Structured Disentangling Networks for Learning Deformation Invariant Latent Spaces

by

Kaushik Koneripalli Seetharam

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved July 2019 by the
Graduate Supervisory Committee:

Pavan Turaga, Chair
Antonia Papandreou-Suppappola
Suren Jayasuriya

ARIZONA STATE UNIVERSITY

August 2019

ABSTRACT

Disentangling latent spaces is an important research direction in the interpretability of unsupervised machine learning. Several recent works using deep learning are very effective at producing disentangled representations. However, in the unsupervised setting, there is no way to pre-specify which part of the latent space captures specific factors of variations. While this is generally a hard problem because of the non-existence of analytical expressions to capture these variations, there are certain factors like geometric transforms that can be expressed analytically. Furthermore, in existing frameworks, the disentangled values are also not interpretable. The focus of this work is to disentangle these geometric factors of variations (which turn out to be nuisance factors for many applications) from the semantic content of the signal in an interpretable manner which in turn makes the features more discriminative. Experiments are designed to show the modularity of the approach with other disentangling strategies as well as on multiple one-dimensional (1D) and two-dimensional (2D) datasets, clearly indicating the efficacy of the proposed approach.

ACKNOWLEDGMENTS

I would like to extend my sincere gratitude to Prof. Pavan Turaga for giving me an opportunity to work under his guidance in the Geometric Media Lab. Over the past year that I have been associated with the lab, I have gained tremendous amount of knowledge related to computer vision and machine learning. I would like to thank Dr. Rushil Anirudh and Suhas Lohit for actively being a part of my thesis and giving me the right amount of guidance. Our periodic discussions were not only helpful in generating new ideas, but also helped me imbibe critical thinking and research skills. I would like to thank the other members of our lab, Kowshik Thopalli and Rajhans Singh, with whom I have had wonderful discussions and consequently learnt a lot. I would like to thank my academic advisor Toni Mengert for being very helpful in all administration related issues. Finally, I would like to thank my mother, Prof. M.K. Nagamani who has provided unconditional emotional and financial support during the course of my Master's program, without which none of this would have been possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

Chapter 1

INTRODUCTION

Prior to the deep learning era, computer vision systems made use of features that were hand-engineered and task-oriented. One of the desired goals for these features was to be invariant to certain nuisance factors in the data such as affine transforms, blur etc. Works such as Lowe (2004), Gopalan *et al.* (2012) have achieved it, but these methods are hand-engineered. Modern day computer vision systems are primarily data-driven where features are *learned* by enforcing suitable constraints on the learning paradigm. Being dependant on data, it makes them susceptible to learn any noise that is implicitly there in the data (due to noise in data generating process). Note that "noise" in this case is any undesired transformation of the original data which we parametrize with a mathematical model. Thus it becomes important to learn features that are invariant to these noise factors.

Figure 1.1 illustrates the concept of invariance that is desired in the feature representation of multi-dimensional signals. Specifically, it illustrates blur, affine and rate invariance for images and time-series. Common kinds of nuisance/noise factors that creep into image datasets that are used to train computer vision systems are: affine variations, blur and brightness/illumination. Similarly time series data have a lot of rate variations. Let us collectively call them as "parametric deformation models". Thus, having a deformation invariant feature representation is crucial for many downstream computer vision and machine learning applications.

Typically in signal processing applications, *noise* corresponds to Additive White Gaussian Noise (AWGN) which is *added* to the range of the signal. This means that the signal at the receiver is assumed to be a sum of the "clean" signal and a noise term. Here, let us define noise in a slightly different way as "task-based undesired factors of variations". This

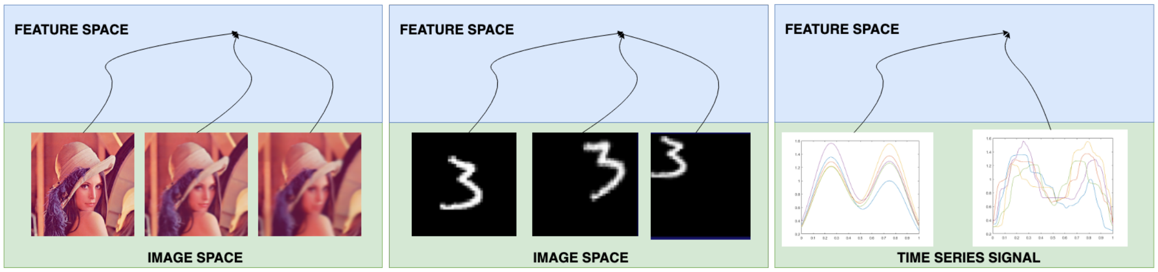

Figure 1.1: Visualization of the concept of invariance - This figure illustrates commonly desired invariances in the feature representation of image and time series data. The first and second figures represent blur and affine transform invariance. The third figure represents rate invariance in time series data.

means that certain factors of variation are desirable for some tasks and are undesirable for other tasks. For example, if the task is image classification, affine transformations can be considered noise as it can disrupt the classifier's performance. However, if the task is to generate new images with similar geometric variations as that of a particular image in the dataset, affine transforms are not considered noise and is in fact desirable to know them explicitly. Since the term "nuisance factor" is task dependant, it is desirable to have an invariant representation to these nuisance factors, along with the parameters that generate these nuisance factors.

Consider the clean version of a signal given by the equation:

$$\mathbf{y} = \phi(\mathbf{x})$$

Let us consider noise such that it acts on the domain of the signal. This is modeled using the equation:

$$\mathbf{y} = \phi(\mathcal{N}(\mathbf{x}; \mathbf{w})) \tag{1.1}$$

where $\mathcal{N}(.)$ is the noise that acts on the domain of the signal $\mathbf{x}$. As we saw earlier, for many signal processing applications, noise is modelled to be "additive" white gaussian which acts on the range of the signal. The above equation represents noise that is "transformative" in nature which acts on the domain of the signal. The goal is to *undo* these transformations

$\mathcal{N}(.)$ so as to retrieve the original clean signal $\phi(\mathbf{x})$ along with the parameters of $\mathcal{N}(.)$. We consider two kinds of signals for our experiments: 1-dimensional (time series) and 2-dimensional (images).

Recent works in computer vision such as Cohen and Welling (2016) have focused on achieving invariant and equivariant representations of data during the learning process. A closely related line of work is learning disentangled representations which is primarily useful in controlling the factors of data generation. In this work, we show how to achieve invariance to nuisance factors by disentangling them out from the remainder of the representation. More specifically, we provide an unsupervised data-driven approach to disentangle these nuisance factors from the canonical representation of the signal. This method is completely unsupervised in the sense that we do not even provide a reference canonical representation. It is interesting to see that the network learns a canonical representation *on its own* from the data.

1.1 Our Contributions

1. We propose a training strategy to disentangle deformation parameters from the semantic content of the signal.

2. We show modularity of approach with other disentangling strategies and improvement in class-discriminative ability of the latent space. We also show experiments indicating the efficacy of our approach for time-series alignment.

3. Finally, we show experiments on a real world dataset of hand action recognition. Using our approach, we show improvement in classification accuracy when the data is corrupted with rate variations.

Organization of Thesis: Chapter 2 begins with historically understanding the importance of invariant representations for computer vision applications. We then look at disentangled representations in unsupervised methods in Section 2.2. Finally, we conclude the chapter by discussing the focus of this thesis: to achieve invariance through disentanglement. Chapter 3 briefly describes the proposed method for one dimensional (1D) and two dimensional (2D) data. Chapter 4 presents experimental results for 1D and 2D data. It also shows the modularity of the proposed method with other existing disentangling frameworks. Chapter 5 concludes the work and discusses future lines of research.

Chapter 2

DISENTANGLED REPRESENTATIONS

This chapter begins with an introduction to the idea of disentangling latent spaces in an autoencoding model. This is followed by understanding the importance of invariant representations in computer vision. Finally, we look at how disentangled representations implicitly promote invariance and why we need to achieve interpretable disentangled representations.

2.1 Introduction to Disentanglement

Unsupervised representation learning is central to several problems in machine learning. The idea of autoencoders (Hinton and Salakhutdinov (2006)) has been at the forefront of data-driven unsupervised learning. They are neural network models that are trained with the goal of reconstructing the input. In the process of doing so, they learn a low-dimensional embedding/representation of the input. These low-dimensional embeddings (also called the latent spaces) can be used for downstream computer vision tasks such as image classification. One of the major questions that needs to be addressed is regarding the semantic interpretability of these representations. Furthermore, with the rise of unsupervised generative models like Variational Autoencoders (Kingma and Welling (2013)) and Generative Adversarial Networks (Goodfellow *et al.* (2014)), the interpretability of the latent space is all the more important in order to control the factors of variation when generating new data.

The idea of having disentangled representations is to "assign semantic meaning to different parts of the latent space". Disentangled representations contribute greatly towards interpretability of latent spaces where, each variables/chunks (group of contiguous variables) represent a certain factor of variation. As a result, disentangling enables the easy modification of latent variables in order to generate desired variations in the image space. Figure 2.1 illustrates that, when each of the latent variables/chunks (depending on framework) are perturbed, there is a corresponding smooth variation in the image space with respect to a semantic attribute. More importantly, each chunk seems to capture a different factor of variation such as hair color, gender, brightness etc.

Figure 2.1: Illustration of disentangled latent space - Interpolations taken from Hu *et al.* (2018). This figure illustrates disentangled representations captured by the latent chunk/variables. The arrow marks indicate the interpolation observed when that particular chunk/variable is varied while keeping others constant.

After the introduction of neural network based generative models like VAE and GANs, there has been a lot of work on disentangled representations. The literature in disentanglement can be broadly classified into two streams: a) works that propose new methods to disentangle latent spaces, b) works that use existing frameworks and discover new aspects to disentangle with minor modifications to the method. Kulkarni *et al.* (2015) introduced a weakly supervised way to learn disentangled representations where you can specify apriori what each latent variables represent. Mathieu *et al.* (2016) proposed a method to break down latent space into chunks of variables where each chunk captures one aspect of variation. Methods that directly build on generative frameworks like β-VAE (Higgins *et al.* (2017)), InfoGAN (Chen *et al.* (2016)) propose disentanglement by promoting statistical independence among latent variables. Makhzani *et al.* (2015) adversarially train the latent space to match the desired distribution hence promoting disentanglement.

Coming to the second stream of works, (Shu *et al.* (2018)) propose a new way to disentangle texture, shading and illumination information from the data. (Xing *et al.* (2018)) disentangles appearance from geometry information. Hu *et al.* (2018) builds on Mathieu *et al.* (2016) to improvise the mixing chunks of latent space and shows disentangling capabilities. Worrall *et al.* (2017) provides a way to achieve equivariant feature space while being able to disentangle. However, the values obtained at the latent space are not interpretable. Liu *et al.* (2018) propose a method to disentangle appearance from pose.

As we have seen above, there has been a lot of success in neural network architectures that achieve unsupervised disentanglement like β-VAE (Higgins *et al.* (2017)), Mixing Autoencoders (Hu *et al.* (2018)), Factor VAE (Kim and Mnih (2018)) etc. However, these approaches suffer from some drawbacks: (a) they cannot guarantee *a priori* which dimensions of the latent space representation correspond to a given attribute, and (b) the latent space loses its discriminative nature in the presence of structured nuisance transforms in the training set. As a result, these methods require post-hoc analysis before they can be successfully deployed into downstream applications. A priori guarantees of factors of variation is generally hard because these factors (such as thickness, style, hair-color etc.) cannot be controlled using an analytic expression. However, geometric factors of variations that can be expressed analytically, are also not captured in an interpretable fashion and we choose them as the focus of this work. In the following section we will look at the idea of invariant representations.

