Learnable factored image representation for visual discovery

Théophile Dalens

PSL UNIVERSITÉ PARIS

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL

Préparée à l'École normale supérieure

Learnable factored image representations for visual discovery

Soutenue par
Théophile DALENS
Le 16 septembre 2019

Ecole doctorale n° 386
SCIENCES MATHÉMATIQUES DE PARIS-CENTRE

Spécialité
MATHÉMATIQUES

Composition du jury :

Valérie GOUET-BRUNET IGN	*Présidente*
Alexei EFROS University of California Berkeley	*Rapporteur*
Patrick PÉREZ Valéo	*Rapporteur*
Josef SIVIC INRIA	*Directeur de thèse*
Mathieu AUBRY ENPC	*Directeur de thèse*

ENS | PSL

Contents

Résumé

L'analyse temporelle quantitative à grande échelle de données textuelles a eu un grand impact sur la compréhension des tendances sociales. Une analyse similaire à grande échelle des collections d'images temporelles permettrait de nouvelles applications en médecine, en science ou en histoire de l'art. Cependant, l'analyse temporelle des données visuelles est une tâche notoirement difficile. Le principal défi est que les images n'ont pas un vocabulaire donné d'éléments visuels, par analogie avec les mots du texte, qui pourraient être utilisés pour une telle analyse. De plus, les objets représentés dans les images varient considérablement en apparence en raison du point de vue de l'appareil photographique, de l'éclairage ou des variations intra-classe. L'objectif de cette thèse est de développer des outils pour analyser les collections d'images temporelles afin d'identifier et de mettre en évidence les tendances visuelles à travers le temps.

Cette thèse propose une approche pour l'analyse de données visuelles non appariées annotées avec le temps en générant à quoi auraient ressemblé les images si elles avaient été d'époques différentes. Pour isoler et transférer les variations d'apparence dépendantes du temps, nous introduisons un nouveau module bilinéaire de séparation de facteurs qui peut être entraîné. Nous analysons sa relation avec les représentations factorisées classiques et les auto-encodeurs basés sur la concaténation. Nous montrons que ce nouveau module présente des avantages par rapport à un module standard de concaténation lorsqu'il est utilisé dans une architecture de réseau de neurones convolutionnel encodeur-décodeur à goulot. Nous montrons également qu'il peut être inséré dans une architecture récente de traduction d'images à adversaire, permettant la transformation d'images à différentes périodes de temps cibles en utilisant un seul réseau.

Nous appliquons notre modèle à une collection de plus de 13 000 voitures fabriquées entre 1920 et 2000 et à un ensemble de portraits d'annuaires d'écoles secondaires entre 1930 et 2009. Cela nous permet, pour une nouvelle image d'entrée donnée, de générer une "vidéo historique en continu" révélant les changements dans le temps en variant simplement l'année cible. Nous montrons qu'en analysant ces vidéos générées, nous pouvons identifier

les déformations des objets dans le temps et en extraire des changements intéressants dans le style visuel au fil du temps.

Abstract

Large-scale quantitative temporal analysis of heritage text data has made a great impact in understanding social trends. Similar large-scale analysis of temporal image collections would enable new applications in medicine, science or history of art. However, temporal analysis of visual data is a notoriously difficult task. The key challenge is that images do not have a given vocabulary of visual elements, in analogy to words in text, that could be used for such analysis. In addition, objects depicted in images vary greatly in appearance due to camera viewpoint, illumination, or intra-class variation. The objective of this thesis is to develop tools to analyze temporal image collections in order to identify and highlight visual trends over time.

This thesis proposes an approach for analyzing unpaired visual data annotated with time stamps by generating how images would have looked like if they were from different times. To isolate and transfer time dependent appearance variations, we introduce a new trainable bilinear factor separation module. We analyze its relation to classical factored representations and concatenation-based auto-encoders. We demonstrate this new module has clear advantages compared to standard concatenation when used in a bottleneck encoder-decoder convolutional neural network architecture. We also show that it can be inserted in a recent adversarial image translation architecture, enabling the image transformation to multiple different target time periods using a single network.

We apply our model to a challenging collection of more than 13,000 cars manufactured between 1920 and 2000 and a dataset of high school yearbook portraits from 1930 to 2009. This allows us, for a given new input image, to generate a "history-lapse video" revealing changes over time by simply varying the target year. We show that by analyzing the generated history-lapse videos we can identify object deformations across time, extracting interesting changes in visual style over decades.

Acknowledgement

First, I'd like to thank my PhD advisors Josef and Mathieu, who encouraged me to push all the way from the beginning to the end. I have learned a lot during these years, thanks to you!

I want to thank Alyosha Efros and Patrick Pérez, for reviewing my thesis, and Valérie Gouet-Brunet and all the members of the jury.

I'd like to thank Alyosha again for welcoming me to Berkeley for a few weeks, along with all the people I've met there! Thanks to Pascal Monasse for giving me the opportunity to teach computer science.

Many thanks to everyone at INRIA for making this thesis more enjoyable. This includes: Ivan, for welcoming me with Josef a long time ago for a project on Google Glass, Francis and Jean, David and all the personnel at INRIA. Thanks to my friends and colleagues who shared my office and the neighbouring ones, who participated in the HashCode, who helped me manage the cluster, and everyone who made this journey much more enjoyable: Jean-Baptiste, Rémi, Robin, Gauthier, Loïc, Gül, Julia, Anton, Yana, Vincent, Aymeric, Guillaume, Antoine, Ignacio, Antoine, Zong, Relja, Minsu, Bumsub, Maxime, Vadim, Federico, Thomas, Dimitri, Tatiana, Nastya, Loucas, Guilhem, Tuan-Hung, Andrei, Pascal, Sesh, Matthew.

Je ne peux pas citer tous les amis qui m'ont apporté leur soutien, mais je pense à vous. Merci ! Je pense aussi à mes amis qui m'ont toujours conseillé de ne pas faire de thèse !

Merci à mes parents qui ont énormément fait pour que j'arrive jusqu'ici. Merci aussi à mes frères et leurs familles, ma belle-soeur et mes beaux-parents !

Enfin, merci à ma femme Lamiae pour avoir été à mes côtés pendant toute cette période. Cette thèse n'aurait pas été possible sans tout le soutien que tu m'as apporté.

Chapter 1

Introduction

1.1 Motivation and objectives

Data-driven analysis through time can have - and already has had - a significant impact on understanding of our world. Analysis of vocabulary used in books through centuries [Michel et al., 2011] shows that large collections of books can be used to perform quantitative analysis and shed light on the evolution of how people write about different topics. Speech analysis has been used to help understand the evolution of language acquisition [Roy et al., 2006]. Similarly, leveraging large-scale time stamped *visual data* can enable new applications. Databases of medical imagery have been used to assess disease progression in radiology [Marcus et al., 2010; Tang and et al., 2018]. Art museums have collections of sculptures, paintings, or objects associated with the date of their creation. Recent efforts are making such art collections publicly available [Dijkshoorn et al., 2018; The Watermark Project; The Visual Arts Data Service]. While art historians are needed to analyze these trends, there is also interest to use machine learning tools to help this analysis [Crowley and Zisserman, 2013; Gonthier et al., 2018; Shen et al., 2019]. Examples of trend analysis include identifying and highliting differences over time. Similarly, in architecture, large scale image collections can be used, with proper annotation, to analyze the characterstic elements of architecture design from place to place [Doersch et al., 2013] or through time [Lee et al., 2015].

While all these problems touch different domains, they share common characteristics. They concern large collections of images that are time stamped. The objective of this thesis is to develop tools to analyze these different types of time stamped image collections, in order to identify and highlight visual trends over time.

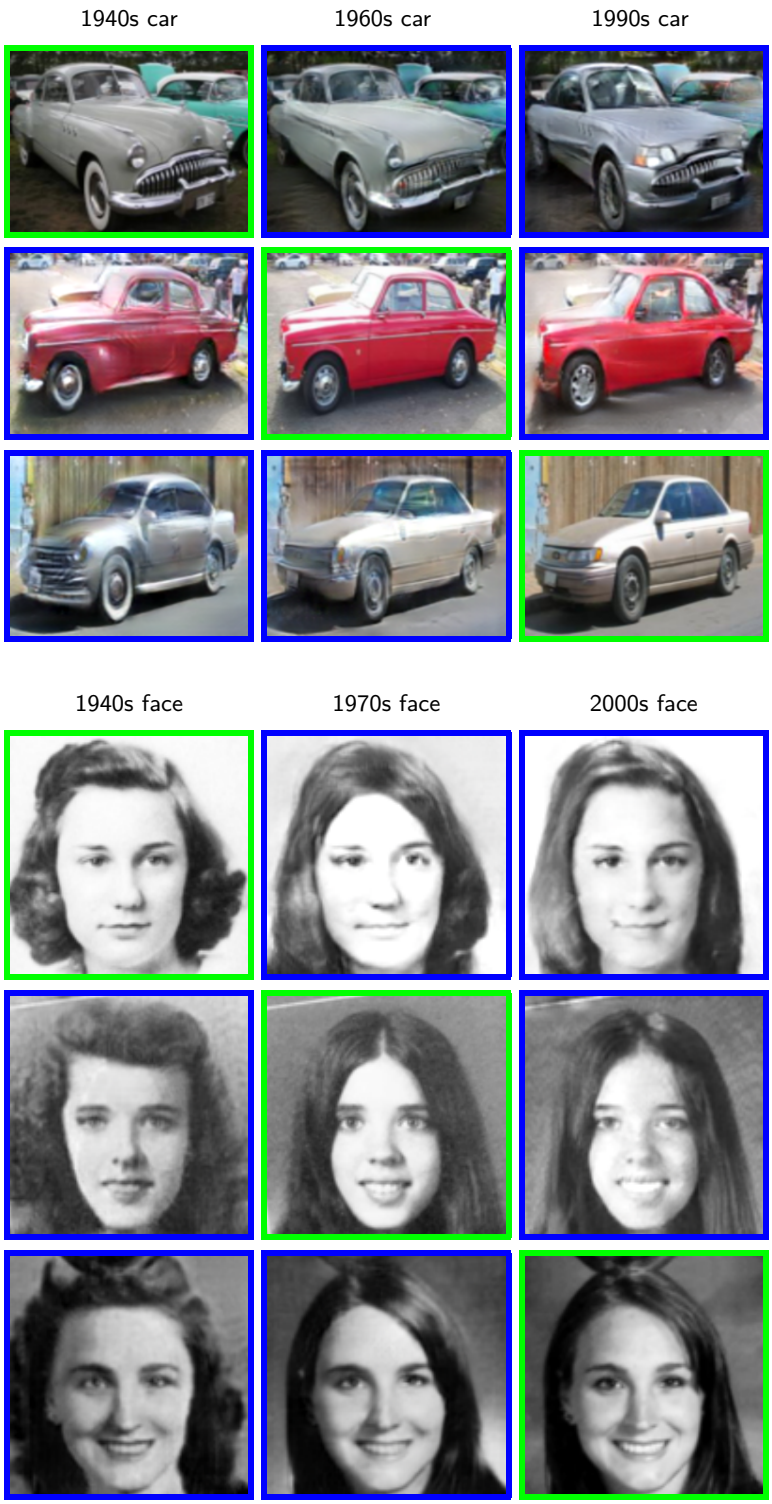


Figure 1.1: **Bilinear image translation.** Our goal is to take as input (in green) an image of a face or an object such as a car, and generate what it would have looked like in another time-period (in blue). Each row shows temporal translation for a different input portrait or car picture. We further show that analyzing changes between the generated images can reveal structural deformations in object shape and appearance over time.

1.2 Approach

In this thesis we propose to perform temporal analysis of visual data by temporal image translation. First, we learn from the input time-stamped image collection a temporal translation model that takes an image as input and translates it to a different time period, while preserving its time-independent characteristics, as illustrated in figure 1.1. Second, we perform temporal analysis by comparing the resulting generated imagery. We describe these two steps next.

Time translation. The time translation is an operation that takes an image and a target time period, and outputs a modified image that has the characteristics of this target time period, while keeping the visual characteristics of the input image that do not depend on time. We illustrate this in figure 1.1 with two examples. We "translate" cars and change their appearance to make them look like they have been built at another period. In these examples, wheels, headlights, and windshields change their appearance over time. The model has no prior notion of these elements, yet it learns what are the characteristic elements and how they evolve over time from the data. At the same time, the type of car, viewpoint, and general colour don't change, as these visual characteristics are not dependent on the time when the car was made. We also apply the same method to historical yearbook portraits of faces, making them appear as coming from another decade while preserving the face identity. In this thesis we investigate and evaluate different approaches to perform such temporal image translation.

Visual discovery. We aim to use image translation for *visual discovery*. Once an image is translated, time related variations for this specific image instance can be identified by simple pixel comparisons. This would be difficult otherwise and would require finding correspondence across different object instances and viewpoints. We can use the images we produce to compare the same car or face at different times (figure 1.2) and highlight the different trends over time. We can also identify typical and atypical images of different time periods (figure 1.3).

1.3 Challenges

We proposed to approach temporal analysis of visual data using temporal image translation, that is answering question "what would this picture be, if this car was made 20 years earlier?". This question is, however, a very difficult one to answer and poses the following key scientific challenges.

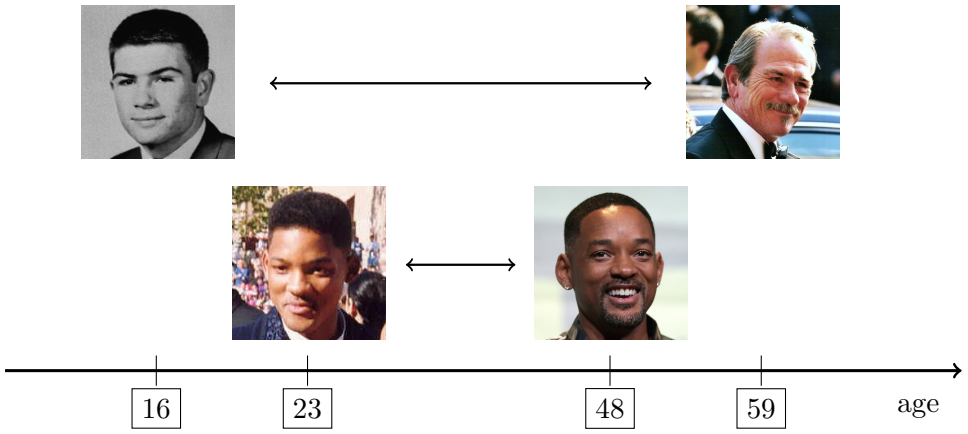
Figure (1.2): **Analyzing trends over time.** We show how translated images allow us to examine differences between cars manufactured in different time periods. We show the original cars from the 1990s (top row), their translation into the 1940s (middle row) and the absolute differences (bottom row) between the original image and the translated image. The absolute differences are shown as heatmaps where bright yellow color indicates a high difference. In this example the most salient differences are: different appearance of wheels, the presence of a runner board in the 1940s and the bulky distinctive hood in the 1940s.

Portraits that are characteristic of the 2000s that are very different from 1970s portraits. Long, straight or curly blonde hair are common in 2000s yearbooks, but are not featured much in 1970s yearbooks.

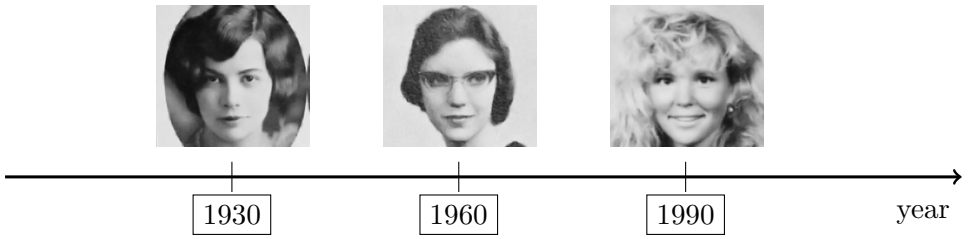
Portraits that are characteristic of the 1970s that are very different from 2000s portraits. Short hair is more common in the 1970s than in the 2000s.

Figure (1.3): **Visual mining.** Analyzing our translated images, we identify images that are characteristic of a specific time period, but would not pass in another period. This way, we identify the most striking differences between two decades.

Unpaired data. The first difficulty is that while we have pictures of cars made in different years, we do not have any example of a car transforming to appear to be made in a different year. We use the term *unpaired* to describe our data, meaning that we only observe single instances of objects at a specific time. While we know the time associated to a particular object, we do not have access to a picture of the same object translated to a different time. Most datasets are naturally unpaired. For example, there is no definitive way to know how a 20 years old woman in 1930 would have dressed and changed her hair, had she been transported to the year 2018 for a yearbook picture. Similarly, while we have pictures of cars manufactured throughout



Example of paired data. As it is possible to have images of the same person at a different age, an "age translation" model could be trained from paired data.



Example of unpaired data. Portraits of three different people taken in a different time period. It is not possible to know how a portrait of the same person would have looked like had it been taken in a different time period.

Figure (1.4): Illustration of the difference between paired (top) and unpaired data (bottom). In this work we focus on unpaired data, which is more challenging.

the 20th century, it is not possible to know how a particular car would have looked like had it been made in different time periods. By contrast, data is called *paired* when it contains the same object with different values for the varying attribute. In some situations, it is possible to obtain paired data. For example, it is possible to take multiple pictures of the same person as they age. Such a dataset could be used in order to try to make people appear older or younger. We illustrate the difference between paired and unpaired data in figure 1.4.

Working with unpaired data makes our task harder. If we knew how a Clio-equivalent would look like in 1950, we could train our model to reproduce this known appearance. However, we aim to tackle the problem with unpaired data, as it is more common in practice and often easier to obtain. In addition, evaluation in the unpaired setting is a major challenge. Because we have no direct way to compare each image generated by our model to a ground truth reference, we have to take special care when designing metrics to evaluate our approach.

Figure (1.5): **Challenge: intra-class variation.** The car dataset that we use presents very different types of cars (city cars, sedans, sports, utility cars and others).

