailed reviews of possible applications. These include, for example, classification [43], image segmentation [44], object detection [45], and image generation from text descriptions [46, 47]. Pretraining on a large image dataset can again be useful to reduce time and data requirements by providing a highly informative representation, for instance with iGPT [37]. Use of transformers in time series tasks is thus far mostly limited to regression, classification, time series forecasting, and their applications [48–50]. Here, the embeddings needed for the transformers are obtained by linear projection.

Chapter 4

Method

In this chapter we present our proposed style transfer method: a variational autoencoder with disentangled latent space that is fast and well-suited for style transfer tasks involving time series. We begin with a detailed discussion of both the deep generative model our method is build on, and the model architecture. Then, we detail how stylized samples are obtained through disentanglement. We conclude by describing our training procedure.

4.1 Generative Model

We need a generative model that allows for fine control and sufficiently complex generation. Thus, a variational autoencoder with a deep generative model is a sensible choice. The commonly used Gaussian model (2.6), however, is not suitable for style transfer tasks, because it entails minimizing a MSE Loss, as shown in section 2.2.1. Reconstruction and generation thus focus heavily on content preservation, accurately capturing the general trend but ignoring stylization. Generally, the reconstruction loss effects which properties of the reconstruction are prioritized. This makes a careful choice essential. A suitable reconstruction loss for stylized generation is the perceptual loss

$$\mathcal{L}(x, x') \coloneqq \lambda_c \mathcal{L}_c(x, x') + \lambda_s \mathcal{L}_s(x, x').$$

It considers both content and style alignment and allows for fine control with the weights λ_c and λ_s. The choice of a generative model for our VAE is similarly important. We want to select a corresponding model that allows for the same emphasis and control over stylized generation. To find this generative model, consider the more general

$$\begin{aligned} z &\sim \mathcal{N}(0, \mathcal{I}) \\ \eta(x)|z &\sim \mathcal{N}(f'_\theta(z), \sigma^2 \mathcal{I}). \end{aligned}$$

Here, a conditional distribution $p_\theta(y|z)$ of the transformed data $y = \eta(x)$ is given. If η is continuous and invertable this uniquely defines the conditional distribution $p_\theta(x|z)$ by setting $g = \eta^{-1}$ in the following proposition. If for example $\eta = \log$, then $p_\theta(x|z)$ has a log-normal distribution.

Proposition 4.1
Let y be an absolutely continuous random variable with density f_y. If g is continuously invertable then $x = g(y)$ is also absolutely continuous with density

$$f_x(x) = f_y(g^{-1}(x))\left|\frac{d}{dx}g^{-1}(x)\right|.$$

If η is not invertable the unique existence of $p_\theta(x|z)$ cannot be guaranteed. However, if η is approximately invertable instead, i.e. if there exists an approximation $\tilde{\eta}^{-1}$ with $\tilde{x} \coloneqq \tilde{\eta}^{-1}(\eta(x)) \approx x$, then the distribution $p(\tilde{x}|z)$ is close to any potential $p(x|z)$. Whereas in the model (2.6) samples of the conditional distribution are obtained with

$$x' = f_\theta(z_j) + w,\ w \sim \mathcal{N}(0, \sigma^2\mathcal{I}),$$

we now sample from the (approximate) conditional distribution $p(\tilde{x}|z)$ with

$$x' = \tilde{\eta}^{-1}(\eta(f_\theta(z_j)) + w),\ w \sim \mathcal{N}(0, \sigma^2\mathcal{I}),$$

where we have additionally set $f'_\theta(z) = \eta(f_\theta(z))$. The former adds noise directly to our observation. This has a profound negative impact, because noisy samples, which have a different style, are modeled with a high likelihood. Conversely, the latter adds more complex noise by varying abstract properties. Instead of a smooth and blurry $f_\theta(z)$ that averages out potential reconstructions, we obtain a sample $f_\theta(z)$ that has average features, in a way, the most representative reconstruction possible. Intuitively, if η encodes content and style features this puts high likelihood on samples with similar content and style. To find good parameters θ, we maximize the likelihood $p_\theta(\eta(x)|z)$ as in section 2.2 by approximating the posterior $p_\theta(z|\eta(x))$ with $q_\psi(z) = \mathcal{N}(g_\psi(x), h_\psi(x))$. Thus, we minimize

$$\min_{\theta,\psi}\ \hat{\mathcal{L}}_{\text{rec}}(x) + \mathcal{L}_{\text{KL}}(x) = \frac{1}{2\sigma^2}\sum_{j=1}^{m}\|\eta(x) - \eta(f_\theta(z_j))\|_2^2 + \text{KL}(q_\psi \,\|\, p(z)).$$

Recall that $f_\theta(z_j)$ is not a direct reconstruction of x but instead an indirect reconstruction with matching properties close to $\eta(x)$. If η is approximately invertable, this also implies $f_\theta(z) \approx x$. To relate this model back to the style transfer task, we specify $\eta(x)$ based on (convolutional) feature activations $\phi_i(x)$, $1 \leq i \leq N_\phi$. Specifically,

$$\eta^c_{ij}(x) = \phi_i(x)_j$$

and

$$\eta^s_j(x) = \begin{cases} \mu(\phi_i(x)),\ j = i & \text{if } 0 < j \leq N_\phi \\ \sigma(\phi_i(x)),\ j = 2i & \text{if } N_\phi < j \leq 2N_\phi \end{cases}.$$

We discuss feature choice more thoroughly in section 5.8.1, but briefly note here that for convolutional features that are learned by a neural autoencoder, $\eta = [\eta^c, \eta^s]$ is approximately invertable. Finally, we propose the deep generative model

$$\begin{aligned} z &\sim \mathcal{N}(0, \mathcal{I}) \\ \eta^c(x)|z &\sim \mathcal{N}(\eta^c(f_\theta(z)), \sigma_c^2 \mathcal{I}) \\ \eta^s(x)|z &\sim \mathcal{N}(\eta^s(f_\theta(z)), \sigma_s^2 \mathcal{I}) \\ q_\psi(z) &= \mathcal{N}(g_\psi(x), h_\psi(x)) \end{aligned} \tag{4.1}$$

which we can fit to a dataset $X = \{x_1, x_2, \ldots, x_N\}$ by minimizing the β-VAE loss

$$\frac{1}{N} \sum_{i=1}^{N} \min_{\theta,\psi} \lambda_c \mathcal{L}_c(x_i) + \lambda_s \mathcal{L}_s(x_i) + \lambda_{\text{KL}} \mathcal{L}_{\text{KL}}(x_i), \tag{4.2}$$

where, as in (3.1) and (3.2) and with $z \sim q_\psi(z)$,

$$\begin{aligned} \mathcal{L}_c(x) &= \sum_i \|\phi_i(x) - \phi_i(f_\theta(z))\|_2^2 \\ \mathcal{L}_s(x) &= \sum_i \|\mu(\phi_i(x)) - \mu(\phi_i(f_\theta(z)))\|_2^2 + \|\sigma(\phi_i(x)) - \sigma(\phi_i(f_\theta(z)))\|_2^2. \end{aligned}$$

The loss (4.2) shows that (4.1) indeed is the corresponding model we have been looking for, allowing for fine control over the relative importance of content, style, and latent structure. Note that following the discussion in section 2.2.1, the above β-VAE formulation with

$\beta = \lambda_{\text{KL}}$ is equivalent to a VAE formulation with

$$\sigma_c^2 = \frac{\lambda_{\text{KL}}}{\lambda_c} \text{ and } \sigma_s^2 = \frac{\lambda_{\text{KL}}}{\lambda_s}.$$

Higher values of λ_s and λ_c thus encourage increased reconstruction quality through decreased feature variation, but also cause a less structured latent space. In the following we scale the weights with

$$\lambda_c' = \frac{\lambda_c}{d},\ \lambda_s' = \frac{\lambda_s}{d},\ \lambda_{\text{KL}}' = \frac{\lambda_{\text{KL}}}{d_{\text{latent}}}$$

to make them less dependent on d and d_{latent}, the dimension of the data and the latent space, respectively.

4.2 Model Architecture

To complete the specification of the VAE model discussed in the previous section, we have to define the encoders g_ψ and h_ψ, and the decoder f_θ. We use transformer models for all three. This allows our model to attend to long range dependencies in the data, and therefore to quickly generate correctly stylized time series even for complex styles. An overview of our model architecture can be found in figure 4. Similar to GPT [51] and BERT [38] models, we only use a transformer encoder, consisting of N_l transformer layers. Because we want to consider the full bidirectional context at each position we do not mask any connections in the self-attention layers. Following an analysis by Xiong et al. [52], we change the position of the layer normalization. Normalizing before applying the attention and feed-forward layer makes transformer training more stable and less dependent on the learning rate schedule. Before we can propagate our m-dimensional time series $x \in \mathbb{R}^{d_{\text{len}} \times m}$ through the transformer architecture, we first have to embed it into a higher-dimensional representation $\bar{x} \in \mathbb{R}^{d_{\text{len}} \times d_{\text{model}}}$. In recent works [48, 49] that apply transformer to time series, a simple linear projection is used. We found that additionally encoding context information using a convolution v_θ with a small kernel size of 5, improves training speed significantly. To this representation we add positional information with the sinusoidal positional encoding P from equation (2.8).

Similar to DSAE [35], our model allows for style transfer by disentangling content and style in the latent space. The latent space thus consists of a time-dependent compo-

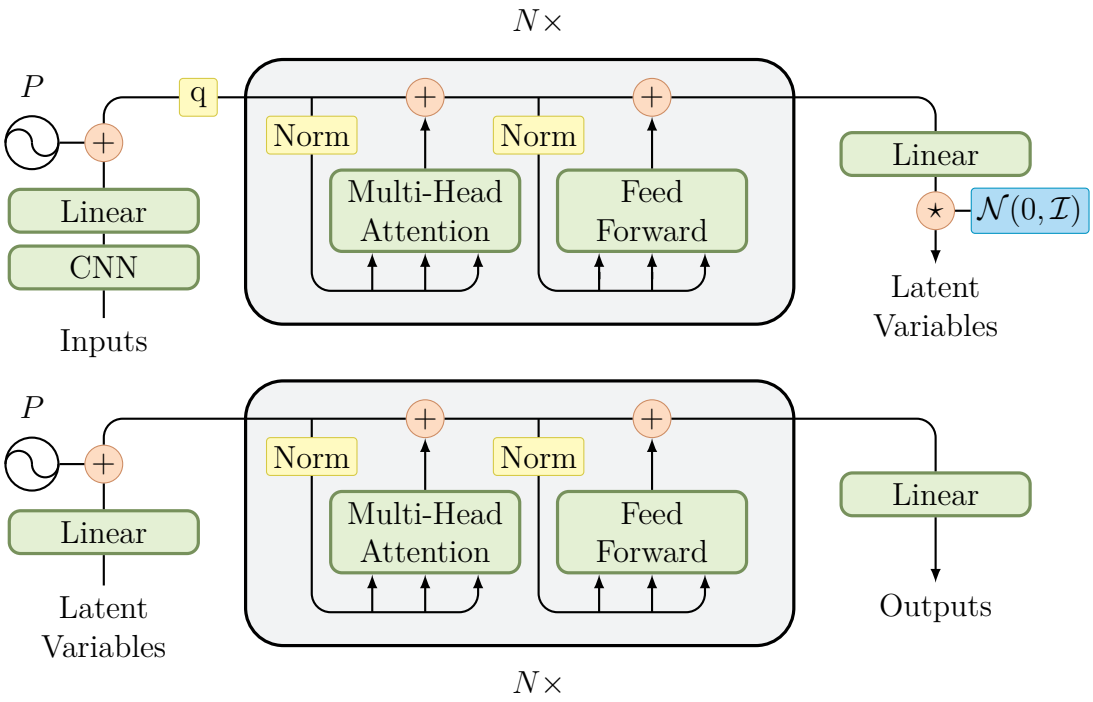


Figure 4. The proposed transformer architecture. The model consists of three parts: two variational encoders (above) and the variational decoder (below). A query token q is appended only for the style encoder.

nent $z^c \in \mathbb{R}^{d_{\text{len}} \times d^c_{\text{latent}}}$ for content and a time-invariant component $z^s \in \mathbb{R}^{d^s_{\text{latent}}}$ for global style information. We approximate the joint posterior with independent Gaussians, i.e. $h_\psi(x) = \text{Diag}(h_\psi(x)_1^2, \ldots, h_\psi(x)_d^2)$. To reduce the computational burden and to make our model more efficient, we use only one transformer g_ψ per component to jointly learn both the mean and the standard deviation of the latent variables. For the time-dependent z^c the parameters are obtained through a linear projection of the learned transformer representation at the respective position; for the time-invariant z^s a query-based approach similar to Carion et al. [45] is used. To this end, we append an all-zero query token $q \in \mathbb{R}^{d_{\text{model}}}$ to the beginning of the embedded time series. Additionally, each vector in the embedding is extended to include a query flag in its last position. This flag is set to 1 for the token and 0 everywhere else. Now, the transformer architecture can learn to encode all relevant global information in the representation at the query position. After a pass through the transformer, the latent parameters can be extracted from the query result with a linear projection. After sampling from the posterior, the latent variables are again embedded with a linear projection and positional encoding. To summarize, one

pass through our model is computed with

$$
\begin{aligned}
\bar{x} &= v_{\psi_e}(x)W_e + b_e + P \\
z_i^c &= g_{\psi_c}(\bar{x})_{i\cdot}W_m^c + b_m^c + \left(g_{\psi_c}(x)_{i\cdot}W_v^c + b_v^c\right) w_c,\ w_c \sim \mathcal{N}(0, \mathcal{I}) \\
z^s &= g_{\psi_s}(\bar{x}_q)_{1\cdot}W_m^s + b_m^s + \left(g_{\psi_s}(x)_{i\cdot}W_v^s + b_v^s\right) w_s,\ w_s \sim \mathcal{N}(0, \mathcal{I}) \\
\bar{z} &= \left[z_i^c,\ z^s\right]^T W_z + b_z + P \\
x' &= f_{\theta_o}(\bar{z})W_o + b_o.
\end{aligned}
$$

This, we abbreviate by

$$
\begin{aligned}
[z^c(x), z^s(x)] &\sim q_\psi(x) \\
x' &= f_\theta(z^c(x), z^s(x))
\end{aligned}
$$

with marginals $z^c(x) \sim q_\psi^c(z^c|x)$ and $z^s(x) \sim q_\psi^s(z^s|x)$, and parameters

$$
\begin{aligned}
\psi &= [\psi_e, \psi_c, \psi_s, W_e, W_m^c, W_v^c, W_m^s, W_v^s, b_e, b_m^c, b_v^c, b_m^s, b_v^s] \\
\theta &= [\theta_o, W_z W_o, b_z, b_o].
\end{aligned}
$$

In addition, we also need to find good values for the model hyperparameters

$$
\vartheta = [N_l, N_h, d_{\text{model}}, d_{\text{ff}}, d_{\text{latent}}^c, d_{\text{latent}}^s],
$$

which include the number of layers N_l, the number of attention heads N_h, and the dimensions $d, d_{\text{ff}}, d_{\text{latent}}^c$, and d_{latent}^s of the model, the hidden feed-forward layer, the content latent space, and the style latent space, respectively. As in the time-series-based Informer [49], we set $N_h = 8$. We optimize the remaining hyperparameters with Bayesian hyperparameter tuning. Details are discussed in section 5.4.1.

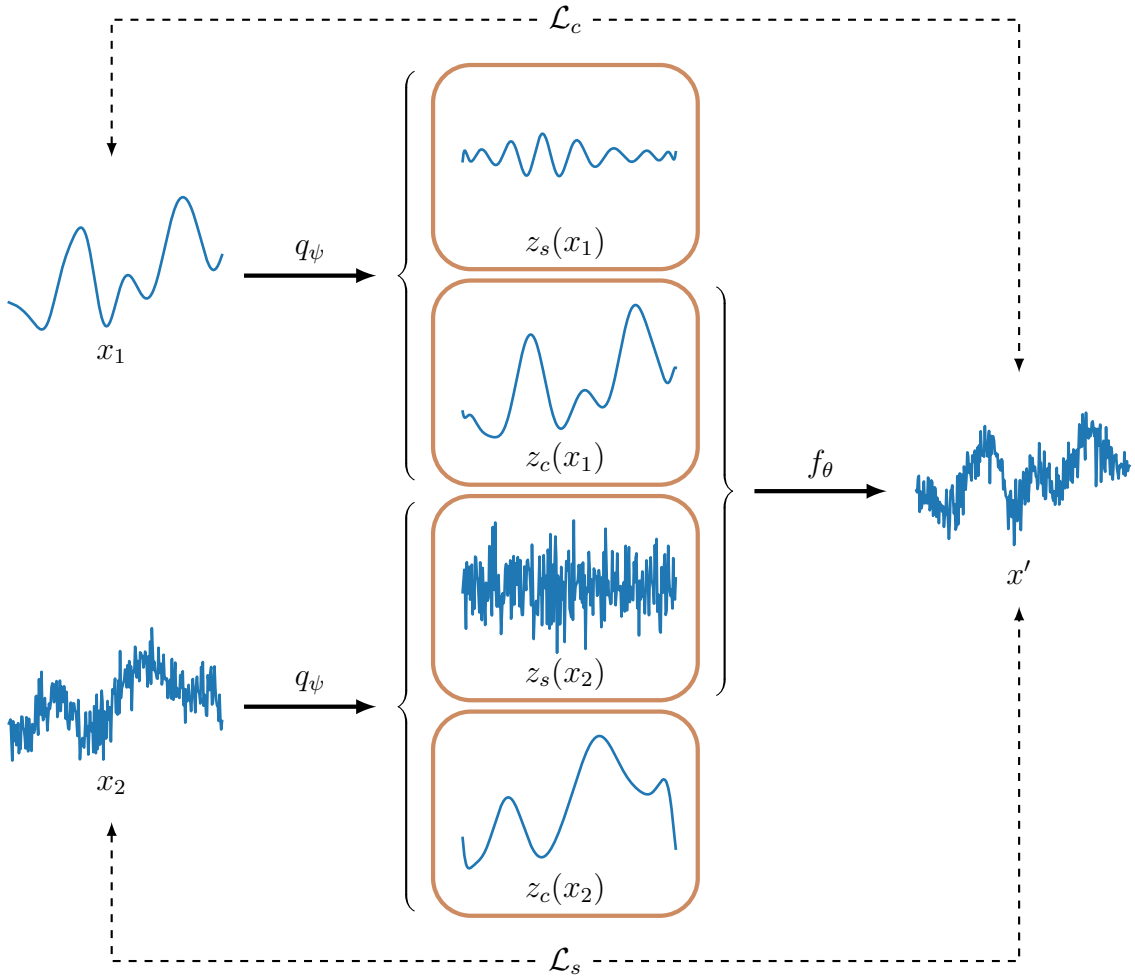


Figure 5. A sketch of latent swapping during training and inference.

4.3 Disentanglement and Inference

After training our transformer model, we aim to perform style transfer by swapping in a different style latent variable. For a time series x and a style sample s, we compute

$$\begin{aligned} [z^c(x), z^s(x)] &\sim q_\psi(x) \\ [z^c(s), z^s(s)] &\sim q_\psi(s) \\ y' &= f_\theta(z^c(x), z^s(s)). \end{aligned} \tag{4.3}$$

The sample y' is a good solution to the style transfer problem (3.5) if both

$$\mathcal{L}_c(x, y') \quad \text{and} \quad \mathcal{L}_s(s, y')$$

are small. To guarantee this, our model has to achieve two things. First, q_ψ has to separate and encode all relevant content and style information in the respective latent variables, and second, f_θ has to properly use all this encoded information. This is the case, for example, when reconstruction quality is high, and content and style are properly disentangled in the latent space. Intuitively, since $x' = f_\theta(z^c(x), z^s(x))$ then contains the same content and style information as x, $z^c(x)$ must contain the content information and $z^s(x)$ must contain the style information. Even in cases where we are only able to obtain imperfect disentanglement, our method may still produce accurately stylized samples, as long as f_θ ignores all residual style information in $z^c(x)$ and content information in $z^s(x)$.