2.2 Invariance in Computer Vision

Finding low dimensional, task-specific feature representation is the backbone of any computer vision application. It is often desired that the features have some specific properties. Two of the most commonly desired properties are invariance and equivariance. Figure 2.2 illustrates the difference between the two kinds of representations. Equivariant representation means that any change in the input space should yield a proportional change in the feature space. Mathematically, $f(\mathcal{T}_\theta(x)) = \mathcal{F}_\theta(f(x))$, where $\mathcal{T}(.)$ is a transformation in the signal space, $\mathcal{F}(.)$ is a transformation in the feature space and $f(.)$ is a function that transforms the signal to the feature space. Invariant representation means that any change in input space should not change the representation in the feature space. Mathematically, $f(\mathcal{T}_\theta(x)) = f(x)$.

Let us focus on invariant representations. As mentioned before, real world data comes with a lot of nuisance transformations like affine, illumination, blur etc. Thus it is important for features to be invariant to these nuisance factors. Such invariant representations have proven to be crucial for the robustness of computer vision system that are developed. Traditionally achieving invariant representations have been approached by using various statistical and differential geometric frameworks (Begelfor and Werman (2006), Mumford (1994)). Existence of such invariant representations have been theoretically proved, and im-

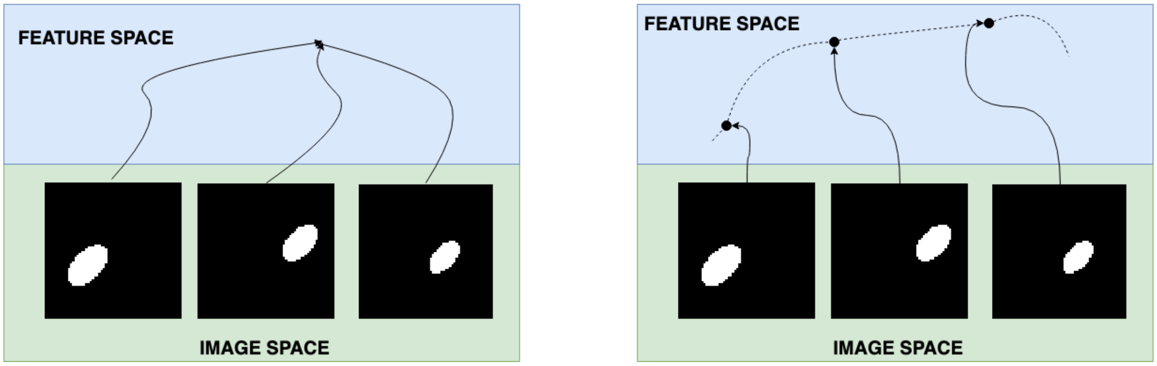

Figure 2.2: Invariant versus equivariant representations - The first row represents the UMap plots for the MAE framework. The left picture represents the latent space embedding of the MAE trained on regular MNIST data. The middle picture represents latent space embedding of MAE trained on AffNIST data. The right picture represents latent space embedding of MAE along with our training strategy trained on AffNIST data.

proved methods to obtain affine representations have also been proposed (Vedaldi (2008)). Feature extraction methods such as SIFT (Lowe (2004)) and SURF (Bay *et al.* (2006)) were proposed with the claim of scale invariance.

The main problem that still persisted was the necessity to hand-tune the parameters of these methods using experiment-based thresholds. However, they were very popular in the computer vision community for mainstream tasks like object detection until the rise of data-driven-learning era. With the advent of deep learning, feature learning became a data driven process. Learning invariant and equivariant representations were now determined by imposing suitable constraints on the training procedure. Convolutional networks were then designed with the goal of achieving such representations (Cohen and Welling (2016)). The most common aspects to which invariance is desired for images are nuisance transforms like affine, blur, illumination/lighting. Jaderberg *et al.* (2015) proposed the Spatial Transformer Networks to learn features invariant to transformations like affine, projective and thin-plate spline. (Lu *et al.* (2019)) proposed a methd to disentangle blur and in turn capture a blur invariant representation. (Lohit and Turaga (2017)) proposed a method to capture invariant representations for illumination/lighting. Invariant representations are also being explored for point cloud (Qi *et al.* (2017), Som *et al.* (2018)) and time series data (Lohit *et al.* (2019)).

2.3 Invariance Through Disentanglement

In the previous section, we have seen the importance of having invariant representations in computer vision applications. Referring to Figure 2.1, in the process of achieving disentangled representations, specific chunks/variables are actually invariant to other factors of variations. For example, in Figure 2.1, the blue variable is invariant to all changes except brightness, red variable is invariant to all changes except saturation, and so on.

However, we wish to go a step further and be able to obtain these transformation parameters along with the invariant feature representations which could potentially be useful for various applications. Consider the scenario of a mobile robot with a camera trying to move around. The robot has to be "self-aware" when it moves i.e. know its motion with respect to its surrounding objects and have a frame of reference. In other words, it has to possess egomotion. This idea of "self-awareness" is drawn from biology where living organisms developed the ability of visual perception for the purpose of moving and acting in the world. Thus it becomes important to know these transformation parameters which could potentially guide the motion of the robot. As a second example, consider data augmentation with geometric transforms. If we want to generate new data with certain geometric transformations from the dataset, disentangled representations could be useful to know these transformations explicitly. Agrawal *et al.* (2015) propose a way to learn from such scenario where you only have access to successive frames at a time. However, when we have access to an entire dataset of images, we should be able to retrieve the transformation parameters with respect to some canonical representation and this is explored in this work.

We address these goals by proposing an unsupervised training strategy to disentangle geometric factors of variations from the semantic content of the data. As desired, we will be able to achieve invariant feature representations along with the transformation parameters. Since the latent space structure corresponding to the deformation factors is enforced analytically (by using the spatial and temporal warping layers described later), we directly obtain a factored representation leading to a robust semantic latent space. We demonstrate that in the presence of deformations in a dataset, the proposed strategy is able to “undo” these deformations and also maintains/improves the class-discriminative ability of this feature space. Furthermore, we cement our claims by showing the modularity of our approach with other disentangling networks.

Chapter 3

DEFORMATION DISENTANGLING STRATEGY

In this chapter we propose a training strategy to disentangle deformation parameters from image and time series data. As described previously, we specifically focus on deformations that manifest as affine variations for image data and rate variations for time-series data. This training strategy is completely unsupervised, modular and yields interpretable latent representations. We train the autoencoder architecture with the regular $\mathcal{L}_2$ loss or any other loss as required by the disentangling strategy that is being used. In the following sub-sections, we illustrate the strategy for both images and time series.

3.1 Disentangling Affine Transforms

For a general autoencoding framework, we illustrate how our training strategy can be applied to disentangle affine transforms.

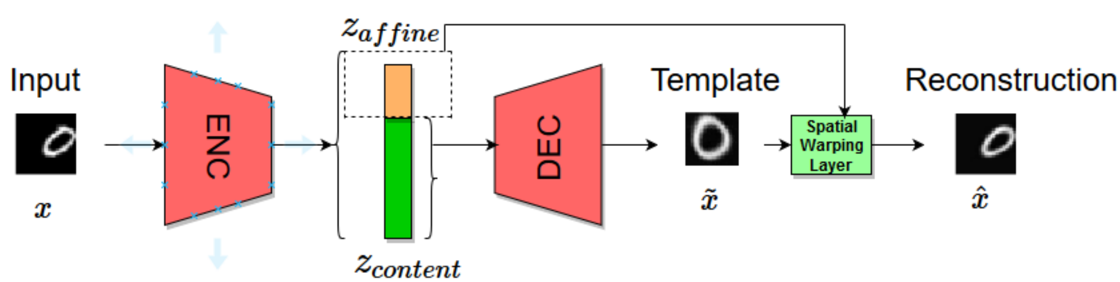

Figure 3.1: Disentangling Affine Parameters in an Unsupervised Fashion Using a Spatial Warping Layer - The encoder is used to generate two chunks: one to capture affine transform parameters and the other to capture content information. The output of decoder is a canonical/template representation which is transformed using a spatial warping layer with the predicted affine transforms.

Figure 3.1 illustrates the proposed strategy to disentangle affine transforms from the semantic content of the signal. The broad idea is to bifurcate the latent space into 2 chunks:

the affine chunk and the content chunk. As we will see in the next chapter, the content chunk can also have disentangled representations as governed by various strategies in literature. This approach is interpretable in the sense that, the affine chunk learned by the encoder is actually a 2x3 affine matrix and can be actually used to transform an image with respect to a "template" predicted by the decoder. It is important to note that only the content chunk is fed to the decoder with the idea of making it output an "affine-invariant" image which we consider to be a template/canonical representation. Finally, a spatial-warping layer (Jaderberg *et al.* (2015)) is used to transform the predicted template using the learned affine transform. Using the above described framework, we show experiments on two datasets and two disentangling frameworks as described in the next chapter.

3.2 Disentangling Rate Transforms

Since we are dealing with time series, consider an autoencoding framework based on Temporal Convolution Networks (Bai *et al.* (2018)) and/or LSTM (Hochreiter and Schmidhuber (1997)). Figure 3.2 illustrates how to disentangle rate variations (γ) from the semantic content of the time series.

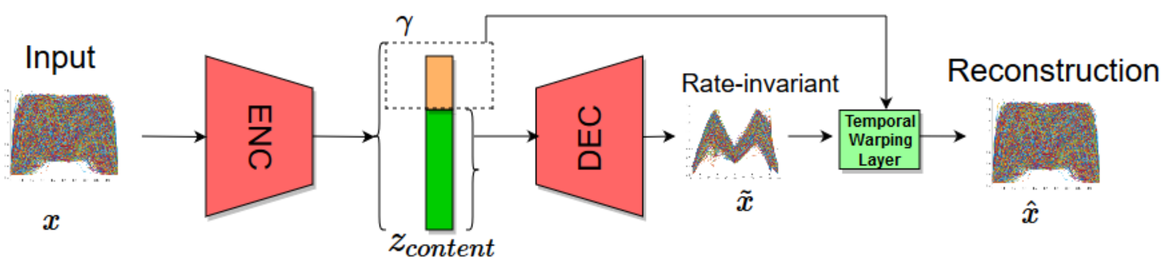

Figure 3.2: Disentanging rate parameters in an unsupervised framework using a temporal warping layer - The encoder is used to generate two chunks: one to capture rate transform parameters and the other to capture content information. The output of decoder is a canonical/template representation which is transformed using a temporal warping layer with the predicted rate transforms.

It is important to note that γ is of the same length as the input sequence and $z_{content}$ is made as small as possible. Similar to the previous case, the latent space is bifurcated into 2 chunks: the rate chunk and the content chunk. The content chunk is a low dimensional embedding of the time series signal. However, as in the case of images, we do not consider any disentangled representations for the content chunk of the time series.

Similar to the spatial warping layer by Jaderberg *et al.* (2015), for time series signals we make use of a "temporal warping layer" proposed by Lohit *et al.* (2019) to perform warping of time-series in a differentiable manner so as to be integrable within a neural network. We show that the above described framework can be used to perform time series alignment of signals while maintaining/improving class-discriminative ability of latent space. We show experiments on datasets with increasing level of difficulty.