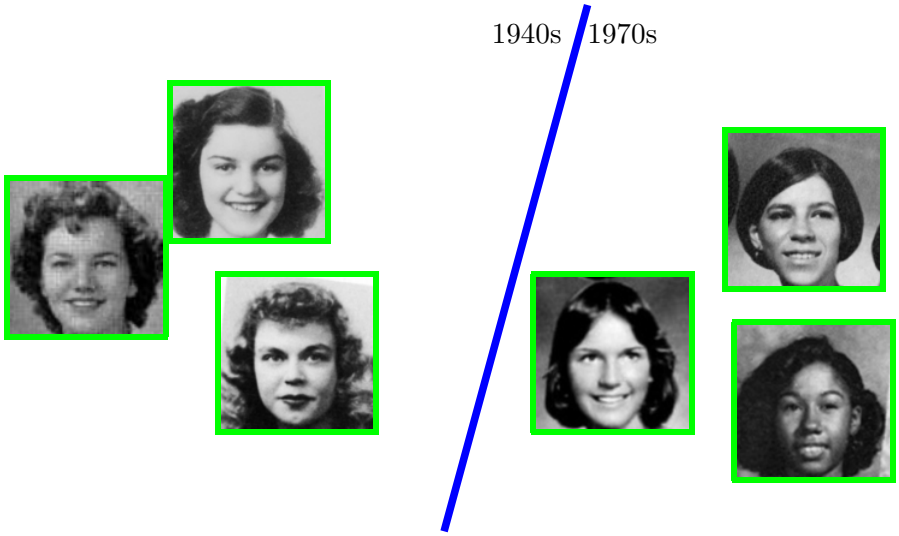
Figure (1.6): **Challenge: viewpoint variation.** The object can be imaged from a variety of viewpoints.

Figure (1.7): **Challenge: illumination variation.** Car pictures are generally taken outside, with different illumination conditions, but some can be also taken inside.

Appearance variations. Another major challenge is the high appearance variations of the objects in the input datasets. For example, cars of the 1990s include very diverse types of cars of cars, ranging from pick-up tracks to sports cars (figure 1.5). This increases the amount of appearance variation the model has to learn in order to translate between different time periods. This variability is intrinsic to the objects we consider.

Other factors of appearance variation include camera viewpoint and illumination, as illustrated in figures 1.6 and 1.7, respectively. Our objective is to isolate the time-dependent variation (e.g. the type of car tyres) from other appearance variation such as viewpoint, illumination or car type that are largely independent of time. The variation in viewpoint means that depicted objects are not aligned, making it difficult to directly compare images at the pixel-level, and to analyze changes in the shape of objects. In addition, analyzing changes in colors and textures is challenging due to variation in illumination. While it is possible to explicitly control for viewpoint and illumination when collecting data, we wish to consider real-world datasets collected in uncontrolled conditions.

Generating images. Finally, another challenge is related to the complexity of the output. This is in contrast to image classification that outputs only a score whether an image belongs to a specific class. In other words, our model not only needs to understand that certain parts of the input image



Classification. Deciding whether a particular picture comes from 1940s or 1970s is solving the classification problem. The output of this problem is a binary value for each image.

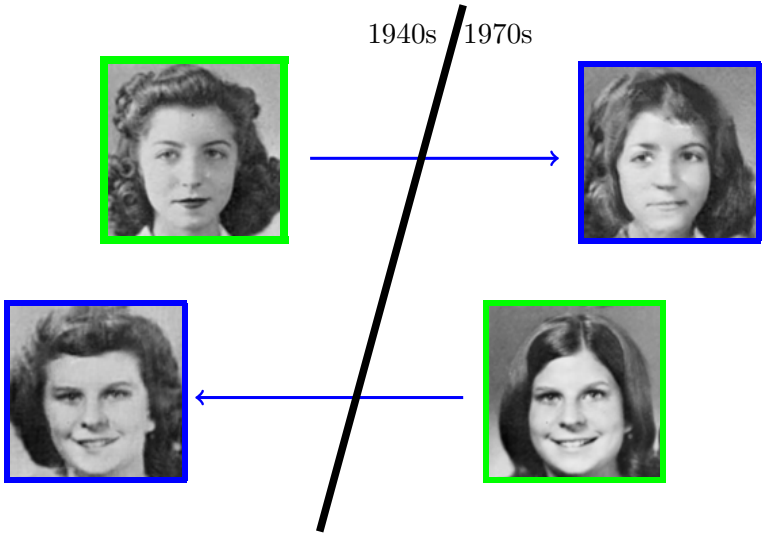


Image translation. In temporal translation we wish to translate an input image (green) into a different time period (blue). In contrast to classification the output is another image.

Figure (1.8): **Challenge: High-dimensional output.** Contrary to the classification task that has a binary output (top), the translation task (bottom) has a high dimensional output.

are characteristic for a specific time period in order to classify the image correctly, but needs to generate the appearance of those parts the way they look like in another time period, as illustrated in figure 1.8. It needs to understand what elements are typical of the time periods, but also how to change them to make them relate to other time periods (figure 1.8). In detail, in image classification the target output is a label, which is often a scalar.

In image translation the target output is a vector of high-dimension ($\mathcal{R}^{h \times w}$), where (h, w) is the dimension of the target output image.

In image translation, producing images that look realistic is also very challenging. The visual quality of the translated images is important in order to identify the key changes in appearance variation across time. It is hard to produce images that look realistic, sharp and capture the characteristic appearance of the target time period. Producing images as output presents also the additional challenge of performing automatic quantitative evaluation.

1.4 Contributions

This section lists the main contributions of this thesis.

The first contribution, described in chapter 4, is a new trainable bilinear factorization module that encourages factor separation by architecture design. We show how this module relates to other existing approaches including principal component analysis. We show its advantages compared to standard concatenation-based factor representation when used in a bottleneck auto-encoder architecture. We then show that our bilinear module can be plugged into a modern unpaired image translation architecture.

Our second contribution is our introduction of models for image translation that features our bilinear module. We investigate an encoder-decoder model with factor separation (section 5.1) and an adversarial traslation model (section 5.2). We demonstrate that learned factors of variation can be used for several visual discovery tasks such as discovering typical or non-typical examples for a particular characteristic attribute (such as a time-stamp).

Our third contribution, described in chapter 6, is the application of image translation models to the task of visual discovery in temporal image collections. We first introduce quantitative metrics that allow us to compare different models for the task of image translation. Then, we apply our image translation models to two temporal datasets: a dataset of portrait pictures acquired from yearbooks in the United States from 1930 to 2009 [Ginosar et al., 2015], and a dataset of cars manufactured between 1920 and 1999 [Lee et al., 2013]. We compare different models including several baselines using the proposed quantitative metrics. We also show the qualitative results obtained by different methods. We provide different ways to visualize the obtained translation results and show how they can be applied for the task of temporal visual discovery.

1.5 Thesis outline

This thesis is organized as follows.

In chapter 2, we overview the previous work this thesis is built upon. In particular, we detail the relevant work in image factorization, image translation and visual discovery.
In chapter 3, we overview the general approach common to the different methods we use for image translation, including the necessary notation.
In chapter 4, we introduce the bilinear operation as a module for image translation. We explain its relation to other methods, especially the principal component analysis.
In chapter 5, we explore image translation models. We investigate different architectures and loss functions. We discuss the strengths and weaknesses of different models.
In chapter 6, we analyze the results of our translation methods, in both a quantitative and qualitative way, and compare them to various baselines. We show results of temporal visual discovery including producing history-lapse videos.

Chapter 2

Related Work

We build upon work on linear and multi-linear models, image translation and temporal analysis. In this chapter we present prior work on these three topics.

2.1 (Multi-)linear image models

Linear and multi-linear models are useful for many computer vision tasks, such as classification, detection or recognition. They allow to represent images as combinations of factors that can be manipulated, which is useful for our goals in this thesis. In this section, we first discuss linear models, starting from principal component analysis, as it is the most direct way to factorize images. We then discuss how mixtures of linear models can be used when a single linear model is not enough to represent the data. Finally we discuss multilinear models, and in particular bilinear models, as they can model interactions that are not possible to obtain with linear models.

Linear models. In this section, we consider N data points $\mathbf{x}^1, \ldots, \mathbf{x}^N \in \mathbb{R}^d$. To simplify notation, we consider that these vectors are centered: $\sum_{1 \leq i \leq N} \mathbf{x}^i = \mathbf{0}$. For example, the set $\{\mathbf{x}^i\}$ can be a set of images with subtracted average image. We discuss linear and multi-linear representations of these data points and their usage.

Principal Component Analysis (PCA [Pearson, 1901]) is the change of basis to $\{\mathbf{e}^1, \ldots, \mathbf{e}^d\}$, an orthonormal basis that allows to project the data points on smaller subspaces with minimal reconstruction error. More specifically, the PCA minimizes quantities

$$\arg\min_{\mathbf{e}_S} \sum_{1 \leq i \leq N} \left\| \mathbf{x}^i - \sum_{1 \leq k \leq S} \langle \mathbf{x}^i, \mathbf{e}^k \rangle \mathbf{e}^k \right\|^2 \tag{2.1}$$

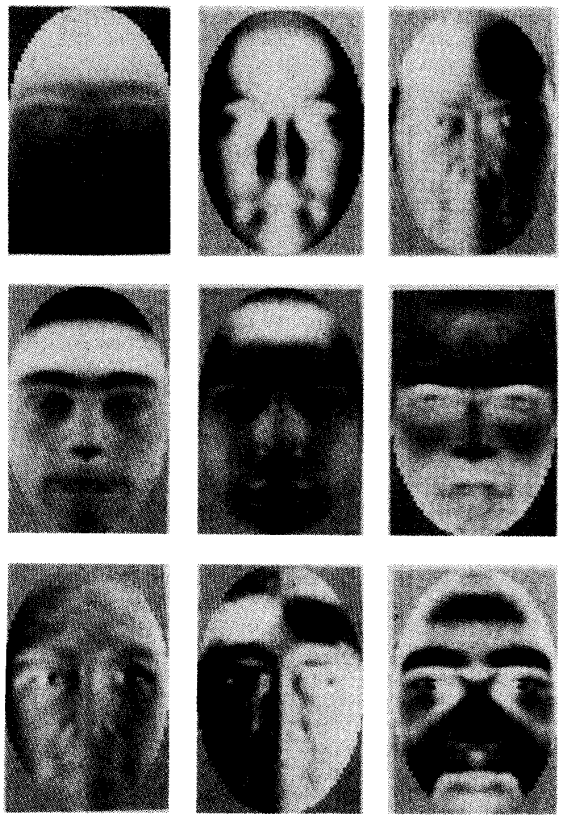


Figure (2.1): The first nine eigenpictures from [Kirby and Sirovich, 1990], ordered left to right and top to bottom by the order given by the PCA, i.e. by how important they are for accurate reconstruction of pictures. Faces can be reconstructed by linear combinations of these eigenpictures. Adding more eigenpictures will lead to better reconstructions, but with diminishing returns.

for each $1 \leq S \leq d$. Thus, PCA allows us to approximate the data points $\mathbf{x}^i$ by

$$\bar{\mathbf{x}}^i = \sum_{1 \leq k \leq S} \lambda_{ik} \mathbf{e}^k \,, \tag{2.2}$$

where $\lambda_{ik} = \langle \mathbf{x}^i, \mathbf{e}^k \rangle$. PCA allows us to reduce the dimensionality of the data points $\mathbf{x}^i$ from d to S in an optimal way, i.e. in a way that minimizes the sum of the reconstruction losses over all data points. The vectors $\mathbf{e}^1, \ldots, \mathbf{e}^N$ are called *principal components*. They can be obtained by performing Singular Value Decomposition (SVD) of X, as they are the right singular vectors of X. They are also the eigenvectors of the matrix $X^T X$ where $X \in \mathbb{R}^{N \times d}$ is the matrix with rows $\mathbf{x}^1, \ldots, \mathbf{x}^N$.

PCA has been applied to collections of images in [Kirby and Sirovich, 1990], and the resulting principal components are called *eigenpictures*. Examples of eigenpictures are shown figure 2.1. [Kirby and Sirovich, 1990] offer a

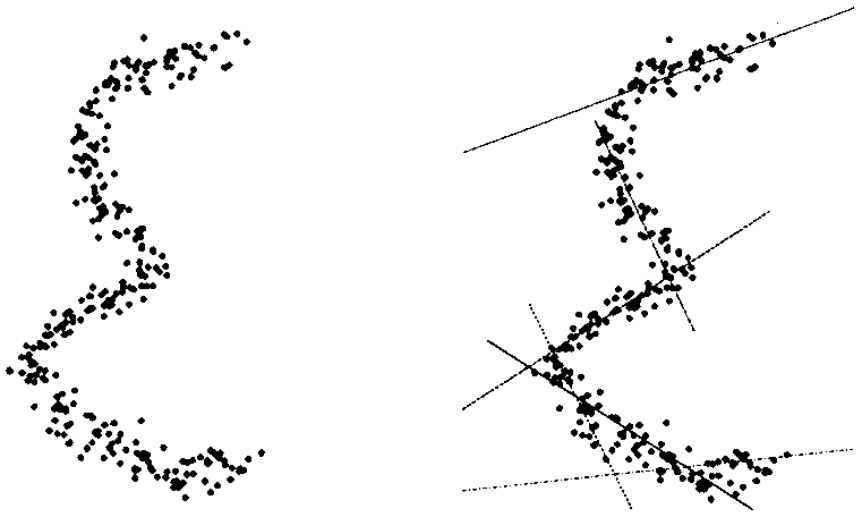


Figure (2.2): Illustation of how a one-dimensional surface (left) can be captured by a mixture of linear models (right) from [Hinton et al., 1997].

measurement of the error made by projecting these face images into this new space, and show that a linear combination of 50 eigenpictures were enough to have a good reconstruction of any face.
In [Turk and Pentland, 1991], this representation allows the authors to perform classification, detection and recognition tasks. To determine if a query image is an image of a face (classification task), the distance from the image to the face space (i.e. the distance between the image and its projection) is compared to a pre-defined threshold. To recognize a face, the authors compute the Euclidean distance between the projection of the query face image and the projection of the faces from the database. The query image is then assigned to the closest face of the dataset, granted that this minimum distance is below a defined threshold.

Principal component analysis has been successfully used for simple face datasets, but this method is not adequate for more complex datasets, for which a low dimensional orthogonal projection would result in too much image degradation. In this thesis we show relationship of our work and principal component analysis, but we apply PCA on an intermediate representation (composed of features from a convolutional neural network) instead of applying it directly on image pixels.

Mixture of linear models. Instead of using a single linear model, be it PCA or factor analysis, [Hinton et al., 1997] has used a mixture of linear models, as illustrated in figure 2.2, to represent images and perform classification. Each different class, e.g. each different digit in a dataset of digit

images, uses its own linear model. Each image $\mathbf{x}^i$ is thus approximated by

$$\tilde{\mathbf{x}}^i = \sum_{1 \leq z \leq m} q_z^i \sum_{1 \leq k \leq S} \lambda_z^{ik} \mathbf{e}_z^k \,, \tag{2.3}$$

where m is the number of classes, q_z^i is the coefficient soft-assigning image i to class z, $\mathbf{e}_z^1, \ldots, \mathbf{e}_z^S$ is the basis obtained by PCA for class z, and the λs are the associated coefficients. The coefficients q_z^i weigh the different z models for the image of index i, and sum to 1: $\sum_z q_z^i = 1$ for each i.

This method is used for classification of digits. The mixture model is trained with an *expectation-maximization* algorithm [Dempster et al., 1977], which alternates between two phases. The expectation phase consists in assigning each training example to their respective linear models. The maximization phase computes the parameters of each linear model.This model is very close to the Gaussian Mixture Model, although it does not whiten the projections. It has also been used for face detection [Yang et al., 1999].

The mixture of linear models is useful for image classification, but does not fully model the possible interactions between factors of variations in images. We next discuss multilinear factorization, which we use in this thesis, in order to achieve this.

Multilinear factorization. In order to factor out style and content for font and face analysis, [Tenenbaum and Freeman, 2000] has explored multiplicative interactions in a bilinear model. They introduce a "symmetric" bilinear model (figure 2.3), which consists of describing both a person and a pose by a vector. With this model, the image $\mathbf{x}^{ij}$ corresponding to person i and pose j can be approximated by the vector $\tilde{\mathbf{x}}^{ij}$ with coordinates:

$$\tilde{\mathbf{x}}^{ij} = \sum_{1 \leq k \leq S} \sum_{1 \leq l \leq T} a_{ik} b_{jl} \mathbf{e}^{kl}, , \tag{2.4}$$

where S and T are the size of the person and pose coefficients, $\{\mathbf{e}^{kl}\}_{1 \leq k \leq S, 1 \leq l \leq T}$ is a set of images, and the image $\tilde{\mathbf{x}}^{ij}$ is obtained by combination of these image given by the outer product of the person and pose coefficients $\mathbf{a}_i$ and $\mathbf{b}_j$. The set of images and coefficients are learned by performing Singular Value Decomposition repeatedly, fixing alternately the set of coefficient vectors $\{\mathbf{a}_i\}_i$ and $\{\mathbf{b}_j\}_j$.

[Vasilescu and Terzopoulos, 2002] extend this idea by considering an arbitrary number of factors of variations. To illustrate this, expression, pose, illumination and person identity are considered. Instead of having eigenfaces, the authors obtain tensorfaces. These tensors can be flattened into vectors, forming images similar to eigenfaces for any considered degree of variation.

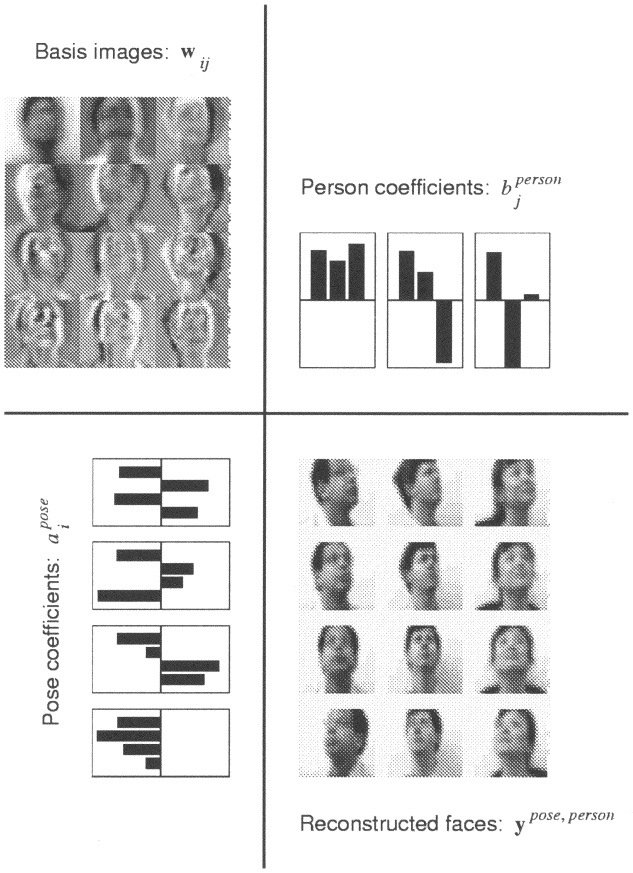


Figure (2.3): Illustation from [Tenenbaum and Freeman, 2000] of a symmetric bilinear model. The parameters (top left) of the model are the basis of the low-dimensional space. Each image is characterized by a vector containing pose information (bottom left) and person information (top right). Images can be reconstructed (bottom right) by making the outer product of these two tensors, and using the resulting matrix as weights for the linear combination of the basis images.