There are several techniques to obtain disentanglement, but they are either not applicable in our setting or do not account for the more general case where the concepts of content and style are not fully independent. The mutual information $\mathrm{I}(q^c_\psi(z^c|x), q^s_\psi(z^s|x))$ is a suitable metric to measure the disentanglement in the latent space. If it is close to zero, $z^c(x)$ contains no information about $z^s(x)$ and vice versa. This motivates the use of the mutual information as an additional regularizer. The high-dimensional integrals are, however, hard to estimate accurately and do not provide a solution for the general case. Careful calibration of the latent space dimensions similar to DSAE [35] is also not feasible, because even the smallest latent space with $d^c_{\text{latent}} = 1$ and $d^s_{\text{latent}} = 1$ might still not be restrictive enough for the univariate or low-dimensional time series we intend to work with.

Instead we propose a new method of active disentanglement, guided by the two separate loss functions unique to the style transfer task. We perform latent swapping as in (4.3) during training and adapt the loss in (4.2) accordingly: For $x' = f_\theta(z^c(x_1), z^s(x_2))$ we compute $\mathcal{L}_c(x', x_1)$ and $\mathcal{L}_s(x', x_2)$. This has two advantages. First, we directly optimize the style transfer problem (3.5), which justifies our approach mathematically, and second, we tailor our disentanglement to the specific notion of content and style implicitly defined through the content and style loss. Indeed, $z^c(x_1)$ and $z^s(x_2)$ only need to learn to store the specific content and style information that is relevant for generation of x'. Note that this technique also works in the more general case where $z^c(x_1)$ and $z^s(x_2)$ contain undesired residual information because f_θ learns to actively ignore this additional information.

Lastly, we briefly discuss some further improvements that can be made during inference. One technique that improves the quality of generated samples is iterative optimization.

Exactly like data-based methods, we can try to get closer to the optimal solution of (3.3) by using gradient descent with back-propagated gradients. Here, a few iterations suffice due to the excellent initialization. This allows us to combine the advantages of data-based methods with the speed and flexibility of model-based approaches. Conceptually, this decreases the inherent amortization error of the VAE discussed in section 2.2. We explore the effect of additional iterations in more detail in section 5.8.4.

Additionally, during evaluation, we can set $z(x) = \mathrm{E}[q_\psi(x)]$ and directly use the mean of the posterior approximation q_ψ instead of sampling from it. This may further improve the quality of the generated time series as we use the best available estimate for the latent variables. Note that on the other hand, if we are interested in more samples with slight variation in content or style or both, we can decode additional posterior samples. Finally, if we want to use more then one style sample, we can compute the mean of their style latent representations, i.e.

$$z^s(S) \coloneqq \frac{1}{N_s} \sum_{i=1}^{N_s} z^s(s_i).$$

This decreases the effect of outliers found in style samples by focusing on characteristics that are present across all samples.

4.4 Pretraining

Transformer models are very flexible and powerful but they also need a lot of training data, in particular, big models like GPT and BERT [38, 51]. Therefore, these models are first trained on an unsupervised tasks with an enormous unlabeled dataset. Although our transformer model is much more compact, proper pretraining still has the potential to improve performance and to decrease the required amount of data significantly [53]. Before we train our model on the main style transfer task, we train on unsupervised generative tasks. Unlike text- and image-based transformers, pretraining on a big task-invariant dataset is not feasible, due to the large diversity of time series from different domains. Instead, we use all available domain-specific training data, including all style samples that are used in the subsequent training. To this end, we adapt ideas from the BERT pretraining procedure which involves a cloze task in which the model learns to predict masked words and randomly substituted words.

Similar to Denoising Autoencoders (DAE) [54], we train our VAE to reconstruct the

original time series from corrupted inputs. In our case, the inputs are corrupted by masking and adding noise to chunks of a fixed length at random positions. Specifically, following the BERT paper, we mask each chunk with a probability of 16%, and add zero-mean standard Gaussian noise to it with a probability of 2%. To mask a position of our input we set the embedded representation to an all-zero mask token, exactly like the query token in section 4.2. Again, we extend every vector in the embedding by a flag that is set to 1 for masked positions and 0 everywhere else. After this initial pretraining step, our model should be able to accurately reconstruct any of the domain specific inputs.

In the second phase of our pretraining procedure we now also include the latent swapping discussed in section 4.3. Specifically, for each batch of training data, we compute the content and style latent variables as usual and then randomly permutate the style latent variables before decoding the latent representation. This pretraining step guides our model to already enforce the disentanglement of content and style. Essentially, our model is now able to perform style transfer across the complete training data. However, in the majority of style transfer applications we have discussed so far, the style dataset is much smaller than the whole dataset and our model thus probably is not yet able to perform the desired style transfer task with the desired accuracy. Nonetheless, this pretraining provides a great initialization that we subsequentially finetune on the main objective.

Chapter 5

Experiments

In this chapter we test the effectiveness of our style transfer method in extensive experiments both on synthetic data and on real data from the domains of finance and speech. First, we introduce the baselines and metrics we use to put our results into context. Then follows a description of the datasets and training procedure. Next, we detail the experimental setups and present the results, which we discuss thoroughly. The chapter ends with an evaluation of the importance of feature choice, style loss and weights, and number of iterations.

5.1 Baselines

We compare our method against a diverse set of baselines, covering the three main categories of feature-based style transfer methods: Data-based methods and model-based methods with fixed style and arbitrary style. As discussed in section 3.1.2, most work on time series style transfer is data-based. Here, we include the methods of Da Silva et al. [31] and El-Laham et al. [30], which use learned features and handcrafted features, respectively. All methods that are based on learned features use the same set of features we obtain by training a CNN-based DAE on the training set. Section 5.8.1 contains further discussions about feature choice and its effect. To compare against model-based methods, we adapt the approach from Syed et al. [27] to time series by training a transformer-based DAE. After we train on the source training set, we finetune the model on the smaller target style dataset. Syed et al. show that this finetuning procedure allows the decoder to successfully produce stylized reconstructions. In our setting, we directly use inputs from the source style and finetune our model with (3.4). This gives further credibility to the results through a clear mathematical motivation and increases robustness. Finally, we include two variants of our proposed method that substitute the transformer architecture with a CNN and a BiLSTM, respectively. As discussed in section 4.3, we also explore the option of 10 or 25 additional gradient descent iterations. Table 1 summarizes all methods.

Notation		Category	Style	Model	Features
I-DAE	[31]	data-based	arbitrary	—	learned
I-HC	[30]	data-based	arbitrary	—	designed
M-FT	[27]	model-based	fixed	transformer	learned
MA-CNN		model-based	arbitrary	CNN	learned
MA-LSTM		model-based	arbitrary	BiLSTM	learned
MA-T		model-based	arbitrary	transformer	learned
MA-T-i		mixed	arbitrary	transformer	learned

Table 1. A summary of several style transfer methods.

5.2 Metrics

There are several aspects to the quality of stylized generation that we aim to measure. These include content preservation, stylization, and predictive utility. Style transfer methods need to find a balanced compromise, especially between the diverging goals of content preservation and stylization. Feature-based methods have fine control over this trade-off with the loss weights λ_c and λ_s. Recall that style transfer aims to find a sample y that combines the content of a content sample x with the style from a style sample s. Because we compare style similarity between two samples with different contents (y and s) and content similarity between two sample of different styles (y and x), it is important to find metrics that are mostly invariant to the properties they do not measure, so that we can properly and independently assess content and style alignment.

One style-invariant metric for content preservation is the feature-based content loss $\mathcal{L}_c(y', x)$. This metric, however, is biased in favor of the models that have directly used it or its features during training. More generally, we can compare the MSE between the two trends of the respective time series. This is less dependent on the different styles then the regular MSE. Trends can be obtained, for example, with a moving average over a fixed number of neighboring elements. Throughout the experiments, we set this window length to 15. Metrics for stylization include the (biased) style loss $\mathcal{L}_s(y', x)$ and the MSE between domain-specific style attributes. For applications in finance, these include the sample autocorrelation

$$\hat{r}_\tau(z) = \frac{\sum_{t=1}^{T-1-\tau}(z_t - \bar{z})(z_{t+\tau} - \bar{z})}{\sum_{t=1}^{T-1}(z_t - \bar{z})^2}$$

of the log-returns

$$z_t = \log \frac{x_{t+1}}{x_t},$$

the volatility (standard deviation of the log-returns)

$$\hat{\sigma}^2(z) = \frac{1}{T-2} \sum_{t=1}^{T-1} (z_t - \bar{z})^2,$$

and the power spectral density

$$\hat{S}_{\omega_k}(z) = \sum_{\tau=-(T-2)}^{T-2} \hat{r}_\tau(z) \exp^{-i\tau\omega_k},$$

which is the discrete Fourier transform of the autocorrelation. Again, any metric over these features is biased towards all models that align them during training. I-HC, in this case, uses the average over all three MSEs as a style loss.

To obtain an unbiased assessment of style alignment, we follow Sajjadi et al. [55], who propose to evaluate generative models with α-precision and β-recall scores, which have the intuition of measuring realism and diversity respectively.

Definition 5.1
Let P and Q be probability distributions and $\alpha, \beta \in [0, 1]$. Then Q has precision α at recall β w.r.t. P if there exists ν_P, ν_Q, and μ such that

$$P = \beta\mu + (1-\beta)\nu_P \;\; \textit{and} \;\; Q = \alpha\mu + (1-\alpha)\nu_Q.$$

Theorem 5.2
If P and Q have finite state space Ω, the set $D(Q, P)$ of all attainable pairs (α, β) is

$$D(Q, P) = \{\delta(\alpha(\lambda), \beta(\lambda))\},$$

where $\delta \in [0, 1]$, $\lambda > 0$, and

$$\alpha(\lambda) \coloneqq \sum_{w \in \Omega} \min(\lambda P(w), Q(w)) \;\; \textit{and} \;\; \beta(\lambda) \coloneqq \sum_{w \in \Omega} \min(P(w), \lambda^{-1} Q(w)).$$

This allows us to efficiently sample the precision-recall-curve, e.g. with

$$\lambda_i = \tan(\frac{i}{m} \cdot \frac{\pi}{2}), 1 \leq i < m. \tag{5.1}$$

Intuitively,

$$\alpha(\beta) = \max_{(\alpha,\beta) \in D(Q,P)} \alpha$$

is the proportion of Q that can be explained by an approximation of P whose quality increases as β increases. For generated samples $y \sim Q$ and style samples $s \sim P$ and if $\alpha(\beta)$ stays high for increasing values of β, this implies that Q can be represented well by P, i.e. y is realistic. Conversely, a high

$$\beta(\alpha) = \max_{(\beta,\alpha) \in D(Q,P)} \beta$$

implies that P can be represented well by Q, i.e. the generated samples y are diverse. In order to find estimates, the authors propose to discretize the data distributions by clustering the joint set of generated samples and style samples. Now, we interpret our distributions as distributions over class labels and apply theorem 5.2. We report the average precision and recall over all λ_i in (5.1) as well as a combined F-score obtained by their harmonic mean. However, for a clear interpretation a visual inspection of the precision-recall curve remains essential.

In addition, we asses the style quality through its indistinguishability from real style samples. To this end, we train a classifier to distinguish between generated and real style samples. To minimize the impact of additional hyperparameters we fit a standard logistic regression model to half of the validation set. We then report the accuracy and recall on the other half. For stylized samples that closely resemble true style samples the accuracy and recall should be close to 50%. Moreover, as the recall is related to how many generated samples are mistaken for real ones, a low recall indicates high realism. Lastly, we consider the predictive utility on the style dataset. Following the TSTR ('train on synthetic, test on real') framework [33], we train a vanilla 2-layer LSTM with a 100-dimensional hidden state to perform a prediction task on the generated samples. Specifically, we learn to predict the next 10 positions from the 50 previous positions. If the trained model generalizes well to the style data, this is an indication of high fidelity and utility as data augmentation. We report the mean absolute error (MAE). Table 2 summarizes all metrics.

Notation	Metric	Category	Biased
$\mathcal{L}_c$	content loss, learned features	content	yes
$\mathcal{L}_{c:d}$	moving average square error	content	no*
$\mathcal{L}_s$	style loss, mean and std of learned features	style	yes
$\mathcal{L}_{s:d}$	style loss, designed features	style	yes*
α^*	average α-precision (realism)	style	no
β^*	average β-recall (diversity)	style	no
F	average F-score (fidelity)	style	no
Acc	accuracy, classification	style	no
Rec	recall, classification	style	no
MAE	mean absolute error, prediction	style, utility	no

Table 2. A summary of several style transfer metrics. Metrics over designed features are biased only if one of the compared models use them during training.

5.3 Data

We now present the different kinds of data we use to investigate our method. First, we study a simple synthetic dataset in which realistic noise is modeled by independent Gaussians. Then, we discuss the data for two real-world applications in finance and speech.

Synthetic Data

We consider the task of refining simulation data. In this case, the style samples are experimental results, which include realistic noise that is attributable, for example, to measurement error. To show the efficacy of our proposed method, we first consider a simple univariate case by modeling the simulation data of interest by samples from a zero-mean Gaussian process. This gives us smooth and aperiodic time series that still exhibit some complexity in their contents. We chose a stationary process whose covariance kernel is the radial basis function (RBF) with bandwidth $\rho^2 = 20$. Thus, we obtain samples at time steps $t_i = i$ via

$$x \sim \mathcal{N}(0, K),\ K_{ij} = \exp\left(-\frac{1}{2\rho^2}|t_i - t_j|^2\right).$$

We assume that this perfectly models the underlying phenomena. Observed style samples are thus obtained from the same distribution, only adding measurement error modeled as

Gaussian white noise with noise level $\sigma_s^2 = 0.5$, i.e.

$$s = c + n,\ c \sim \mathcal{N}(0, K),\ n \sim \mathcal{N}(0, \sigma_s^2 \mathcal{I}).$$

In this setting, style and content are completely separable as they are independent. Our method has to learn to estimate the σ_s and then to add the same independent Gaussian noise. Moving away from the original interpretation, we can also consider the inverse case. Now our method has to estimate the smooth content by removing all noise. Due to the low signal-to-noise ratio of $\sigma_s^{-2} = 2$ this is feasible. In this simplified setting, both tasks have little relevancy outside of diagnosing our methods. Note that the likelihoods $p(x)$ and $p(s)$ cannot be used directly as task-specific metrics for stylization, because for fixed parameters they are both maximized by the mean 0. We can use an extension of Bartlett's Test of homogeneity of variances [56] to obtain p-value estimates for the null hypothesis $\mathrm{H}_0 : x_i \overset{\text{iid}}{\sim} (0, \Sigma)$. Unfortunately, in our case this involves computing the logarithm of the determinant of large near-singular covariance matrices, which is not feasible to to numerical instability. We can, however, consider $\mathcal{L}_\Sigma$, the MSE between the sample covariance $\hat{\Sigma}$ and the true covariance of the style samples.

Financial Data

Next, we consider a style transfer task with a more complex notion of style. Specifically, we are interested in the generation of additional data for extreme events like stock market crashes and financial crises. To this end, we work with time series data from global stock markets. Here, we select five representative and influential stock market indices from markets in the US, Japan, and China including the S&P 500 (^GSPC), the NASDAQ Composite (^IXIC), the NYSE Composite (^NYA), the Nikkei 225 (^N225), and the Hang Sheng Index (^HSI). Daily closing prices in the considered time window from January, 1987 to September, 2022 are available online [57]. As can be seen in figure 7, the indices are highly correlated and follow the same global economic trends.

Price discovery for assets is complex in general, with many outside factors such as specific company decisions, real world events, and the state of the economy playing a large role. Nonetheless, statistical analysis shows the existence of universal empirical properties that are shared between nearly all asset returns over all time periods [58]. These so-called stylized facts include, among others, the absence of significant linear autocorrelations, approximate Gaussianity of the log-returns, and negative correlation between volatility

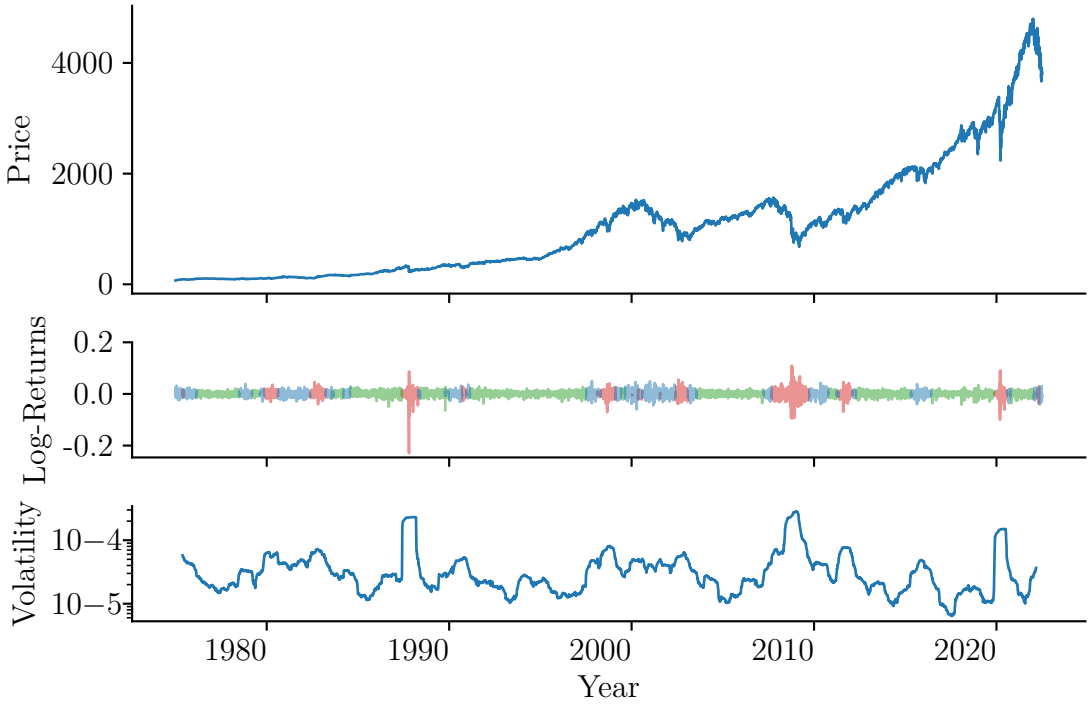
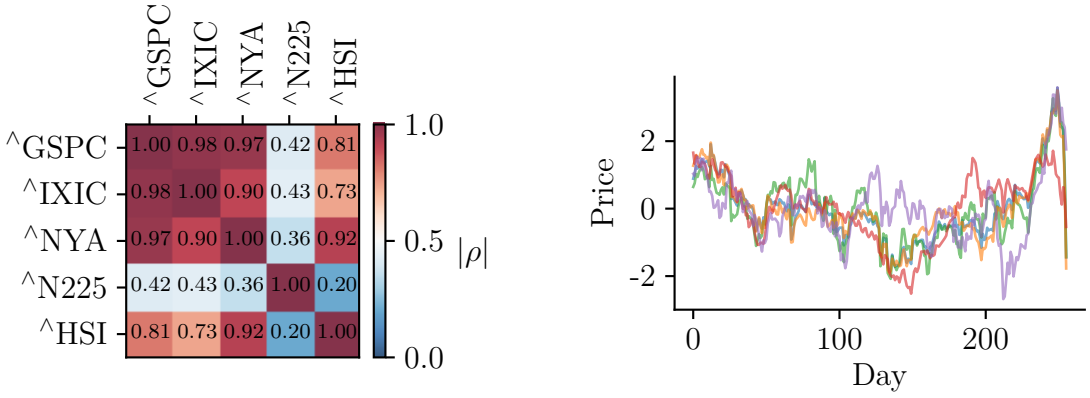


Figure 6. The S&P 500 stock index, its log-returns, and market-wide time-local volatility estimates. Periods of low, middle, and high volatility are marked in green, blue, and red respectively.



(a) Absolute Pearson Correlation

(b) Normalized prices

Figure 7. The five stock indices are highly correlated, in particular the American ^GSPC (blue), ^IXIC (green), and ^NYA (orange). ^N225 (red) and ^HSI (purple) still follow the same global trends.

and asset returns. Finding a stochastic model that is flexible enough to reproduce all stylized facts is highly non-trivial and related problems like parameter estimation become intractable for more complex models. In this context, style transfer is a promising technique that allows us to combine the interpretability of simpler and tractable models with the complex properties of real data.