Chapter 4

EXPERIMENTAL RESULTS

In this chapter, we show our results on different experiments carried out. For image data, we show performance on AffNIST (Tieleman (2013)) and DSprites datasets (Matthey *et al.* (2017)). For modularity, we show experiments on β-VAE (Higgins *et al.* (2017)) and MAE (Hu *et al.* (2018)). For time series signals, we show performance on two synthetic datasets and one real world dataset that are described in the subsequent sections.

4.1 Image Data

We consider the AffNIST and DSprites datasets which have many affine transform variations. We show the modularity of our approach by combining it with the following two disentangling strategies:

1. Mixing Autoencoders (MAE) (Hu *et al.* (2018))

2. β-VAE (VAE) (Higgins *et al.* (2017))

The first method involves disentangling the latent space by breaking it down into chunks. The second method involves promoting statistical independence between the latent variables. Let us see the composition of the two datasets.

4.1.1 AffNIST Dataset

The AffNIST dataset consists of affine transformed variations of the original MNIST handwritten digits dataset. There are about 32 affine variations to each of the $70,000$ images in the dataset, thus giving rise to about two million data points (train + validation + test data). The training set contains about 1.6 million examples with equal distribution among classes. Figure 4.1 illustrates a few examples from the dataset. For our experiments, we use ~ 50000 as training set and about 10000 as test set with equal distribution of classes. The range of variations of affine parameters in this dataset are:

1. Rotations: $+20$ to -20 degrees.
2. Shear: $+0.2$ to -0.2.
3. Scale: 0.8 (shrink) to 1.2 (enlarge).
4. Translations.

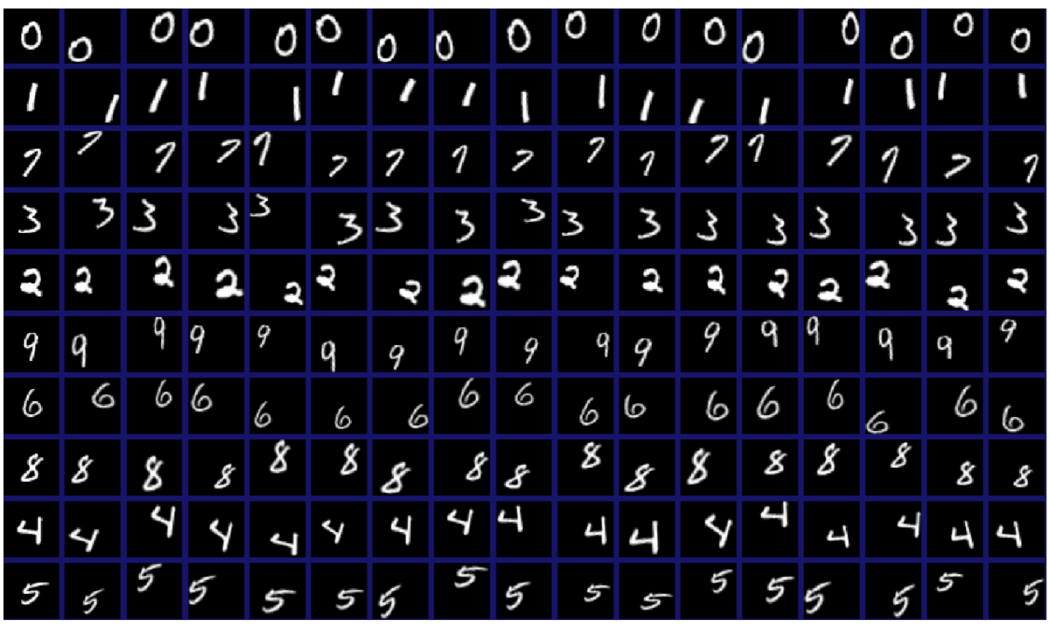

Figure 4.1: Examples from AffNIST dataset - The first column represents each of the 9 digits of different styles. The remaining columns represent an affine transformed version of the first column. Figure taken from webpage of Tieleman (2013)

4.1.2 Disentangling Sprites Dataset

The range of variations present in this dataset are:

1. Shapes: 3 - square, circle, heart

2. Scale: 6 values linearly spaced in range $[0.5, 1]$

3. Rotation: 40 values in range $[0, \pi]$

4. Translation X: 32 values in range $[0, 1]$

5. Translation Y: 32 values in range $[0, 1]$

Thus, the total size of dataset is around $700,000$. For our purpose, we consider a subset of this dataset with a more coarsely sampled set of affine transforms. Specifically, we consider only 11 values along the x and y translations. This leads to a dataset of size around $88,000$. Figure 4.2 shows a few examples from the DSprites dataset.

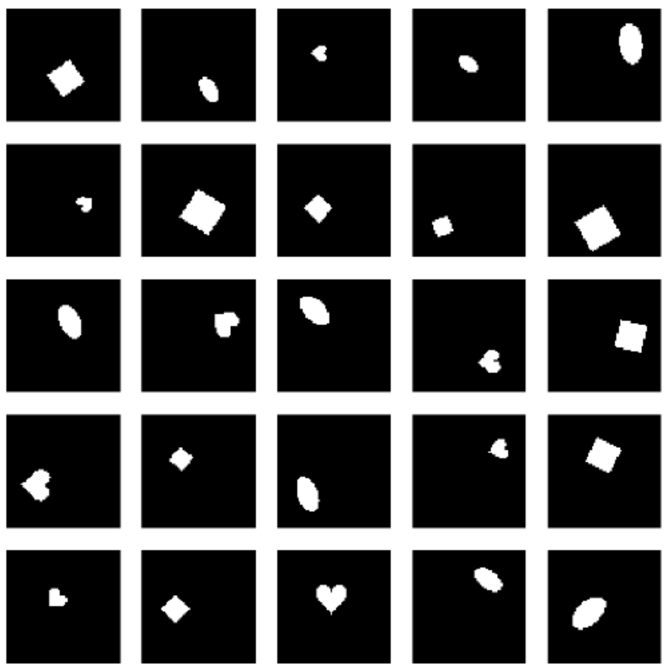

Figure 4.2: Examples from Dsprites dataset - This figure represents various affine transformed versions of all three shapes in the dataset. Figure taken from github page of Matthey *et al.* (2017)

4.1.3 Mixing Autoencoders

This method was originally proposed in Hu *et al.* (2018). The architecture is as shown in Figure 4.3. The idea of mixing is to first take 2 images (batches) and project them

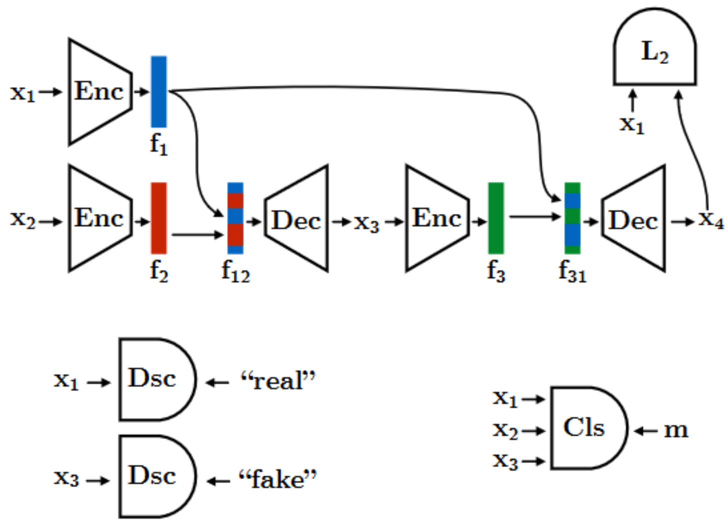

Figure 4.3: Disentangling latent spaces by mixing them - Here the latent space is bifurcated into chunks. The two encoders and decoders shown in the figure have tied weights. f_{12} is performing mixing between f_1 and f_2. f_{31} is performing un-mixing. Finally the autoencoder is trained with a weighted combination of reconstruction error, classification error and discriminator error. Figure taken from Hu *et al.* (2018)

to the latent space f_1 and f_2 respectively. Now, consider chunks (blocks of contiguous dimensions) of the latent space and "mix" the chunks of f_1 and f_2 by multiplying them with appropriate binary masks m^i and adding them. This yields the mixed latent space f_{12}. Now the mixed latent space is fed into the decoder producing an output. Intuitively, this output x_3 has a mixture of features from both x_1 and x_2. These could be anything like texture, style, brightness etc. Now this intermediate representation x_3 is again fed through the encoder to obtain latent space f_3. Ideally f_{12} and f_3 have to be the same. With this goal, "un-mixing" is performed to obtain f_{31} which is fed to the decoder to obtain x_4. The reconstruction loss is computed between x_4 and x_1. It also important to note that, for each image combination x_1 and x_2, there are $2^n - 2$ mixing combinations. For more details on how mixing is performed please refer Hu *et al.* (2018).

The proposed loss function to train the AE end to end is given by:

$$L_{total} = \lambda_1 E_{x_1,x_2}\left[\sum_m |x_4 - x_1|^2\right] + \lambda_2 \sum_m E_{x_1,x_2}\left[\log(Dsc(x_1)) + \log(1 - Dsc(x_3))\right] + \lambda_3 E_{x_1,x_2}\left[-\sum_m \sum_i m^i \log y^i + (1-m^i)\log(1-y^i)\right] \tag{4.1}$$

There are a total of three components to the loss function: a) reconstruction loss, b) classification loss and c) adversarial loss. The adversarial training ensures that the intermediate output x_3 belongs to the image distribution. The classification loss ensures that the "shortcut" problem (Szabó *et al.* (2017)) is avoided. More details about the intuition of these loss functions can be found in Hu *et al.* (2018).

Modified Mixing Architecture

We now describe the architectural modifications with our training strategy so as to learn an affine-invariant and disentangled latent space as illustrated in Figure 4.4

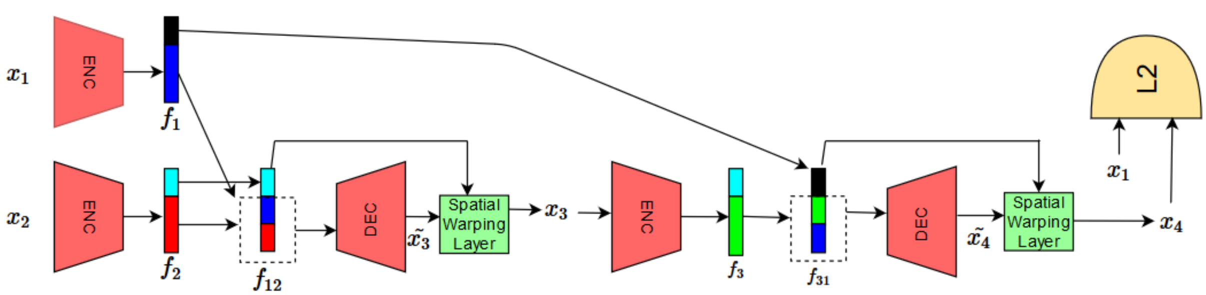

Figure 4.4: Mixing autoencoders with our training strategy - The black and cyan chunks represent the affine components with 6 variables. The blue and red chunks represent the content components.