Bilinear models for Convolutional Neural Networks. These models have rarely been used for more complex images than aligned faces, because the complex variations that occur in natural images are hard to represent using linear models in pixel space. In contrast, we formulate bilinear factor translation in a end-to-end trainable encoder-decoder convolutional neural network (CNN) architecture. Bilinear layers have recently been used in CNNs, for example in the context of fine-grained recognition [Lin et al., 2015] and visual question answering [Fukui et al., 2016], but to the best of our knowledge, they were not used for visual discovery by image translation.

Figure (2.4): Rotation of cars obtained by [Tatarchenko et al., 2016]. In these two examples, the predictions are on the top row, and the ground truths are on the bottom row; the input image fed to the network is the leftmost image.

2.2 Deep Auto-Encoders and Image Translation

In this thesis, we translate images, which means that we modify an image so that a specific aspect of it changes, without changing its other characteristics. We use auto-encoders, i.e. neural networks that are trained to learn a coding of an image (with an *encoder*) and restore images from this coding (with a *decoder*). In this section, we review auto-encoder models and methods to translate images that are most relevant to our work. We first discuss auto-encoders, and how they can be used to modify images. We then focus on the setup where datasets do not have corresponding images with different attributes. We then discuss ways to improve the visual quality of images, especially perceptual losses and adversarial losses. Finally, we discuss cycle-consistent adversarial networks.

Auto-encoding images. Neural network based models such as Boltzmann machines have been long used for image generation [Hinton et al., 2006; Le Roux et al., 2011; Osindero and Hinton., 2008; Salakhutdinov and Hinton, 2008]. In [Vincent et al., 2008], a two-layer auto-encoder is used to denoise images. The authors note that different losses can be minimized to learn the parameters of the auto-encoder and generate an image $\tilde{\mathbf{x}}$ from an original data image $\mathbf{x}$, in particular the squared Euclidean loss

$$l(\mathbf{x}, \tilde{\mathbf{x}}) = \|\tilde{\mathbf{x}} - \mathbf{x}\|_2^2 \,. \tag{2.5}$$

In this thesis we also use, among others, a squared Euclidean loss in order to minimize the difference between a data point and its reconstruction. More recently, [Kingma and Welling, 2015] has used a variational auto-encoder to learn probabilistic models from large datasets, while [Yan et al., 2016] have used a variational auto-encoder to generate images with different attributes.

Image translation with deep auto-encoders. Several works have used auto-encoders in order to modify an input image. In [Tatarchenko et al., 2016], an auto-encoder is trained to modify the viewpoint of an object. They take as input a rendered image of a 3D model of an object, and a target

angle for the translated image. The output is the image of the object from a different viewpoint (see figure 2.4 for examples), along with an estimation of the depth. Because the method uses rendered images from a 3D model, it is possible to compare directly the output of the network to an image with the correct attribute. We say that data is *paired* when it is possible to obtain a ground truth for what a translated image should look like. Because data is paired, the authors train their networks by minimizing the distance between a rotated version of a car obtained by their network and one obtained by a render of the 3D model (ground truth). This distance is a combination of the squared Euclidean loss on pixels and a $L1$ distance on depth. By contrast, we use *unpaired* data in this thesis. As we cannot have a reference for what the image translated by our model should be, the method of [Tatarchenko et al., 2016] is not possible to use in our case.

[Kulkarni et al., 2015] propose another approach to build a factorized representation to generate images. They force specific parts of their code variable to explain the variation between paired images that only differ by the respective attribute. Again, this method is only possible to use when paired data is available.

Another approach, proposed by [Reed et al., 2014], learns a Restricted Boltzmann Machine that models multiplicative interactions between the different factors of variations. This model is also applied on paired data, but the use of multiplicative interactions is also similar to what we use in our convolutional neural networks.

To generate complex photo-realistic without losing low-level details, it is possible to modify the network architecture by adding shortcut connections, as demonstrated for very different sets of paired images in [Isola et al., 2017].

Deep auto-encoders for unpaired data. Several works have translated images with unpaired datasets. In [Cheung et al., 2014], the lack of paired data is solved by using a combination of multiple losses. In addition to a reconstruction loss that is used in paired data setups, they use a covariance loss between two parts of the extracted code, which enforces separation between the different parts of the code. More specifically, their model is an encoder f and a decoder g with $(\mathbf{z}, \hat{\mathbf{y}}) = f(\mathbf{x})$ and $\hat{\mathbf{x}} = g(\mathbf{z}, \hat{\mathbf{y}})$, where $\mathbf{x}$ is an image associated with the time labeled $\mathbf{y}$ in the dataset, $\hat{\mathbf{y}}$ is the observed attribute computed by the encoder f, $\mathbf{z}$ is called a latent variable and $\hat{\mathbf{x}}$ is the reconstruction of image $\mathbf{x}$. The model is trained with a combination of a reconstruction loss between $\mathbf{y}$ and $\hat{\mathbf{y}}$, a cross-entropy loss between $\mathbf{y}$ and $\hat{\mathbf{y}}$,

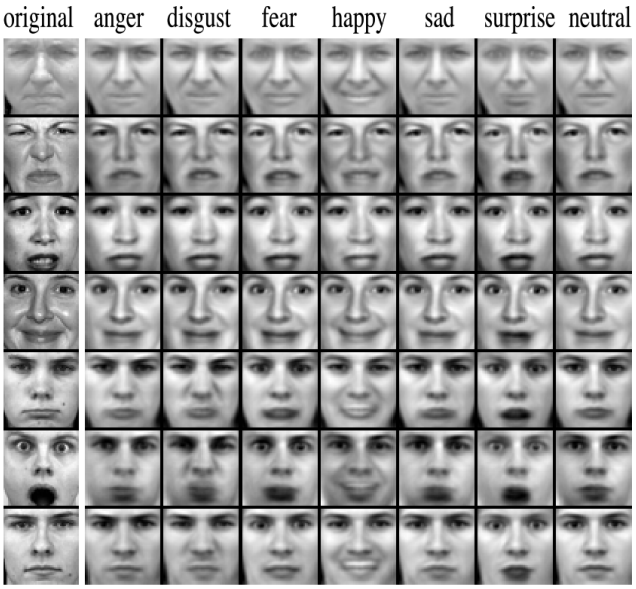


Figure (2.5): Translation of faces into different emotions in [Cheung et al., 2014]. Input images, in the left most column, are encoded into $(\mathbf{z}, \hat{\mathbf{y}})$. Then, the generator takes $\mathbf{z}$ and a different emotion $\mathbf{y}'$ as input, to decode into images shown on the right.

and a cross-covariance loss between $\mathbf{z}$ and $\hat{\mathbf{y}}$:

$$L_{\text{COV}}\left((\hat{\mathbf{y}}^{1,\ldots,N}), (\mathbf{z}^{1,\ldots,N})\right) = \frac{1}{2N}\sum_{ijn}\left((\hat{y}_i^n - \bar{\hat{y}}_i)(z_j^n - \bar{z}_j)\right)^2, \tag{2.6}$$

where $(\hat{\mathbf{y}}^{1,\ldots,N})$ and $(\mathbf{z}^{1,\ldots,N})$ are N associated results of the encoder, and $\bar{\hat{\mathbf{y}}}$ and $\bar{\mathbf{z}}$ their respective means. This cross-covariance loss is computed on mini-batches of size N. The goal of minimizing the the cross-covariance loss is to disentangle the attribute coded by $\mathbf{y}$ from the other factors of variation of the image, which are then encoded by $\mathbf{z}$. After training, the model is able to change $\hat{\mathbf{y}}$ to translate images, without changing the other factors of variation present in the variable $\mathbf{z}$. Examples shown in figure 2.5 show how this model can translate faces to different emotions. In section 5.1, we build on this work by proposing an architecture that uses a separation loss between the attribute and the non-observed factors of variation. We replace the cross-covariance loss with a stronger separation loss in order to increase the variation induced by changing the attribute.

Adversarial networks. As shown by [Denton et al., 2015; Dosovitskiy and Brox, 2016; Johnson et al., 2016; Radford et al., 2016; Wang and Gupta, 2016; Zhu et al., 2016], the visual quality of the output imagery can be enhanced using *adversarial costs* [Goodfellow et al., 2015] that encourage the network to generate images that cannot be distinguished from natural images.

While our work is not focused on improving the realism of the translated images, we make use of adversarial networks to improve their visual quality.

Perceptual loss. Auto-encoders are often trained with a reconstruction loss on pixels. In [Johnson et al., 2016], the authors introduce a perceptual loss to compare feature representations of images. In particular, they use a squared Euclidean loss on a feature reconstruction of both the original and the translated image, as it captures better the perceptual difference between images. Similarly, in our work we use feature construction losses.

Unpaired Translation with Cycle-Consistent Adversarial Networks. In [Zhu et al., 2017], the authors translate images from two domains, $\mathcal{C}_1$ and $\mathcal{C}_2$, with unpaired images. They jointly train two networks, $M_{1\to2}$ and $M_{2\to1}$, translating images from the two domains, such that for an image $I \in \mathcal{C}_1$, $M_{2\to1}\left(M_{1\to2}\left(I\right)\right)$ is close to I. They also use a conditional adversarial loss with two discriminators, D_1 and D_2. D_1 is trained to distinguish between images from the domain 1 $\mathcal{C}_1$, and images translated from domain 2 to domain 1 $M_{2\to1}(\mathcal{C}_2)$. The networks $M_{1\to2}$ and $M_{2\to1}$ use residual building blocks from [He et al., 2016]. We use a similar network in our work. We extend this work by tackling a dataset with more than two domains.

2.3 Visual mining and temporal analysis

Visual mining has been used in computer vision as a tool to accomplish various goals. We call "visual mining" the task of extracting images, or parts of images, from a collection of images that exhibit interesting properties or can be useful for another computer vision task. In this section, we first discuss mining whole images in a dataset of images. Then, we discuss mining discriminative image elements, i.e. parts of images that can be used to differentiate between objects or other attributes. Finally, we discuss how mining discriminative image elements have been used for temporal analysis of visual data.

Mining images. Extracting relevant images from large image collections has been tackled in a number of related work. In [Quack et al., 2008], the authors have used external information such as geographical data and Wikipedia articles in order to mine pictures of interesting landmarks from very large collections of images. [Martin-Brualla et al., 2015] is related to both mining and temporal analysis. In this work, the authors reconstruct time-lapses of various landmarks from a large dataset of pictures across the

world. To achieve this, they mine multiple pictures of the landmarks, as well as a reference viewpoint for each of them, before warping the images to form a time-lapse of each landmark. Similar work is done in [Matzen and Snavely, 2014], in which the authors build a "scene chronology" by clustering 3D points of different scenes obtained from a large set of images. In [Sivic and Zisserman, 2004], objects are mined across different shots of a movie by clustering feature representations of images. These works concern instance-level matching, i.e. finding the same object or scene across different images captured at different times.

In our work, we also mine images that are characteristic of their time in section 6.2. Beyond this, we focus on finding visual patterns that generalize beyond instance-level matching. This has generally been done by mining discriminative image elements.

Mining discriminative image elements. Patches, windows, parts of images are said "discriminative" if they help distinguishing between attributes, such as the object category (dogs or cats), the location (Barcelona or Casablanca) or the time (which year).

For example, [Grauman and Darrell, 2006] extracts typical features that characterize object categories from unsupervised data. To do this, local features (such as SIFT [Lowe, 1999]) are computed in each image and a similarity score is computed for each pair. Then, based on this score, the features are clustered and represent learned image categories. Still for unsupervised data, [Lee et al., 2009] extracts mid-level visual features from convolutional Deep Belief Networks. Discovering mid-level features from unsupervised data has generally been evaluated by using these features to categorize new images [Singh et al., 2012; Sivic et al., 2005; Karlinsky et al., 2009]. This discovery of mid-level patches is done without any supervision, so these models are limited to discovering only visual patterns that are both very common and highly visually consistent. To address this issue, another line of work (including ours) uses image-level supervision.

Image datasets can have annotation for each image. This is called weak supervision as it provides annotations only on. This enabled [Doersch et al., 2012] to obtain geographically representative elements from different places. This work features a dataset of street level images from cities such as Paris, San Francisco or Prague. For each city, they extract patches from these images that are both common in that particular city and do not appear in other ones. To achieve this, they compute an image representation for patches based on histogram of gradients [Dalal and Triggs, 2005] (HOGs) and a scaled-down version of the patch. Then, using a score based on cross-

Figure (2.6): Street images of Paris (top) and London (bottom), and results (right) of visual elements that are characteristic of these two cities from [Doersch et al., 2012].

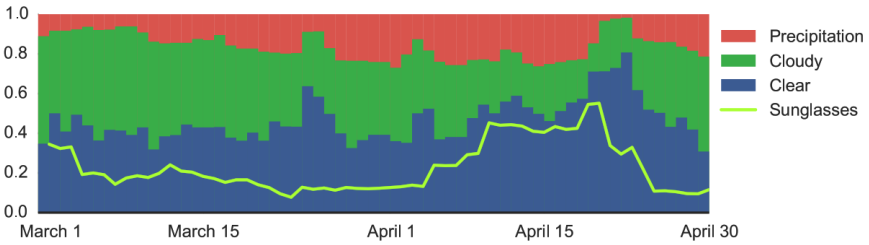


Figure (2.7): Correlation between weather and presence of sunglasses in pictures from [Matzen and Snavely, 2015].

correlation, they look for the nearest neighbor patches and select patches that are only close to those of the same city. The authors refine this selection several times using discriminative learning to only select patches that would be best to train a linear Support Vector Machine (SVM) detector. Figure 2.6 shows examples of the resulting patches. For example, blue street signs are very frequent in Paris, but will rarely be found in other cities. This approach treats the attribute as a discrete variable (e.g. Paris or London, no in-between). Our work differs as we present methods that treat time as a continuous variable.

Temporal analysis. Another approach for retrieving discriminative parts of objects is examining which parts of images are used by CNNs. In [Matzen

and Snavely, 2015], the authors train a CNN classifier on "foveal images", for which all but a small patch of the image has been blurred, and "bubble images", for which all but a small patch of the image has been grayed out. This network is then used to find the bubble images whose classification prediction matches best the true label of the images. The bubble images are then clustered and, again, used to train SVM classifiers. The authors present an example in which they learn an bubble image classifier to predict the month of the year in which pictures of people in NYC have been taken. They show that it is harder to assign the correct time to two pictures when they have been taken in close months (e.g. July and August) than when they have been taken a long time apart (e.g. July and January). We make similar observations when trying to estimate the time period of pictures in this thesis. [Matzen and Snavely, 2015] also analyze temporal trends with their method. Noting that the months of March and April are more distinct than other pairs of consecutive months, they looked at the cluster of patches and saw that the most discriminative one was images of sunglasses. They then correlated the presence of sunglasses with clear weather (see figure 2.7). This work is different from us as we seek to translate full images to exhibit differences instead of extracting patches.

Also closely related to our work, [Lee et al., 2013] aim to discover elements whose appearance changes with time and location. They work on multiple datasets including a dataset of cars that we use. Cars made between 1920 and 1999 are annotated with the time they have been manufactured. We also use this dataset in this thesis. [Lee et al., 2013] first use HOGs to describe patches, and mine discriminative patches by taking those whose nearest neighours across the dataset are only present in images of similar attribute (such as the year). In a second phase, they establish correspondences between patches across the attributes. They iteratively train detectors of similar visual elements, and gradually enlarge the positive training set of the detector by incorporating matching patches of an increasing range of years. By doing this, they fully leverage the continuous character of their labels.

We differ from these works as we analyze temporal effects by looking at generated images, having both a global and local representation of the changes that occur because of time.

Chapter 3

Approach Overview and Notation

In this chapter, we present an overview of our model for temporal image translation, introduce the key notation and provide a detailed roadmap for the rest of the thesis.

3.1 Objectives and approach overview

We seek to discover patterns in changes of appearance related to time in a large collection of images. We address this problem by learning a model that translates images in time to answer question: "What would the input image look like at a different time?". More precisely, we develop a factored model that learns to separate time-specific appearance variation from the rest of the content of the image. This is illustrated in figure 3.1. The outcome of this approach is that, for a given input image, we can produce a "history-lapse video" revealing changes over time by simply varying the target time variable.