Another important stylized fact is the observation that volatility tends to cluster in time. High volatility in the context of finance can be defined as the amount of dispersion in the asset returns. Volatile assets are less predictable and experience faster and more significant price fluctuations. Clusters of high volatility can usually be attributed to impactful world events that cause decreased stability and increased uncertainty, which is mirrored in the stock market. Figure 6 illustrates local volatility in the ^GSPC index. The 1929 Black Thursday stock market crash, the 2008 financial crisis, and the recent COVID-19 pandemic have all caused clearly visible phases of increased log-return dispersion. We estimate the time-local volatility with a rolling variance over the returns in a window of size 200. Averaging over all indices found on Yahoo Finance's list of major world indices [57], we obtain general volatility estimates (c.f. figure 6) that, due to the high correlation among stock market indices, still accurately reflect the volatility for each individual index. Using this index-invariant metric we can categorize our financial data into periods of low, middle, and high volatility by manually setting volatility thresholds. Now, we can formulate the task of generating realistic extreme events as a style transfer task where our style dataset contains all patterns of high volatility.

For the content dataset we can use the more abundant low volatility patterns or simulation data. For the later, we extend a discrete Gaussian Hidden Markov Switching Model, a popular doubly stochastic process with an internal state. This internal state is described by a time-homogeneous Markov chain with finite state space, where each state represents different asset behaviour. One state may encode a low volatility upward trend, while another state may encode patterns with higher volatility. In this model, we sample the independent log-returns via

$$R_t \sim \mathcal{N}(\mu_{s_t}, \Sigma_{s_t}),$$

where μ_i and Σ_i contain the trend and volatility for state s_t at time t. This gives us enough flexibility to generate samples that replicate many stylized facts, including volatility clustering. If we want to create diverse simulation data, we can estimate the model parameter

with the expectation–maximization (EM) algorithm and then sample from the stochastic process. Because we are focused on rare events that lead to high volatility patterns, we instead simulate a stock market crash that induces a regime switch from a low volatility to a high volatility state. Specifically, we sample log-returns according to

$$R_t = \begin{cases} \mathcal{N}(\mu_1, \Sigma_1) & \text{if } t < t_c \\ \log(1 - p_c) & \text{if } t = t_c \\ \mathcal{N}(\mu_2, \Sigma_2) & \text{if } t > t_c \end{cases}$$

with covariances

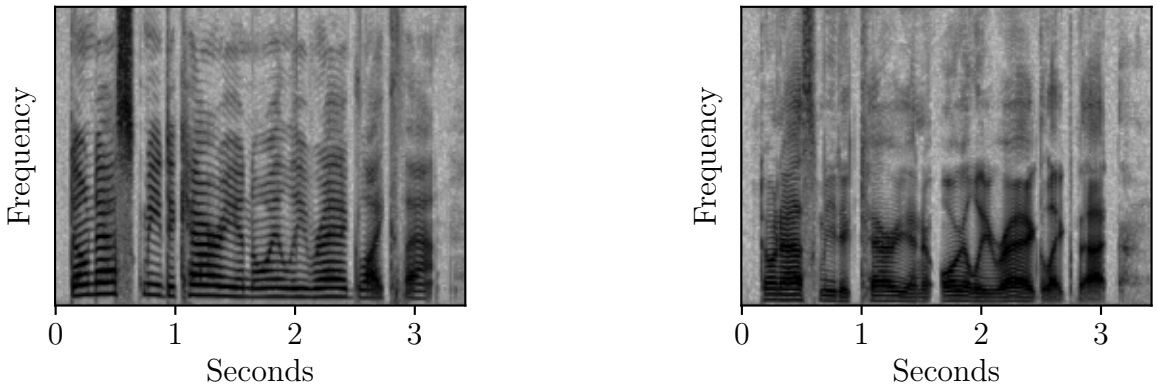
$$(\Sigma_1)_{ij} = \rho_{ij}\, \sigma_1^2 \;\text{ and }\; (\Sigma_2)_{ij} = \rho_{ij}\, \sigma_2^2,$$

where we set the volatility σ_1 and σ_2 to the average volatility across the low and high volatility segments in the data, the trend μ_1 and μ_2 to 0, the correlation ρ_{ij} to the sample correlation, and the loss percentage p_c to 15%. After reconstructing the asset price from the sampled log-returns, we obtain a reasonably accurate initial content sample that we can further improve through style transfer.

Note that the investigated indices are priced at completely different scales, which harms both model-based generation and unbiased evaluation. Thus, we normalize every sample channel to have zero mean and unit variance. Furthermore, we remove any linear trend by first fitting a linear regression model and then subtracting the fitted trend line. Because $\mathcal{L}_{s:d}$ is calculated from log-returns which are not invariant to changes in mean, we calculate the log-returns before subtracting the channel means. For the generated stylized sample, we re-add the style sample means before computing the log-returns. To verify if our methods correctly replicate the higher volatility, we also report $\mathcal{L}_\sigma$, the MSE between the volatility of the stylized sample and the true volatility of the style sample. This metric is again biased towards I-HC which actively aligns the volatilities.

Speech Data

Finally, we experiment on high-dimensional speech data. Here we are interested in preserving the spoken words but changing the speaker identity. We consider the TIMIT dataset [59], which contains 5.4 hours of spoken sentences from 630 speakers (70% male, 30% female), recorded at a sample rate of 16kHz. We pre-process the raw speech waveforms with the sparse fast Fourier transform by computing the local frequency spectrum



(a) Female Speaker (b) Male Speaker

Figure 8. Spectrograms of the recorded sentence 'Don't ask me to carry an oily rag like that' from two different speakers.

every 32 ms. The obtained log-magnitude spectrograms have 257 frequency channels and are subsequentially split into overlapping sequences of length 256, equivalent in length to a few seconds. Figure 8 illustrates the different characteristics of female and male speech. The fundamental frequency in the female speaker's utterance is higher which leads to larger gaps between the lines that correspond to the harmonic frequencies.

5.4 Training

To train the parameters of our model and the baselines, we use Adam [60], an adaptive gradient-based optimizer that is well suited for stochastic gradient descent. Note that this also applies to data-based baselines, where the randomly initialized data sample is the parameter that is optimized. At each iteration, we update our parameters with

$$\theta_t = \theta_{t-1} - \zeta_t \frac{\hat{m}_t}{\sqrt{\hat{v}_t}}$$

where $\hat{m}_t$ and $\hat{v}_t$ are unbiased estimators for the mean and centered variance of our gradients. Similar to gradient descent with momentum, theses estimates reuse information about previous gradients. They are updated via

$$\hat{m}_t = \beta_m \hat{m}_{t-1} + (1 - \beta_m) g_t$$
$$\hat{v}_t = \beta_v \hat{v}_{t-1} + (1 - \beta_v) g_t^2,$$

where g_t is the current gradient, accumulated over a data batch. Following the authors, we set the exponential decay rates to $\beta_m = 0.9$ and $\beta_v = 0.999$. Larger batch sizes have the potential to increase the overall performance by providing better gradient estimates, especially for transformer models [61]. Thus, we set the batch size as high as possible within the limitations of the available system memory. To accommodate all models, this is 64 for the synthetic data, and 32 for the finance and speech data. We use the learning rate schedule

$$\zeta_{t_i} = \begin{cases} \zeta & \text{if } i \leq n \\ \beta^{n-i}\zeta & \text{else} \end{cases}$$

with base learn rate ζ and learning rate decay β. The learning rate is updated after each iteration for data-based methods and after each training epoch (i.e. after using every training sample once) for model-based methods. We set n so that we train all models on the base learning for the first 20% of all epochs. For data-based methods we set $n = 1$.

5.4.1 Hyperparameter Tuning

In addition to the parameters θ, we also have a set ϑ of hyperparameters that cannot be learned through gradient-based optimization. These include, for example, the learning rate ζ and hyperparameters that are related to the network topology like the number of layers. We optimize these with respect to a validation metric that describes style transfer quality. To this end, we split our data into training, validation, and test sets, containing 80%, 10%, and 10% of the data respectively. Since testing all possible combinations is prohibitively expensive, we use Bayesian hyperparameter tuning. To this end, we model our validation metric by a surrogate distribution $p(y|\vartheta)$ and our hyperparameters by a prior $p(\vartheta)$. In our case, we use Gaussian process regression and uniform priors. This allows us, for every set of hyperparameters, to predict the value of the validation metric and the uncertainty of that prediction. After testing a configuration ϑ, we update the surrogate model according to Bayes' rule. For the next configuration, we select ϑ with the highest expected improvement over the current best threshold y^*, maximizing

$$\mathrm{E}_{y^*}(\vartheta) \coloneqq \int_{y*}^{\infty} (y - y^*) p(y|\vartheta) p(\vartheta)\, dy.$$

Thus, we always choose relevant candidates that are likely to produce good results or that we are still uncertain about. Gaussian process regression is illustrated in figure 9. For fixed λ_c and λ_s a sensible choice for the validation metric is the perceptual loss (3.3)

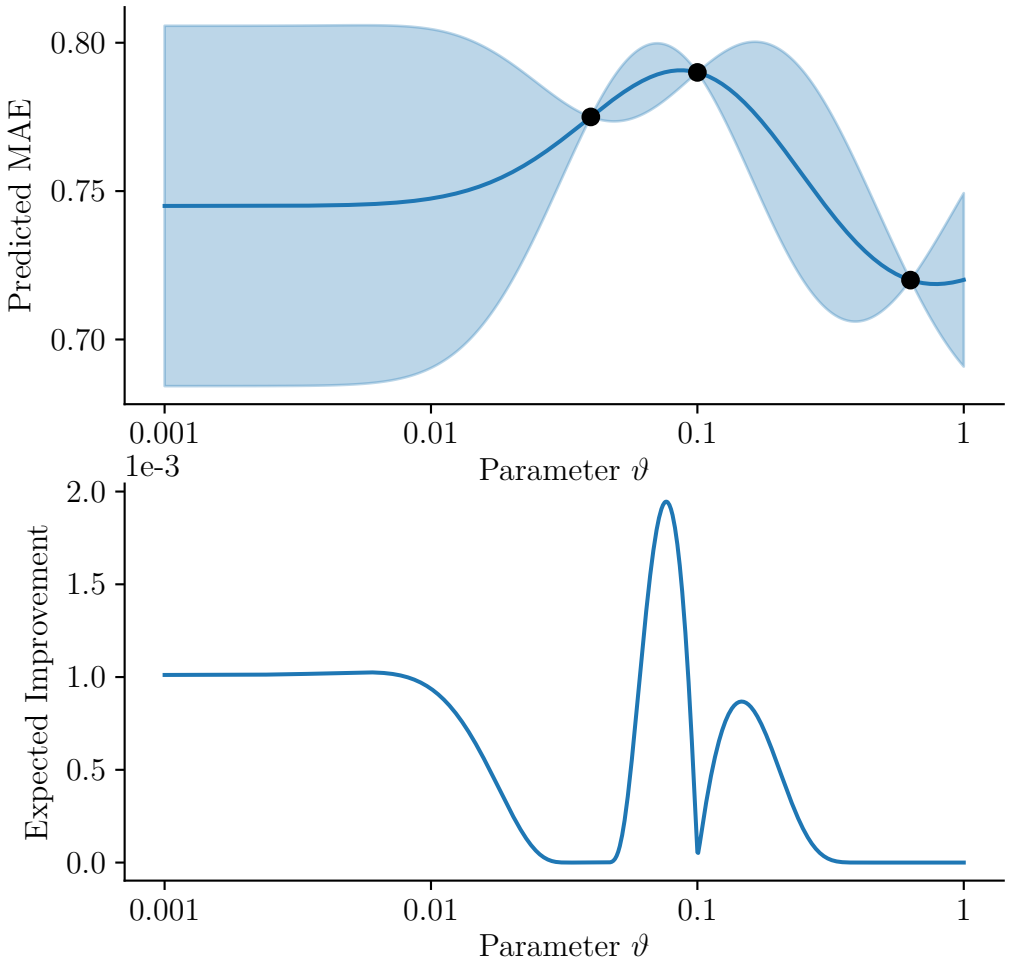


Figure 9. Sketch of Gaussian process regression with one parameter and three samples. The plots depict a 95% credible region and the expected improvement $\mathrm{E}_{y^*}(\vartheta)$, respectively.

that we aim to optimize during training. The choice of λ_c and λ_s reflects our task-specific priorities: When we want to create synthetic data for data augmentation in downstream task, content preservation might be less important then in a setting where we want to enhance realism in a simulation. If not specified otherwise, we set $\lambda_c = 1$, $\lambda_s = 10$. The effects of different weights are studied further in section 5.8.3.

In all experiments, we choose a log-uniform prior over $[10^{-6}, 10^{-2}]$ for the learning rate ζ, a uniform prior over $[0.8, 1]$ for the learning rate decay β, and a log-uniform prior

over $[10^{-6}, 10]$ for λ_{KL}. All intervals are broad enough that optimal parameters are still inside the bounds. For model hyperparamaters we generally chose a uniform categorical prior. Specifically, we select $N_l \in \{2, 3, 4\}$, $d_{\text{model}} \in \{32, 64, 128\}$, $d^c_{\text{latent}} \in \{1, 2, 4, 8\}$, and $d^s_{\text{latent}} \in \{1, 2, 4, 8, 16\}$. For MA-T we additionally choose $d_{\text{ff}} \in \{128, 256, 512\}$. For MA-LSTM we use N_l BiLSTM layers, where the forward and backward pass are averaged so that the representation size stays constant with d_{model} dimensions per position. For MA-CNN we apply N_l convolutions with a kernel size of $k \in \{5, 10, 25\}$, shape-preserving padding, and ReLU non-linearities. For each model on each dataset we use one random seed, with 10 runs of Bayesian hyperparameter tuning for model-based methods and 20 runs for data-based methods. For MA-T-i, we first select the model parameters and then, separately, select the learning rate and learning rate decay for the iterative improvements. Data-based methods use 500 iterations if not specified otherwise. We chose all parameters with the goal to minimize, as a validation metric, the perceptual loss (3.3) with features depending on the model.

To acquire the convolutional features our content and style loss is based on, we train a denoising autoencoder on the full training set. After corrupting the inputs as in section 4.4 we propagate through N_l layers of convolutions followed by N_l layers of deconvolutions. Between every layer we apply a ReLU non-linearity. As a reconstruction loss we use the MSE, which is also the validation metric that we use to minimize the hyperparamters with Bayesian hyperparameter tuning, using the same priors as above. From the trained CNN, we then select the first layer convolutional features as content features and the second layer convolutional features as style features. In section 5.8.1 we discuss feature choice in more depth.

5.5 Synthetic Data

We now evaluate all methods on the synthetic data from section 5.3. To this end, we sample 10 time series of length 5000 from the smooth and noisy Gaussian processes. The larger samples are then split into overlapping chunks of length $d_{\text{len}} = 256$. We consider style transfer in both directions and train each model twice, separately optimizing the hyperparameters. All reported results for model-based methods are obtained after 50 epochs, i.e. every sample in the training set is seen 50 times. The focus of this fundamental style transfer task is to show the efficacy of our method and to study the importance of several model components in a completely controlled environment. As the notion of style

is relatively simple, we do not expect the data-based methods to get stuck in local optima that are far from the optimal solution to (3.3). Because our method optimizes the same objective as I-DEA, our goal for this set of experiments is thus to match the results obtained by the data-based I-DEA baseline. I-HC is not able to handle our real-valued data that prohibits the computation of log-returns. For the sake of completeness we report results using the absolute logarithm

$$\log^* x := \begin{cases} \log|x| & \text{if } x \neq 0 \\ 0 & \text{if } x = 0 \end{cases}$$

but we do not expect the designed features, which are highly specific to finance data, to correctly identify the desired style.

Tables 3 and 4 show the quantitative results on all metrics for the style transfer from smooth to noisy and from noisy to smooth data, respectively. Three randomly selected samples are shown in figures 10 and 11. As to be expected, I-DEA performs well. It generates realistic, diverse, and indistinguishable time series, as indicated by a high F-score and accuracy. The sample covariance matches the true style covariance closely and both content and style loss are low. Our method is able to successfully produce samples of a similar high quality in a fraction of the time, with F-scores that are less than 0.5% lower. Generally, our method outperforms other model-based baselines on the majority of metrics, in particular on stylization metrics. Because these other models are not able to accurately capture the intricacies of the style, they are less restricted overall and can attain better content preservation. This is especially true for M-FT, which is not able to produce believable noisy time series. Instead, a visual inspection shows that M-FT produces orderly and almost periodic noise. This could be the case, because the model does not have access to a style sample during inference, and thus can only remember the most basic style properties from the training. The BiLSTM variant of our method still produces samples of reasonable quality and outperforms the CNN variant in every style metric across both tasks, likely due to the fact that CNNs are less adapted to time series generation tasks. I-HC is neither able to preserve the content of its input nor able to capture any task-specific notion of style. It provides the best estimates of the covariance matrix in the noisy case, but considering that its generated samples look similar on both task, this is most likely due to pure chance.

We note, however, that our method produces noisy samples that are more distinguishable from style samples than other methods. A visual inspection of the generated time series in figure 11 shows no obvious defects, so this may be due to artifacts in the generated samples that are not visually distinct but can be found by the classifier. This problem is not resolved with additional iterations but could potentially be tackled with adversarial training techniques. We argue that the higher distinguishability is not a significant problem since our generated samples demonstrate high fidelity and high predictive utility, in the latter even beating the I-DAE baseline. This sufficiently demonstrates the efficacy of our method to create realistic and useful time series.

We strengthen this perspective through further visualization. To this end, figures 17 and 18 show the precision-recall-curves of all sample distributions. Again, we see that MA-T is on par with I-DEA. Initially, there is a small quality gap, but if we improve our method's output with only 10 more iterations of gradient-based optimization, this gap closes quickly. Furthermore, after 25 iterations our method is able to beat the I-DAE baseline on both the content and style loss in style transfer to noisy data. We further analyze the relationship of number of iterations and perceptual quality in section 5.8.4.

Finally, we compare low dimensional embeddings of the style dataset and the set of generated samples. Here, Principal Component Analysis (PCA) [62] and t-Distributed Stochastic Neighbor Embedding (TSNE) [63] each provide a joint two-dimensional embedding that we can visually inspect. While PCA computes the two principal components and discards the rest of the information, TSNE aims to preserve the inherent structure of the high-dimensional data. Figures 21 and 23 show the PCA embeddings and figures 22 and 24 the TSNE embeddings of selected methods. Because we create style samples at each position of the larger Gaussian process, the style dataset produces long connected lines, with neighbouring points belonging to data samples that are identical up to a small shift. Both types of embeddings contain essentially the same information, but TSNE tends to produce more readable results with shorter lines that are are less entangled. Ideally, we would like our generated samples to be distributed in the same regions that the style samples are located. For the noisy style in particular, our generated samples should form point clouds that are less structured than the style dataset, since in this case we want to add noise that is independent from the content. Overall, both I-DAE and our method fulfill these requirements, they form dense point clouds around the style dataset. M-FT, on the other hand, produces straight lines that indicate that no independent noise

is learned. The samples generated by MA-LSTM tend to concentrate in the middle and are thus less diverse than our method. To summarize all the evidence, our method generates high-quality stylized samples in both tasks, matching the quality of slower iterative methods.

5.6 Financial Data

Now that we have shown our method's efficacy in style transfer tasks, we will show its effectiveness in a relevant application. Our style in this set of experiments is extracted from realistic high-volatility stock data. A deeper understanding of this kind of rare event data is of high relevancy, for example to make reliable predictions during economic crashes and pandemics. To accurately train any prediction model, augmentation of the data, which is limited to the very few historical events recorded to date, is essential. We consider two content datasets: low volatility stock market data and simulation data. The former has more realistic contents, which could also give our generated samples more credibility, but is only available in limited quantities. After categorizing our data according to volatility, we again split the data into overlapping time series of length $d_{\text{len}} = 256$, yielding roughly 8.000 samples of which 1558 exhibit high volatility and 4963 exhibit low volatility patterns. In all pretraining steps across all methods, we include the full set of 8.000 samples, since this may help to better generalize beyond the otherwise rather small training set. Both pretraining and actual training last for 100 epochs, after which all models have converged as indicated by a stagnant training loss.

Table 5 shows the quantitative results on all metrics for style transfer from low to high volatility data. Classification accuracy and recall are high across all methods, likely due to the smaller validation set. Notably, our method beats the iterative I-DEA baseline in every metric. In contrast to the simple style in the synthetic experiments, finding the optimal solution to (3.3) is non-trivial and an iterative approach thus much more likely to get stuck in an undesirable local optimum. Moreover, our method beats all other baselines in all unbiased style metrics. Additional iterations further increase the performance gap yielding realistic time series with high fidelity and predictive utility. A few samples are shown in figures 12 and 13. We note that I-HC produces visually distinct time series that do not capture the correct style characteristics despite having very closely matching volatility and autocorrelations. Instead, the generated samples show noisy behaviour not found in any historic data. This may stem from the inflexibility of the designed

features, which do not allow I-HC to adapt to the specific desired style. For example, visual inspection of the samples in figure 12 indicates that the high volatility data has higher correlation between the five indices. This characteristic, is successfully replicated by our method, its BiLSTM variant, M-FT, and I-DEA but is outside of the scope of the designed features.