While retaining the broad idea of mixing, we allocate the first chunk (of size six) to be the affine chunk and the remaining to be the content chunk. It is important to note that, the content chunk is also disentangled i.e. the dark blue and red chunks represent different factors of variations. Also, according to our training strategy, we feed only the mixed content chunk (dark blue and red chunks) to the decoder. The loss function to train the network is a simple $\mathcal{L}_2$ error given by:

$$L_{total} = E_{x_1,x_2}\left[\sum_m |x_4 - x_1|^2\right] \tag{4.2}$$

From our experiments we observe that the discriminator and classifier are not crucial to see disentanglement in the latent space for the datasets we consider.

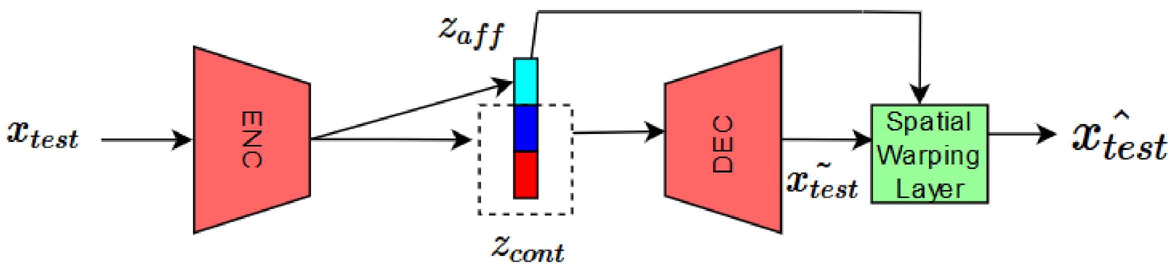

Figure 4.5: Inference architecture for modified mixing framework - The architecture is that of a simple autoencoder but with the latent space disentangled. There are three chunks in this case. The cyan chunk represents affine transform parameters and the dark blue and red chunks represent other factors of variation in the data.

Figure 4.5 shows the inference architecture. As we can notice, during the testing phase, this entire framework is just that of a simple autoencoder, but with the latent space disentangled.

As mentioned previously, a by-product of such a training procedure is template estimation i.e. the affine transform predicted by the network is w.r.t a template output by the decoder. The template need not be an "upright" version of the image, but can be any transformed version, as long as all images get warped to the same transform. As we might expect, when we train the network multiple times, we end up with different looking templates.

However, the tendency is to end up with a template which has an "average" orientation of all the images in the dataset as also observed by Jaderberg *et al.* (2015). Below we describe the architectural details used to implement the algorithm.

Architectural Details

We make use of a fully connected architecture with layer sizes $784 - 512 - 256 - 128 - 64 - 26$ for the encoder and $26 - 64 - 128 - 256 - 784$ for the decoder. The latent space dimension is 26 $(6 + 10 + 10)$ where the first six variables capture the values of an affine matrix and the remaining is divided into two chunks of ten variables each to capture the disentangled representation of the content chunk. We make use of LeakyRelu activation function with a negative slope of 0.2, Adam optimizer with learning rate of 0.0002 and momentums of 0.5 and 0.999 respectively. Also, we randomly sample the masks instead of using all the $2^n - 2$ (subtract 2 because all zeros and all ones do not constitute any mixing) combinations of possible masks, where n is the number of chunks. For all these experiments, we have divided the latent space into three chunks out of which one captures affine information. Let us now visualize and analyze the results on two datasets: AffNIST and DSprites.

AffNIST Results

Figure 4.6 and Figure 4.7 represent the estimated template and the latent space traversals respectively. During inference stage, we use the architecture given in Figure 4.5. According to Figure 4.5, the second row of Figure 4.6 represents $x_{\tilde{test}}$ i.e. the output of the decoder. As it can be noticed, all the images have the same (or very close) values of scale, translation and rotation, thus conforming to the notion of a template. Figure 4.7 represents latent space traversal along different chunks. The first row represents x_{test} when linear interpolation is performed between two digits (8 and 9) in the latent space. The second row represents

x_{test} for interpolation only along the affine chunk of the latent space. We can see that the identity of the image is retained while only inducing affine variations. The third and fourth rows represent interpolation along 2 different chunks in content part of the latent space. As it can be noticed, the two chunks represent variations in style and identity respectively.

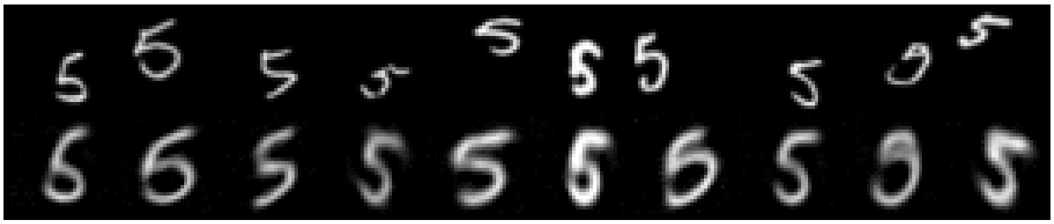

Figure 4.6: Estimated template by mixing autoencoders - AffNIST dataset - The first row indicates some examples of the number 5 from the AffNIST dataset. As it can be seen, there are a lot of translation, scale and rotational variations. After the training procedure, the templates learned by the autoencoder are illustrated in row 2.

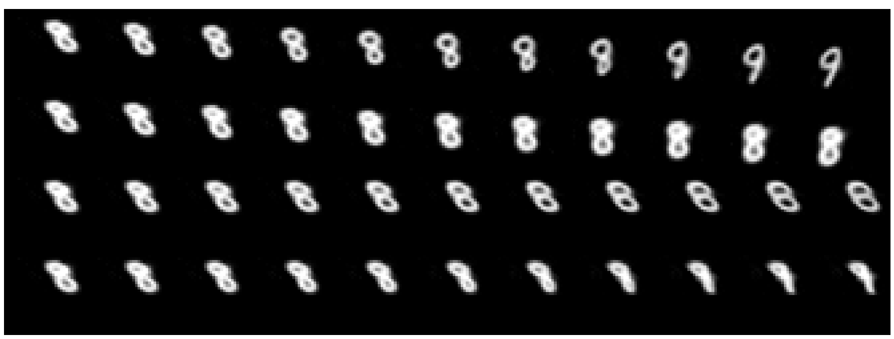

Figure 4.7: Latent Chunk Traversals for Mixing Autoencoders - Affnist dataset - The first row represents the overall interpolation between the 2 digits. The second row represents interpolation only along the affine chunk of the latent-space. The third and fourth chunks represent interpolations along chunks 2 and 3 respectively.

An interesting phenomenon we observed is the latent space clustering with and without using our training strategy. When a dimensionality reduction technique such as t-SNE (Maaten and Hinton (2008)) or UMap (McInnes *et al.* (2018)) is applied to the latent space to visualize it in two dimensions, and when the embeddings are colored with respect to classes, we notice that using our training strategy promotes better clustering. This is suggestive of the fact that affine invariant latent spaces are helpful for basic vision tasks like image classification.

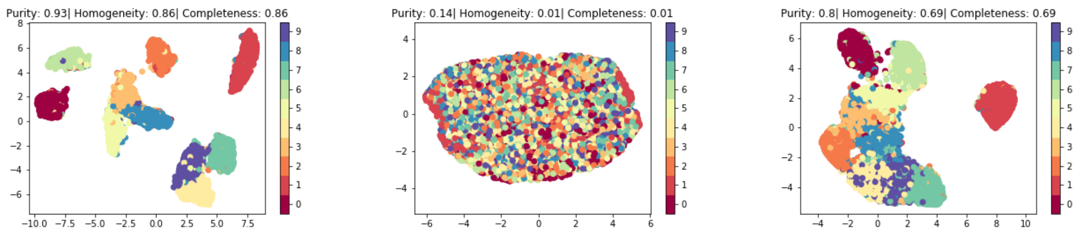

Figure 4.8: Visualization of Latent Representations for Mixing Autoencoders - Affnist Dataset - The figure represents the UMap plots for the MAE framework. The left picture represents the latent space embedding of the MAE trained on regular MNIST data. The middle picture represents latent space embedding of MAE trained on AffNIST data. The right picture represents latent space embedding of MAE along with our training strategy trained on AffNIST data.

Mixing Autoencoders	Purity	Homogeneity	Completeness
Clean MNIST	0.93	0.86	0.86
AffNIST	0.14	0.01	0.01
AffNIST + our strategy	**0.8**	**0.69**	**0.69**

Table 4.1: Clustering metrics of the latent space for mixing autoencoder framework - This table represents the clustering metrics for the MAE framework. As it can be noticed, by using our training strategy the class-discriminative ability of the latent space improves.

Figure 4.8 represents the latent space embeddings for the MAE architecture. When the normal/clean MNIST (LeCun *et al.* (1998)) is trained using MAE, the latent space is not only disentangled into chunks, but also well clustered according to classes i.e. the latent space is class discriminative. However, when nuisance transforms like affine transforms are introduced into the data, we notice that the latent space, although disentangled is not class-discriminative anymore. These embeddings cannot be used even for simple vision tasks such as classification. By introducing our strategy, we notice that the class-discriminative ability of these embeddings are regained. Thus, a by-product of our training mechanism is improved class-discriminatve ability of the latent space. This is also reflected in the cluster purity metrics in Table 4.1

DSprites Results

Similar experiments were conducted for the DSprites dataset with the MAE architecture. We notice that all the predicted templates have the same scale and position. However, the rotational invariance is something that the architecture struggles to capture. As before, Figures 4.9 and 4.10 represent the templates and latent space traversals for the DSprites dataset.

Figure 4.9: Estimated template by mixing autoencoders - DSprites dataset - The first row indicates some examples of the number 5 from the DSprites dataset. There are a lot of translation, scale and rotational variations. After the training procedure, the templates learned by the autoencoder ($\tilde{x_3}$ in Figure 4.4) are illustrated in row 2.

As we can notice from Figure 4.9, the output of decoder is invariant to translations and scales thus conforming to the notion of a template. Figure 4.10, illustrates the latent space traversals of the MAE framework. When we vary the affine chunk, we notice that the only change in the image is that of affine transform. When the chunks in the content space is varied, we can notice that only the "identity" of the image changes (in this case, its from heart to circle). This is because it is the only factor of variation that hasn't been captured yet.

Figure 4.11 illustrates the latent space embeddings of the Mixing Autoencoder framework with and without using our training strategy. Even though disentangled, without using

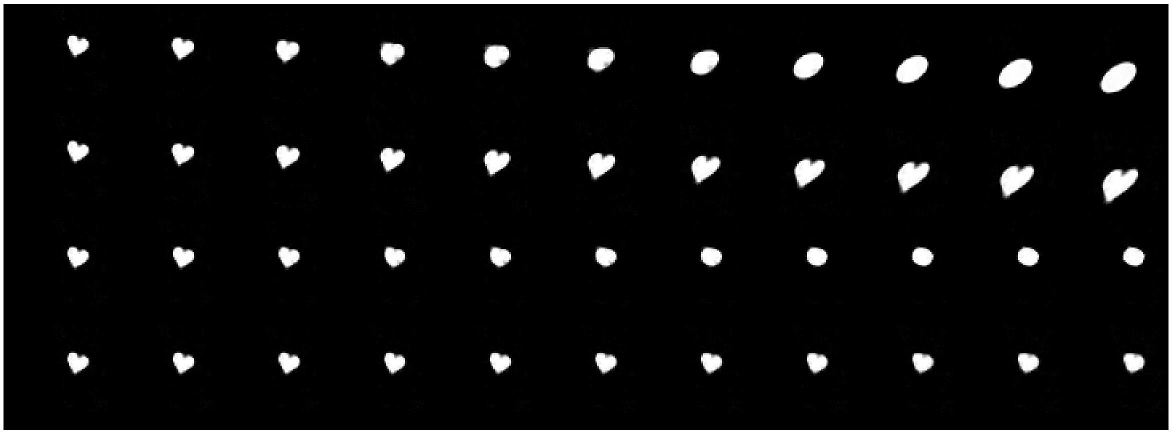

Figure 4.10: Latent chunk traversals for mixing autoencoders - DSprites Dataset - The first row represents the overall interpolation between the 2 shapes, square and circle. The second row represents interpolation only along the affine chunk of the latent-space. The third and fourth rows represent interpolations along chunks 2 and 3 respectively.