More formally, our main hypothesis is that an image is completely determined by a time variable $\mathbf{y}$ and a latent content variable $\mathbf{z_y} \in \mathbb{R}^C$. For example, the appearance of a particular car from the 1930s is a combination of two sets of factors. The first set of factors, determined by a time vector $\mathbf{y}$, is related to the period the car was built in, due to cars exhibiting time-characteristic features, such as running boards or round mud guards for cars from the 1930s. The second set of factors, encoded in a content vector $\mathbf{z_y} \in \mathbb{R}^C$, is independent of the time and includes, for example, viewpoint and body type (e.g. sedan vs. pick-up truck). Our factorization model will be trained in an end-to-end manner from a dataset of images with time stamps. We assume the dataset is *unpaired*, i.e. each object is observed at only one time. For example, we observe a particular car made at one specific time period and we do not know how that particular car would have looked like if it had been made at a different time period. The objective of the training is

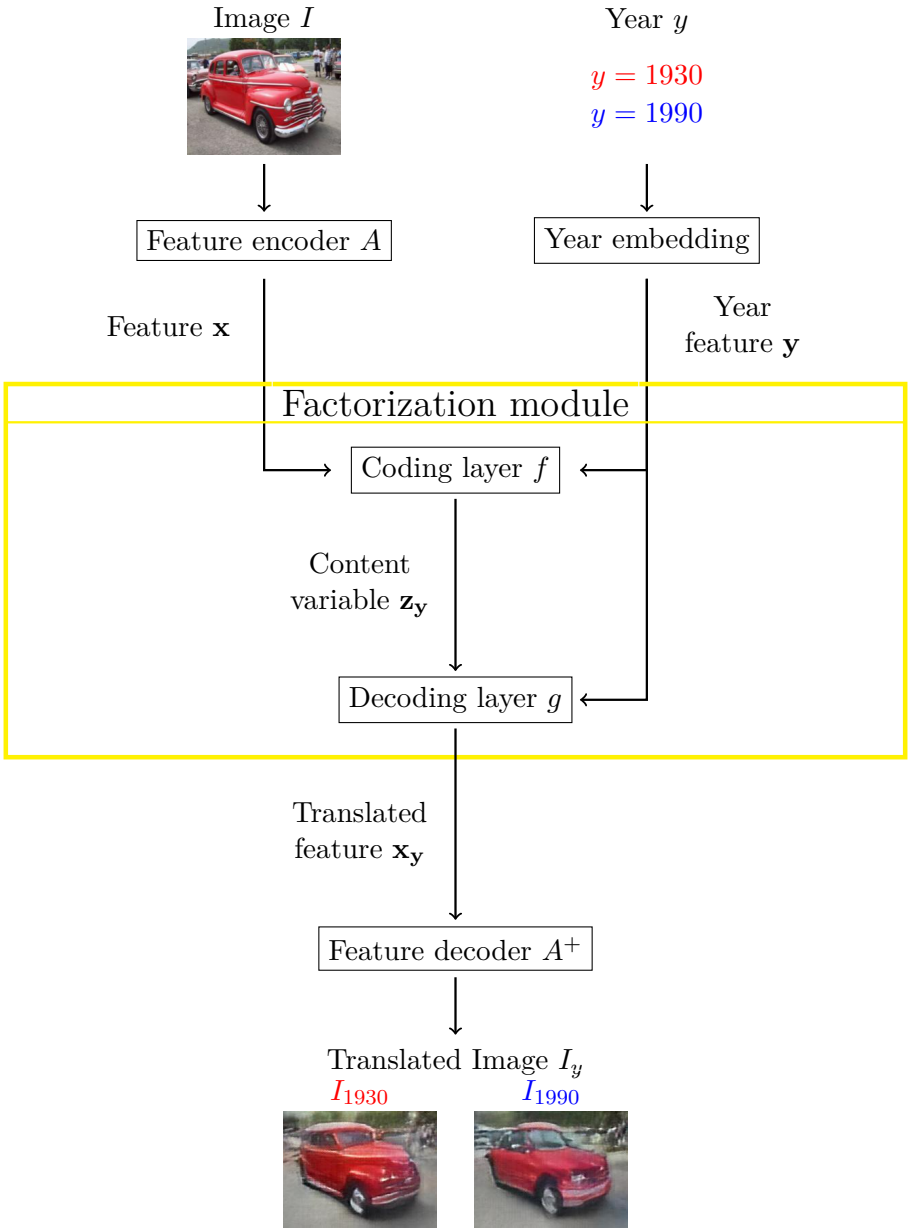


Figure (3.1): **Convolutional neural network architecture for factored visual discovery.** We assume that an image is completely determined by (i) a latent content variable $\mathbf{z_y}$ capturing the appearance factors independent of time such as car type and color, and (ii) the time feature $\mathbf{y}$ capturing the appearance factors specific to certain time, such as the running boards for cars made in 1930s. The input image I is encoded into a feature vector $\mathbf{x}$, which is transformed together with year variable $\mathbf{y}$ using a mapping $f(\mathbf{x}, \mathbf{y})$ into a latent content representation $\mathbf{z_y}$. A second mapping $g(\mathbf{z}, \mathbf{y})$ then combines the latent content with the input time feature $\mathbf{y}$ to produce new feature $\mathbf{x_y}$. The reconstructed feature $\mathbf{x_y}$ is finally decoded into image I_y. Parameters of the factorization module together with the encoder and decoder are learnt in an end-to-end manner.

to isolate the time dependent appearance variation from such unpaired input training data. This is achieved by optimizing a cost function that includes a reconstruction term, possibly together with optional translation and separation terms. The reconstruction term ensures that the model reconstructs well the input training images at their given time period. The translation term ensures that when the input image is translated to different time periods, the output images exhibit characteristics of the training images from those target periods. The separation terms encourages the independence of factors related to time and factors encoding the time-agnostic content of the image.

3.2 Notation

As illustrated in figure 3.1, we assume each input image is associated with a time stamp, represented by a feature vector $\mathbf{y} \in \mathbb{R}^T$. We denote by $\mathbf{x} \in \mathbb{R}^K$ a K-dimensional feature encoding of image I obtained with an encoder: $\mathbf{x} = A(I)$. Our factorization module (see figure 3.1) computes a mapping of these features into a latent content factor $\mathbf{z_y} = f(\mathbf{x}, \mathbf{y})$. Then the inverse mapping $\mathbf{x_y} = g(\mathbf{z_y}, \mathbf{y})$ decodes the content factor $\mathbf{z_y}$ jointly with the time dependent factor $\mathbf{y}$ into a new image feature $\mathbf{x_y}$ that is reconstructed into the output image $I_\mathbf{y}$: $I_\mathbf{y} = A^+(\mathbf{x_y})$. The specific form of f and g is discussed in chapters 4 and 5.

3.3 Detailed Roadmap

In chapter 4, we introduce a trainable bilinear factorization module which allows us to represent multiplicative interactions between time and content. used typically in neural network architectures. We also provide details of our efficient implementation of bilinear factorization, which allows us to apply our bilinear module on large-scale datasets.

In chapter 5, we discuss different image translation models. We discuss two different architectures based on an bottleneck auto-encoder and a recent adversarial image translation architecture. We show how our bilinear factorization module can be inserted in both architectures and discuss in details the different loss functions used.

In chapter 6, we discuss the quantitative and qualitative results obtained with our temporal translation approach. We first introduce the two datasets we experimented on, and discuss the metrics that we use to quantitatively evaluate our approach and to compare it other methods. Finally, we demonstrate how to use our temporal image translation model for visual discovery

in unpaired image collections annotated with time stamps. We show how to use our model to generate history-lapse videos that translate the input image into different time periods while preserving the identity of the depicted object. We analyze the generated images sequences to extract interesting changes in style over time.

Chapter 4

Bilinear Factorization Module

In this chapter we introduce a trainable bilinear factor separation module that can be plugged into different image translation architectures. In section 4.1 we present and detail this bilinear module. In section 4.2 we describe how this module relates to other factor separation approaches such as Principal Component Analysis, style and content factorization as well as simple feature vector concatenation. Finally, as speed is important for training from large datasets, in section 4.3 we explain our efficient implementation of the bilinear module. Chapter 5 will then show how to employ our bilinear module in different image translation architectures.

4.1 Module description

In this section we describe our new bilinear factored representation module, which is visualized in figure 4.1a, and that can be used as the core factorization module in the visual discovery architecture shown in figure 3.1 in chapter 3. Our bilinear factor separation module is composed of two bilinear layers. The first bilinear layer $f(\mathbf{x}, \mathbf{y})$, given by eq. (4.2), combines the input feature $\mathbf{x}$ and given input year $\mathbf{y}$ in a multiplicative fashion into a content vector $\mathbf{z_y}$. The second bilinear layer, given by eq. (4.1), combines the content vector $\mathbf{z_y}$ with the input time $\mathbf{y}$ in a multiplicative way into the reconstructed feature $\mathbf{x_y}$.

The key idea of our approach is to combine the content vector $\mathbf{z} \in \mathbb{R}^S$ and the style vector $\mathbf{y} \in \mathbb{R}^T$ in a multiplicative way, using a bilinear layer. In particular, the decoding layer $g(\mathbf{z_y}, \mathbf{y})$ reconstructs the feature $\mathbf{x_y}$ from the latent content variable $\mathbf{z_y}$, representing the input $\mathbf{x}$ interpreted using the

given input time vector $\mathbf{y}$, and the given input time variable $\mathbf{y}$ as

$$\mathbf{x}_{\mathbf{y}}^{k} = g(\mathbf{z}_{\mathbf{y}}, \mathbf{y})^{k} = \sum_{s=1}^{S} \sum_{t=1}^{T} W_{g}^{s,t,k} \mathbf{z}_{\mathbf{y}}^{s} \mathbf{y}^{t}\,, \tag{4.1}$$

where W_g are the learnable weights of the decoding layer g and a^b denotes component b of the vector a (and similarly for tensor components). Note that if $\mathbf{y}$ is a one hot encoding of the time period y, matrices $W_g^{\cdot,t,\cdot}$ can be interpreted as linear weights specific to the period y.

Before we can proceed with translation using equation (4.1), the latent content vector $\mathbf{z}_{\mathbf{y}}$ needs to be estimated from the input image feature $\mathbf{x}$. We compute $\mathbf{z}_{\mathbf{y}}$ using another bilinear layer as

$$\mathbf{z}_{\mathbf{y}}^{s} = f(\mathbf{x}, \mathbf{y})^{s} = \sum_{k=1}^{K} \sum_{t=1}^{T} W_{f}^{k,t,s} \mathbf{x}^{k} \mathbf{y}^{t}, \tag{4.2}$$

where W_f are the weights of the encoding layer f and $\mathbf{z}_{\mathbf{y}}^{s}$ is the component s of the latent content variable $\mathbf{z}_{\mathbf{y}}$ representing the input $\mathbf{x}$ interpreted using the given input time vector $\mathbf{y}$. Note also that similarly to W_g, if $\mathbf{y}$ is a one hot encoding of the time period y, matrices $W_f^{\cdot,t,\cdot}$ can be interpreted as weights of a linear layer for time period t. Also note that the latent content variable $\mathbf{z}_{\mathbf{y}}$ depends on the given input time vector $\mathbf{y}$ and can be thought of as a "projection" of the input representation $\mathbf{x}$ onto an "image subspace" specific to period y.

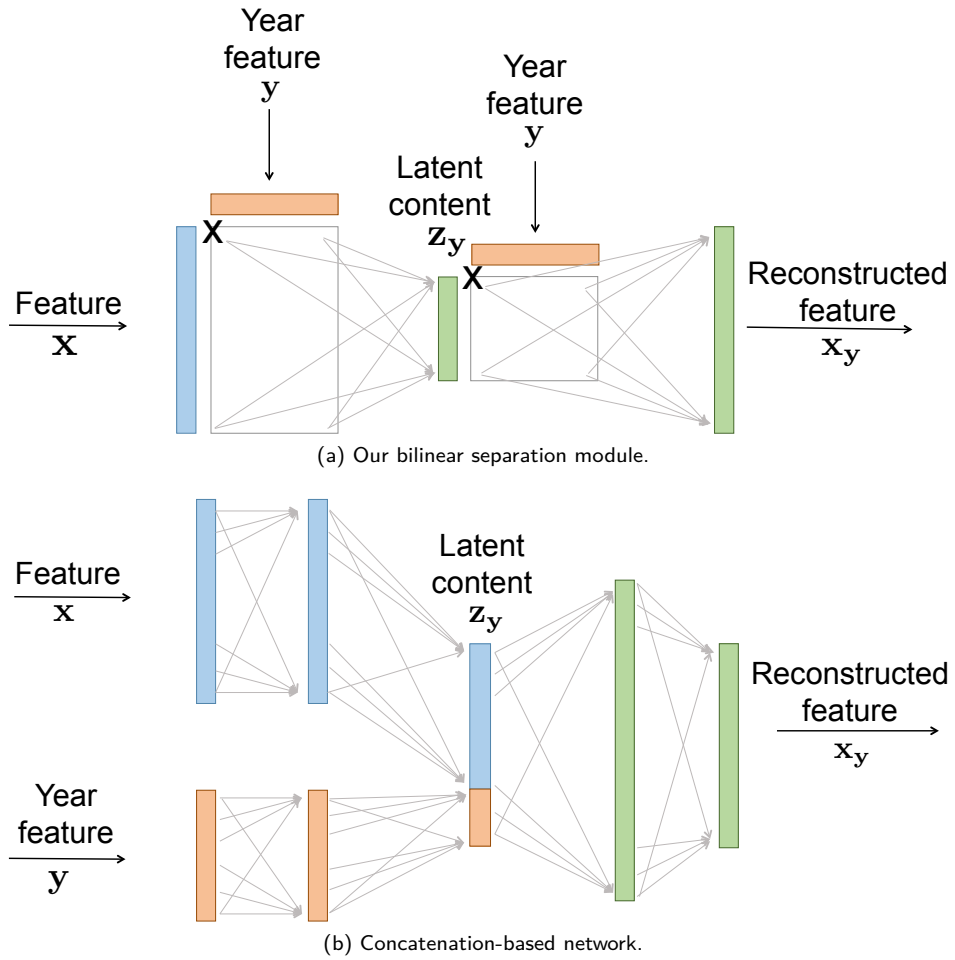
4.2 Relation to other approaches

In the following we discuss the relation of our bilinear module to (i) image translation based on concatenation, (ii) principal component analysis, and (iii) style and content factorziation.

4.2.1 Relation to concatenation-based generation methods

The standard way to combine variables in a CNN architecture is to concatenate them (or their transformed versions. See for example [Dosovitskiy et al., 2016]. Since the latent variables in concatenation based factorization methods are often estimated independently in a feed forward way, we omit the index $\mathbf{y}$ for the $\mathbf{z}$ variable in this paragraph. The output of a decoder $g(\mathbf{z}, \mathbf{y})$ of a single decoding layer g applied to a concatenation of input features z and y is written as

$$\mathbf{x}_{\mathbf{y}}^{k} = g(\mathbf{z}, \mathbf{y})^{k} = \sum_{i=1}^{S} W_{g}^{i,k} \mathbf{z}^{i} + \sum_{i=S+1}^{S+T} W_{g}^{S+i,k} \mathbf{y}^{i}, \tag{4.3}$$



(a) Our bilinear separation module.

(b) Concatenation-based network.

Figure (4.1): **Factorization architectures.** Our bilinear factorization module (top) and the standard concatenation-based architecture (bottom) have two principal differences. First, our module captures multiplicative interactions between time $\mathbf{y}$ and content $\mathbf{z_y}$, while concatenation implies additive interactions. Then, we explicitly include dependency on time $\mathbf{y}$ in computing latent content $\mathbf{z_y}$. More layers can be included in our bilinear module similar to the concatenation architecture.

where W_g are the weights of the fully connected layer g. This equation is to be compared with equation (4.1), where the interaction between $\mathbf{y}$ and $\mathbf{z}$ is multiplicative and not additive.

While the output is in practice often a result of several layers, it builds on this simple additive combination of the latent style and time. The resulting architecture is visualized in figure 4.1b, and we compare its representational power to our bilinear module in chapter 6.

4.2.2 Relation to principal component analysis (PCA)

In the case where $\mathbf{y}$ is a one hot vector representing each time period, there is a direct relation between our bilinear module and PCA. Indeed, our module can in this case be rewritten for each time period independently as the succession of two linear layers. If a $L2$ reconstruction loss is used between the output and the input, these two successive linear layers are equivalent, up to a simple transformation of the feature space, to performing PCA decomposition followed by reconstruction [Baldi and Hornik, 1989; Sanger, 1989]. Thus, our architecture is equivalent in this case to performing PCA on each time period independently, a property that we verified experimentally: for each time period $\mathbf{y}$, we reconstruct input feature $\mathbf{x}$ in the basis given by the PCA of image features in that period. In contrast to PCA, however, our bilinear module can naturally be inserted in an encoder-decoder architecture. This in turn allows training the entire factored image representation (including the bilinear module, encoder and decoder) in an end-to-end manner using more complex loss functions that go beyond simple $L2$ reconstruction.

4.2.3 Relation to bilinear style and content factorization

By representing vectors as a bilinear combination of time and latent content latent variables in concatenation based factorization methods, our model is related to the bilinear style and content separation of [Tenenbaum and Freeman, 2000]. Our approach has, however, three fundamental differences. First, we assume the style (in our case time) vector is known, given and fixed for each input instance, and thus we only need to estimate the content vector. Second, because we assume the style is known, we can estimate the content vector with a simple linear operation. This is in contrast to [Tenenbaum and Freeman, 2000] who have to resort to an algorithm based on an iterative SVD. Because of this key difference the model of [Tenenbaum and Freeman, 2000] also requires multiple observations of the same content with different styles, i.e. paired data, which our model does not and can operate on unpaired data collections. Finally, we include our bilinear separation approach as a module in a CNN architecture, which allows for end-to-end parameter learning.

4.3 Efficient implementation of the bilinear module

We implement our module and networks in Torch [Collobert et al., 2011]. Torch provides an implementation of the bilinear operation called “nn.Bilinear”. However, we found this implementation can be too slow for our purposes, so we implemented another version. In this section, we detail the differences between these two implementations and give performance results.