Table 6 shows the quantitative results for style transfer from simulation to high volatility data, while figures 12 and 13 show style transfer for a selection of style samples that also contain a real stock market crash. The results are essentially the same as with low volatility content data. Our method still beats the I-DEA baseline and produces the generated samples with the highest predictive utility. The main difference is that F-scores and classification accuracy are now indicating low stylization. This can be attributed to the fact that all stylized sample share the same content: a step drop in asset price that is rare, even among high volatility patterns. Thus it is very easy to distinguish generated samples from most style samples. As this is by design, we can ignore the reported scores on these metrics. On the contrary, a visual examination of the stylized samples indicates that style transfer was successful. The sharp crash in the simulation is softened and stretched over multiple days, a behaviour that can also be observed in the given COVID-19 related crash of the style sample. Additionally, the post-crash segment of the generated sample now exhibits more realistic and more intense fluctuations.

Figures 19 and 20 show the precision-recall-curves of all methods and both style transfer tasks. Generally, recall is higher than precision, because creating realistic samples in this setting is more difficult than creating diverse samples, especially because our validation set is small and thus less diverse anyway. The curves confirm what we concluded from the other metrics: iterative methods and methods based on CNNs are less suited than methods based on LSTMs and transformers. This can be attributed to the fact that the latter architectures can correctly identify complex long-range dependencies.

5.7 Speech Data

Lastly, we briefly explore the efficacy of our method on high dimensional data with high complexity. In this experiment, we aim to extract the speaker-specific style from a small set of utterances. This task is challenging, since both content and style arise from the interplay of many different frequency bands. Hence, it is not enough to find the correct

frequencies in the spectrogram and add realistic noise. Rather, our method has to extract high level concepts such as uttered phonemes from the data, discard all additional information, and then reconstruct them again with a new style that visually looks completely distinct. Spectrograms of two generated samples can be found in figure 16. It is obvious that both our method and a data-based approach fail to identify any style information. This does not change for high values of λ_s. Instead, the methods produce noisy reconstructions of the content sample with high magnitudes in the same frequencies bands.

This failure mode can be attributed to two potential shortcomings of our method. Either our method is unable to correctly identify high-level information across the large number of channels or our features do not accurately capture the desired notion of style. Because our method was able to extract high-level concepts such as correlation from the financial data in section 5.6, the second explanation seems more plausible. Recall that in section 4.1 we have discussed that with an approximately invertable feature mapping μ, any reconstruction tends to be close to the input not only in the feature space, but also approximately in position-wise values. Throughout this thesis we use convolutional features that are learned on a reconstruction task. Thus, our feature mapping is approximately invertable and our set of features is incentivised to not drop any relevant information. This is ideal for the low-dimensional data encountered so far, but for speech style transfer this is detrimental, as we want to discard any low-level content information in favor of the correct high-level concepts of phonemes and words. This hints at a potential solution to the poor performance. By using a different method of feature acquisition, which does not focus on reconstruction, but rather on content extraction, we can learn the appropriate high-level features. For instance, we could train on phoneme or word detection and classification. An in-depth exploration of this idea is beyond the scope of this thesis and will be left for future work.

5.8 Ablation

Now that we have seen the effectiveness and limitations of our approach, we will conclude this thesis with a discussion of several important factors that play a role in our method's performance. We will explore the effects of feature choice, style loss, loss weights, and finally, the number of iterations.

5.8.1 Feature Choice

Up to this point we have deliberately avoided an in-depth discussion of the learned features and their selection procedure. Indeed, feature acquisition in the context of style transfer can be seen as a separate problem. First, we define the notion of content and style and only then do we attempt to solve the resulting style transfer problem. Thus, the optimization of learned features has to be evaluated in an additional outer loop. In the inner loop we select model hyperparamters based on the perceptual loss, and in the outer loop we select the method-invariant style transfer hyperparamters. These includes the method of feature acquisition, the kind of content and style loss, and the loss weights λ_c and λ_s.

As the trade-off between content preservation and stylization is highly non-trivial, it is not always possible to find a suitable unbiased and feature-invariant validation metric to jointly evaluate the content and style features. In this case, an expert has to make a manual choice that considers the task-specific priorities and all available metrics and visualization techniques across all methods. For instance, in our experiments with the synthetic data, a visual inspection of the generated samples and a low F-score indicate that features based on log-return properties are not able to accurately capture the desired style, despite achieving the lowest covariance loss λ_Σ, which on a first glance might have been a candidate for a method-invariant style transfer metric. In the cases where style transfer can be accurately captured by a validation metric, all method-invariant design choices can be made to optimize this metric, and all hyperparameters can be selected with Bayesian hyperparameter tuning. For example, if our goal is to create synthetic samples as data augmentation for a downstream task, the predictive utility may be a suitable validation metric.

To speed up this process, recall that feature choice, and other design choices, are mostly method-invariant. Thus, it is feasible to only consider a data-based iterative method,

which has no trainable parameters and only two major hyperparameters: the learning rate and the learning rate decay. This allows for much faster prototyping and informs design choices that are still relevant for other methods. For instance, we can verify the choice to base the content loss on early low-level features, and the style loss on later high-level features. Here, figure 25 shows the effect of selecting content and style features from different layers. Indeed, high-level content features harm content preservation while low-level style features harm stylization. Table 7 shows the corresponding metrics. Our proposed feature selection outperforms the alternatives in many metrics including predictive utility, if only by a small margin. Furthermore, we can use this iterative prototyping approach to analyze stylization by only optimizing the style loss, i.e. by setting $\lambda_c = 0$ in (3.3). Starting from a random initialization, this gives us visual insight in to the captured style characteristics, independent from content information. Figure 27 shows the results of style replication in smooth and noisy data, which verifies that our selected features accurately capture the desired style.

5.8.2 Loss Choice

As discussed in section 3.1.1, there are several ways to define a style loss over the selected style features. We consider the two most popular choices based on feature co-occurrences as in (3.1) and on feature mean and standard deviations as in (3.2). We deploy the same prototyping technique as in the previous section and compare the results of a data-based method on the two losses in figure 26. Both losses yield high-quality results, but the one based on feature means produces slightly more realistic time series, both visually and as indicated by the a higher F-score in table 7.

5.8.3 Content-Style Trade-Off

At their core, style transfer tasks have the two goals of content preservation and stylization, which are often at odds with each other. For instance, in section 5.5, proper stylization adds independent noise which can only decrease any metric of content similarity. For practical applications, the task-specific priorities inform which goal needs to be prioritized, and thus how the loss weights λ_c and λ_s should be set. Recall the setting from section 5.6, where our goal is to create realistic rare event data for high volatility stock market events. If we want to use this data as data augmentation, the predictive utility can be used as validation metric to automatize the selection of the loss weights. Intuitively, we expect that putting high importance on stylization yields samples that

are more similar to the style dataset and thus also have high predictive utility. Table 8 shows the predictive utility of our method's stylized samples along with all other metrics for different values of the loss weights. Moreover, we perform Bayesian hyperparameter jointly over the model hyperparameters and the loss weights. To this end, we use a log uniform prior over $[10^{-6}, 10^{2}]$ and $[10^{-4}, 10^{4}]$ for λ_c and λ_s, respectively. As to be expected, increased importance of stylization steadily improves style metrics like λ_s and $\lambda_{s:d}$ but deteriorates content metrics. On the other hand, for $\lambda_s = 1$ and $\lambda_c = 1$, we see almost no stylization and the inputs are simply reproduced. Since the low volatility data itself contains useful information, the samples then have low MAE, but do not have the specific high-volatility information that translates to downstream performance gains. For partially stylized results, the predictive utility initially follows the predicted behaviour with a decreasing MAE. For very large style weight λ_s, the MAE increases again, most likely because no realistic content information could be generated. The Bayesian optimization approach, is able to correctly identify the optimal relative importance of style.

5.8.4 Number of Iterations

Lastly, we investigate the effect of additional iterations. In the experimental results in sections 5.5 and 5.6, we have seen the benefits of additional optimization iterations on the style transfer loss in (3.3). Throughout all style transfer tasks the content loss, style loss, F-score, and predictive utility could be significantly improved, in some cases yielding over 5% improvement in absolute F-score percentages, and over 50% lower style loss values. In particular, in the style transfer tasks from smooth to noisy and from low to high volatility data, this allows us to bridge the gap to or even exceed the best performing baselines. Up until now, we have only focused on a small number of iterations to ensure the fast inference of our method, and on a large number of iterations to ensure the quality of the data-based methods. Now, we explore the alternatives and investigate how high the quality of the generated samples of our method can be if we do not care about efficiency, and how efficient the inference of data-based methods can become if we tolerate a small drop to overall quality.

To this end we test both I-DEA and M-TA on the style transfer task from smooth to noisy data. We report the results after 5, 10, 25, 100, and 500 iterations in table 9, each time fitting a separate learning rate and learning rate decay. For MA-T we use the same trained model from section 5.5 for all runs. We note that, both methods, in the end, still

perform on a similar level. MA-T starts out much stronger due to the great initialization, but this gap is decreased with every iteration and almost non-existent after 500 iterations. The biggest difference is the efficiency, the number of iterations needed to achieve close to the optimal performance. Here MA-T is superior, as most of the time it needs only half the number of iterations or less to achieve the same quality. Moreover, it is likely that for more complex tasks, our method is at an additional advantage, as it is much less likely to get stuck in local optima. We can see this in the experiments in finance and speech where I-DEA is clearly outperformed by our method. This highlights that our method is not only useful as a tool to speed up inference, but also to improve style transfer quality. Indeed, the only major disadvantage of our generated samples with artificial noise was their high distinguishability, and with additionally iterations this issue becomes much less pronounced, as indicated by the classification accuracy and recall, which decrease significantly by nearly 17%.

Chapter 6

Conclusion

In this thesis, we developed a fast model-based style transfer framework based on a newly proposed loss-guided disentanglement. Our approach enables fast and data-efficient time series style transfer on several applications across several diverse datasets, both on univariate and on multivariate time series data. We showed the effectiveness of our method both through visualization and extensive experiments. An evaluation with unbiased metrics yields empirical evidence that we are able to match or exceed the generational capabilities of all recent baselines in many important aspects. Predictive utility, in particular, is consistently the highest with our approach, which indicates that we successfully generate stylized samples with high fidelity and high practical value as data augmentation. We note that neither our method nor any of the data-based baselines could accurately capture the complex content and style of high-dimensional speech data. We believe that this is due to a flawed feature acquisition and not a direct limitation of our method. A more detailed analysis of our method's efficacy on this high-dimensional dataset will be left to future work. As a concluding remark we note that there are several ways our method could still be improved upon, including adverserial training or with a more expressive latent space that also captures the correlation of positions close in time.

Chapter 7

Appendix

7.1 Figures and Tables

We now present the results on all style transfer tasks. We refer the reader to chapter 5 for a detailed discussion and interpretation. The following pages are structured as follows. First, we plot randomly chosen stylized samples for each tasks. Then, we illustrate the α-recall and β-recall curves. Next, we visualize low dimensional embeddings of the synthetic data. The last set of figures explores different method-invariant design choices. Finally, we state the quantitative results over all discussed metrics and tasks. The tables follow the same order in which they appear in the discussion. Furthermore, in each column we mark in bold numbers the best value, and every other value that is not less than 5% smaller and not more than 5% bigger. For values in percentages, we mark all values within 1% of absolute percentage points. For classification accuracy and recall we consider as the best value either the lowest value or 50%, depending on which is bigger.

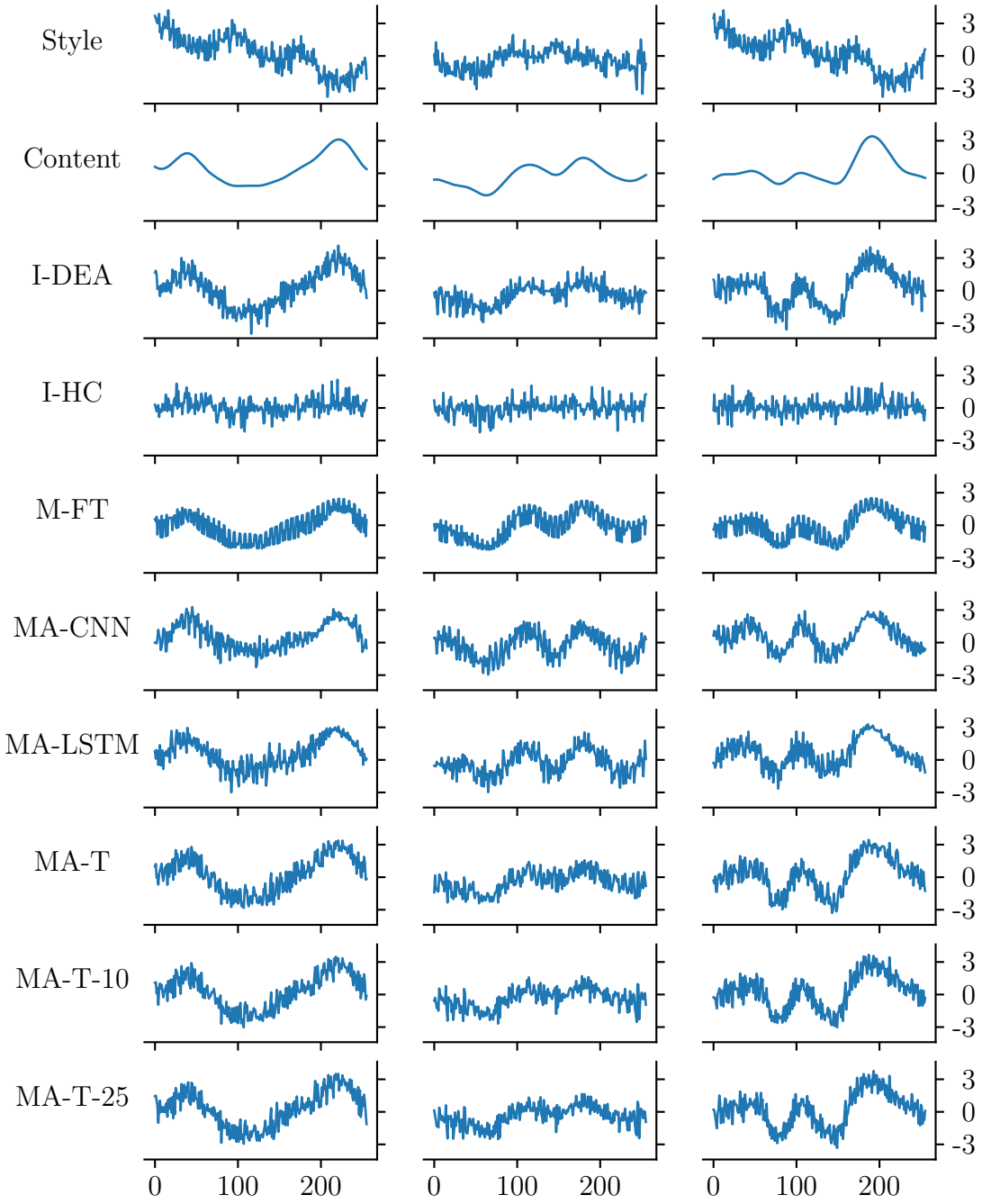


Figure 10. Style transfer from smooth to noisy data on randomly chosen data samples.

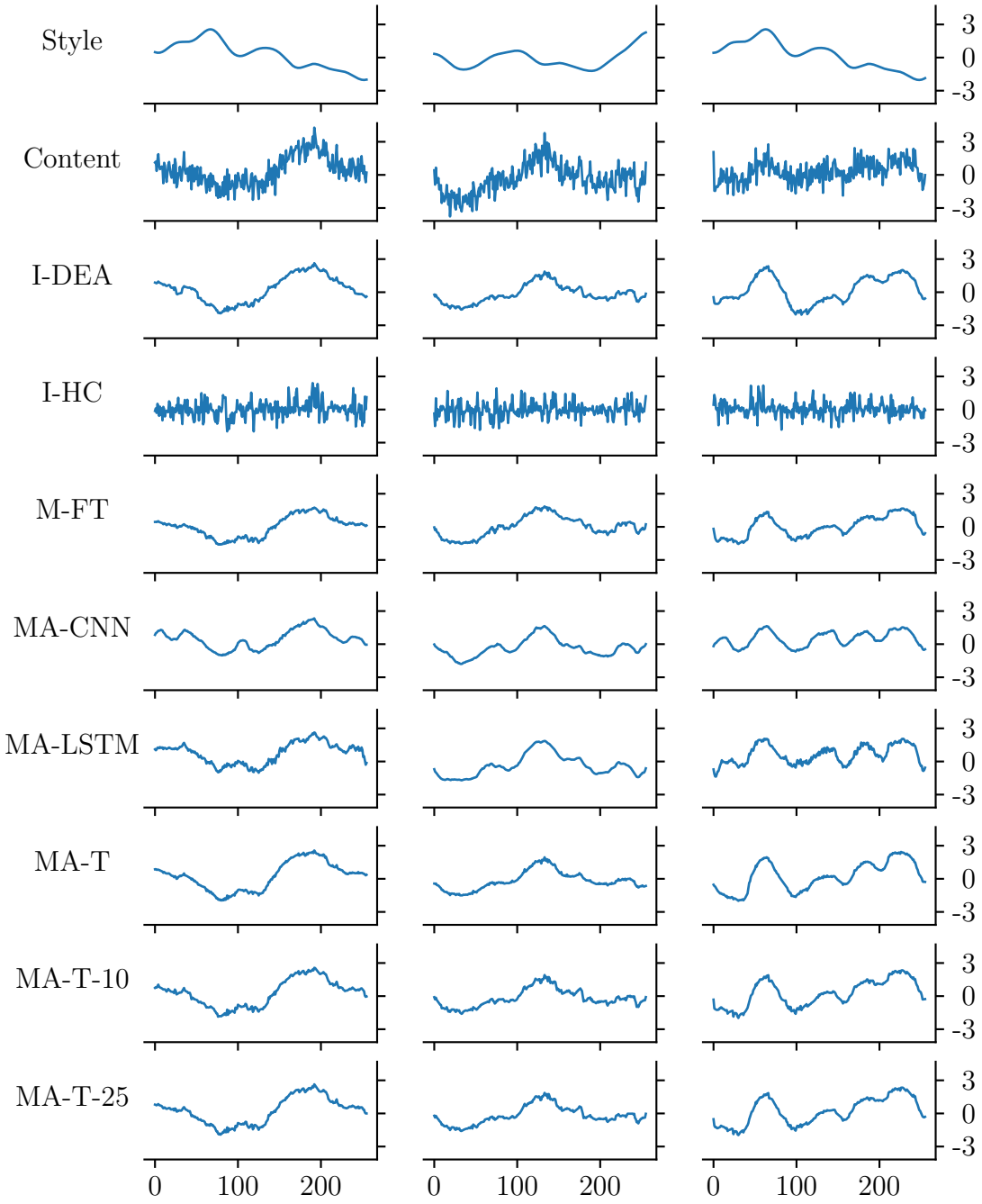


Figure 11. Style transfer from noisy to smooth data on randomly chosen data samples.