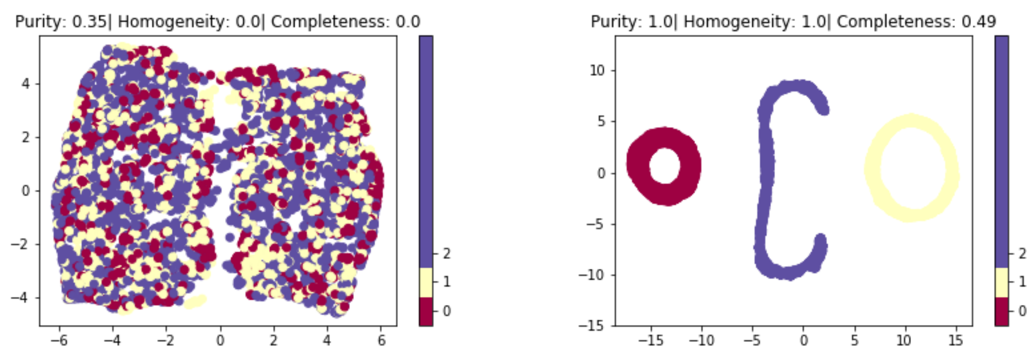

Figure 4.11: Visualization of latent representations for mixing autoencoders - DSprites dataset - The figure on the left represents the latent space embedding when the vanilla MAE framework is trained on the DSprites dataset. The figure on the right represents the latent space embedding when MAE framework along with our approach is trained on Dsprites dataset.

our strategy, the latent space is not class-discriminative as it can be seen from figure on the left. However, on using our strategy, we see that the latent embedding separates out into distinct clusters.

4.1.4 β - Variational Autoencoder

The variational autoencoder was originally introduced in (Kingma and Welling (2013)). It is basically a neural network parametrization of latent variable models. A very simple modification to this idea was to scale up the Lagrangian multiplier to the KL divergence term of the cost function, which was proposed in β-VAE (Higgins *et al.* (2017)). Figure 4.12 represents the architecture of a VAE. The loss function used to train a VAE architecture

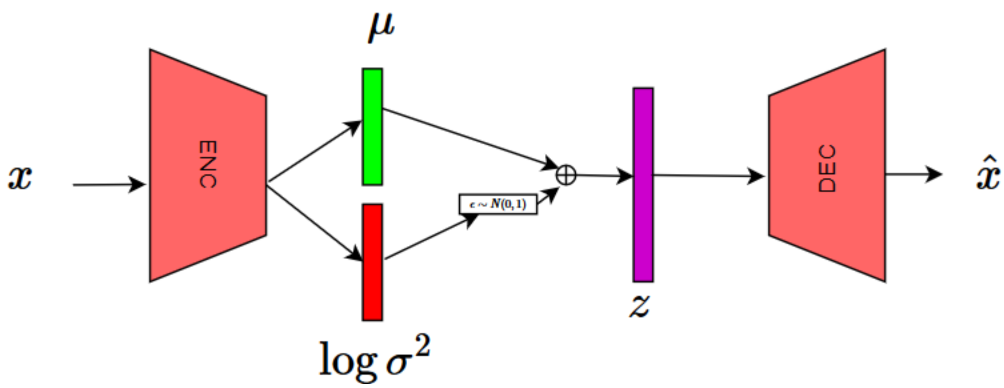

Figure 4.12: β-VAE architecture - This figure represents the architecture of a Variational Autoencoder. The encoder is used to predict the mean and variance parameters of a Gaussian distribution. The re-parametrization trick is used to sample from this distribution which in turn is fed to the decoder.

is given by:

$$L_{total}(\theta, \phi) = E_{z \sim q_\theta(z|x_i)} [\log p_\phi(x_i|z)] - \mathcal{KL}(q_\theta(z|x_i)||p(z)) \tag{4.3}$$

where, $q_\theta(.)$ is the encoder distribution, $p_\phi(.)$ is the decoder distribution and $\mathcal{KL}(.)$ is the KL divergence calculated between the encoder distribution and a Gaussian prior. By assuming that the decoder distribution is also Gaussian, the first term in the loss function, reduces to the $\mathcal{L}_2$ reconstruction error between input and decoder's output. By assuming that the prior on the latent space is normal Gaussian, the KL divergence term can be calculated using a closed form expression. Finally, the loss function used to train the VAE is given by:

$$L_{total} = \sum_{i=1}^{n} ||x_i - \hat{x}_i||^2 - \left(\sigma_i^2 + \mu_i^2 - \log \sigma_i - 1\right) \tag{4.4}$$

To train the β-VAE, the modified loss function is going to be:

$$L_{total} = \sum_{i=1}^{n} ||x_i - \hat{x}_i||^2 - \beta \left(\sigma_i^2 + \mu_i^2 - \log \sigma_i - 1\right) \tag{4.5}$$

Modified β-VAE Architecture

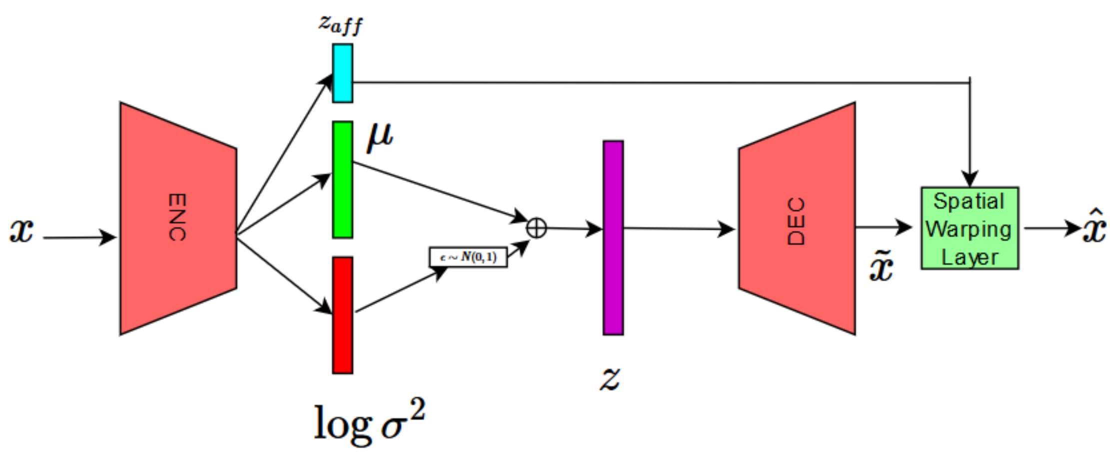

Figure 4.13: Modified β-VAE architecture - This figure represents the architectural modification to the β-VAE. The encoder is used to predict the mean, variance of a Gaussian distribution AND affine matrix parameters. The re-parametrization trick is used to sample from this distribution which in turn is fed to the decoder. However, as discussed previously, the affine chunk is not fed into the decoder.

We make the architectural modifications as shown in Figure 4.13 to incorporate our strategy into β-VAE. As it can be seen, the affine chunk predicted by the encoder is used to directly transform the output of the decoder $\tilde{x}$ to get $\hat{x}$. Let us now visualize some results with these architectural modifications.

AffNIST Results

Figure 4.14 illustrates the outputs of the decoder of a VAE. As it an be seen, the outputs are invariant to scale, rotation and translation in the inputs, thus conforming to the notion of a template. Figure 4.15 illustrates the latent space traversals in the β-VAE framework. As we expect, interpolation along affine chunk leads to only affine transform variations in

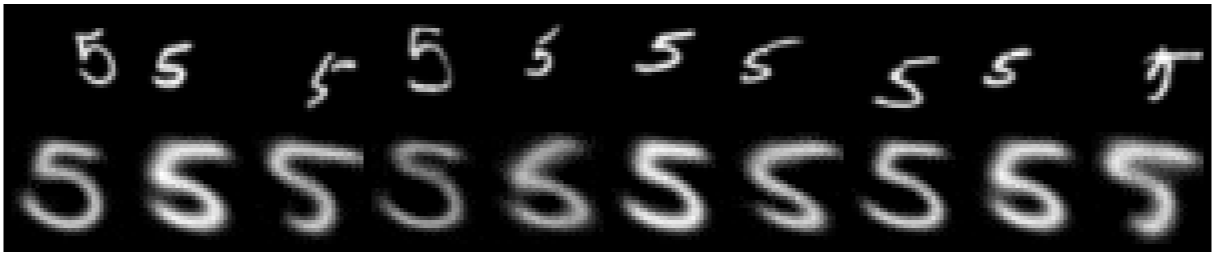

Figure 4.14: Estimated template by mixing autoencoders - AffNIST dataset - The first row indicates some examples of the number 5 from the AffNIST dataset. There are a lot of translation, scale and rotational variations. After the training procedure, the templates learned by the autoencoder (output of decoder in Figure 4.4) are illustrated in row 2.

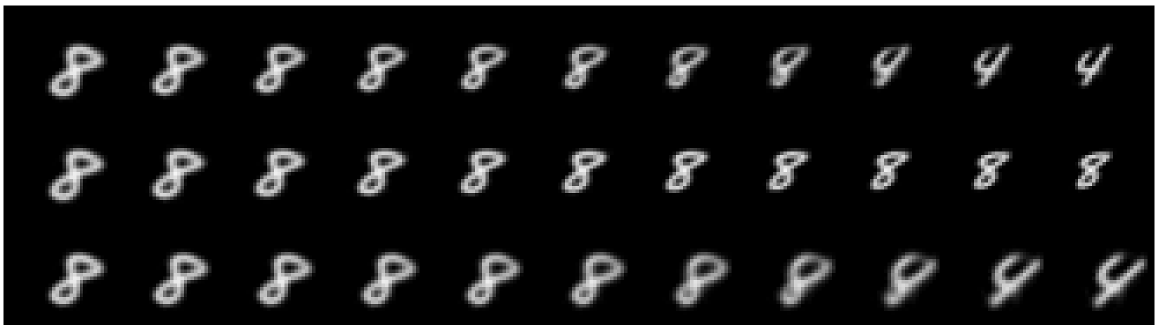

Figure 4.15: Latent chunk traversals for β-VAE - AffNIST dataset - The first row represents the overall interpolation between the 2 numbers (8 and 4). The second row represents interpolation only along the affine chunk of the latent-space. The third row represents interpolation along the non-affine part of the latent space.

the image and interpolating along the non-affine/content chunk leads to only variation in the identity of the digit.