Let's call n the number of inputs processed simultaneously by the graphics card. This set of inputs is called a minibatch. In equation 4.1, we give the definition of the bilinear layer for a pair of vectors. In the neural network, the operation is implemented for matrices, in order to concatenate several input images into a single minibatch that can be processed more efficiently on graphics processing units. With a minibatch of size n, the bilinear operation can be written as

$$\mathscr{B}(A,B,W)_{i,c} = \sum_{\substack{1\leq a\leq n_A \\ 1\leq b\leq n_B}} B_{i,b}W_{c,b,a}A_{i,a} \qquad \forall i = 1,\ldots,n\,, \tag{4.4}$$

where $\mathscr{B}(A,B,W) \in \mathbb{R}^{n\times n_C}$ is the output matrix holding n vectors of output dimension n_C, $A \in \mathbb{R}^{n\times n_A}$ and $B \in \mathcal{R}^{n\times n_B}$ are the input matrices holding n vectors of dimension n_A and n_B respectively, and $W \in \mathcal{R}^{n_C\times n_B\times n_A}$ denotes the parameters of the bilinear operation. Note that there is no additive bias in the operation.
The default bilinear module in the Torch implementation makes this operation in the following order. First, for each $c = 1\ldots n_C$, the quantity

$$BW_{c,*,*} \in \mathbb{R}^{n\times n_B}$$

is computed with a call to the matrix multiplication routine, where $W_{c,*,*} \in \mathbb{R}^{n_B\times n_A}$ is the matrix obtained by taking the c-th sub-matrix of W:

$$(BW_{c,*,*})_{i,a} = \sum_{1\leq b\leq n_B} B_{i,b}W_{c,b,a} \qquad \forall i = 1,\ldots,n\,\forall b = 1,\ldots,n_B\,. \tag{4.5}$$

Then, the result of the bilinear operation is obtained by making the element wise product between $BW_{c,*,*}$ and A, and summing over the index a, which gives $C_{i,c}$ for each $c = 1\ldots n$.
While the matrix multiplication, element wise product and sum are fast operations done on the GPU, the loop over $c = 1\ldots n$ submatrices in the mini-batch and the associated function call overheads can result in a significant time loss.
We can reduce the number of loops by rewriting the bilinear equation, for each $c = 1,\ldots,n_C$ and $i = 1,\ldots,n$ as:

$$\mathscr{B}(A,B,W)_{i,c} = \sum_{\substack{1\leq a\leq n_A \\ 1\leq b\leq n_B}} \sum_{1\leq h\leq n_A n_B} I_{h,b,a}B_{i,b}W_{c,b,a}A_{i,a}\,, \tag{4.6}$$

where $I \in \mathbb{R}^{n_A n_B\times n_A\times n_B}$ is a sparse indexing matrix given by

$$I_{h,b,a} = \begin{cases} 1 & \text{if } h = 1+(b-1)n_A + a\,, \\ 0 & \text{otherwise.} \end{cases} \tag{4.7}$$

By switching the order of the sums we can rewrite equation 4.6 as:

$$\mathscr{B}(A,B,W)_{i,c} = \sum_{1 \leq h \leq n_A n_B} \sum_{\substack{1 \leq a \leq n_A \\ 1 \leq b \leq n_B}} B_{i,b} I_{h,b,a} A_{i,a} W'_{h,c}\,, \tag{4.8}$$

where $W' \in \mathbb{R}^{n_A \times n_B, n_C}$ is a 2-dimensional matrix with the same coefficients as W:

$$W'_{1+(b-1)*n_A+a,c} = W_{c,b,a}\,. \tag{4.9}$$

Finally, we can see that we have re-written the bilinear operation as

$$\mathscr{B}(A,B,W) = \mathscr{B}(A,B,I)W'\,. \tag{4.10}$$

We have implemented a bilinear layer using this formulation. Note that the bilinear operation $\mathscr{B}(A,B,I)$ (which can be seen as an outer product of each line of A and B) takes $n_A n_B$ loops instead of n_C, and can thus lead to significant gain when $n_A n_B < n_C$.
In our implementation of the model, the bilinear operation is used twice. First, the code $\mathbf{z} \in \mathbb{R}^S$ is obtained by applying the bilinear operation on the input feature $\mathbf{x} \in \mathbb{R}^K$ and the year feature $\mathbf{y} \in \mathbb{R}^T$, where S, K and T are the dimensions of the content vector, the input feature vector, and the timed vector, respectively. In this case, using the default implementation of the bilinear module, we have $n_C = S$ loops, which is small: typically $S = 16$ and in all of our tests $S \leq 128$. (see section 6.4 for a more detailed discussion about the size of $\mathbf{z}$.). Second, the code is decoded into $\mathbf{x}_y \in \mathbb{R}^K$. In this case we have $n_C = K$, which is 64,896 for our auto-encoder model. We thus use our bilinear layer implementation instead, which reduces the number of loops to $K \times T$, which is 128 for a code of size $K = 16$ and a time feature of size of $T = 8$. Our new implementation is in this case approximately 300 times faster. With this implementation, each iteration of our bilinear layer is as fast as the baseline concatenation module. For our auto-encoder network, it takes 3 minutes to do a full pass on a dataset of 10,000 images on a GTX 1080 GPU, i.e. a full pass on 1M images would take about 5 hours. Furthermore, compared to a concatenation baseline with a similar number of parameters, our model converges in a smaller number of iterations to a lower value of the objective.

Chapter 5

Image Translation Models

In chapter 4, we have discussed how we separate time and content information using a bilinear module and how this module compares with other factorization techniques and architectures. While it is possible to apply the bilinear factor separation directly on the image pixels [Tenenbaum and Freeman, 2000] with some success on simple images, such as letters or digits, this approach does not work well for complex natural images. Instead, we perform bilinear factor separation on higher-level image features extracted using a convolutional neural network. This is achieved by employing our bilinear factor seperation module as part of a convolutional neural network architecture. This model can be trained end-to-end.In this chapter we plug-in the bilinear factor separation module in two different convolutional network architectures for image translation.

The first one, described in section 5.1, is a bottleneck auto-encoder. We explain how the presence of a bottleneck allows us to perform image translation. We describe how to employ the bilinear module in the auto-encoder, compare it with standard concatenation, and discuss different losses used for training.

In section 5.2, we show how to employ the bilinear factorization module in a modern image translation architecture based on CycleGAN [Zhu et al., 2017]. We show that employing our bilinear factorization module allows us to learn a single encoder/decoder for all time periods instead of having a specific encoder/decoder for each pair of source and target periods. We explain how this model is trained to translate images thanks to a domain-adversarial loss.

5.1 Image translation using a bottleneck auto-encoder

In this section we show how to employ our bilinear factorization module in a bottleneck auto-encoder for temporal image translation. We first give an intuition of how to perform image translation using a bottleneck auto-encoder

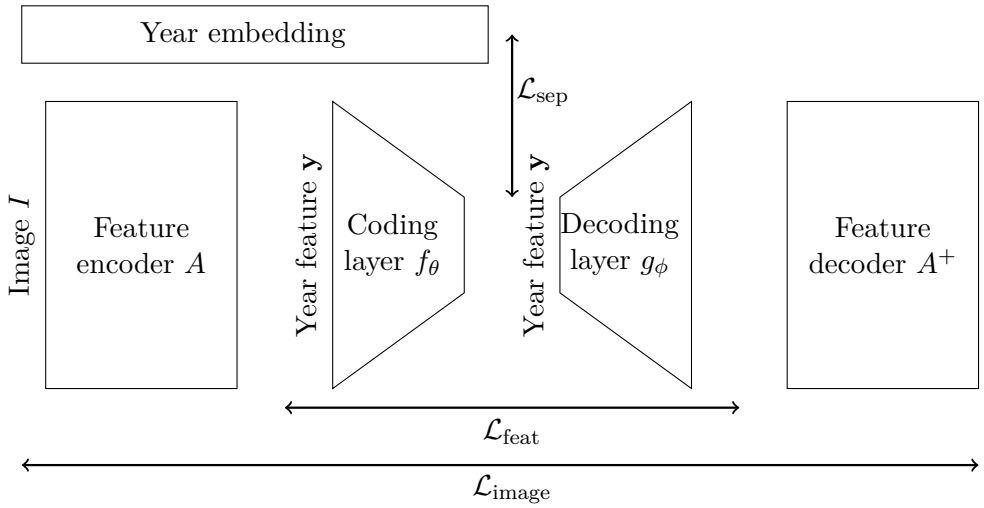


Figure (5.1): **The bottleneck auto-encoder.** In this model, modification with time is enforced by the bottleneck and the separation loss $\mathcal{L}_{\text{sep}}$, which separates time and content. The proximity to the original image is enforced by the reconstruction loss on the feature $\mathcal{L}_{\text{feat}}$, which compares the input and output of the factorization module, and the reconstruction loss on the image $\mathcal{L}_{\text{image}}$, which compares the original image to the output of the model.

and then discuss the different losses used for training. Finally, we discuss the influence of the bottleneck code size on the model.

5.1.1 Intuition and Architecture

The bottleneck auto-encoder model is composed of three parts (figure 5.1): the feature encoder, the bottleneck factorization module, and the feature decoder. The feature encoder transforms an image into a vector representation that we refer to as "feature vector" that is easier to manipulate than the image itself. The goal of the factorization module is to translate this feature vector from its original time to its target time. We achieve that by compressing the encoded feature into a code vector of small size, before combining it with the year information. Because the code size is small, the model has to be efficient when storing information in the code. The intuition is that forcing the size of the code to be small encourages the code to contain only the time-independent information as the time-dependent information (in the form of the target time period) is added after the bottleneck during decompression. In detail, the time information is added either with the bilinear operation described in chapter 4, or with the standard feature vector concatenation. We will show experimentally in chapter 6 the benefits of our bilinear module. The feature obtained by combining these vectors can be decoded into the translated image.

More formally, we denote A the encoder, which takes as input an image I and outputs a feature vector $\mathbf{x}$: $\mathbf{x} = A(I)$. Then, the encoding layer f and the decoding layer g form the factorization module: $\mathbf{z_y} = f(\mathbf{x_y}, \mathbf{y})$ where $\mathbf{z_y}$ is the latent content variable, and $\mathbf{x_y} = g(\mathbf{z_y}, \mathbf{y})$ where $\mathbf{x_y}$ is the translated feature. We denote by A^+ the decoder that takes this translated and outputs a translated image $I_\mathbf{y}$: $I_\mathbf{y} = A^+(\mathbf{x_y})$. The factorization, composed of a coding layer f and a decoding layer g, can either be a concatenation module or a bilinear module. The concatenation module is composed of a linear layer for a simple linear layer $\mathbf{z} = f(\mathbf{x})$, and a linear layer on the concatenation of $\mathbf{z}$ and $\mathbf{y}$ for g as described in equation (4.3). For this module the latent code variable $\mathbf{z}$ does not explicitly depend on $\mathbf{y}$. The bilinear module described in section 4.1.

Architecture details. Our encoder is based on the AlexNet [Krizhevsky et al., 2012] architecture. We use the decoder with a deconvolutional architecture similar to [Dosovitskiy and Brox, 2015], but with batch normalization ([Ioffe and Szegedy, 2015]) in order to make the training more stable.

AlexNet has 5 convolutional layers followed by 3 fully connected layers. Features can be extracted from any of these layers, and we have to choose the best features for our task. Using features extracted from the earlier layers make the feature inversion easier and closer to the original image, while deeper features have a higher-level representation of the image. Based on preliminary results as well as the comparison of the reconstructions based on different features by [Dosovitskiy and Brox, 2015], we opted to use features from the 4th layer of AlexNet, *c*onv4.

Training details. The parameters of the encoder are initialized by training it for classification on the ImageNet [Russakovsky et al., 2015] dataset. The decoder is pre-trained to minimize the squared distance between the original image and the auto-encoded image $\left\|A^+\left(A\left(I\right)\right) - I\right\|_2^2$, without any modification of the feature vector.

After this pre-training, we train the model in two phases. First,we train the factorization module, f and g. Then, we fine-tune the whole model. We discuss the losses that we use for this training and fine-tuning in the next section.

5.1.2 Training loss

Our full loss for training the network can be written as the combination of three losses:

$$\mathcal{L}_{\text{total}}(M) = \mathcal{L}_{\text{image}}(M) + \lambda_{\text{feat}}\mathcal{L}_{\text{feat}}(f, g) + \lambda_{\text{sep}}\mathcal{L}_{\text{sep}}(f)\,, \tag{5.1}$$

where M is our bottleneck auto-encoder model, and λ_{feat} and λ_{sep} are hyper-paramaters weighing the reconstruction loss on the image $\mathcal{L}_{\text{image}}(M)$, the reconstruction loss on the feature vectors $\mathcal{L}_{\text{feat}}(f, g)$ and the optional separation loss $\mathcal{L}_{\text{sep}}(f)$. We detail these losses in the next paragraphs.

Reconstruction loss on the image. The reconstruction loss on the image is a combination of a $L1$ loss and an adversarial loss:

$$\mathcal{L}_{\text{image}}(M) = \mathop{\mathbb{E}}_{I}\left[\|M(I, y_I) - I\|_1\right] + \lambda_{\text{adv}}\mathcal{L}_{\text{adv}}(f)\,, \tag{5.2}$$

where λ_{adv} is a hyper-parameter, and y_I is the time corresponding to an image I. The goal of the $L1$ loss is to match an auto-encoded image $M(I, y_I)$ with the original I as much as possible. As it tends to produce blurry images (see figure 6.4), the objective of the adversarial term $\mathcal{L}_{\text{adv}}$ is to make images look more realistic, i.e. look more like a natural image, for example presenting sharp edges. To improve the stability of the training, we use the least-squares generative adversarial network loss, as described in [Mao et al., 2017]. More precisely, we train a fully convolutional discriminator network D that takes as input either a real image I from the dataset, or an image generated by our model $M(I, y_I)$, where y_I is the time period corresponding to image I, and evaluates for each location of the image, in a convolutional way, if it comes from the dataset or our model. The generator M is trained to both provide a good reconstruction and confuse the discriminator. The adversarial loss for the generator thus is:

$$\mathcal{L}_{\text{adv}}(M) = \mathop{\mathbb{E}}_{I}\left[\left(D(M(I, y_I)) - 1\right)^2\right]\,, \tag{5.3}$$

where $D(M(I, y_I))$ is the output of the discriminator for the generated image $M(I, y_I)$. In other words, the generator tries to produce images that are indistinguishable from real images by the discriminator, i.e. the discriminator outputs label 1. Note that this adversarial loss also depends on the discriminator D, which is trained jointly with our model using the following loss function

$$\mathcal{L}_{\text{disc}}(D) = \mathop{\mathbb{E}}_{I}\left[D\left(M(I, y_I)\right)^2 + \left(D(I) - 1\right)^2\right]\,, \tag{5.4}$$

where the expectation is taken over the set of training images I with associated time stamps y_I. The first term, $D\left(M(I, y_I) - 0\right)^2$, is the loss on generated

images, which the discriminator is trying to map to class 0, and the second term, $(D(I)-1)^2$, is the loss on real images I, which the discriminator is trying to map to class 1. The intuition is that the discriminator tries to separate real and generated images while the generator tries to produce images that are indistinguishable from real photographs. Note that losses given by (5.3) and (5.4) are applied in a fully convolutional manner as described in [Zhu et al., 2017]. This adversarial loss is encouraging the output images to look like real images but does not attempt to discriminate between different time periods. Adversarial loss that discriminates between different time periods will be discussed in section 5.2.2.

Reconstruction loss on the feature vector. In addition to the above reconstruction loss on the pixel-level, we pre-train the bottleneck modules with a reconstruction loss on the feature vector level:

$$\mathcal{L}_{\text{feat}}(f,g) = \mathop{\mathbb{E}}_{\mathbf{y}\in\mathcal{Y}} \left\| \mathbf{x}^i_{\mathbf{y}} - g(f(\mathbf{x}^i_{\mathbf{y}}),\mathbf{y}) \right\|_2^2 , \tag{5.5}$$

where $\mathbf{x}^i_{\mathbf{y}}$ is the input image and $g(f(\mathbf{x}^i_{\mathbf{y}}),\mathbf{y})$ its reconstruction with the given ground truth time $\mathbf{y}$. As discussed in chapter 4, this is analoguous to Principal Component Analysis in the case of a bilinear bottleneck module. For both the bilinear and concatenation modules, the feature-level reconstruction loss given by equation (5.5) ensures that the features are preserved when the target year is the year of the original image (no translation).

Separation loss. Finally, the concatenation model can be complemented with a separation loss $\mathcal{L}_{\text{sep}}$. The aim is to learn a model where the code and time vector are two independent variables. For example, when the model is applied to images of faces, the code can capture the identity of the person and attribute the year of the photograph. Here, knowing the year of the photograph does not convey any information about the identity of the person. It is this intuition we would like to capture. We explain in the following paragraph the details of the separation loss. Please note that we only use it for the bottleneck auto-encoder with a concatenation module. As will be seen in chapter 6, we found that this type of separation loss is not necessary to achieve good results with our bilinear factorization module.

More formally, we assume that the code $\mathbf{z}\in\mathbb{R}^K$ and time vector $\mathbf{y}\in\mathcal{Y}$ are realizations of random variables Z and Y, with probability distributions $p(Z)$ and $p(Y)$, respectively. The aim is to enforce that Z and Y are independent, i.e. that

$$P(Z,Y) = P(Z)P(Y)\,. \tag{5.6}$$

As by definition the joint probability of Z and Y can be written as

$$P(Z,Y) = P(Y)P(Z|Y)\,, \tag{5.7}$$

it can be seen from equation (5.6) that independence can be expressed with the equation

$$P(Z|Y) = P(Z)\,. \tag{5.8}$$

Thus, Z and Y are independent if and only if the conditional distribution $P(Z|Y = \mathbf{y})$ are the same for all $\mathbf{y} \in \mathcal{Y}$. In other words, knowing $\mathbf{y}$ is not going to bring any additional information about Z. To simplify the computation, we further assume Z has a Gaussian distribution $N(\boldsymbol{\mu}, \Sigma)$. Since $\mathbf{z}$ is the output of a linear layer and an affine transformation can be absorbed by this layer, we can further assume without loss of generality that $\boldsymbol{\mu} = \mathbf{0}$ and $\Sigma = \mathtt{I}$, i.e. $P(Z) = N(\mathbf{0}, \mathtt{I})$. Plugging-in the Gaussian form of $P(Z)$ into constraint (5.8), leads to

$$P(Z|\mathbf{y}) = N(\mathbf{0}, \mathtt{I}), \forall \mathbf{y} \in \mathcal{Y}. \tag{5.9}$$

In other words, the code Z is independent of time Y when for each value $\mathbf{y}$ of the time the observed conditional distribution of codes $\mathbf{z}$ is a unit Gaussian. We translate the constraint (5.9) into the following separation loss

$$\mathcal{L}_{\text{sep}}(f) = \sum_{\mathbf{y} \in \mathcal{Y}} \text{KL}\left(P(Z|Y = \mathbf{y})\right), \mathcal{N}(\mathbf{0}, \mathbf{I}))\,, \tag{5.10}$$

where $KL(.,.)$ is the Kullback-Leibler (KL) divergence between two distributions.Note that the separation loss is minimized exactly when the constraint (5.9) is satisfied, i.e. the conditional distribution of Z for each $\mathbf{y}$ follows a unit Gaussian. To further simplify the computation, we assume that the dimensions z_k of $\mathbf{z}$ are independent and Gaussian, which leads to the following form of the separation loss

$$\mathcal{L}_{\text{sep}} = \sum_{\mathbf{y} \in \mathcal{Y}} \sum_{k=1}^{K} \frac{1}{2} \left(-\log((\sigma_k^{\mathbf{y}})^2) + (\mu_k^{\mathbf{y}})^2 + (\sigma_k^{\mathbf{y}})^2 - 1 \right)\,, \tag{5.11}$$

where μ_k^y and $(\sigma_k^{\mathbf{y}})^2$ are the mean and variance of the k-th dimension of the codes $\{\mathbf{z}_1^{\mathbf{y}}, \ldots, \mathbf{z}_i^{\mathbf{y}}, \ldots \mathbf{z}_{N_{\mathbf{y}}}^{\mathbf{y}}\}$ extracted from the images annotated with time $\mathbf{y}$:

$$\mu_k^y = \frac{1}{N_{\mathbf{y}}} \sum_{i}^{N_{\mathbf{y}}} z_{i,k}^{\mathbf{y}}\,, \tag{5.12}$$

$$(\sigma_k^{\mathbf{y}})^2 = \frac{1}{N_{\mathbf{y}}} \sum_{i}^{N_{\mathbf{y}}} (z_{i,k}^{\mathbf{y}} - \mu_{y,k})^2\,. \tag{5.13}$$

Note that the loss $\mathcal{L}_{\text{sep}}(f)$ is summed over all dimensions of $\mathbf{z}$ (inner sum) and all possible values of time $\mathbf{y}$ (outer sum). Note again that above separation loss is enforcing independence of random variables Z and Y by pulling the mean and variance of codes $\mathbf{z}^{\mathbf{y}}$ extracted from images annotated with a certain time $\mathbf{y}$ towards 0 and 1, respectively.