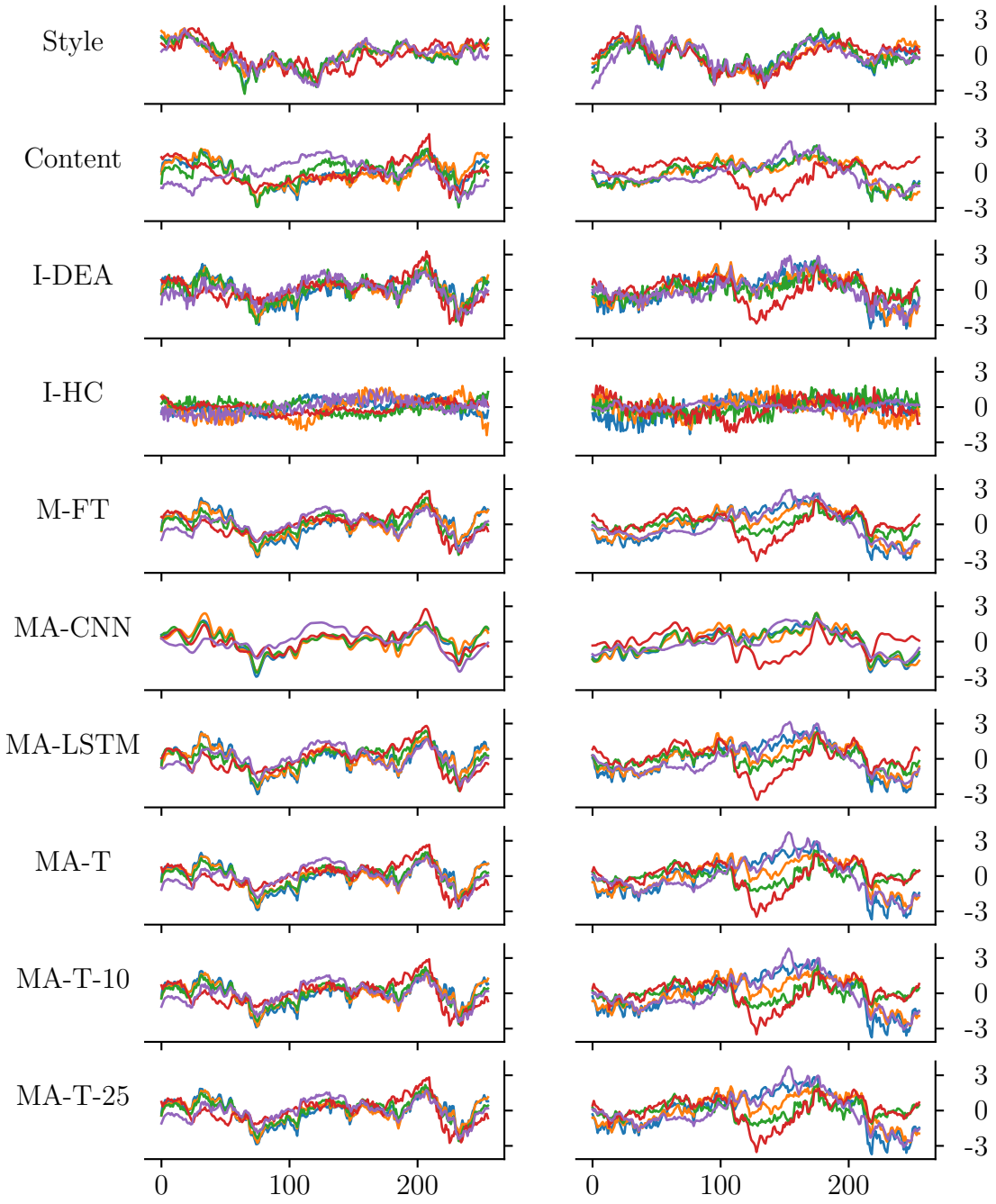


Figure 12. Style transfer from low to high volatility data on randomly chosen samples.

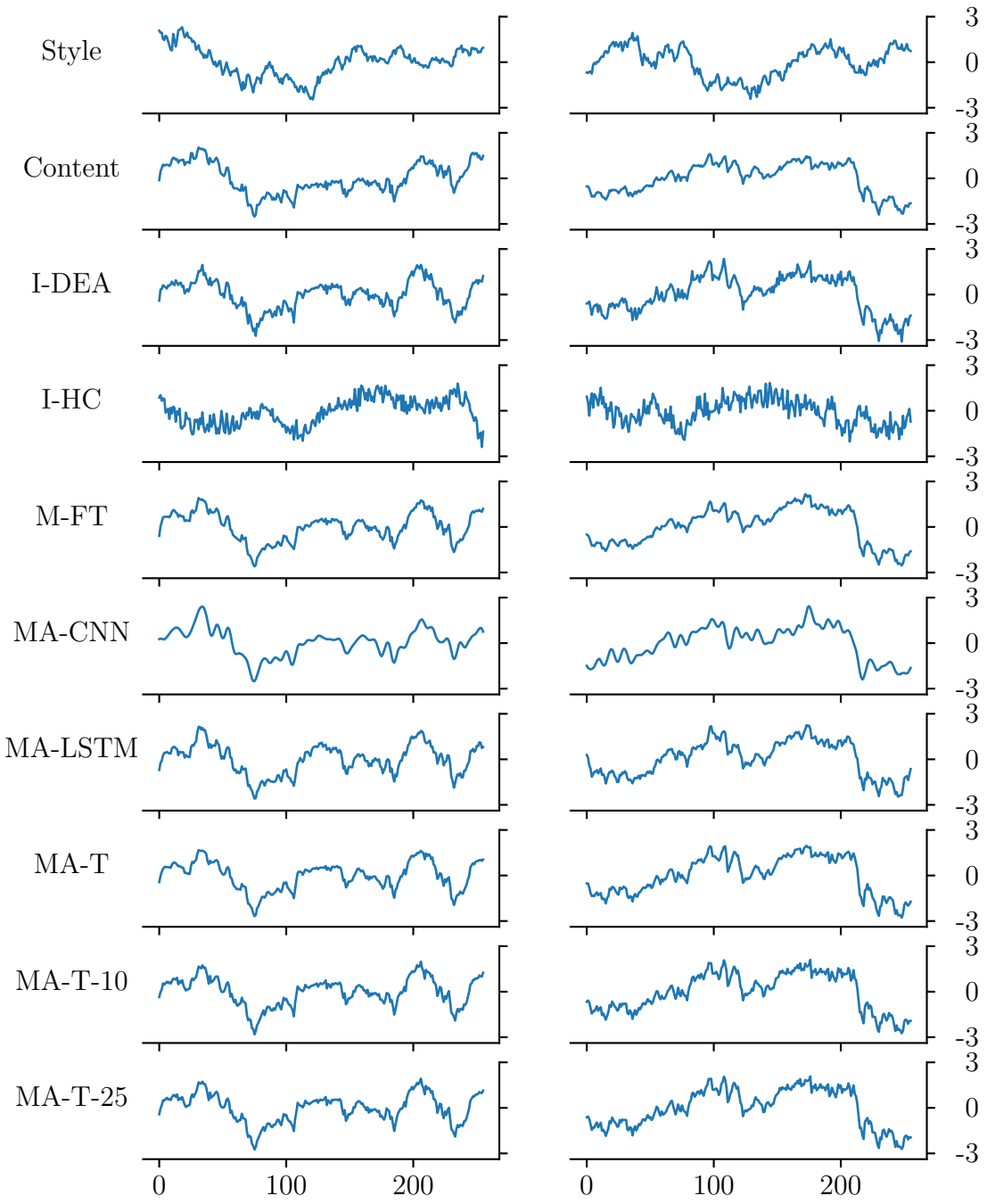


Figure 13. Style transfer from low to high volatility data on randomly chosen samples. This plot shows the NASDAQ Composite index from figure 12.

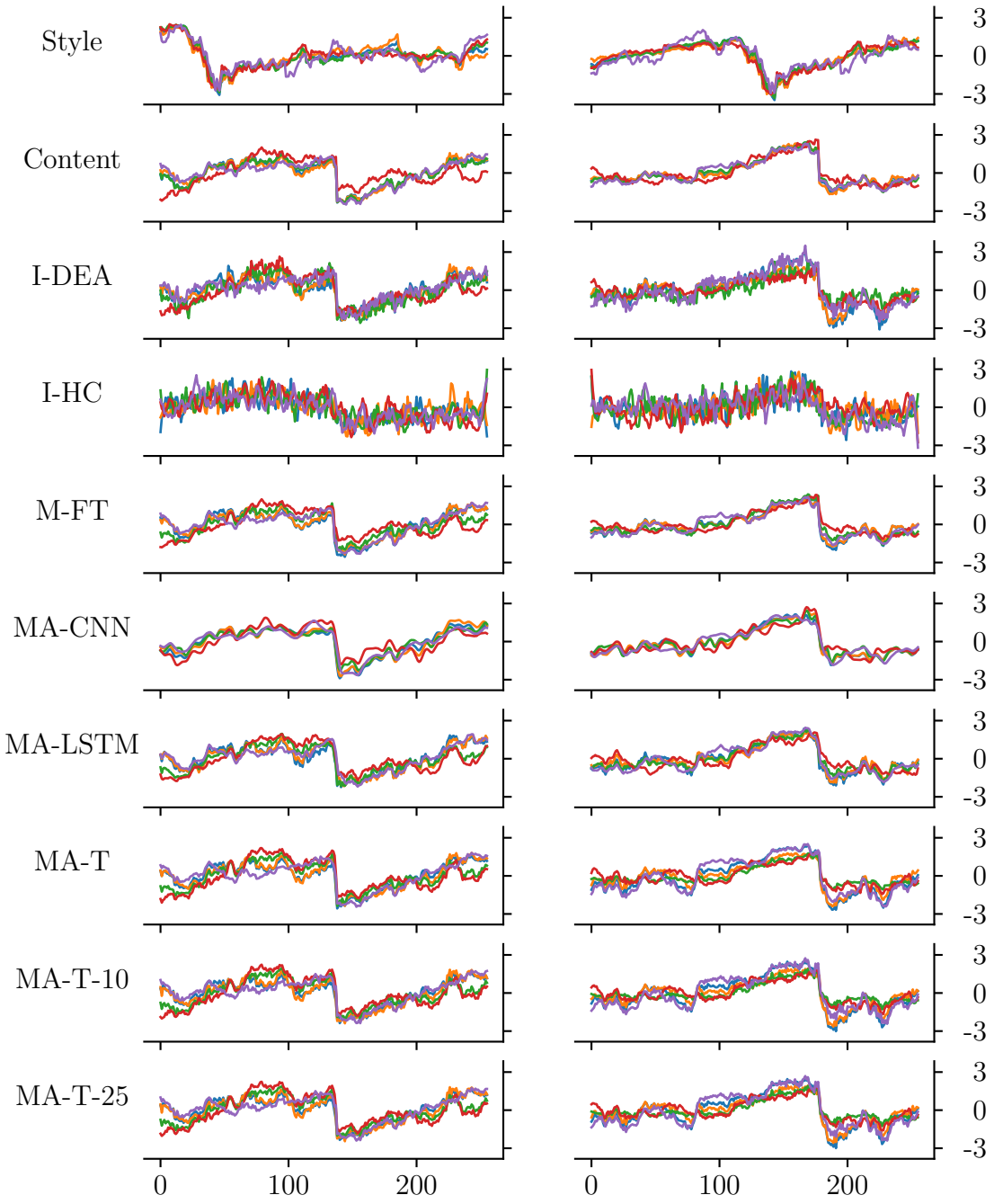


Figure 14. Style transfer from simulation to high volatility data on manually chosen samples.

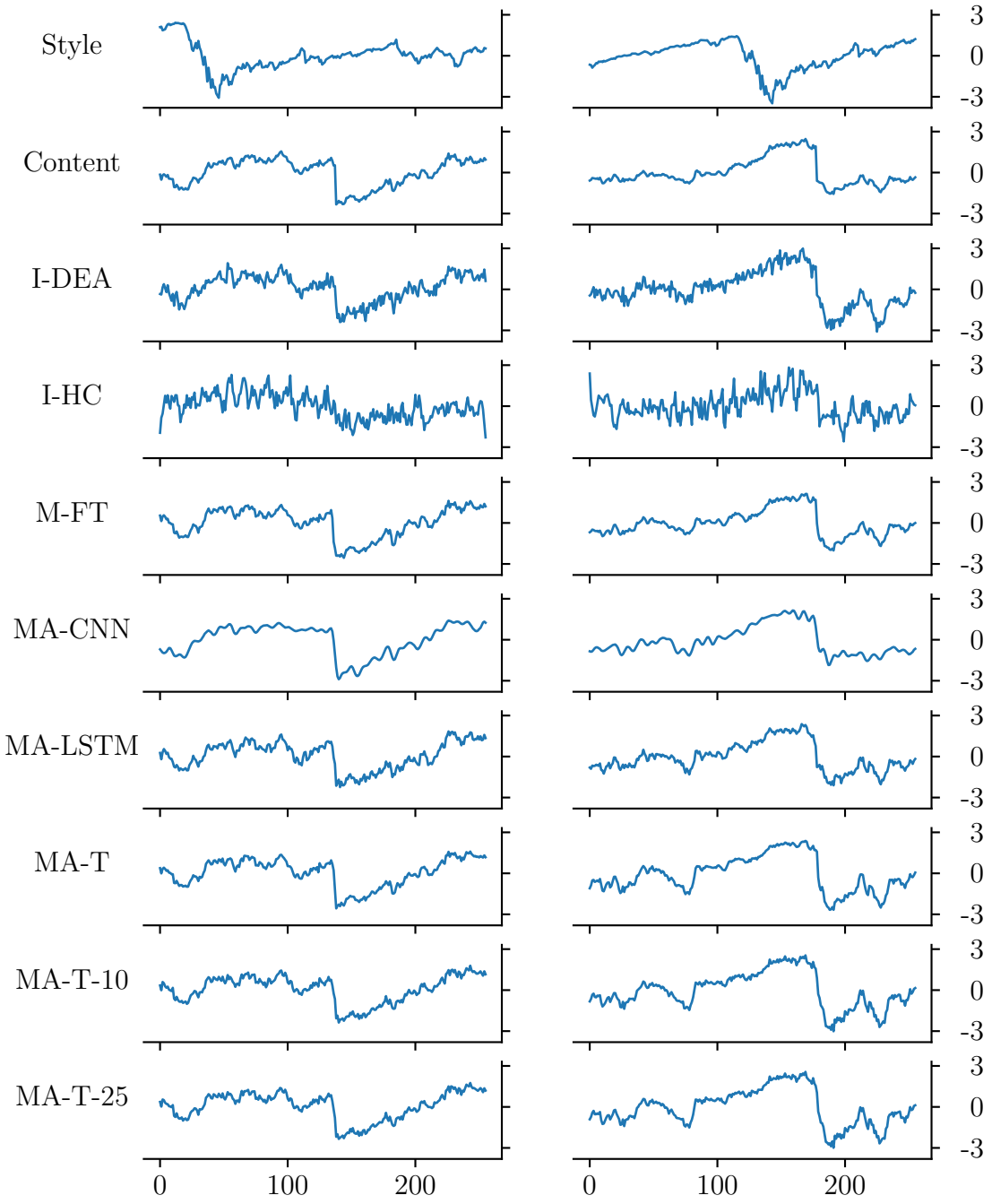


Figure 15. Style transfer from simulation to high volatility data on manually chosen samples. This plot shows the NASDAQ Composite index from figure 14.

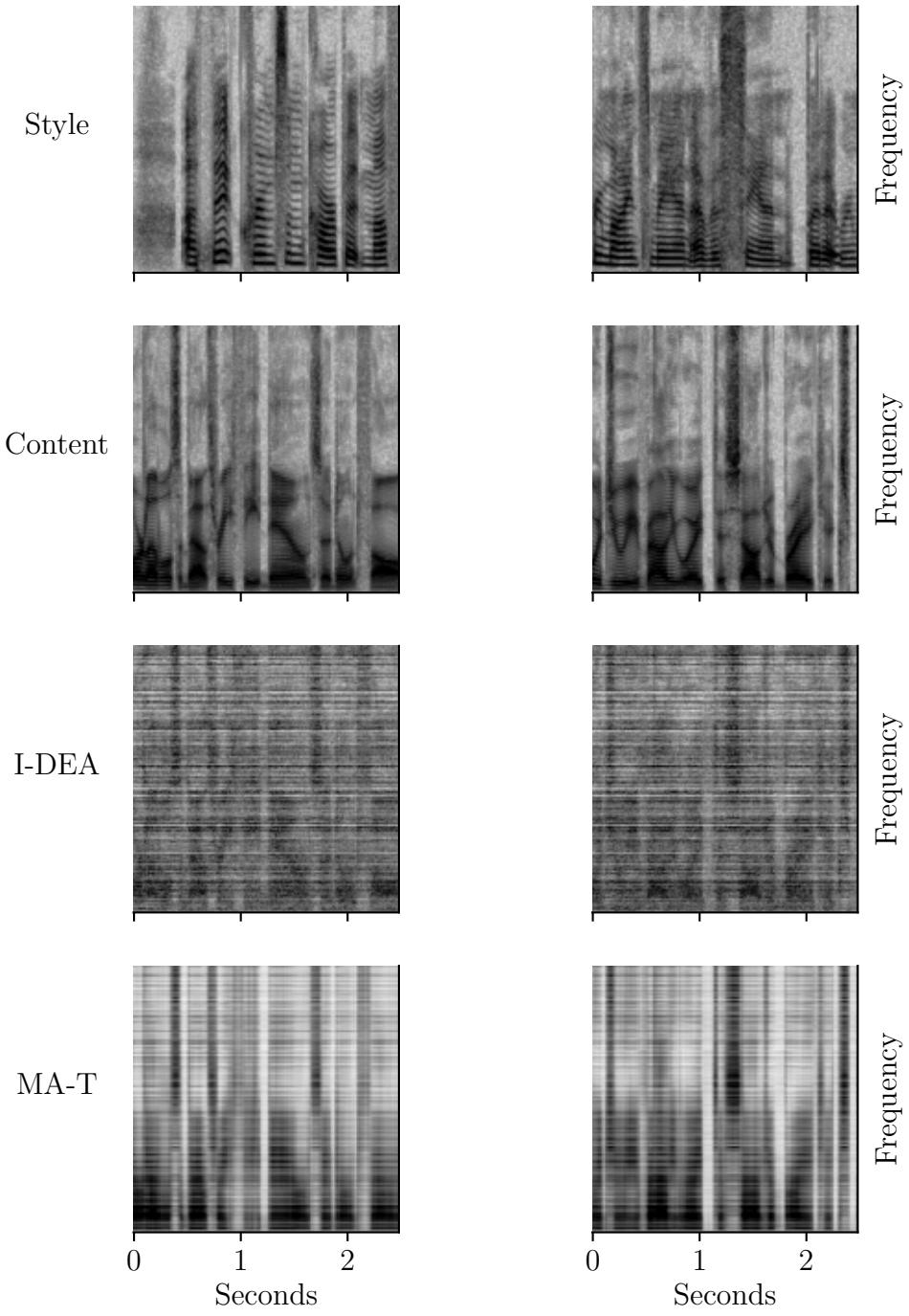


Figure 16. Style transfer from male to female speech on randomly chosen samples.

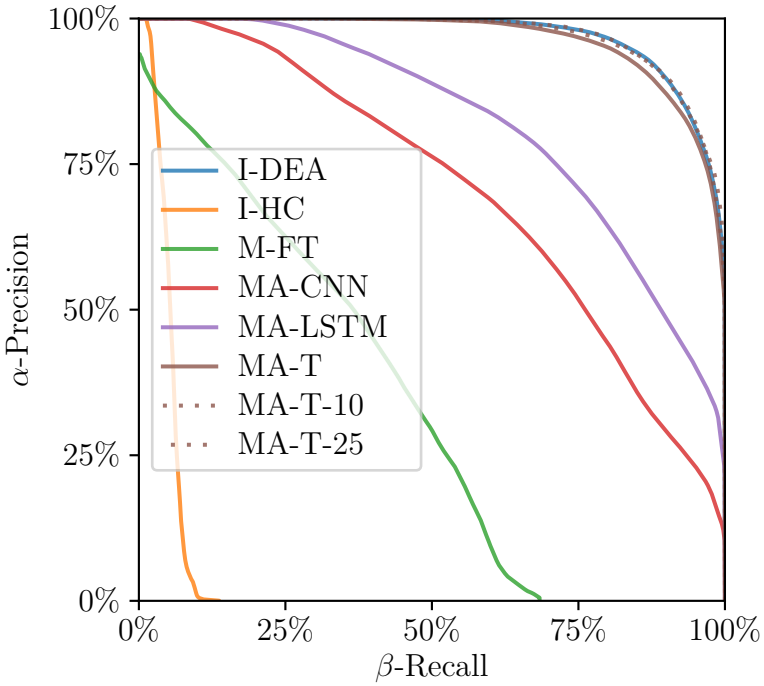


Figure 17. α-Precision and β-Recall for style transfer from smooth to noisy data.