Just as in the case of MAE, an interesting phenomenon we observed is the latent space clustering with and without using our training strategy. When a dimensionality reduction technique such as t-SNE or UMap is applied on the latent space to visualize in two dimensions, and when the points are colored with respect to classes, we notice a huge difference in the way clusters are formed. Figure 4.16 represents t-SNE embeddings of the affine invariant latent space. Our training strategy implicitly promotes better clustering thus indicating improved class-discriminative ability of the latent space. This is also reflected in the cluster purity metrics shown is Table 4.2

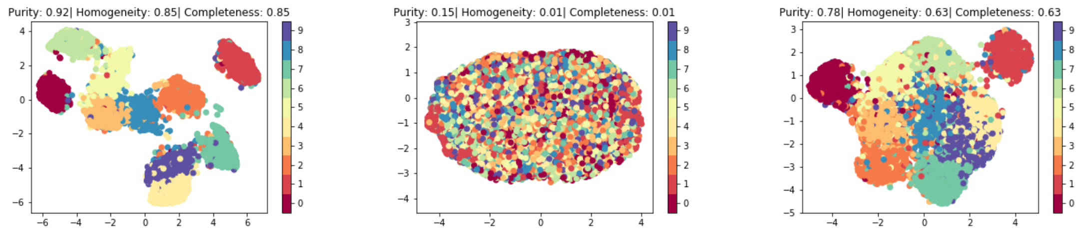

Figure 4.16: Visualization of latent representations for β-VAE - AffNIST dataset - The first row represents the UMap plots for the β-VAE framework. The left picture represents the latent space embedding of the MAE trained on regular MNIST data. The middle picture represents latent space embedding of β-VAE trained on AffNIST data. The right picture represents latent space embedding of β-VAE along with our training strategy trained on AffNIST data.

β-VAE framework	Purity	Homogeneity	Completeness
Clean MNIST	0.92	0.85	0.85
Affine MNIST	0.15	0.01	0.01
Affine MNIST + proposed strategy	**0.78**	**0.63**	**0.63**

Table 4.2: Clustering metrics of the latent space for variational autoencoder framework - This table represents the clustering metrics for the VAE framework. As it can be noticed, by using our training strategy the class-discriminative ability of the latent space improves.

DSprites Results

Figure 4.17: Estimated template for β-VAE - DSprites dataset - The first row represents some examples from the dataset. The second row shows the corresponding affine-invariant templates predicted by the decoder.

Figure 4.17 illustrates the templates predicted by the β-VAE framework for DSprites dataset. As before, these templates are scale and translation invariant. In Figure 4.18, the first row represents the overall interpolation between two shapes. The second row illustrates

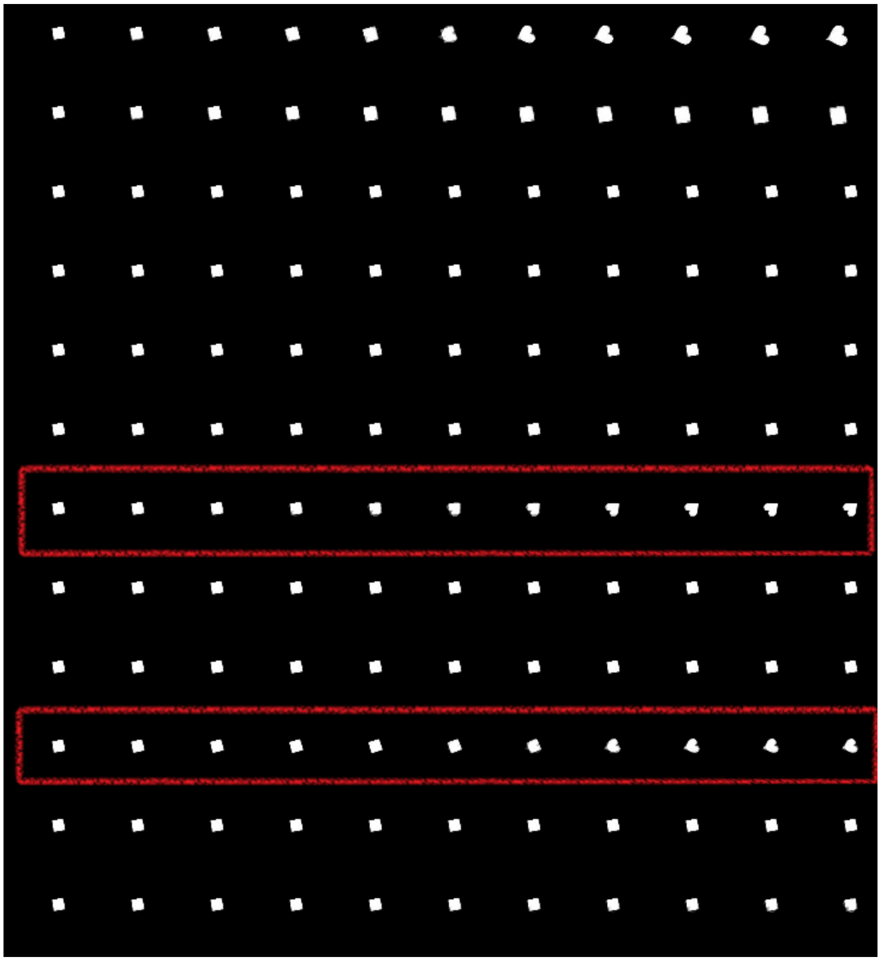

Figure 4.18: Latent traversals for β-VAE - DSprites dataset - The first row represents the overall interpolation between the 2 shapes, square and heart. The second row represents interpolation only along the affine chunk of the latent-space. The remaining rows represent the interpolation along each latent variable. The highlighted rows capture the identity information.

interpolation only along affine chunk. The two rows highlighted in red show that they are the only ones capturing identity/class information. The rest of the latent variables do not capture anything (because the affine chunk already captures affine variations, and the only other factor variation remaining to be captured is the class information).

4.2 Time Series Data

We will first look at the idea of rate variations followed by how to mathematically model them. Then we will see some experiments on datasets with increasing levels of complexity to show the efficacy of our approach.

Rate Variations

Consider the action of “wearing a jacket” or “picking up a bootle”. These are common actions that are performed in our day to day lives. However, these actions can be performed with different speeds/rates. In Figure 4.19, we illustrate the idea of rate variations. Top left and bottom left figures show the action of “wearing a jacket” performed at two different rates. The different colors show rate variations happening within the actions. Top right and bottom right graphs show the x,y,z rate variations summed over all joints. It is interesting to see how the same action with different rate variations have very different time series. If a classifier is trained on a signal with a particular rate, it will perform poorly on a signal with a different rate. Thus, it is important to achieve a rate-invariant representation for these signals.

Modeling rate variations

Let $\alpha_1(t)$ represent the original time series (signal) and $\alpha_2(t)$ be the rate distorted version of the signal. We will model these rate variations using a parameter γ. As we will see next, we assume that the rate variations belong to a specific class of functions Γ which obey some properties. Let γ be a 1-differentiable function defined on the domain [0,T]. Then, $\gamma \in \Gamma$ should obey the following properties:

1. $\gamma(0) = 0$

2. $\gamma(T) = 1$

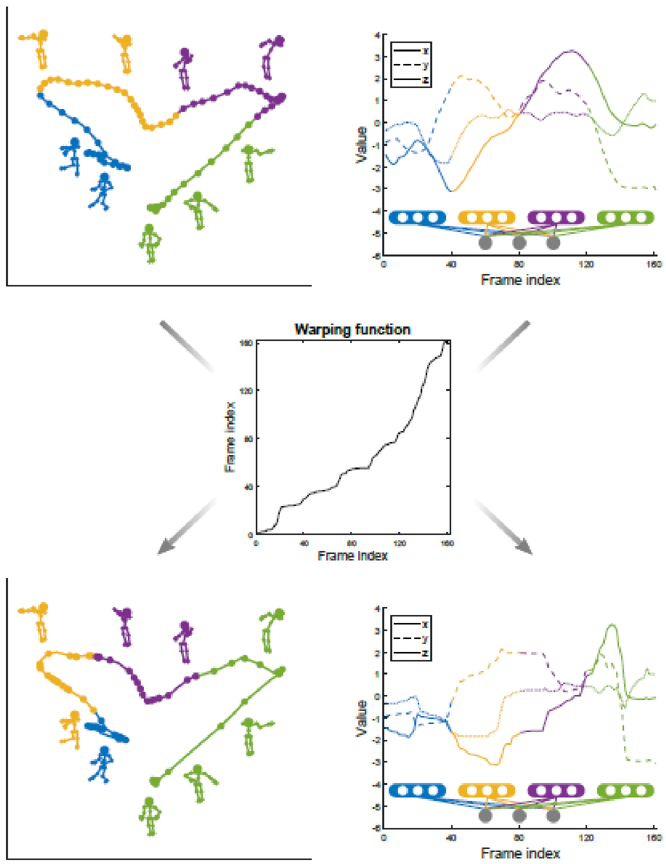

Figure 4.19: Illustration of rate variations in time series signals - Top left and bottom left figures represent the action of "wearing a jacket" performed at two different rates. Top right and bottom right graphs illustrate the x,y,z time series summed over all joints of the person. Figure taken from Lohit *et al.* (2019)

3. $\gamma(t_1) < \gamma(t_2)$ for $t_1 < t_2$

The last property indicates that γ is a monotonically increasing function *and* is order preserving. This is especially important, because order has to be preserved in time series data such as action sequences. Let $\dot{\gamma}$ be the first differential of γ. Along with the monotonicity property, we can see that: $\gamma(t) = \int_0^t \dot{\gamma}(t)dt$. Now, $\gamma(T) = \int_0^T \dot{\gamma}(t)dt = \gamma(T) - \gamma(0) = 1 - 0 = 1$. Thus the area under the $\dot{\gamma}$ curve is 1 which suggests that it is like a probability density function (pdf). This indicates that γ is like a cumulative distribution function (cdf).

Time series signals are typically captured using devices like Microsoft Kinect or MoCap systems. Thus, the data is discrete in nature. The same properties described above can be extended to discrete time signals as:

$$\gamma(t) = \sum_{i=0}^{t} \dot{\gamma}(i) \text{ and } \gamma(T) = \frac{1}{T}\sum_{i=0}^{T} \dot{\gamma}(i) = 1$$

Equipped with this modeling of rate variation parameter γ, we will now use the framework described in Figure 3.2 to disentangle γ from the input signal. Similar to image data, we employ a completely unsupervised training approach by using an autoencoder framework. It shows a general autoencoder framework that can be used to disentangle rate variations from a time series signals. The input to the architecture is some rate-distorted signal and the output of the decoder is a rate-invariant/aligned signal. Similar to the spatial warping layer, here we make use of the temporal warping layer proposed in Lohit *et al.* (2019). It is also important to note that the output of the encoder is actually interpreted as $\dot{\gamma}$ which is then used to *construct* γ using the properties described above. We will now look at experiments on three datasets.