5.1.3 Discussion

The size of the bottleneck inside the model M, i.e. code $\mathbf{z_y}$, is a critical parameter for the bottleneck auto-encoder. It influences the result in two different ways. Decreasing the dimension of $\mathbf{z}$ corresponds to decreasing the dimension of the linear space of the possible values of $g(\mathbf{z_y}) = \mathbf{x_y}$, and leads to the following observations.On one hand, as a result of the information loss corresponding to this dimensionality reduction, the reconstruction error $\|\mathbf{x} - \mathbf{x_y}\|$ will be larger, and the reconstructed image will lack details. On the other hand, the network has to rely more on the time specific code $\mathbf{y}$ to reconstruct the training data. As a result, the reconstructed images will exhibit more strongly the characteristic features of the images in the target period, which is part of our goal. In the extreme case where only the time specific code $\mathbf{y}$ is preserved and the dimension of the bottleneck $\mathbf{z}$ is zero, the best the network could do in terms of the reconstruction loss on the training data is to produce an output close to an average training image for the given time period.

We find that the concatenation module is more sensitive to the bottleneck code size than the bilinear module. For simple datasets, such as for faces, a balance between preserving image quality and generating typical images for each period can be found with both modules. In other cases, such as with the car dataset we use, we find that only the bilinear module allows us to produce strong appearance variations when changing the time period while preserving the characteristic features of the particular object depicted in the image. We compare results obtained with both modules in experiments presented in section 6.4.

Because the presence of a bottleneck is detrimental to image quality with both modules, we explore another image translation architecture without any bottleneck discussed in the next section.

5.2 Adversarial translation

In this section we introduce the domain adversarial translation model. First, we give an intuition of how this model performs image translation. We then discuss the different losses used for training. Finally, we compare this model to the bottleneck auto-encoder presented in section 5.1 and CycleGAN [Zhu et al., 2017].

5.2.1 Intuition and Architecture

A successful approach to image translation that does not use a bottleneck auto-encoder, CycleGAN, was recently introduced in [Zhu et al., 2017]. The

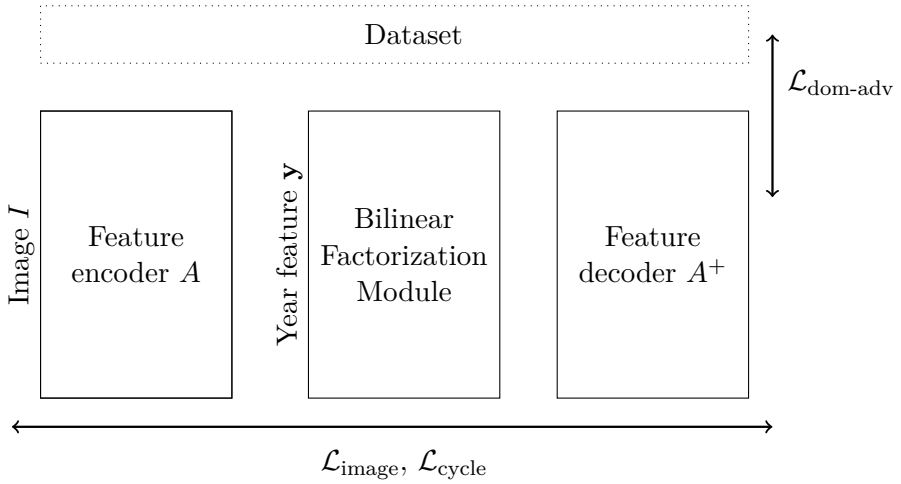


Figure (5.2): **The adversarial translation model.** In this model, modification with time is enforced by the domain adversarial loss $\mathcal{L}_{\text{dom-adv}}$, which compares the output of the model to other images of the same time period. The proximity to the original image is enforced by the reconstruction loss on the image $\mathcal{L}_{\text{image}}$, which compares the original image to the output of the model, and the cycle loss $\mathcal{L}_{\text{cycle}}$, which compares the original image to an image translated to another time period then translated again to its original time.

work is focused on the case of two image domains, such as two different time periods in our case.

One of the main drawbacks of CycleGAN for our task is that it does not build a shared factored representation, and thus requires to train two translation networks per domain pair. It also does not allow to represent continuous quantities, such as time. We extend the architecture of [Zhu et al., 2017] to multiple domains. To achieve this we (i) share the network weights for translations between all periods, and (ii) add a factorization module as described in chapter 4 as a central block, in parallel to a shortcut connection. We show this model in figure 5.2.

Architecture details. The network is based on [Johnson et al., 2016]. It uses residual building blocks ([He et al., 2016]), i.e. blocks of convolutional layers with the same dimension for the input and output, with an additive operation that adds the input of the block to the output. This network allows minimal quality loss in the reconstruction, because there is little dimensionality reduction. Instead, the network can translate images because it is trained with a domain adversarial loss, which forces the result of the translation to appear like images of the target time period. We can thus remove the bottleneck in our bilinear factor separation, which makes it equivalent to a single bilinear layer. Interestingly, while the architecture with a single bilinear layer is simpler than the full bilinear module, it typically has more parameters since it does not have a bottleneck.

5.2.2 Training Loss

To train our translation architecture denoted M, we use a combination of multiple losses:

$$\mathcal{L}_{\text{total}}(M) = \mathcal{L}_{\text{id}}(M) + \lambda_{\text{cycle}}\mathcal{L}_{\text{cycle}}(M) + \lambda_{\text{dom-adv}}\mathcal{L}_{\text{dom-adv}}(M)\,, \tag{5.14}$$

where $\mathcal{L}_{\text{id}}$, $\mathcal{L}_{\text{cycle}}$, and $\mathcal{L}_{\text{dom-adv}}$ are the identity loss, the cycle consistency loss and the domain adversarial loss detailed in the following paragraphs, and the λs are the relative weights of each loss.

Identity loss. We use a L_1 reconstruction loss on the generated image without translation, as we did for the bottleneck model:

$$\mathcal{L}_{\text{id}}(M) = \mathop{\mathbb{E}}_{I}\left[\|M(I, y_I) - I\|_1\right], \tag{5.15}$$

where the expectation is taken over all images I and y_I is the time period associated to image I. As the target year is the year of the original image, this loss enforces that the image is modified as little as possible. The loss also improves the stability of the training.

Cycle consistency loss. Similar to [Zhu et al., 2017], we also consider a cycle consistency loss

$$\mathcal{L}_{\text{cycle}}(M) = \mathop{\mathbb{E}}_{I,y}\left[\|M(M(I, y), y_I) - I\|_1\right], \tag{5.16}$$

where the expectation is taken over all images I and all possible target time periods y, and y_I is the time period associated to image I. This loss ensures that a source image I translated to time y, $M(I, y)$, can be converted back to the source image I when using the source time y_I as the input to the generator.

Domain adversarial loss. Finally, to enforce that the generated images look different for each time period, we rely on a domain adversarial loss

$$\mathcal{L}_{\text{dom-adv}}(M) = \mathop{\mathbb{E}}_{I,y}\left[\left(1 - D^y\left(M(I, y)\right)\right)^2\right], \tag{5.17}$$

where the expectation is taken over all images I and all possible target time periods y, and $D^y\left(M(I, y)\right) \in [0, 1]$ indicates if $M(I, y)$ resembles images from the dataset with attribute y. The intuition is that the generated image $M(I, y)$ for time period y should be classified correctly by the discriminator D. We train a single discriminator D that takes an image as input and is

trained to output a vector that contains one hot encoding of the input image time period. D^y denotes the component of the output vector corresponding to time y. The discriminator is trained jointly with the generator M using the following loss

$$\mathcal{L}_{\text{dom-disc}}(D) = \mathop{\mathbb{E}}_{I}\left[(1 - D^{y_I}(I))^2\right] + \beta \mathop{\mathbb{E}}_{I,y}\left[D^y(M(I,y))^2\right], \tag{5.18}$$

where β is a hyper-parameter. The first term encourages that the discriminator predicts 1 for the correct time period y_I for training images I. The second term encourages that for the translated images $M(I, y)$ the discriminator output for period y, D^y should be zero. In other words, the discriminator should be able to detect that the generated image does not belong to the target period. Note again that this loss is evaluated in a fully convolutional manner as described in [Zhu et al., 2017].

5.2.3 Discussion

This approach has some similarities with the aforementioned bottleneck auto-encoder described in section 5.1. In both cases, a $L1$ loss is used to compare an original image to the output of the network after translation is performed. With the auto-encoder, an adversarial loss sharpens the results and makes them more realistic. This addition is not necessary in the case of the adversarial translation model, because its domain adversarial loss, which is necessary for translation, also makes images more realistic. With this domain adversarial loss, the adversarial translation model does not rely on a bottleneck and is thus capable of outputting images of better quality.

We note here the differences between CycleGAN and our network. In CycleGAN, only two classes are considered, while we can consider an arbitrary number of classes (in our case time periods) using a single network. We do not need one generator network and one discriminator network for each class, but we use one generator network for all classes and one discriminator for all classes. The only parts of our networks that use more parameters when considering new classes are the bilinear factorization module in the middle of the generator, and the last layer of the discriminator which determines the time period of an image.

Chapter 6

Visual Discovery in Unpaired Image Collections

In the previous chapter, we have described how we incorporate the bilinear module in different image translation architectures. The goal of this chapter is to experimentally compare these architectures and demonstrate their use for visual discovery. The objectives are threefold. First, we wish to validate whether the previously described architectures can perform temporal image translation. Second, we wish to compare their performance to sensible baselines on the image translation task. Finally, we wish to demonstrate visual discovery in unpaired image collections via learning and applying the proposed temporal image translation models.

This chapter is organized as follows. In section 6.1, we introduce the datasets and metrics that we use to evaluate our models. In section 6.2, we present preliminary results obtained with the bottleneck auto-encoder. These results demonstrate the difficulty of temporal translation on complex visual data and have been for us the motivation for developing our bilinear factorization module (described in chapter 4).

In the remainder of this chapter we present results of our bilinear module incorporated in the two image translation architectures described in chapter 5. In section 6.3, we analyze the effect of different architectures and losses on the results of temporal image translation. In section 6.4, we provide an in depth comparison of our bilinear module with the the standard feature concatenation. In section 6.5, we give examples of the new type of image analysis enabled by our model: generating history-lapse videos and discovering trends by analyzing changes across time.

6.1 Datasets and metrics

To illustrate the generality of our method, we use two datasets of historical images for our evaluation. The historical car dataset [Lee et al., 2013] is a set of cars made in a span of 80 years. The historical yearbook portraits [Ginosar et al., 2015] contains pictures of students taken across 80 years. For both datasets, we grouped the images by decades and selected three decades with clear differences to analyse the results of our method (40s, 70s and 2000s for the faces and 40s, 60s and 90s for the cars). We thus obtained 3600 training images and 300 testimages for the face dataset across three decades, and 7299 training images and 1258 test images for the car dataset. Below we give details of the two datasets and introduce appropriate metrics to quantitatively compare the results of the different approaches.

Historical car dataset. We show results on the challenging car dataset introduced in [Lee et al., 2013], containing modern images of cars together with their construction date, between 1920 and 1999. This dataset presents several difficulties, with cars having different appearance and different backgrounds. The cars are also imaged in a variety of lighting conditions and from a variety of viewpoints. To localize the cars in the image, we first run a standard Faster R-CNN car detector [Ren et al., 2015] pre-trained on Pascal VOC images. The detected bounding boxes are then resized to fit the input of the networks without preserving the aspect ratio. Despite this pre-processing, this dataset remains a great challenge.

Historical yearbook portraits. We also applied the proposed model to the historical photograph collection of American high school yearbook portraits from 1930s to 2000s of [Ginosar et al., 2015]. Aligned face images are known to be relatively easy to analyze, and results on this type of data have been demonstrated for example in [Ginosar et al., 2015; Reed et al., 2014; Yan et al., 2016].

Evaluation metrics. Quantitative evaluation for our task is difficult as there is no ground truth for how a specific car or portrait looked like in both 1920s and 1990s. We evaluate the different methods by looking at a reconstruction metric and a classification metric. Our reconstruction metric is simply the $L1$ distance between the input image and its translation into its ground truth period. While this is not a very precise metric of image similarity, we found that for our task it correlated well with percieved image quality. As pointed above, this reconstruction metric is limited since it can only be used together with the ground-truth time period, and doesn't evaluate wether the generated image shows characteristics of the target time period. This is the goal of our classification metric. Following [Isola et al., 2017], we use an off-the-shelf classifier to assess whether our model is able to generate

images that "look" like the images of the intended time period. This metric is related to the "inception score" [Salimans et al., 2016], object detection evaluation in [Wang and Gupta, 2016] and the "semantic interpretability" measure in [Zhang et al., 2016]. In detail, we trained a classifier on the *conv4* features of an AlexNet network trained on ImageNet [Russakovsky et al., 2015] to predict time periods on real images and applied it (i) to a validation set of previously unseen real images, and (ii) to images generated by the different methods.

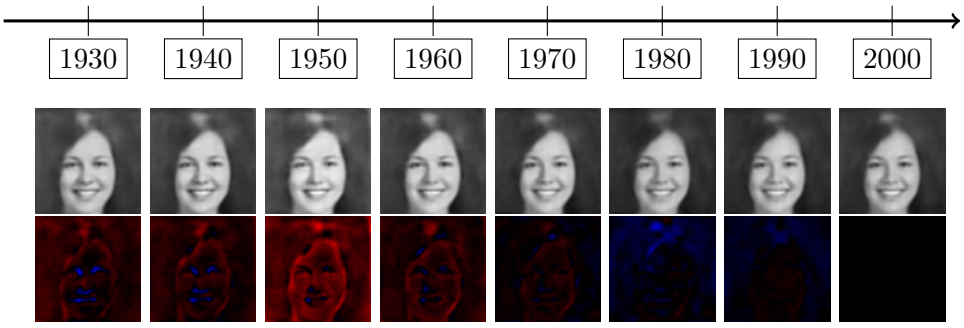
For each model and dataset, we report two numbers: (i) the percentage of correctly classified images in the case when the input image is translated into its ground truth time period ("Same year"), and (ii) the percentage of correctly classified images when the input image is translated to a different time period ("Modified year"). The first score evaluates whether the model preserves the temporal characteristic of the image when the time period is unchanged. The second score evaluates whether the model manages to change the time characteristics of the generated image in a way that fools the classifier. Higher numbers are better for both metrics; they can be interpreted as a success of the translation model to change the time period of the input image. The quantitative results are discussed in sections 6.3 and 6.4. In the following section, we present preliminary results using the bottleneck auto-encoder architecture, which served as a motivation for developing our bilinear factorization module.

6.2 Preliminary results with the bottleneck architecture

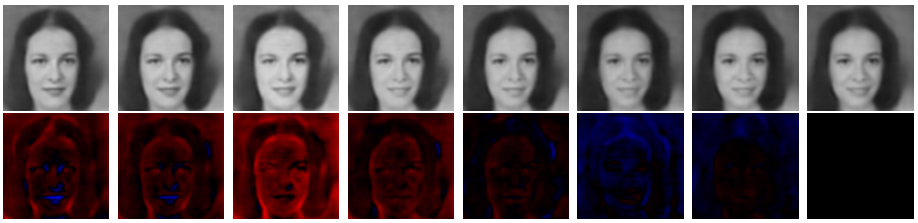
In this section, we describe the preliminary results obtained with the bottleneck auto-encoder architecture with a concatenation factorization module. We present results only on the easier face dataset, which contains relatively well aligned images. We found that this type of architecture was not able to learn image translation models on more varied image data such as the dataset of historical cars, where objects are not aligned and imaged from a variety of viewpoints and under different illuminations. These negative results served as the main motivation for developing the bilinear factorization module, which is evaluated in sections 6.3 and 6.4.

Architecture details. In this section, we use the bottleneck auto-encoder architecture described in section 5.1. The year embedding $\mathbf{y}$ is a continuous mapping of the time t, defined by:

$$y_i(t) = K_t \exp\left(\frac{(i - \alpha t)^2}{2\sigma^2}\right), \tag{6.1}$$

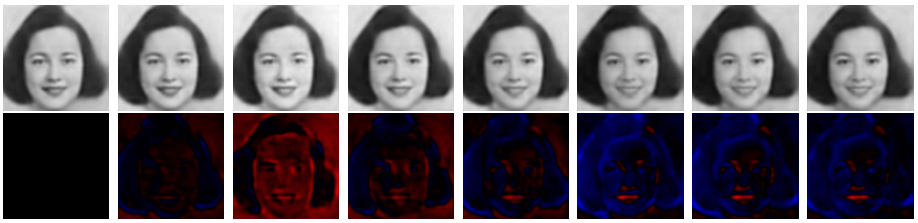


(a) Face image from the 2000s reconstructed at different time periods.



(b) Face image from the 2000s reconstructed at different time periods.

(c) Face image from the 1930s reconstructed at different time periods.



(d) Face image from the 1930s reconstructed at different time periods.