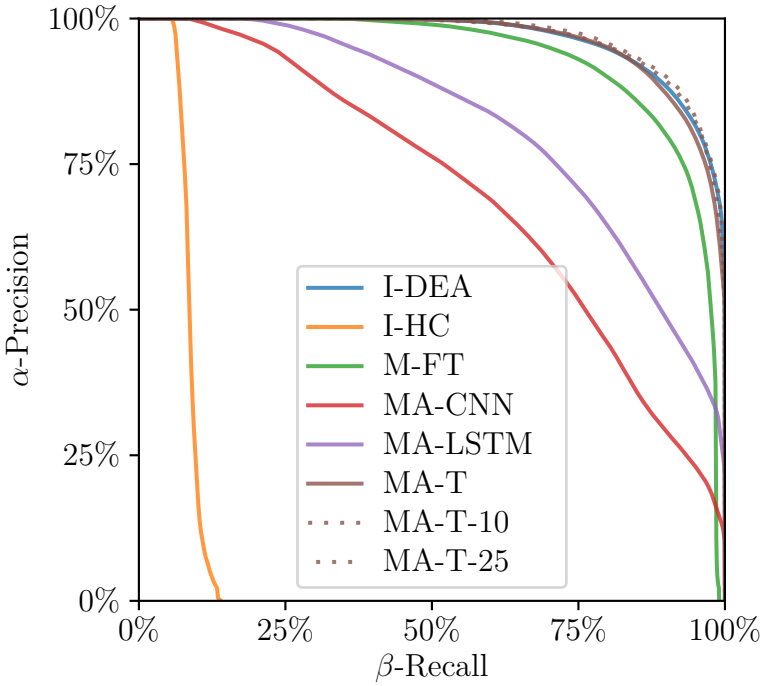


Figure 18. α-Precision and β-Recall for style transfer from noisy to smooth data.

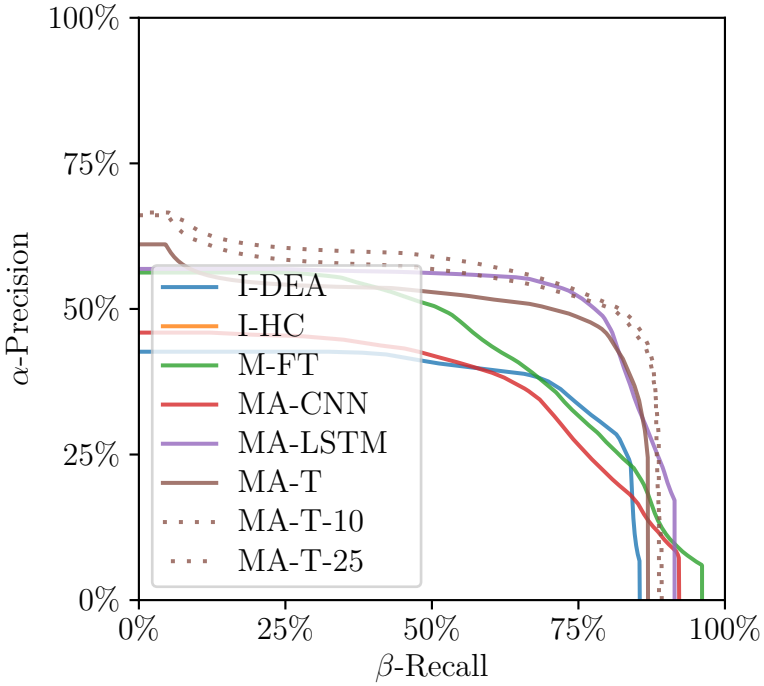


Figure 19. α-Precision and β-Recall for style transfer from low to high volatility data.

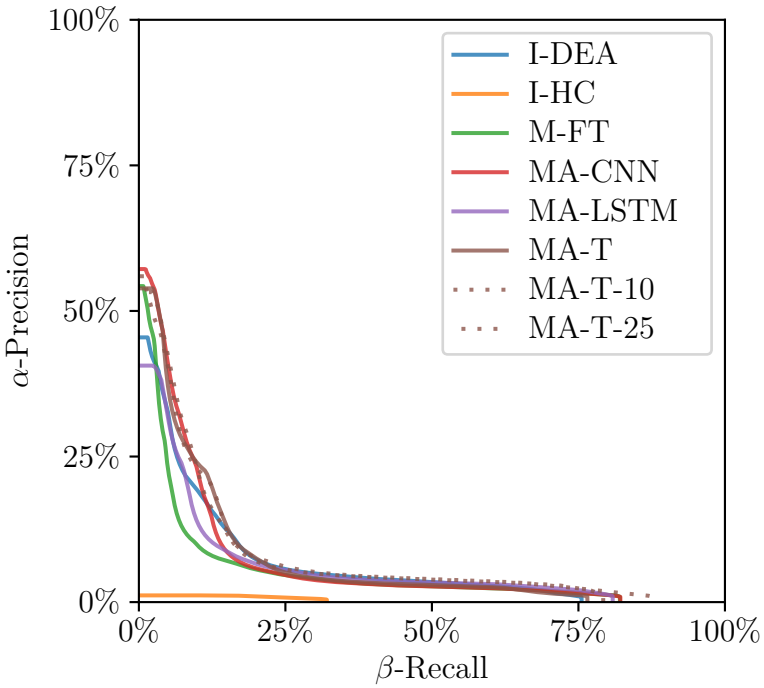
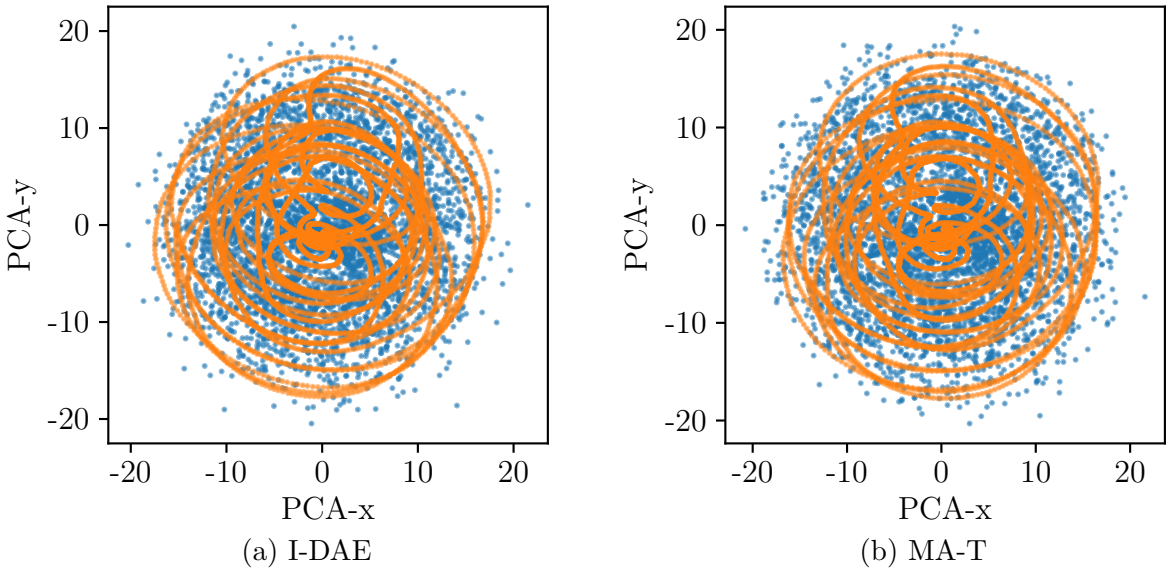
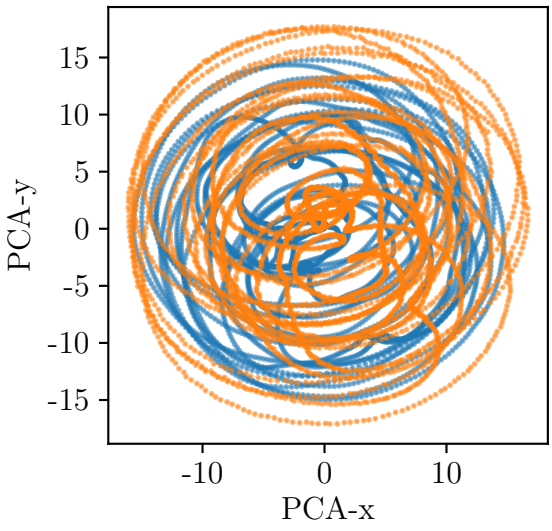


Figure 20. α-Precision and β-Recall for style transfer from simulation to high volatility data.

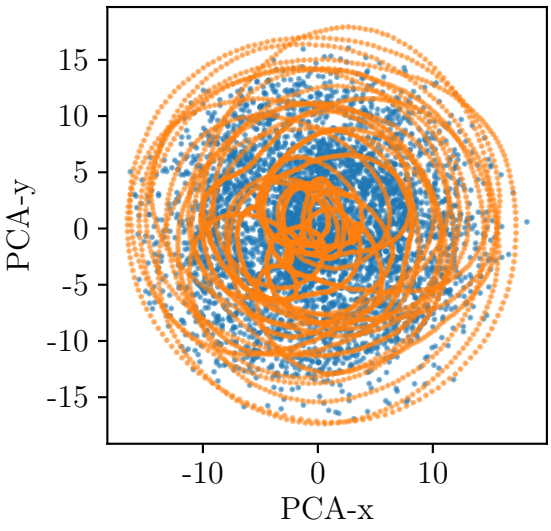


(a) I-DAE

(b) MA-T



(c) M-FT



(d) MA-LSTM

Figure 21. PCA plots comparing low dimensional representations of generated samples and style samples in style transfer from smooth to noisy data.

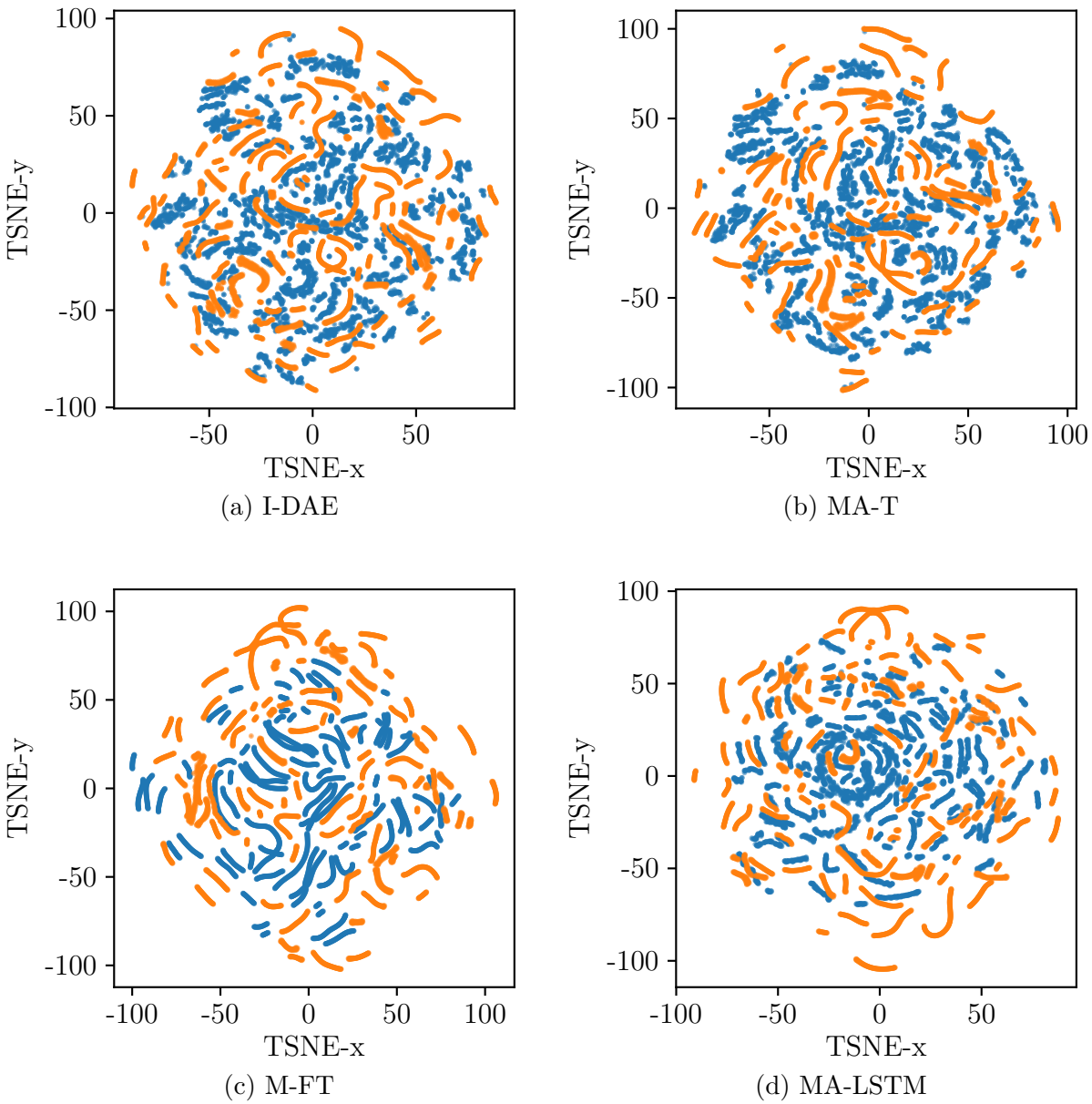
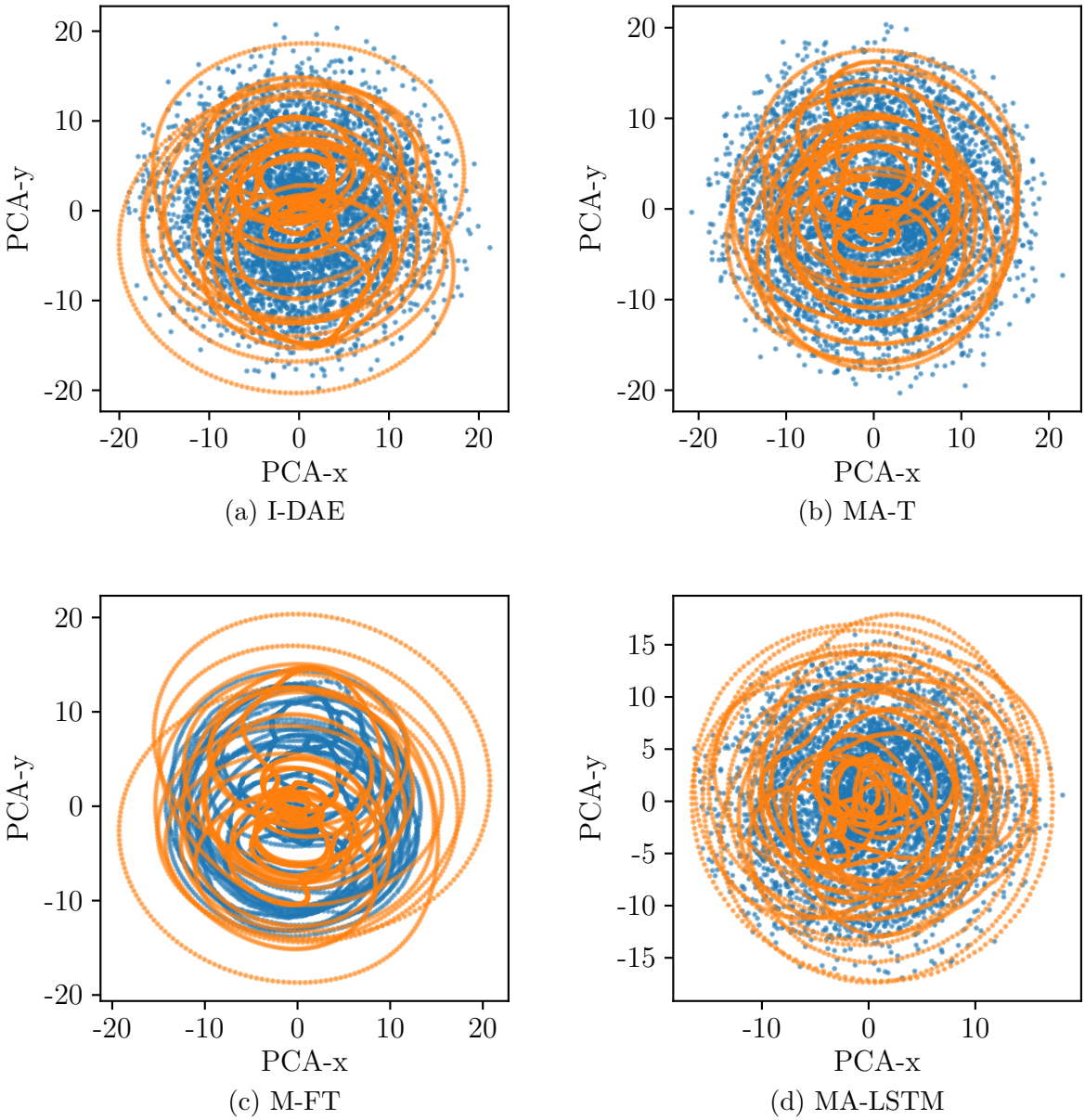


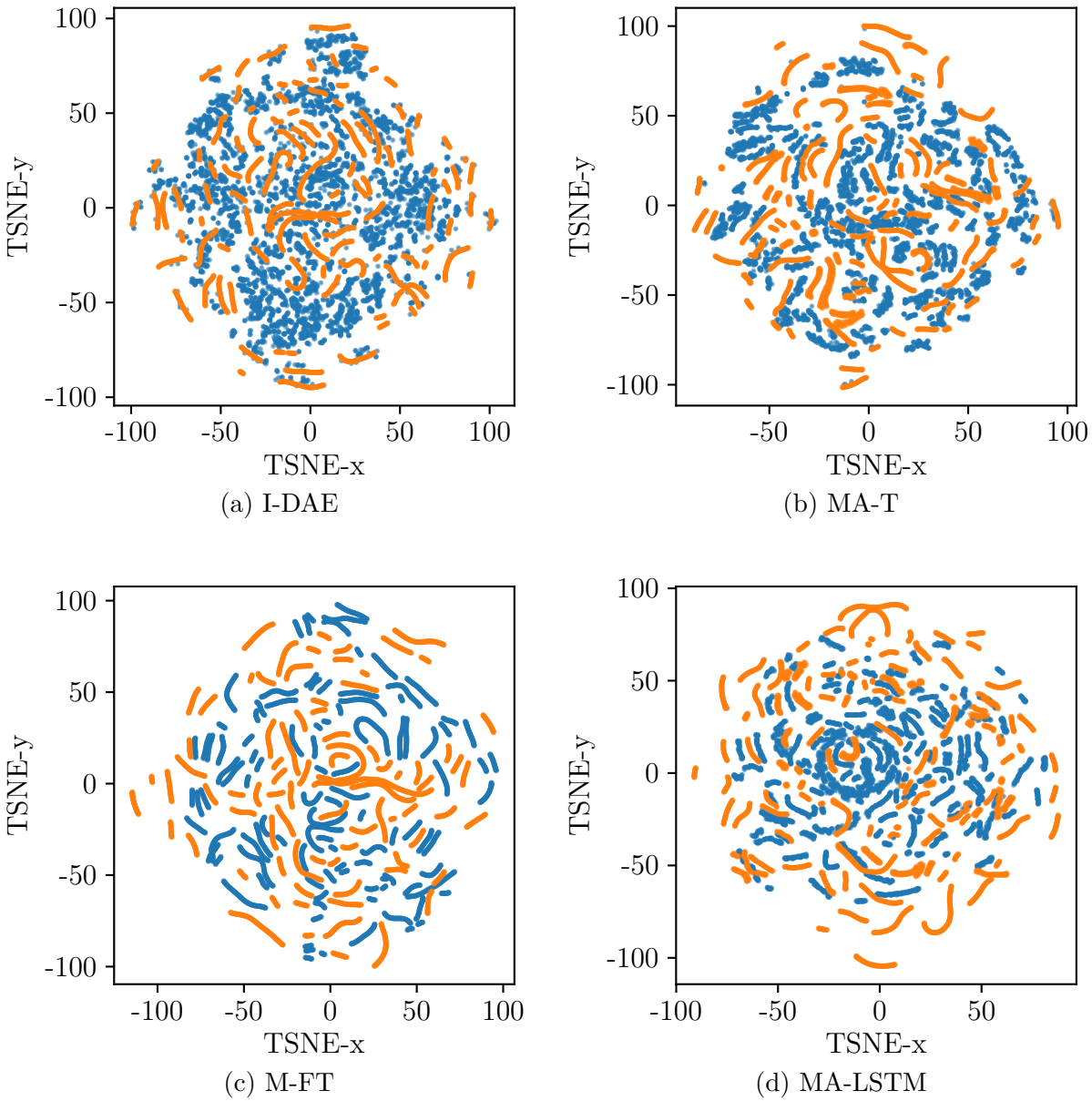
Figure 22. TSNE plots comparing low dimensional representations of generated samples and style samples in style transfer from smooth to noisy data.