4.2.1 Synthetic Dataset 1 - Bimodal Gaussian

The first dataset we use to show rate disentanglement is the bimodal Gaussian data. We generate the data using the equation:

$$X(t) = \alpha * \mathcal{N}(0.25, 0.15)(t) + \beta * \mathcal{N}(0.75, 0.15)(t) \tag{4.6}$$

where,

$$\alpha = 0.35 + 0.3 * \mathcal{U}(0,1)$$

$$\beta = 0.35 + 0.3 * \mathcal{U}(0,1)$$

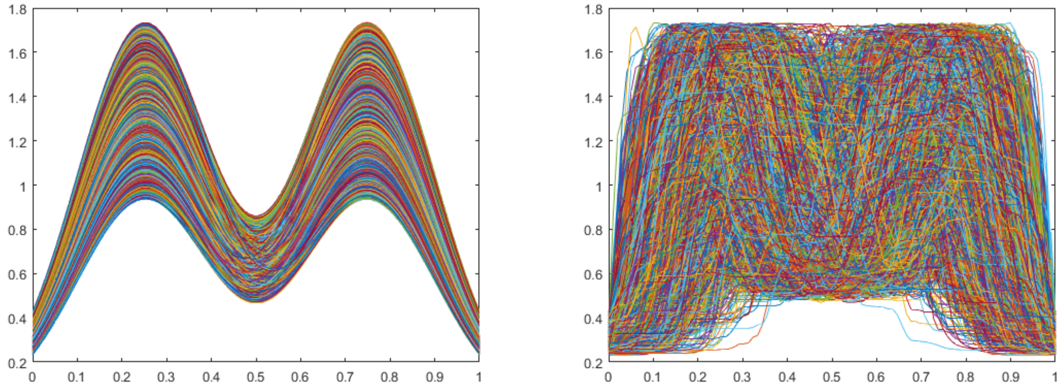

Figure 4.20: Visualization of bimodal gaussian data - The figure on the left illustrates the "clean" bimodal Gaussian data generated using the equation 4.6. The figure on the right illustrates the same bimodal Gaussian signal with rate distortions.

$\mathcal{N}(.)$ representing a Gaussian distribution and $\mathcal{U}(.)$ representing a uniform distribution. The dataset is illustrated in Figure 4.20.

Figure 4.20 (left) represents the "clean" bimodal Gaussian data. Each sequence is of 100 time steps. Figure 4.20 (right), represents the warped version using some rate variation γ. This serves as the input to the autoencoder architecture in Figure 3.2. According to our training strategy, the output of the decoder should be a "rate-invariant" / template version of the input. Finally, the rate parameters γ learned by the encoder is used to directly transform the output of the decoder to obtain a reconstruction which has to be same as the input.

It is interesting to see that the rate-invariant signal predicted by the autoencoder (Figure 4.21) is very similar to Figure 4.20(left), the original rate invariant version of the input to the AE. On applying some post processing steps to Figure 4.21(left), we obtain Figure 4.21(right). Although the reconstructed signal is not exactly Gaussian, it captures the modalities of the Gaussian distribution and more importantly aligns the sequences by undoing the rate variations. The post processing steps are only done to obtain a "cleaned-up" version and is described in the following sub-section.

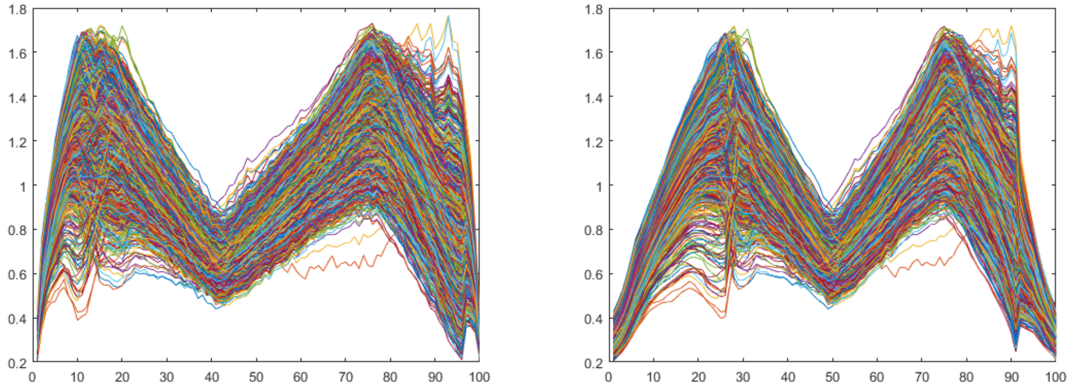

Figure 4.21: Visualization of rate invariant bimodal gaussian - The figure on the left illustrates the output of the decoder X_{temp}(also called the template). Figure on the right illustrates the template after some post-processing steps.

Post Processing Step

Referring to Figure 3.2,

$$\hat{x} = \tilde{x} \circ \gamma$$

Let γ_μ represent the arithmetic mean of all the γ's predicted for the dataset. Now,

$$\gamma_\mu \circ \gamma_\mu^{-1} = \gamma_I$$

where γ_I is the identity rate variation. Hence, we can see that

$$\hat{x} = \tilde{x} \circ \gamma_\mu \circ \gamma_\mu^{-1} \circ \gamma$$

The post-processed output that we see is the output of $\tilde{x} \circ \gamma_\mu$. We have chosen γ_μ so that the mean of the γ when the post-processed output is fed into the autoencoder, will be identity.

Figure 4.22: Two classes of time series signals considered - These figures represent the two classes of time series signals considered to experiment the unsupervised rate-disentangling capability using the proposed framework.

Figure 4.23: Warped version of two class time series signals - These figures represent the warped version of Figure 4.22. They have been warped using some rate-variation parameters.

4.2.2 Synthetic Dataset 2 - Multi Class Time Series

The next dataset we consider, consists of two classes of waveforms as illustrated in Figure 4.22. These are the original “clean” waveforms. On applying rate variations, we obtain Figure 4.23 which inturn are fed to the autoencoder architecture in Figure 3.2.

The equations that govern the waveforms generated for the two classes are as follows:

$$\begin{aligned} y_1(t) &= \alpha * \mathcal{N}(0.5, 0.15) \\ \alpha &= 0.35 + 0.3 * \mathcal{U}(0, 1) \end{aligned} \tag{4.7}$$

$$
y_2(t) = \begin{cases} \beta * t/10 & 0 \leq t \leq 10 \\ \beta - \frac{(t-10)*\beta}{50-10} & 11 \leq t \leq 50 \\ \beta - \frac{((40-t)-50)*\beta}{90-50} & 51 \leq t \leq 90 \\ \beta * \frac{(10-t)}{10} & 91 \leq t \leq 100 \\ 0 & \text{else} \end{cases} \tag{4.8}
$$

$$
\beta = 0.35 + 0.3 * \mathcal{U}(0,1)
$$

When the autoencoder is trained with the 2 classes of waveforms, the template predicted as a result is as illustrated in Figure 4.24.

Figure 4.24: Autoencoder template prediction - These figures represent the templates predicted for the two class of signals by the autoencoder.

Figure 4.25: Autoencoder template post processed - These figures represent the templates predicted for the two class of signals by the autoencoder after post-processing.

Finally, applying the post-processing steps discussed in the previous sub-section, we get more "clean" looking waveforms as shown in Figure 4.25. It is very interesting to note that, the autoencoder is able to learn a template representation per class and doesn't collapse all classes to the same template.

In the next section, we present a set of experiments where the goal is to perform hand-actions classification using MoCap data corresponding to the joints of the hand. This is a time-series dataset (Garcia-Hernando *et al.* (2018)) and we show that using our proposed approach, we can obtain a rate-invariant representation and in turn achieve better classification accuracy.

4.2.3 ICL Hand Pose Dataset

Here, we demonstrate experiments on a real world dataset of hand actions. The dataset contains 3D hand pose sequences with 21 joint locations per frame of 45 daily hand action categories interacting with 26 objects, such as "pour juice", "put tea bag" and "read paper". These action sequences are performed by six subjects, each of them performing each action for about four to five times. The actions are recorded using an accurate motion capture system. There are 600 training sequences and 575 test sequences. As the sequences are of varying lengths, we uniformly sample the sequences such that all sequences contain 50 samples. If the sequences are shorter than 50 samples, we use zero padding. As there are 21 joints per frame, each input sequence is of dimension 50x63 (21x3 = 63).

Real world data is often not very clean and contains a lot of nuisance factors. In order to illustrate the effectiveness of this training strategy, we artificially introduce rate variations into this dataset. We set the sequence length to 100 such that the original sequence is placed between 25 and 75 and the remaining time steps are set to zero. Now, random "affine warps" are introduced into the data which has the form $\gamma(t) = at + b$, where t $\in [25, 75]$, a $\in [0.75, 1.25]$ and b $\in \{0, 1, 2, ...49\}$.

Architectural details

The TCN based classifier is composed of 1 Conv1D layer and 1 FC layer of size 45 (num. of classes). It uses convolution filters of size 8 and feature maps of depth 32. Training is done using momentum optimizer with learning rate of 1e-3 and momentum of 0.9. The TCN-AE is composed of the encoder and decoder. Encoder is made up of 3 Conv1D and 1 FC layer. Convolutional layers have filter size 16 and 63 feature maps are used at every layer. The FC layer (latent space) has 7 dimensions for TCN-AE + proposed approach and 10 dimensions for the vanilla TCN-AE. Similarly, the decoder is also composed of 3 Conv1D layers with the same number of filter and feature map size. A FC network/multi-layer perceptron is used to train the features obtained from latent space of TCN-AE. The FC network is composed of 2 hidden layers with 80 dimensions. Experiments on TCN-AE + proposed approach were conducted on both features learned as well as templates learned and we observe similar performance on both. The reported numbers are obtained by training the templates using the TCN classifier described above.

Description of Experiments

We conduct experiments in three settings to show the efficacy of the proposed approach. The first setting is to train the affine warped version of the ICL dataset using a TCN-based classifier. Note that this is a supervised experiment. The second setting is to train a TCN based autoencoder and evaluate the learned features (latent representations) for class-discriminative ability. The third setting is to train a TCN based autoencoder along with proposed method and evaluate the rate-invariant latent space. Note that the second and third settings are completely unsupervised.

We then plot the t-SNE embeddings of these feature/latent representations and also calculate the classification accuracies. Figure 4.26 show the t-SNE plots for these settings described above in the same order and Tables 4.3 and 4.4 show the classification accuracies and clustering metrics for the corresponding experiments. Interestingly we observe that training in an unsupervised fashion using our approach does perform marginally better than directly training the affine-warped data with a classifier. We also notice that our proposed method is a huge improvement over a vanilla TCN-based autoencoder whose latent space prove to be *not* class-discriminative. However, this marginal improvement over supervised setting comes along with a huge increase in number of parameters. Depending on the application, computational and storage resources available, performance requirements and label constraints, a suitable strategy can be chosen.

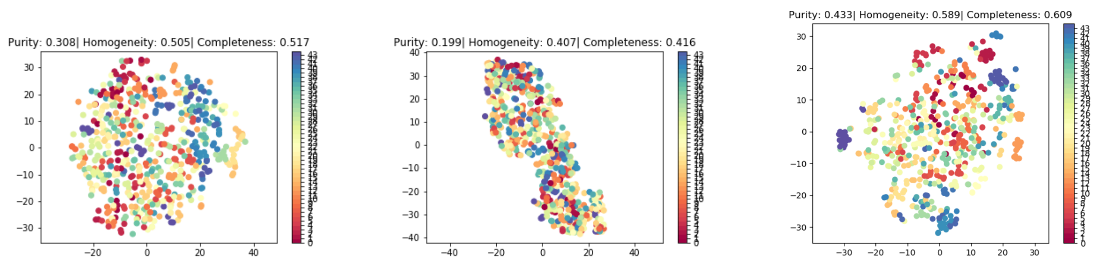

Figure 4.26: t-SNE plots of feature representation for ICL Dataset - The first figure represents the t-SNE plot of penultimate layer of a TCN-based classifier. The second and third figures represent the t-SNE plots of latent space of a TCN based autoencoder without and with our training strategy. We can clearly see that the proposed strategy improves the class-discriminative ability of the latent space while making it rate-invariant.