Figure (6.1): **Results of the bottleneck auto-encoder with concatenation factorization module.** For each face the top row shows the generated appearance over time and the bottom row shows the differences compared to the reconstruction at the true time period. Blue represents darker pixels and red represents lighter pixels in the reconstruction of the specific period compared to the true period. In these examples we can see that most of the difference is located around the lips.

where σ and α are hyper-parameters and K_t is a normalization factor set so that $\|\mathbf{y}(t)\|_2 = 1$. This mapping provides a soft discretization of continous time into bins representing different time periods. We use concatenation to combine the code $\mathbf{z}$ and the year embedding $\mathbf{y}$. Both $\mathbf{z}$ and $\mathbf{y}$ have a code

Network part	Encoder: $\mathbf{z} = f_\theta(A(I))$							Concat $(\mathbf{z}, \mathbf{y})$
layer type	conv	max	conv	max	conv	conv	conv	concat
kernel size	11	3	5	3	3	3	3	-
dimension	21	11	11	6	6	6	6	6
nb features	96	96	256	256	384	384	8	8 + 8
stride	4	2	1	2	1	1	1	1
batch norm	-	-	-	-	-	-	-	-
non-linearity	ReLu	-	ReLu	-	ReLu	ReLu	-	-

Network part	Decoder: $I_y = A^+\left(g_\phi(\mathbf{z}, \mathbf{y})\right)$									
layer type	conv	conv	up	conv	up	conv	up	conv	up	conv
kernel size	5	5	2	5	2	5	2	5	2	5
dimension	6	6	12	12	24	24	48	48	96	96
nb features	256	256	256	256	256	128	128	64	64	3
stride	1	1	1	1	1	1	1	1	1	1
batch norm	BN	BN	-	BN	-	BN	-	BN	-	-
non-linearity	lReLu	lReLu	-	lReLu	-	lReLu	-	lReLu	-	-

Table 6.1: Details of the auto-encoder architecture used in section 6.2.

size of 8. Details of this architecture are presented in table 6.1.

6.2.1 Visualizing differences over time

We first use our network to visualize trends over time from the perspective of a specific image. A first possible approach would be to simply look at the reconstructed images with the same content code $\mathbf{z}$ (the same person) for different time periods, as visualized in the top row for each person shown in figure 6.1. Looking at the images, the main observation is the change in the facial expression. As was previously noted in [Ginosar et al., 2015], people in later pictures tend to smile more.

6.2.2 Quantitative analysis of reconstruction errors

While in 6.2 we have shown temporal translation results for specific individuals, here we try to extract patterns by analyzing the reconstruction errors of translated images across decades. In figure 6.2 we show eight plots, one for each time period showing the reconstruction error over time for all 100 testing instances in that time period. Within each graph the curves are shifted so that the error is zero for the ground truth period. The colors of individual curves correspond to the specific years inside the decade – cold colors represent the early years and warm color the later years.

The first observation is that the evolution pattern of errors is consistent for all test samples inside each decade. For example, graph (a) shows that images from 1930s have overall low reconstruction error for the neighbouring 1940s but also 1960s and 1970s. The reconstruction error is higher in 1950s and 1980s. Also, the smallest errors are often attained for the ground truth decade.

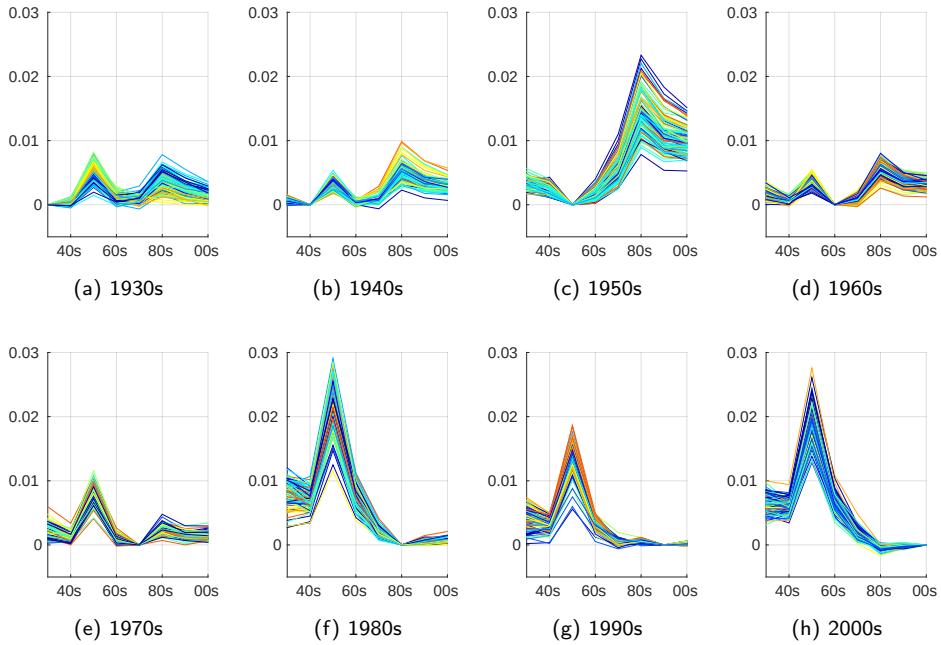


Figure (6.2): Reconstruction errors (y-axis) for different time periods (x-axis). See text.

Second, one can notice clear trends for the years inside the decade shown by consistent pattern of colors for the individual curves, despite the fact that the annotations used during training are exactly the same for all the years inside a decade. This effect is especially evident for the early decades. For example, inspecting graph (a) the images from early 30s encoded by blue and cyan colors have low reconstruction errors for 1950s but are not reconstructed well in 1980s. Interestingly, the reverse is true for the images of late 30s encoded by yellow and red colors that have relatively high reconstruction errors for 1950s bur are reconstructed relatively well in 1980s.

Finally, one can notice three clearly different groups. The first group is formed by 30s, 40s, 60s and 70s for which the errors are generally small but are slightly higher in the 50s and after (including) 80s. The second salient group is formed by the 80s, 90s and 2000s, which have a relatively high error for the earlier periods. Finally, the last group is the 50s, that consistently appeared as an outlier decade in this experiment.

6.2.3 Finding typical and non-typical examples

In this section we report results of visual data mining based on generated imagery. We generate a reconstruction of each test image for all time-periods and mine the test set for example faces that follow a specific temporal pattern in the reconstruction error. Results are shown in figure 6.3 and analyzed

(a) Test images for which the average reconstruction error across all time periods is the lowest.

(b) Test images for which the average reconstruction error across all time periods is the highest.

(c) Test images for which the range of the reconstruction errors across time is the lowest.

(d) Test images for which the range of the reconstruction error across time is the highest.

(e) Test images from the 2000s most different from the 1970s.

(f) Test images from the 1970s most different from the 2000s.

(g) Test images from the 1930s most different from the 1950s.

(h) Test images from the 1950s most different from the 1930s.

Figure (6.3): **Visual data mining based on generated imagery.** We generate a reconstruction of each test image for all time-periods and mine the test set for example faces that follow a specific temporal pattern in the reconstruction error. See discussion in section 6.2.3.

next.

Average MSE over time. One of the simplest statistics to look at is the mean square error of the reconstruction, that we can average over all time periods. Figures 6.3a and 6.3b show the images that have a small and large average MSE over time.

Range of reconstruction errors. To avoid the bias in the MSE error, we sort here images based on the range of the reconstruction error, i.e. the absolute difference between the reconstruction error in the best and the worst year. Intuitively, if the range is large, then the images are easy to reconstruct in one period but very hard to reconstruct in some other period. On the contrary, if the range of errors is small, the image is just as likely to come from any period. The results are shown in figures 6.3c and 6.3d. We observe that the images with the highest range of errors are the minorities (figure 6.3d) that are badly reconstructed using the early decades because they are absent from the yearbook collections, but are well reconstructed in the recent years. The least distinctive images (figure 6.3c) with low range of errors are faces without explicit features associating them to any specific year.

Comparison of specific decades. The above analysis mines for the most striking differences across the entire test collection. We can, however, use our method to pinpoint images with large differences in reconstruction errors between two specific decades. For example, figures 6.3e and 6.3f highlight the differences between the 1970s and the 2000s. These images highlight two dominant differences: there are much more girls that are smiling and have long hair in the 2000s (figure 6.3e), and much more girls with shorter, dark, but voluminous hair in the 1970s (figure 6.3f). There are some outliers, which is expected as we are mining eight images out of only 100 test images for a specific period. Similarly, we compare in figures 6.3g and 6.3h the 1930s and 1950s and find that faces from the 1930s, which are the most different from the 1950s are the ones with very curly well combed hair. On the other hand, the short hairstyle with more volume around the ears is characteristic of the 1950s compared to the 1930s.

6.2.4 Discussion

In this section we have reported results using the bottleneck autoencoder with concatenation factorization. While these results were promising, we found this architecture has only limited ability to capture appearance variation over time. While it could extract (some) temporal variation from the face dataset, where the faces are fairly well spatially aligned, this model was not able perform temporal translation for the car dataset (results not shown),

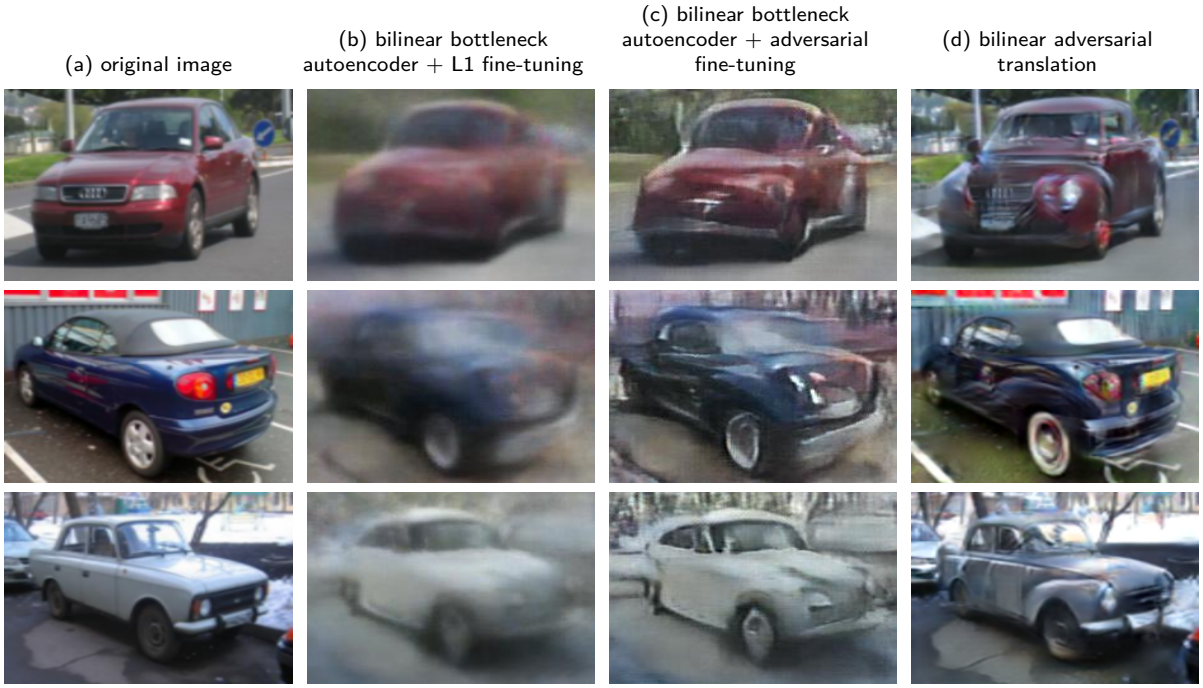


Figure (6.4): **Comparison of different architectures and losses.** From left to right: (a) original input image in the 1990s ; (b) translation into 1940s generated with a bilinear bottleneck autoencoder fine-tuned using only a $L1$ reconstruction loss produces rather blurry output ; (c) Fine-tuning using an additional adversarial loss produces sharper results ; (d) the best results are obtained using the bilinear adversarial translation model.

which has much larger variation of appearance due to intra-class variation, changes in viewpoint, background and illumination conditions. These results led us to develop the bilinear factorization module (chapter 4) which we employ in two different temporal translation architectures (as described in chapter 5. We report results for these models in the following sections.

6.3 Analysis of architectures and losses for image translation

In the previous section, we have shown preliminary results obtained with the bottleneck auto-encoder and the concatenation factorization module. While the results were promising on the aligned face dataset of yearbook portraits, this architecture was not able to capture the larger image variation in the more complex and non-aligned car dataset. These results served as the primary motivation for developing the bilinear factorization module (chapter 4) and incorporating this module in two different temporal translation architectures (chapter 5): (i) the bottleneck auto-encoder (the same as used in the previous section) as well as (ii) the recent adversarial image

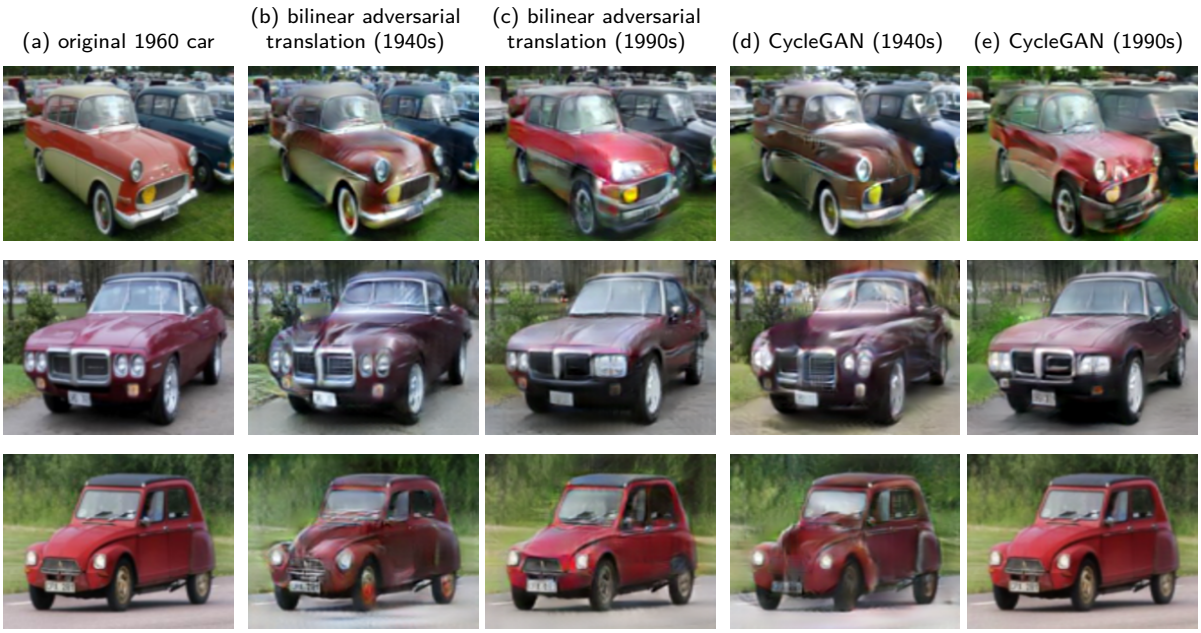


Figure (6.5): **Comparison between our bilinear adversarial translation model and CycleGAN [Zhu et al., 2017].** (a) The input image (1960s). (b)-(c) translation by our approach into 1940s and 1990s, respectively. (d)-(e) translation by CycleGAN into 1940s and 1990s, respectively. Both our bilinear adversarial translation and CycleGAN produce sharp images and clear changes over time with comparable visual quality of results. However, our bilinear adversarial translation architecture has the advantage of being a single model whereas a separate CycleGAN model needs to be trained for each target time period.

translation architecture [Zhu et al., 2017]. In this section we report we report experimental results of comparing these two architectures. We show that the bilinear factorization module makes it possible to learn temporal variation using the bottleneck auto-encoder even from the challenging car dataset as it builds factorization capability directly into the architecture. However, we further demonstrate that even better results can be obtained by inserting our bilinear factorization module in the recent adversarial image translation architecture [Zhu et al., 2017]. In this case our bilinear module enables image translation to multiple different target time periods using a single network. We begin with the analysis of visual quality of the results, then perform quantitative analysis and finally discuss the typical failure modes.

In the next section 6.4 we analyze the differences between the standard concatenation based factorization module and our new bilinear factorization module when used in the bottleneck auto-encoder. Finally, in section 6.3 we demonstrate how to apply our bilinear adversarial translation model for the task of visual discovery in an unpaired image collection annotated with time stamps.

Qualitative analysis. We focus our qualitative analysis on the more challenging car dataset. Qualitative results on the face dataset are shown in section 6.5 and on our project webpage [web]. First, in figure 6.4, we compare the results of translation of cars from the 1990s to the 1940s using our bilinear factorization module inserted into the bottleneck auto-encoder architecture without (b) and with (c) adversarial loss as well as inserted into the adversarial translation framework (d). The results in column (b) show that using only the $L1$ reconstruction loss with a bottleneck auto-encoder architecture results in blurry images that show some changes related to the time period but are hard to interpret. As shown in column (c), adding an adversarial loss to the bottleneck auto-encoder produces more realistic images with clearer differences between the time periods. However, there is still significant loss in image quality, due to the dimensionality reduction in the bottleneck. Finally, images produced with the adversarial translation (column (d)) are the most realistic and make changes over time easiest to interpret.

Second, in figure 6.5, we show a comparison between our bilinear adversarial translation model and CycleGAN [Zhu et al., 2017], which is the state-of-the-art model for image translation. For CycleGAN, we have trained three pairwise models, in order to cover transformations from and to each time period. Note that our bilinear adversarial model is only trained once with the three time periods. While the images are slightly different, the overall image quality and the temporal changes are similar. However, our bilinear adversarial translation architecture has the advantage of being a single model that is faster to train, as most of the weights of the model are shared. Note that the adversarial translation using a concatenation based factorization module produces similar quality of results and is not shown in the figure.