(a) I-DAE (b) MA-T

(c) M-FT (d) MA-LSTM

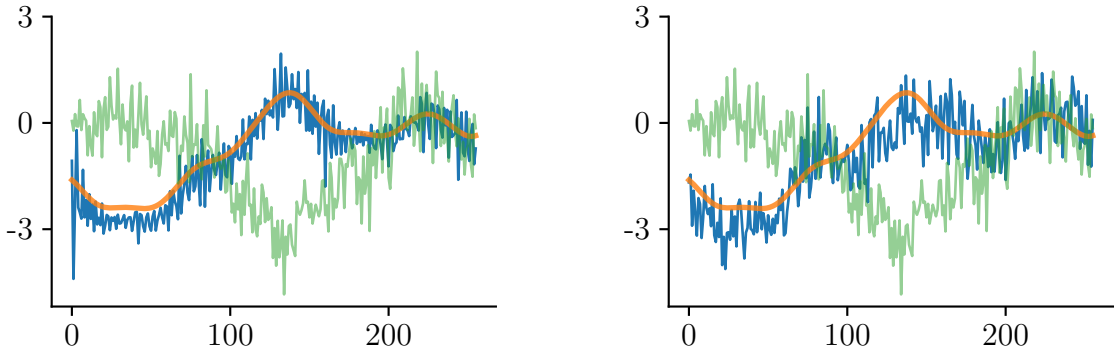
Figure 23. PCA plots comparing low dimensional representations of generated samples and style samples in style transfer from noisy to smooth data.



(a) I-DAE (b) MA-T

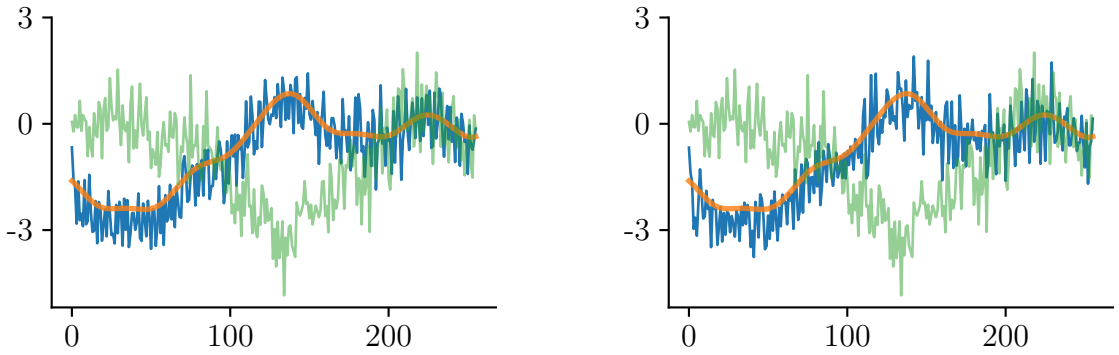
(c) M-FT (d) MA-LSTM

Figure 24. TSNE plots comparing low dimensional representations of generated samples and style samples in style transfer from noisy to smooth data.



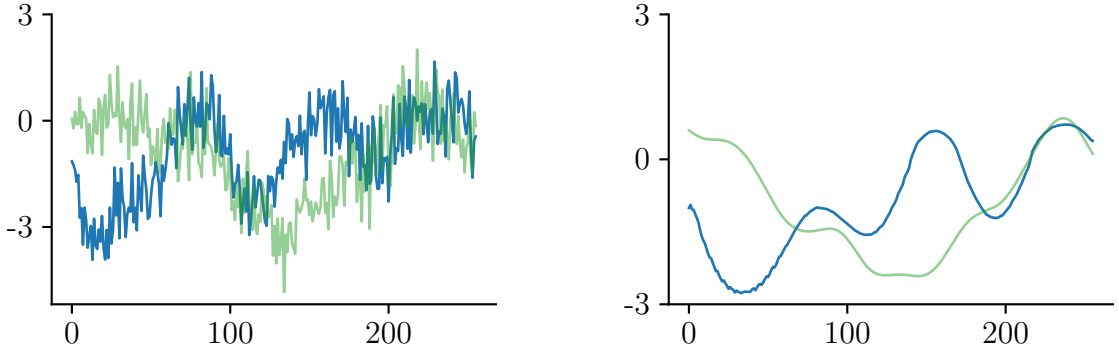
(a) content: 1st layer, style: 1st layer (b) content: 3rd layer, style: 2nd layer

Figure 25. Data-based style transfer from smooth to noisy data with different features. Depicted is a generated sample (blue) with the content (orange) and style (green) sample.



(a) feature co-occurence as in (3.1) (b) feature mean and std as in (3.2)

Figure 26. Data-based style transfer from smooth to noisy data with different style losses. Depicted is a generated sample (blue) with the content (orange) and style (green) sample.



(a) Noisy data (b) Smooth data

Figure 27. Data-based style replication with different style samples. Depicted is a generated style sample (blue) with the original style sample (green).

	content		style		
	$\mathcal{L}_c \downarrow$	$\mathcal{L}_{c:d} \downarrow$	$\mathcal{L}_s \downarrow$	$\mathcal{L}_{s:d} \downarrow$	$\mathcal{L}_\Sigma \downarrow$
I-DAE	$3.88 \cdot 10^{-2}$	$6.37 \cdot 10^{-1}$	$6.69 \cdot 10^{-4}$	$3.31 \cdot 10^{-1}$	$\mathbf{4.73 \cdot 10^{-2}}$
I-HC	$3.91 \cdot 10^{-2}$	$7.29 \cdot 10^{-1}$	$3.30 \cdot 10^{-2}$	$\mathbf{2.95 \cdot 10^{-2}}$	$1.25 \cdot 10^{-1}$
M-FT	$\mathbf{2.23 \cdot 10^{-2}}$	$\mathbf{2.49 \cdot 10^{-1}}$	$1.45 \cdot 10^{-2}$	$8.49 \cdot 10^{-1}$	$6.04 \cdot 10^{-2}$
MA-CNN	$2.49 \cdot 10^{-2}$	$2.73 \cdot 10^{-1}$	$1.24 \cdot 10^{-2}$	$3.30 \cdot 10^{-1}$	$7.65 \cdot 10^{-2}$
MA-LSTM	$2.49 \cdot 10^{-2}$	$\mathbf{2.17 \cdot 10^{-1}}$	$1.16 \cdot 10^{-2}$	$3.30 \cdot 10^{-1}$	$6.30 \cdot 10^{-2}$
MA-T	$2.69 \cdot 10^{-2}$	$3.66 \cdot 10^{-1}$	$3.23 \cdot 10^{-3}$	$3.31 \cdot 10^{-1}$	$5.87 \cdot 10^{-2}$
MA-T-10	$2.78 \cdot 10^{-2}$	$3.55 \cdot 10^{-1}$	$8.06 \cdot 10^{-4}$	$3.30 \cdot 10^{-1}$	$5.04 \cdot 10^{-2}$
MA-T-25	$2.73 \cdot 10^{-2}$	$3.58 \cdot 10^{-1}$	$\mathbf{5.68 \cdot 10^{-4}}$	$3.29 \cdot 10^{-1}$	$5.38 \cdot 10^{-2}$

	style					
	$\alpha^* \uparrow$	$\beta^* \uparrow$	F $\uparrow$	Acc $\downarrow$	Rec $\downarrow$	MAE $\downarrow$
I-DAE	**71.02**%	**71.06**%	**71.04**%	**50.54**%	51.47%	$6.95 \cdot 10^{-1}$
I-HC	13.71%	7.04%	9.31%	**48.82**%	**48.41**%	$8.86 \cdot 10^{-1}$
M-FT	57.23%	53.47%	55.28%	99.84%	99.68%	$7.03 \cdot 10^{-1}$
MA-CNN	58.55%	58.83%	58.69%	56.94%	55.04%	$8.50 \cdot 10^{-1}$
MA-LSTM	64.76%	65.04%	64.89%	53.97%	54.76%	$\mathbf{6.77 \cdot 10^{-1}}$
MA-T	**70.92**%	**70.92**%	**70.92**%	92.67%	88.88%	$\mathbf{6.73 \cdot 10^{-1}}$
MA-T-10	**70.69**%	**70.68**%	**70.68**%	88.54%	84.18%	$\mathbf{6.53 \cdot 10^{-1}}$
MA-T-25	**70.90**%	**70.92**%	**70.91**%	88.90%	84.26%	$\mathbf{6.70 \cdot 10^{-1}}$

Table 3. Results on style transfer from smooth to noisy data, $\lambda_c = 1$, $\lambda_s = 10$.

	content		style		
	$\mathcal{L}_c \downarrow$	$\mathcal{L}_{c:d} \downarrow$	$\mathcal{L}_s \downarrow$	$\mathcal{L}_{s:d} \downarrow$	$\mathcal{L}_\Sigma \downarrow$
I-DAE	$2.63 \cdot 10^{-2}$	$4.03 \cdot 10^{-1}$	$\mathbf{4.38 \cdot 10^{-4}}$	$9.27 \cdot 10^{-1}$	$2.52 \cdot 10^{-2}$
I-HC	$3.90 \cdot 10^{-2}$	$7.37 \cdot 10^{-1}$	$3.35 \cdot 10^{-2}$	$\mathbf{3.00 \cdot 10^{-1}}$	$\mathbf{1.29 \cdot 10^{-2}}$
M-FT	$\mathbf{2.07 \cdot 10^{-2}}$	$\mathbf{1.91 \cdot 10^{-1}}$	$1.31 \cdot 10^{-2}$	$8.40 \cdot 10^{-1}$	$4.54 \cdot 10^{-2}$
MA-CNN	$2.36 \cdot 10^{-2}$	$\mathbf{1.98 \cdot 10^{-1}}$	$1.34 \cdot 10^{-2}$	$8.54 \cdot 10^{-1}$	$4.95 \cdot 10^{-2}$
MA-LSTM	$2.32 \cdot 10^{-2}$	$2.15 \cdot 10^{-1}$	$1.09 \cdot 10^{-2}$	$8.49 \cdot 10^{-1}$	$3.02 \cdot 10^{-2}$
MA-T	$2.73 \cdot 10^{-2}$	$3.86 \cdot 10^{-1}$	$7.10 \cdot 10^{-4}$	$8.96 \cdot 10^{-1}$	$3.33 \cdot 10^{-2}$
MA-T-10	$2.58 \cdot 10^{-2}$	$3.83 \cdot 10^{-1}$	$4.87 \cdot 10^{-4}$	$8.92 \cdot 10^{-1}$	$2.65 \cdot 10^{-2}$
MA-T-25	$2.56 \cdot 10^{-2}$	$3.77 \cdot 10^{-1}$	$\mathbf{4.45 \cdot 10^{-4}}$	$8.92 \cdot 10^{-1}$	$2.63 \cdot 10^{-2}$

	style					
	$\alpha^* \uparrow$	$\beta^* \uparrow$	F $\uparrow$	Acc $\downarrow$	Rec $\downarrow$	MAE $\downarrow$
I-DAE	**71.03**%	**71.08**%	**71.06**%	52.57%	**50.70**%	$1.10 \cdot 10^{-1}$
I-HC	13.68%	7.07%	9.32%	**47.21**%	**46.85**%	$9.02 \cdot 10^{-1}$
M-FT	69.26%	68.77%	69.01%	**50.58**%	**47.33**%	$1.13 \cdot 10^{-1}$
MA-CNN	63.85%	63.32%	63.59%	56.04%	54.36%	$7.83 \cdot 10^{-2}$
MA-LSTM	69.23%	69.19%	69.21%	68.27%	70.65%	$8.22 \cdot 10^{-2}$
MA-T	**70.84**%	**70.91**%	**70.87**%	52.97%	55.76%	$\mathbf{5.42 \cdot 10^{-2}}$
MA-T-10	**71.04**%	**71.06**%	**71.05**%	51.06%	51.83%	$7.26 \cdot 10^{-2}$
MA-T-25	**70.97**%	**71.02**%	**71.00**%	52.33%	50.46%	$7.44 \cdot 10^{-2}$

Table 4. Results on style transfer from noisy to smooth data, $\lambda_c = 1$, $\lambda_s = 10$.

	content		style		
	$\mathcal{L}_c \downarrow$	$\mathcal{L}_{c:d} \downarrow$	$\mathcal{L}_s \downarrow$	$\mathcal{L}_{s:d} \downarrow$	$\mathcal{L}_\sigma \downarrow$
I-DAE	$1.55 \cdot 10^{-2}$	$2.13 \cdot 10^{-1}$	$3.23 \cdot 10^{-3}$	$1.64 \cdot 10^{-0}$	$4.19 \cdot 10^{-0}$
I-HC	$4.64 \cdot 10^{-2}$	$1.78 \cdot 10^{-1}$	$5.14 \cdot 10^{-2}$	$\mathbf{2.61 \cdot 10^{-3}}$	$\mathbf{3.47 \cdot 10^{-3}}$
M-FT	$\mathbf{1.34 \cdot 10^{-3}}$	$\mathbf{2.16 \cdot 10^{-2}}$	$1.09 \cdot 10^{-3}$	$2.84 \cdot 10^{-1}$	$6.42 \cdot 10^{-2}$
MA-CNN	$8.32 \cdot 10^{-3}$	$5.57 \cdot 10^{-2}$	$1.20 \cdot 10^{-3}$	$2.96 \cdot 10^{-1}$	$5.56 \cdot 10^{-2}$
MA-LSTM	$3.26 \cdot 10^{-3}$	$4.19 \cdot 10^{-2}$	$8.72 \cdot 10^{-4}$	$2.75 \cdot 10^{-1}$	$5.38 \cdot 10^{-2}$
MA-T	$2.21 \cdot 10^{-3}$	$3.28 \cdot 10^{-2}$	$6.70 \cdot 10^{-4}$	$2.86 \cdot 10^{-1}$	$7.29 \cdot 10^{-2}$
MA-T-10	$2.28 \cdot 10^{-3}$	$3.33 \cdot 10^{-2}$	$\mathbf{5.10 \cdot 10^{-4}}$	$2.67 \cdot 10^{-1}$	$7.31 \cdot 10^{-2}$
MA-T-25	$2.88 \cdot 10^{-3}$	$4.14 \cdot 10^{-2}$	$\mathbf{5.34 \cdot 10^{-4}}$	$2.58 \cdot 10^{-1}$	$4.66 \cdot 10^{-2}$

	style					
	$\alpha^* \uparrow$	$\beta^* \uparrow$	F $\uparrow$	Acc $\downarrow$	Rec $\downarrow$	MAE $\downarrow$
I-DAE	44.08%	53.35%	48.27%	95.92%	100.0%	$3.18 \cdot 10^{-1}$
I-HC	7.43%	7.46%	7.45%	100.0%	100.0%	$3.75 \cdot 10^{-1}$
M-FT	42.30%	52.66%	46.92%	81.25%	93.75%	$3.25 \cdot 10^{-1}$
MA-CNN	40.24%	47.63%	43.62%	87.50%	100.0%	$4.54 \cdot 10^{-1}$
MA-LSTM	40.13%	46.06%	42.89%	79.69%	100.0%	$3.47 \cdot 10^{-1}$
MA-T	48.35%	55.26%	51.58%	81.25%	100.0%	$3.18 \cdot 10^{-1}$
MA-T-10	**53.31**%	**58.97**%	**56.00**%	**70.31**%	**90.63**%	$\mathbf{3.06 \cdot 10^{-1}}$
MA-T-25	42.39%	50.12%	45.93%	85.94%	96.88%	$\mathbf{3.02 \cdot 10^{-1}}$

Table 5. Results on style transfer from low to high volatility data, $\lambda_c = 1$, $\lambda_s = 10$.

	content		style		
	$\mathcal{L}_c \downarrow$	$\mathcal{L}_{c:d} \downarrow$	$\mathcal{L}_s \downarrow$	$\mathcal{L}_{s:d} \downarrow$	$\mathcal{L}_\sigma \downarrow$
I-DAE	$\mathbf{8.00 \cdot 10^{-3}}$	$1.15 \cdot 10^{-1}$	$3.23 \cdot 10^{-3}$	$7.83 \cdot 10^{-1}$	$2.79 \cdot 10^{-1}$
I-HC	$1.37 \cdot 10^{-1}$	$1.69 \cdot 10^{-1}$	$1.14 \cdot 10^{-2}$	$\mathbf{5.05 \cdot 10^{-2}}$	$\mathbf{2.71 \cdot 10^{-3}}$
M-FT	$9.93 \cdot 10^{-3}$	$\mathbf{5.75 \cdot 10^{-2}}$	$4.43 \cdot 10^{-3}$	$6.76 \cdot 10^{-1}$	$1.21 \cdot 10^{-1}$
MA-CNN	$2.49 \cdot 10^{-2}$	$1.32 \cdot 10^{-1}$	$4.90 \cdot 10^{-3}$	$8.99 \cdot 10^{-1}$	$1.70 \cdot 10^{-1}$
MA-LSTM	$1.37 \cdot 10^{-2}$	$1.33 \cdot 10^{-1}$	$3.60 \cdot 10^{-3}$	$6.86 \cdot 10^{-1}$	$1.13 \cdot 10^{-1}$
MA-T	$1.03 \cdot 10^{-2}$	$1.63 \cdot 10^{-1}$	$3.13 \cdot 10^{-3}$	$6.57 \cdot 10^{-1}$	$1.07 \cdot 10^{-1}$
MA-T-10	$9.37 \cdot 10^{-3}$	$1.54 \cdot 10^{-1}$	$\mathbf{2.27 \cdot 10^{-3}}$	$6.69 \cdot 10^{-1}$	$1.05 \cdot 10^{-1}$
MA-T-25	$9.34 \cdot 10^{-3}$	$1.51 \cdot 10^{-1}$	$\mathbf{2.25 \cdot 10^{-3}}$	$6.60 \cdot 10^{-1}$	$1.03 \cdot 10^{-1}$

	style					
	$\alpha^* \uparrow$	$\beta^* \uparrow$	F $\uparrow$	Acc $\downarrow$	Rec $\downarrow$	MAE $\downarrow$
I-DAE	11.58%	16.88%	13.74%	**98.08**%	**100.0**%	$\mathbf{3.06 \cdot 10^{-1}}$
I-HC	1.63%	4.56%	2.41%	**98.46**%	**100.0**%	$4.51 \cdot 10^{-1}$
M-FT	16.43%	17.57%	16.99%	**98.46**%	**100.0**%	$3.52 \cdot 10^{-1}$
MA-CNN	15.22%	20.94%	17.63%	**98.07**%	**100.0**%	$5.29 \cdot 10^{-1}$
MA-LSTM	**17.75**%	**22.67**%	**19.91**%	**98.46**%	**99.23**%	$3.31 \cdot 10^{-1}$
MA-T	14.75%	18.98%	16.61%	**97.69**%	**100.0**%	$\mathbf{3.09 \cdot 10^{-1}}$
MA-T-10	13.46%	14.51%	13.97%	**98.46**%	**100.0**%	$3.19 \cdot 10^{-1}$
MA-T-25	15.41%	20.29%	17.52%	**98.46**%	**100.0**%	$\mathbf{2.98 \cdot 10^{-1}}$

Table 6. Results on style transfer from simulation to high volatility data, $\lambda_c = 1$, $\lambda_s = 10$.

	content	style		
	$\mathcal{L}_{c:d}\downarrow$	$\mathcal{L}_{s:d}\downarrow$	$\mathcal{L}_{\Sigma}\downarrow$	MAE$\downarrow$
I-DAE	$3.47 \cdot 10^{-1}$	$\mathbf{3.31 \cdot 10^{-1}}$	$5.43 \cdot 10^{-2}$	$\mathbf{6.92 \cdot 10^{-1}}$
I-S1	$\mathbf{2.35 \cdot 10^{-1}}$	$3.58 \cdot 10^{-1}$	$\mathbf{4.87 \cdot 10^{-2}}$	$\mathbf{7.10 \cdot 10^{-1}}$
I-C3	$1.59 \cdot 10^{-0}$	$\mathbf{3.46 \cdot 10^{-1}}$	$7.87 \cdot 10^{-2}$	$7.48 \cdot 10^{-1}$
I-GRAM	$3.28 \cdot 10^{-1}$	$\mathbf{3.48 \cdot 10^{-1}}$	$7.94 \cdot 10^{-2}$	$\mathbf{7.02 \cdot 10^{-1}}$

	style				
	$\alpha^*\uparrow$	$\beta^*\uparrow$	F$\uparrow$	Acc$\downarrow$	Rec$\downarrow$
I-DEA	**71.01**%	**71.01**%	**71.01**%	**50.06**%	**48.57**%
I-S1	**71.09**%	**71.09**%	**71.09**%	**48.37**%	**44.68**%
I-C3	**70.17**%	**70.26**%	**70.22**%	**48.51**%	**49.66**%
I-GRAM	**70.70**%	**70.71**%	**70.70**%	53.19%	51.18%