Affine warped ICL data	Purity	Homogeneity	Completeness
TCN - Classifier	0.308	0.505	0.517
TCN - AE	0.199	0.407	0.416
TCN - AE + proposed method	**0.433**	**0.589**	**0.609**

Table 4.3: Cluster purity metrics of the feature representations of the three methods - The table represents three cluster purity metrics: purity, homogeneity and completeness for the experiments described in this section. It can be seen that the proposed method outperforms the other two.

Method	Accuracy	#params
TCN - Classifier	72.86%	160k
TCN - AE	43.66%	1.1 million
TCN - AE + proposed method	**76.6%**	**1.25 million**

Table 4.4: Classification accuracies of the three methods - The table represents classification accuracies for the experiments described in this section. It can be seen that the proposed method outperforms the other two.

Figure 4.27 illustrates example data from the ICL dataset for action no. 1 (corresponding to “charge cell phone”) and joint no. 4. Figure on the left illustrates the original signal. The figure in the center illustrates the signal after introducing affine warps. The figure on the right illustrates the template output by the autoencoding architecture. This corresponds to $\tilde{x}$ in Figure 3.2. The original signal is aligned between time-steps 25 to 75. However, after introducing affine warps, the signal is spread out between 0 to 100 time steps. From the template, we can see that it not only centers the data (approximately), but also tries to find a canonical/template representation and aligns them.

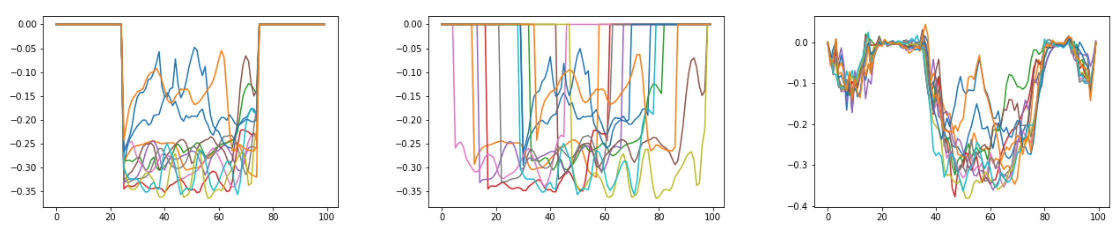

Figure 4.27: Example data from ICL dataset before and after warping - The first figure represents the original signal from the dataset. The middle figure represents the signal after introducing affine warps into it. The rightmost figure represents the template predicted by the autoencoding framework.

Chapter 5

CONCLUSION

In this thesis, we have presented a training strategy to disentangle deformation parameters from the semantic content of the signal. For image data, we show experiments on the AffNIST and DSprites datasets where affine parameters were successfully disentangled along with other factors like style and identity. We have also shown its modularity by integrating it with other disentangling strategies like MAE and VAE. As a by product of this strategy, we were able to observe improved class discriminative ability of the latent space. For time series data, we showed that the proposed strategy can be used to align sequences that have been distorted by rate variations. We create and use synthetic datasets to show the efficacy of the proposed approach. Finally, we present experiments on a real-world dataset of hand action recognition and clearly show that the features learned are more class discriminative.

We provide a general framework to estimate the parameters of a noise model which typically acts on the domain of the signal. We show that in the case of geometric factors of variation like affine and rate transforms, it is possible to disentangle these factors in an explicit manner. Furthermore, these disentangled nuisance factors are interpretable i.e. can be used to transform our signal to the desired form. As mentioned previously, interpretable disentangled representations have significance in many application like egomotion which is used to guide robots in an active perception scenario.

This work could be extended in multiple directions. The strategy could be extended to perform warping from one signal to another. This could potentially be the data-driven version of dynamic time warping. With the help of latent space embeddings, we showed that the features learned are in-fact class discriminative. We hope to connect the affine invariant properties of the proposed approach back to classical affine invariant shape analysis, and study how the classical affine invariant metrics can be realized within deep neural networks. Since this is a general framework to disentangle interpretable factors, it can be extended to perform disentanglement of other factors of variations such as illumination, blur, etc.

REFERENCES

Agrawal, P., J. Carreira and J. Malik, “Learning to see by moving”, in “Proceedings of the IEEE International Conference on Computer Vision”, pp. 37–45 (2015).

Bai, S., J. Z. Kolter and V. Koltun, “An empirical evaluation of generic convolutional and recurrent networks for sequence modeling”, arXiv preprint arXiv:1803.01271 (2018).

Bay, H., T. Tuytelaars and L. Van Gool, “Surf: Speeded up robust features”, in “European conference on computer vision”, pp. 404–417 (Springer, 2006).

Begelfor, E. and M. Werman, “Affine invariance revisited”, in “2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’06)”, vol. 2, pp. 2087–2094 (IEEE, 2006).

Chen, X., Y. Duan, R. Houthooft, J. Schulman, I. Sutskever and P. Abbeel, “Infogan: Interpretable representation learning by information maximizing generative adversarial nets”, in “Advances in neural information processing systems”, pp. 2172–2180 (2016).

Cohen, T. and M. Welling, “Group equivariant convolutional networks”, in “International conference on machine learning”, pp. 2990–2999 (2016).

Garcia-Hernando, G., S. Yuan, S. Baek and T.-K. Kim, “First-person hand action benchmark with rgb-d videos and 3d hand pose annotations”, in “Proceedings of Computer Vision and Pattern Recognition (CVPR)”, (2018).

Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, “Generative adversarial nets”, in “Advances in neural information processing systems”, pp. 2672–2680 (2014).

Gopalan, R., S. Taheri, P. Turaga and R. Chellappa, “A blur-robust descriptor with applications to face recognition”, IEEE transactions on pattern analysis and machine intelligence **34**, 6, 1220–1226 (2012).

Higgins, I., L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed and A. Lerchner, “beta-vae: Learning basic visual concepts with a constrained variational framework.”, ICLR **2**, 5, 6 (2017).

Hinton, G. E. and R. R. Salakhutdinov, “Reducing the dimensionality of data with neural networks”, science **313**, 5786, 504–507 (2006).

Hochreiter, S. and J. Schmidhuber, “Long short-term memory”, Neural computation **9**, 8, 1735–1780 (1997).

Hu, Q., A. Szab, T. Portenier, P. Favaro and M. Zwicker, “Disentangling factors of variation by mixing them”, in “The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)”, (2018).

Jaderberg, M., K. Simonyan, A. Zisserman *et al.*, “Spatial transformer networks”, in “Advances in neural information processing systems”, pp. 2017–2025 (2015).

Kim, H. and A. Mnih, "Disentangling by factorising", arXiv preprint arXiv:1802.05983 (2018).

Kingma, D. P. and M. Welling, "Auto-encoding variational bayes", arXiv preprint arXiv:1312.6114 (2013).

Kulkarni, T. D., W. F. Whitney, P. Kohli and J. Tenenbaum, "Deep convolutional inverse graphics network", in "Advances in neural information processing systems", pp. 2539–2547 (2015).

LeCun, Y., C. Cortes and C. J. Burges, "The mnist database of handwritten digits, 1998", URL http://yann. lecun. com/exdb/mnist **10**, 34 (1998).

Liu, B., X. Wang, M. Dixit, R. Kwitt and N. Vasconcelos, "Feature space transfer for data augmentation", in "Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition", pp. 9090–9098 (2018).

Lohit, S. and P. Turaga, "Learning invariant riemannian geometric representations using deep nets", in "The IEEE International Conference on Computer Vision (ICCV) Workshops", (2017).

Lohit, S., Q. Wang and P. Turaga, "Temporal transformer networks: Joint learning of invariant and discriminative time warping", in "Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition", pp. 12426–12435 (2019).

Lowe, D. G., "Distinctive image features from scale-invariant keypoints", International journal of computer vision **60**, 2, 91–110 (2004).

Lu, B., J.-C. Chen and R. Chellappa, "Unsupervised domain-specific deblurring via disentangled representations", in "Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition", pp. 10225–10234 (2019).

Maaten, L. v. d. and G. Hinton, "Visualizing data using t-sne", Journal of machine learning research **9**, Nov, 2579–2605 (2008).

Makhzani, A., J. Shlens, N. Jaitly, I. Goodfellow and B. Frey, "Adversarial autoencoders", arXiv preprint arXiv:1511.05644 (2015).

Mathieu, M. F., J. J. Zhao, J. Zhao, A. Ramesh, P. Sprechmann and Y. LeCun, "Disentangling factors of variation in deep representation using adversarial training", in "Advances in Neural Information Processing Systems", pp. 5040–5048 (2016).

Matthey, L., I. Higgins, D. Hassabis and A. Lerchner, "dsprites: Disentanglement testing sprites dataset", URL https://github. com/deepmind/dsprites-dataset/.[Accessed on: 2018-05-08] (2017).

McInnes, L., J. Healy and J. Melville, "Umap: Uniform manifold approximation and projection for dimension reduction", arXiv preprint arXiv:1802.03426 (2018).

Mumford, D., "Pattern theory: a unifying perspective", in "First European congress of mathematics", pp. 187–224 (Springer, 1994).

Qi, C. R., H. Su, K. Mo and L. J. Guibas, “Pointnet: Deep learning on point sets for 3d classification and segmentation”, in “The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)”, (2017).

Shu, Z., M. Sahasrabudhe, R. Alp Guler, D. Samaras, N. Paragios and I. Kokkinos, “Deforming autoencoders: Unsupervised disentangling of shape and appearance”, in “Proceedings of the European Conference on Computer Vision (ECCV)”, pp. 650–665 (2018).

Som, A., K. Thopalli, K. Natesan Ramamurthy, V. Venkataraman, A. Shukla and P. Turaga, “Perturbation robust representations of topological persistence diagrams”, in “Proceedings of the European Conference on Computer Vision (ECCV)”, pp. 617–635 (2018).

Szabó, A., Q. Hu, T. Portenier, M. Zwicker and P. Favaro, “Challenges in disentangling independent factors of variation”, arXiv preprint arXiv:1711.02245 (2017).

Tieleman, T., “affnist”, URL https://www. cs. toronto. edu/˜ tijmen/affNIST/, Dataset URL https://www. cs. toronto. edu/˜ tijmen/affNIST/.[Accessed on: 2018-05-08] (2013).

Vedaldi, A., *Invariant representations and learning for computer vision*, Ph.D. thesis, Citeseer (2008).

Worrall, D. E., S. J. Garbin, D. Turmukhambetov and G. J. Brostow, “Interpretable transformations with encoder-decoder networks”, in “Proceedings of the IEEE International Conference on Computer Vision”, pp. 5726–5735 (2017).

Xing, X., R. Gao, T. Han, S.-C. Zhu and Y. N. Wu, “Deformable generator network: Unsupervised disentanglement of appearance and geometry”, arXiv preprint arXiv:1806.06298 (2018).