Table 7. Effect of method-invariant design choices on style transfer from smooth to noisy data on I-DEA, $\lambda_c = 1$, $\lambda_s = 10$. I-S1 uses style features from an earlier layer, I-C3 uses content features from a later layer, and I-GRAM uses a style loss based on feature co-occurrence. Also compare figures 25 and 26.

	content		style		
	$\mathcal{L}_{c}\downarrow$	$\mathcal{L}_{c:d}\downarrow$	$\mathcal{L}_{s}\downarrow$	$\mathcal{L}_{s:d}\downarrow$	$\mathcal{L}_{\sigma}\downarrow$
$\lambda_s/\lambda_c = 10^0$	$\mathbf{2.53 \cdot 10^{-3}}$	$\mathbf{5.27 \cdot 10^{-2}}$	$8.91 \cdot 10^{-3}$	$\mathbf{1.69 \cdot 10^{-0}}$	$3.71 \cdot 10^{-1}$
$\lambda_s/\lambda_c = 10^1$	$1.30 \cdot 10^{-2}$	$2.29 \cdot 10^{-1}$	$4.68 \cdot 10^{-3}$	$\mathbf{1.73 \cdot 10^{-0}}$	$3.45 \cdot 10^{-1}$
$\lambda_s/\lambda_c = 10^2$	$4.95 \cdot 10^{-2}$	$4.97 \cdot 10^{-1}$	$3.27 \cdot 10^{-3}$	$\mathbf{1.71 \cdot 10^{-0}}$	$3.73 \cdot 10^{-1}$
$\lambda_s/\lambda_c = 10^3$	$2.69 \cdot 10^{-1}$	$1.54 \cdot 10^{-1}$	$\mathbf{6.70 \cdot 10^{-4}}$	$\mathbf{1.67 \cdot 10^{-0}}$	$3.66 \cdot 10^{-1}$
$\lambda_s/\lambda_c \approx 10^{2.3}$	$7.21 \cdot 10^{-2}$	$5.69 \cdot 10^{-1}$	$3.18 \cdot 10^{-3}$	$\mathbf{1.70 \cdot 10^{-0}}$	$\mathbf{3.22 \cdot 10^{-1}}$

	style					
	$\alpha^*\uparrow$	$\beta^*\uparrow$	F$\uparrow$	Acc$\downarrow$	Rec$\downarrow$	MAE$\downarrow$
$\lambda_s/\lambda_c = 10^0$	44.71%	52.43%	48.26%	99.23%	**100.0**%	$\mathbf{3.04 \cdot 10^{-1}}$
$\lambda_s/\lambda_c = 10^1$	46.47%	56.11%	50.96%	98.72%	**100.0**%	$3.36 \cdot 10^{-1}$
$\lambda_s/\lambda_c = 10^2$	46.60%	55.28%	50.57%	99.74%	**100.0**%	$3.26 \cdot 10^{-1}$
$\lambda_s/\lambda_c = 10^3$	**50.11**%	**58.73**%	**54.08**%	100.0%	**100.0**%	$4.01 \cdot 10^{-1}$
$\lambda_s/\lambda_c \approx 10^{2.3}$	45.07%	54.95%	49.52%	**96.43**%	**100.0**%	$\mathbf{3.14 \cdot 10^{-1}}$

Table 8. Effect of λ_c and λ_s on style transfer from low to high volatility data with MA-T. The last row is the result of including λ_c and λ_s in the Bayesian hyperparameter tuning.

	content		style		
	$\mathcal{L}_c \downarrow$	$\mathcal{L}_{c:d} \downarrow$	$\mathcal{L}_s \downarrow$	$\mathcal{L}_{s:d} \downarrow$	$\mathcal{L}_\Sigma \downarrow$
I-DAE-5	$6.30 \cdot 10^{-2}$	$1.25 \cdot 10^{-0}$	$5.28 \cdot 10^{-3}$	$\mathbf{3.30 \cdot 10^{-1}}$	$9.43 \cdot 10^{-2}$
I-DAE-10	$5.37 \cdot 10^{-2}$	$1.10 \cdot 10^{-0}$	$3.79 \cdot 10^{-3}$	$\mathbf{3.30 \cdot 10^{-1}}$	$6.90 \cdot 10^{-2}$
I-DAE-25	$3.56 \cdot 10^{-2}$	$5.56 \cdot 10^{-1}$	$8.77 \cdot 10^{-4}$	$\mathbf{3.31 \cdot 10^{-1}}$	$\mathbf{4.62 \cdot 10^{-2}}$
I-DAE-100	$\mathbf{2.71 \cdot 10^{-2}}$	$\mathbf{3.55 \cdot 10^{-1}}$	$4.78 \cdot 10^{-4}$	$\mathbf{3.31 \cdot 10^{-1}}$	$5.25 \cdot 10^{-2}$
I-DAE-500	$\mathbf{2.67 \cdot 10^{-2}}$	$\mathbf{3.47 \cdot 10^{-1}}$	$\mathbf{4.44 \cdot 10^{-4}}$	$\mathbf{3.31 \cdot 10^{-1}}$	$5.43 \cdot 10^{-2}$
MA-T	$\mathbf{2.69 \cdot 10^{-2}}$	$3.66 \cdot 10^{-1}$	$3.23 \cdot 10^{-3}$	$\mathbf{3.31 \cdot 10^{-1}}$	$5.87 \cdot 10^{-2}$
MA-T-5	$\mathbf{2.77 \cdot 10^{-2}}$	$\mathbf{3.50 \cdot 10^{-1}}$	$1.04 \cdot 10^{-3}$	$\mathbf{3.31 \cdot 10^{-1}}$	$\mathbf{4.85 \cdot 10^{-2}}$
MA-T-10	$2.78 \cdot 10^{-2}$	$\mathbf{3.55 \cdot 10^{-1}}$	$8.06 \cdot 10^{-4}$	$\mathbf{3.31 \cdot 10^{-1}}$	$5.04 \cdot 10^{-2}$
MA-T-25	$\mathbf{2.73 \cdot 10^{-2}}$	$\mathbf{3.58 \cdot 10^{-1}}$	$5.68 \cdot 10^{-4}$	$\mathbf{3.29 \cdot 10^{-1}}$	$5.38 \cdot 10^{-2}$
MA-T-100	$\mathbf{2.68 \cdot 10^{-2}}$	$\mathbf{3.52 \cdot 10^{-1}}$	$\mathbf{4.54 \cdot 10^{-4}}$	$\mathbf{3.31 \cdot 10^{-1}}$	$5.50 \cdot 10^{-2}$
MA-T-500	$\mathbf{2.64 \cdot 10^{-2}}$	$\mathbf{3.45 \cdot 10^{-1}}$	$\mathbf{4.45 \cdot 10^{-4}}$	$\mathbf{3.31 \cdot 10^{-1}}$	$5.45 \cdot 10^{-2}$

	style					
	$\alpha^* \uparrow$	$\beta^* \uparrow$	F $\uparrow$	Acc $\downarrow$	Rec $\downarrow$	MAE $\downarrow$
I-DAE-5	39.76%	39.85%	39.30%	52.23%	51.43%	$7.48 \cdot 10^{-1}$
I-DAE-10	62.61%	63.34%	62.97%	54.86%	59.97%	$6.90 \cdot 10^{-1}$
I-DAE-25	**71.37%**	**71.39%**	**71.38%**	48.74%	51.55%	$\mathbf{6.53 \cdot 10^{-1}}$
I-DAE-100	**71.01%**	**71.01%**	**71.01%**	49.84%	49.38%	$\mathbf{6.80 \cdot 10^{-1}}$
I-DAE-500	**71.01%**	**71.01%**	**71.01%**	**50.06%**	**48.57%**	$6.92 \cdot 10^{-1}$
MA-T	**70.92%**	**70.92%**	**70.92%**	92.67%	88.88%	$6.73 \cdot 10^{-1}$
MA-T-5	**70.58%**	**70.60%**	**70.59%**	92.01%	87.88%	$\mathbf{6.53 \cdot 10^{-1}}$
MA-T-10	**70.69%**	**70.68%**	**70.68%**	88.54%	84.18%	$\mathbf{6.53 \cdot 10^{-1}}$
MA-T-25	**70.90%**	**70.92%**	**70.91%**	88.90%	84.26%	$\mathbf{6.70 \cdot 10^{-1}}$
MA-T-100	**70.83%**	**70.82%**	**70.83%**	83.64%	80.73%	$6.99 \cdot 10^{-1}$
MA-T-500	**70.98%**	**70.97%**	**70.97%**	75.75%	75.87%	$7.08 \cdot 10^{-1}$

Table 9. Effect of number of iterations in style transfer from smooth to noisy data, $\lambda_c = 1$, $\lambda_s = 10$.

References

[1] L. A. Gatys, A. S. Ecker, and M. Bethge. "Image Style Transfer using Convolutional Neural Networks". In: *Computer Vision and Pattern Recognition*. 2016, pp. 2414–2423.

[2] V. Dumoulin, J. Shlens, and M. Kudlur. "A Learned Representation For Artistic Style". In: *International Conference on Learning Representations*. 2017.

[3] M. J. Schervish. *Theory of Statistics*. Springer New York, 1955.

[4] D. Kingma and M. Welling. "Auto-Encoding Variational Bayes". In: *International Conference on Learning Representations*. 2014.

[5] S. Kullback and R. A. Leibler. "On Information and Sufficiency". In: *The Annals of Mathematical Statistics* 22 (1951), pp. 79–86.

[6] I. Higgins, L. Matthey, A. Pal, C. P. Burgess, X. Glorot, M. M. Botvinick, S. Mohamed, and A. Lerchner. "β-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework". In: *International Conference on Learning Representations*. 2017.

[7] C. Cremer, X. Li, and D. Duvenaud. "Inference Suboptimality in Variational Autoencoders". In: *International Conference on Machine Learning*. Vol. 80. 2018, pp. 1078–1086.

[8] D. Rezende and S. Mohamed. "Variational Inference with Normalizing Flows". In: *International Conference on Machine Learning*. Vol. 37. 2015, pp. 1530–1538.

[9] J. Campos, S. Meierhans, A. Djelouah, and C. Schroers. "Content Adaptive Optimization for Neural Image Compression". In: *Computer Vision and Pattern Recognition Workshops* (2019).

[10] Y. Yang, R. Bamler, and S. Mandt. "Improving Inference for Neural Image Compression". In: *Neural Information Processing Systems* 33 (2020), pp. 573–584.

[11] C. E. Shannon. "A Mathematical Theory of Communication". In: *The Bell System Technical Journal* 27 (1948), pp. 379–423.

[12] S. Hochreiter and J. Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9 (1997), pp. 1735–80.

[13] Y. Yu, X. Si, C. Hu, and J. Zhang. "A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures". In: *Neural computation* 31 (2019), pp. 1235–1270.

[14] B. Lindemann, T. Müller, H. Vietz, N. Jazdi, and M. Weyrich. “A Survey on Long Short-Term Memory Networks for Time Series Prediction”. In: *Procedia CIRP* 99 (2021), pp. 650–655.

[15] A. Graves and J. Schmidhuber. “Framewise Phoneme Classification with Bidirectional LSTM and other Neural Network Architectures”. In: *Neural Networks* 18 (2005), pp. 602–610.

[16] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. “Attention is All You Need”. In: *Neural Information Processing Systems* 30 (2017), pp. 6000–6010.

[17] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler. “Efficient Transformers: A Survey”. In: *Computing Surveys* (2022).

[18] J. L. Ba, J. R. Kiros, and G. E. Hinton. “Layer Normalization”. In: *arXiv:1607.06450* (2016).

[19] L. Neumann and A. Neumann. “Color Style Transfer Techniques using Hue, Lightness and Saturation Histogram Matching”. In: *Computer Aesthetics in Graphics, Visualization and Imaging*. 2005, pp. 111–122.

[20] X. Huang and S. Belongie. “Arbitrary Style Transfer in Real-Time with Adaptive Instance Normalization”. In: *International Conference on Computer Vision*. 2017, pp. 1501–1510.

[21] Y. Jing, Y. Yang, Z. Feng, J. Ye, Y. Yu, and M. Song. “Neural Style Transfer: A Review”. In: *Transactions on Visualization and Computer Graphics* 26 (2019), pp. 3365–3385.

[22] Y. Deng, F. Tang, X. Pan, W. Dong, C. Ma, and C. Xu. “StyTrˆ2: Unbiased Image Style Transfer with Transformers”. In: *arXiv:2105.14576* (2021).

[23] D. Jin, Z. Jin, Z. Hu, O. Vechtomova, and R. Mihalcea. “Deep Learning for Text Style Transfer: A Survey”. In: *Computational Linguistics* 48 (2022), pp. 155–205.

[24] S. Rao and J. Tetreault. “Dear Sir or Madam, May I Introduce the GYAFC Dataset: Corpus, Benchmarks and Metrics for Formality Style Transfer”. In: *arXiv:1803.06535* (2018).

[25] Y. Cao, R. Shui, L. Pan, M.-Y. Kan, Z. Liu, and T.-S. Chua. “Expertise Style Transfer: A New Task Towards Better Communication between Experts and Laymen”. In: *arXiv:2005.00701* (2020).

[26] R. Pryzant, R. D. Martinez, N. Dass, S. Kurohashi, D. Jurafsky, and D. Yang. “Automatically Neutralizing Subjective Bias in Text”. In: *AAAI Conference on Artificial Intelligence*. Vol. 34. 2020, pp. 480–489.

[27] B. Syed, G. Verma, B. V. Srinivasan, A. Natarajan, and V. Varma. “Adapting Language Models for Non-Parallel Author-Stylized Rewriting”. In: *AAAI Conference on Artificial Intelligence*. Vol. 34. 2020, pp. 9008–9015.

[28] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. “Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks”. In: *International Conference on Computer Vision*. IEEE, 2017, pp. 2223–2232.

[29] N. Takemoto, L. de M. Araújo, T. A. Coimbra, M. Tygel, S. Avila, and E. Borin. “Enriching Synthetic Data with Real Noise using Neural Style Transfer”. In: *International Congress of the Brazilian Geophysical Society*. Vol. 78. 2019.

[30] Y. El-Laham and S. Vyetrenko. “StyleTime: Style Transfer for Synthetic Time Series Generation”. In: *arXiv:2209.11306* (2022).

[31] B. Da Silva and S. S. Shi. “Style Transfer with Time Series: Generating Synthetic Financial Data”. In: *arXiv:1906.03232* (2019).

[32] M. Dogariu, L.-D. Ştefan, B. A. Boteanu, C. Lamba, B. Kim, and B. Ionescu. “Generation of Realistic Synthetic Financial Time-Series”. In: *Multimedia Computing, Communications, and Applications* 18 (2022), pp. 1–27.

[33] C. Esteban, S. L. Hyland, and G. Rätsch. “Real-Valued (Medical) Time Series Generation with Recurrent Conditional Gans”. In: *arXiv:1706.02633* (2017).

[34] J. Yoon, D. Jarrett, and M. Van der Schaar. “Time-Series Generative Adversarial Networks”. In: *Neural Information Processing Systems* 32 (2019).

[35] L. Yingzhen and S. Mandt. “Disentangled Sequential Autoencoder”. In: *International Conference on Machine Learning*. 2018, pp. 5670–5679.

[36] J. Bai, W. Wang, and C. P. Gomes. “Contrastively Disentangled Sequential Variational Autoencoder”. In: *Neural Information Processing Systems* 34 (2021), pp. 10105–10118.

[37] M. Chen, A. Radford, R. Child, J. Wu, H. Jun, D. Luan, and I. Sutskever. “Generative Pretraining from Pixels”. In: *International Conference on Machine learning*. 2020, pp. 1691–1703.

[38] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. “Bert: Pre-Training of Deep Bidirectional Transformers for Language Understanding”. In: *arXiv:1810.04805* (2018).

[39] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. “Language Models are Few-Shot Learners”. In: *Neural Information Processing Systems* 33 (2020), pp. 1877–1901.

[40] K. S. Kalyan, A. Rajasekharan, and S. Sangeetha. “AMMUS: A Survey of Transformer-based Pretrained Models in Natural Language Processing”. In: *arXiv:2108.05542* (2021).

[41] K. Han, Y. Wang, H. Chen, X. Chen, J. Guo, Z. Liu, Y. Tang, A. Xiao, C. Xu, Y. Xu, et al. “A Survey on Vision Transformer”. In: *Transactions on Pattern Analysis and Machine Intelligence* (2022).

[42] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah. “Transformers in Vision: A Survey”. In: *Computing Surveys* 54 (2022), pp. 1–41.

[43] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”. In: *arXiv:2010.11929* (2020).

[44] E. Xie, W. Wang, Z. Yu, A. Anandkumar, J. M. Alvarez, and P. Luo. “SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers”. In: *Neural Information Processing Systems* 34 (2021), pp. 12077–12090.

[45] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko. “End-to-End Object Detection with Transformers”. In: *European Conference on Computer Vision*. 2020, pp. 213–229.

[46] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever. “Zero-Shot Text-to-Image Generation”. In: *International Conference on Machine Learning*. 2021, pp. 8821–8831.

[47] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen. “Hierarchical Text-Conditional Image Generation with CLIP Latents”. In: *arXiv:2204.06125* (2022).

[48] G. Zerveas, S. Jayaraman, D. Patel, A. Bhamidipaty, and C. Eickhoff. “A Transformer-Based Framework for Multivariate Time Series Representation Learning”. In: *Conference on Knowledge Discovery & Data Mining*. 2021, pp. 2114–2124.

[49] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang. “Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting”. In: *AAAI Conference on Artificial Intelligence*. Vol. 35. 2021, pp. 11106–11115.

[50] N. Wu, B. Green, X. Ben, and S. O’Banion. “Deep Transformer Models for Time Series Forecasting: The Influenza Prevalence Case”. In: *arXiv:2001.08317* (2020).

[51] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. “Improving Language Understanding by Generative Pre-Training”. In: (2018).

[52] R. Xiong, Y. Yang, D. He, K. Zheng, S. Zheng, C. Xing, H. Zhang, Y. Lan, L. Wang, and T. Liu. “On Layer Normalization in the Transformer Architecture”. In: *International Conference on Machine Learning*. 2020, pp. 10524–10533.

[53] I. Turc, M.-W. Chang, K. Lee, and K. Toutanova. “Well-Read Students Learn Better: On the Importance of Pre-training Compact Models”. In: *arXiv:1908.08962* (2019).

[54] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. “Extracting and Composing Robust Features with Denoising Autoencoders”. In: *International Conference on Machine learning*. 2008, pp. 1096–1103.

[55] M. S. Sajjadi, O. Bachem, M. Lucic, O. Bousquet, and S. Gelly. “Assessing Generative Models via Precision and Recall”. In: *Neural Information Processing Systems* 31 (2018).

[56] A. C. Rencher. *Methods of Multivariate Analysis*. John Wiley & Sons, 2003.

[57] *Major World Indices*. Web. Yahoo Finance. Sept. 2022.

[58] R. Cont. “Empirical Properties of Asset Returns: Stylized Facts and Statistical Issues”. In: *Quantitative Finance* 1 (2001), pp. 223–236.

[59] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, D. S. Pallett, and N. L. Dahlgren. *DARPA TIMIT Acoustic Phonetic Continuous Speech Corpus*. 1993.

[60] D. P. Kingma and J. Ba. “Adam: A Method for Stochastic Optimization”. In: *arXiv:1412.6980* (2014).

[61] M. Popel and O. Bojar. “Training Tips for the Transformer Model”. In: *The Prague Bulletin of Mathematical Linguistics* 110 (2018), pp. 43–70.

[62] I. T. Jolliffe and J. Cadima. “Principal Component Analysis: A Review and Recent Developments”. In: *Philosophical Transactions of the Royal Society A* 374 (2016).

[63] L. Van der Maaten and G. Hinton. “Visualizing Data using t-SNE.” In: *Journal of Machine Learning Research* 9.11 (2008), pp. 2579–2605.

Declaration

I, Justus Will, avouch that I have created this thesis on my own and without help or sources but those explicitly mentioned and that I have marked any and all citations as such.

Kaiserslautern, November 4, 2022

Justus Will