Quantitative analysis. The visual results are supported by the quantitative results reported in table 6.2. First, for both datasets, there is a clear gap between reconstruction errors for methods using adversarial translation and the bottleneck auto-encoder, with more than an order of magnitude difference. This confirms the visual results form figure 6.4. The reconstruction error for the bottleneck auto-encoder architecture is improving with the size of the bottleneck, but does not reach the quality of the result of the adversarial translation model. Second, the classification results, which are much higher than chance for all adversarial translation frameworks and for some bottleneck auto-encoder architectures demonstrate that both the auto-encoder and adversarial translation models can effectively change the appearance of the input image towards the target period. Finally, the joint results of the three independently trained pairwise CycleGAN models are similar to the results of our single bilinear adversarial translation model, which confirms the visual results from figure 6.5. The adversarial translation models, and

Figure (6.6): **The main failure modes of our bilinear adversarial translation model.** Each column shows one failure mode (unusual viewpoint, open hood, rare car type). In each column the top image shows the input (car from the 1940s, green border) and the bottom image shows the output translation into the 1990s (blue border). In all cases the model fails to modify the input image in a substantial way, but often still modifies some small parts, such as the wheels or the headlights.

in particular our bilinear adversarial translation model, show both a high classification score and a low reconstruction error. This makes these models good candidates to perform visual discovery in unpaired temporal image collections, as we will demonstrate in section 6.5.

Failure cases. Figure 6.6 shows several typical failure examples of our best performing approach, the bilinear adversarial translation model. In general, our method fails on uncommon images, where it typically produces a car that is nearly identical to the original input image, with sometimes localized changes on small parts such as the wheels. In other words, the model defaults to the original input image when it doesn't know how to modify the image correctly. These failure cases typically correspond to unusual viewpoints (figure 6.6(a)), open trucks or boots (figure 6.6(b)), or rare or otherwise unique cars in our dataset (figure 6.6(c)).

6.4 Comparison of bilinear and concatenation factorization modules

In this section, we analyze the differences between the standard concatenation based factorization module and our new bilinear factorization module. We focus this comparison only on the simple bottleneck auto-encoder architecture. In the more powerful framework of adversarial translation, both types of factorization modules produce similar results.

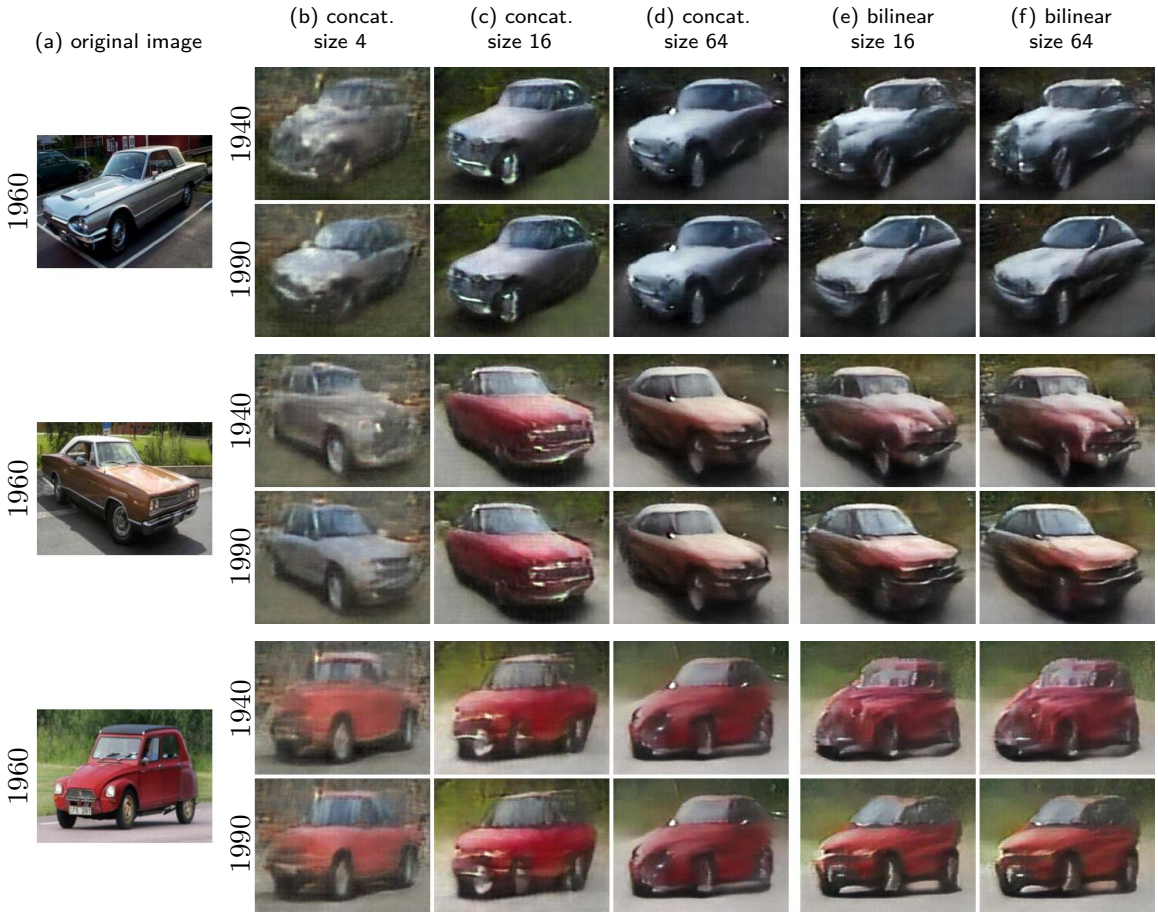


Figure (6.7): **Comparison of bilinear and concatenation modules in the bottleneck auto-encoder architecture.** In each example we show: (a) input original image (1960s); (b-d) concatenation-based translation to 1940s (top) and 1990s (bottom) using different code size of 4 (b), 16 (c) and 64 (d); (e-f) bilinear translation to 1940s (top) and 1990s (bottom) using code size of 16 (e) and 64 (f). Note that concatenation modules produce either limited temporal changes (larger code size) or low quality reconstruction (small code size). On the contrary, the bilinear factorization module (e-f) produces significant temporal variations with reasonable image quality for a range of code sizes.

	cars (3 periods)		
	Same year	Modif. year	$L1$ error
	Classification		Reconstruction
Real images	93%	*n.a.*	*n.a.*
Bilinear auto-encoder (64)	84%	69%	0.13
Bilinear auto-encoder (16)	86%	80%	0.16
Concat. auto-encoder (64)	55%	23%	0.13
Concat. auto-encoder (16)	64%	48%	0.16
Concat. auto-encoder (4)	64%	64%	0.20
Bilinear translation (ours)	83%	63%	0.017
Concat. translation	83%	63%	0.016
CycleGAN [Zhu et al., 2017]	76%	70%	0.015

(a) Quantitative evaluation for the cars dataset [Lee et al., 2013].

	faces (3 periods)		
	Same year	Modif. year	$L1$ error
	Classification		Reconstruction
Real images	96%	*n.a.*	*n.a.*
Bilinear auto-encoder (64)	95%	15%	0.08
Bilinear auto-encoder (16)	99%	90%	0.11
Concat. auto-encoder (64)	80%	11%	0.10
Concat. auto-encoder (16)	81%	9%	0.11
Concat. auto-encoder (4)	96%	81%	0.14
Bilinear translation (ours)	97%	91%	0.008
Concat. translation	98%	88%	0.009
CycleGAN [Zhu et al., 2017]	97%	87%	0.009

(b) Quantitative evaluation for the faces dataset [Ginosar et al., 2015].

Table 6.2: Quantitative evaluation of our models and baselines. We evaluate both a reconstruction error (normalized $L1$ loss, lower is better) and a "inception" type of score (percentage of correctly classified images, higher numbers are better).

Qualitative analysis. As in the previous section, we focus our qualitative analysis on the more challenging car dataset and provide results on the face dataset on the project webpage [web]. Qualitative translation results for different code sizes are shown in figure 6.7 and demonstrate clear differences between the concatenation-based and bilinear models. Indeed, for the code size 16 and 64, the image quality is similar for both types of factorization modules, but the concatenation based module produces almost no differences between the two target time periods, while very visible changes happen using the bilinear module. By reducing the code size of the concatenation based module further, for example to 4, it is possible to force the model to produce some changes, but only at the cost of very low quality images, where a large

part of the image identity has been lost. These results clearly demonstrate the advantage of the bilinear factorization module that we introduced and analyzed in chapter 4.

Quantitative analysis. The visual analysis is supported by the the quantitative results reported in table 6.2a. The reconstruction error is similar using both factorization modules with the same code size, but the classification scores are much higher, on both datasets, using our bilinear module. Obtaining higher classification scores using the concatenation-based model is possible only at the cost of strongly decreasing the code size, and thus reducing the image quality. However, even with a code size of 4, resulting in an important decrease of the reconstruction error, the classification scores of the concatenation module are still lower than the scores of our bilinear module. This demonstrates the clear advantage of our bilinear factorization module when employed in the bottleneck auto-encoder architecture.

6.5 Visual discovery via bilinear image translation

In this section we demonstrate how to apply our bilinear adversarial translation model for the task of visual discovery in an unpaired image collection annotated with time stamps. First, we train a bilinear translation model on the input image collection using the time stamp of each image as its (observed) time variable. Note that the model is able to extract temporal changes over time from the unpaired data, i.e. despite never seeing a particular object instance (e.g. a car or a face) evolve over time. Then, we apply the trained model on a new unseen test image. By varying the input time stamp we generate a history-lapse video that translates the input image into different time periods while preserving the identify of the depicted object. Finally, we analyze the generated image sequence to extract interesting changes in style over time.

History-lapse videos. Given a pre-trained bilinear adversarial translation model we generate a history-lapse for a new test image. This is achieved by varying the input time variable. In detail, using soft-assignment of the year y to define the time vector $\mathbf{y}$, we produce history-lapse videos with dense sampling of time. We generate videos at a sampling rate of 1 frame per year. Please see the videos on the project webpage [web]. Example frames are shown in figure 6.8. The videos reveal interesting changes of car style learnt from the entire dataset but applied on a specific instance of a car depicted from a given fixed viewpoint and captured in given imaging conditions. Note how some parts appear, disappear or are transformed over time. For example, the running board on the side of the car, characteristic for the 1940s gradually disappears over time; the round lights of the 1940s are transformed into

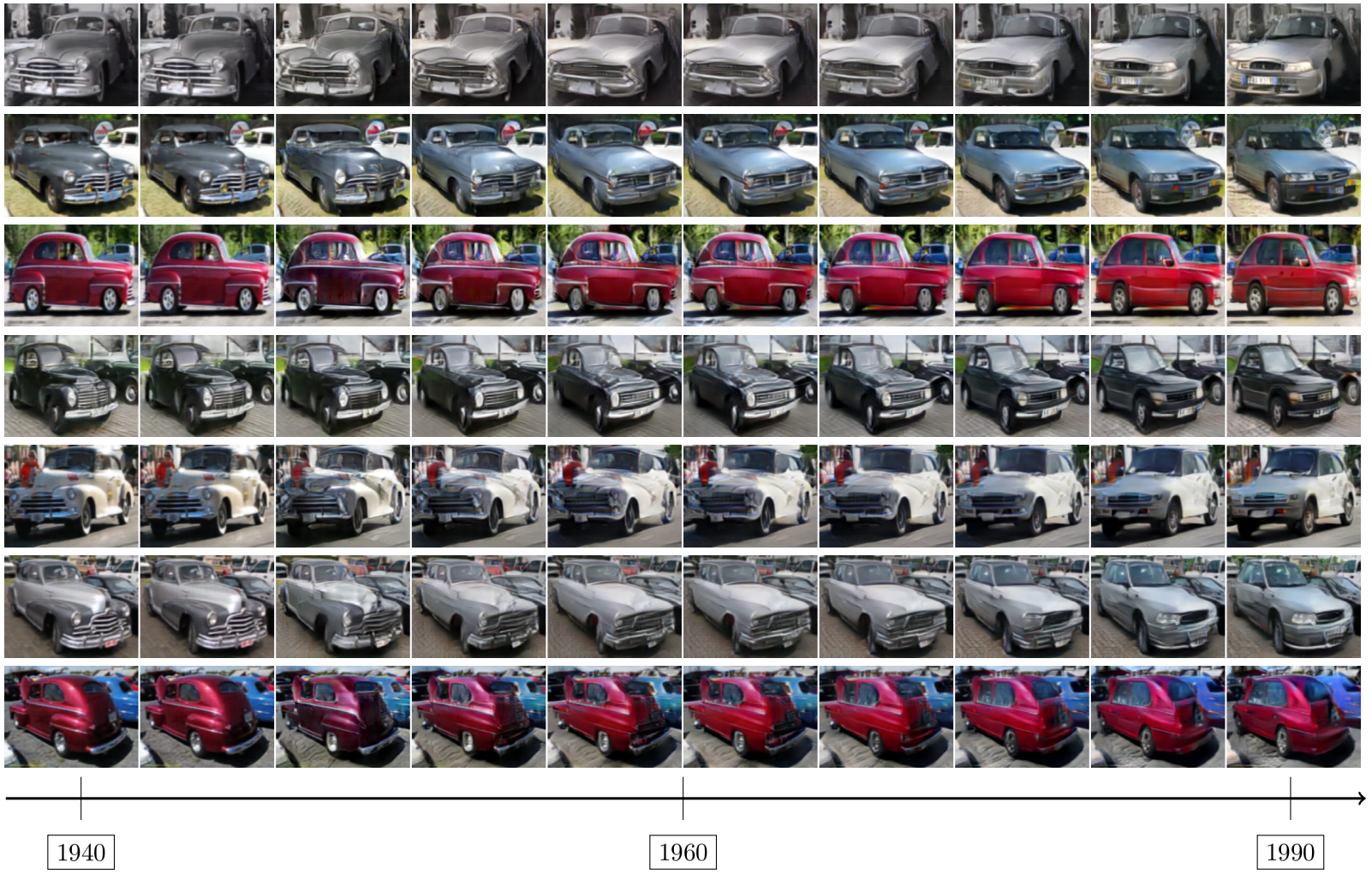


Figure (6.8): Examples of frames from history-lapse videos generated by our bilinear adversarial translation model. Here, input images of cars from the 1940s (left) are translated to different future times (left to right) by varying the input time variable. The output is a video showing changes of car style over time. Please note how the shape and appearance of the input cars changes over time. For example, the curved hood gradually fattens, the round lights become rectangular, the windscreen gets bigger, the front grille gets smaller and the running board on the side gradually disappears. Note also that other characteristics of the input car instances, such as viewpoint, color and the overall shape are preserved. History-lapse videos can be found on the project webpage [web].

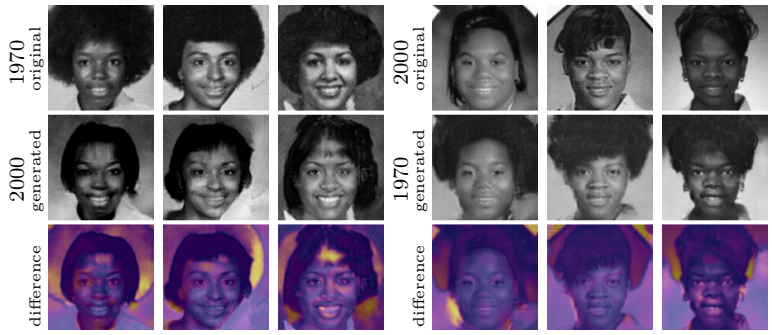
(a) Input images of cars from the 1960s (top row) and their generated translations into 1990s (middle row). The difference images (bottom row) highlight the changes in the headlights of the cars, going from small round ones to large rectangular ones, as well as the increasing size of the car windows and windshield.

(b) Input images of cars from the 1940s (top row) and their generated translations into 1990s (middle row). In addition to the changes in the shape of headlights and the size of windows, similar to (a), the difference images (bottom row) highlight the disappearing metallic bumpers as well as the evolution of the shape of the hood, which flattens over time.

(c) Input images of cars from the 1990s (top row) and their generated translations into 1940s (middle row). The difference images (bottom row) highlight the appearing running board (or footboard) as well as the changing appearance of wheels.

Figure (6.9): **Analyzing trends over time.** Each plate (a-c) shows consistent changes over time. In each plate, we show the original input images (top row), their translation into another period (middle row) and the absolute differences (bottom row) between the top and middle images. The absolute differences are shown as heatmaps where bright yellow color indicates the maximum absolute difference within the image, while dark blue colors correspond to small differences. We superimpose the heatmaps on a low contrast version of the original input image to further highlight the changes produced by our model. Please see more results on the project webpage [web].

(a) Input portraits from the 1940s (top row) and their generated translations into 2000s (middle row). The difference images (bottom row) reveal consistent changes in haircuts, with hair getting longer, and less curly in the 2000s.



(b) Input portraits from the 1970s (top row) and their generated translations into 2000s (middle row). The difference images (bottom row) reveal a particular trend in the hairstyle of African American women, who often had a voluminous afro haircut in the 1970s, which mostly disappeared in the 2000s.

(c) Input portraits from the 1940s (top row) and their generated translations into 2000s (middle row). In addition to changes in haircut, the difference images (bottom row) reveal changes in facial expression with bigger smiles showing more teeth in the 2000s as well as the color of the lips getting lighter, possibly because of changes in the popular lipstick color and use.

Figure (6.10): **Analyzing trends over time.** Each plate (a-c) shows consistent changes over time. In each plate, we show the original input images (top row), their translation into another period (middle row) and the absolute differences (bottom row) between the top and middle images. The absolute differences are shown as heatmaps where bright yellow color indicates the maximum absolute difference within the image, while dark blue colors correspond to small differences. We superimpose the heatmaps on a low contrast version of the original input image to further highlight the changes produced by our model. Please note consistent changes in hairstyle in plates (a) and (b), and smile in plate (c).

rectangular ones in the 1990s; or the windows, initially small, are enlarged over time. Note that the model learns to generate such changes without annotated part correspondence in the training data or seeing a specific object instance evolve over time in training. Please note also that such translation with a continuously changing attribute (here the time variable) would not be possible using the standard CycleGAN approach [Zhu et al., 2017].

Analyzing trends over time. An analysis of appearing, evolving and disappearing parts can be performed by looking at the differences between the images generated at the different time periods and the original input image. This way we can highlight the most important differences for a specific generated time period. Figure 6.9 shows such difference images for several examples from the car dataset. Please note the consistent changes over time. For example, the strong image differences (bright yellow color) at the base of windows and windshields indicate the evolution of their size over time, clearly showing the trend of having larger windows and windshields in later time periods. This trend is consistently exhibited in the input dataset on many different cars. Some elements are also characteristic for a specific period. For example, the running board on the side of the car is often present in the 1940s, but disappears (and hence is highlighted in bright yellow in the difference images) in the later periods. The wheels and headlights are also elements that have characteristic appearance at a specific time period and are often highlighted in the difference images.

Similarly, figure 6.10 reveals the trends in hairstyle and facial expression across time in women's yearbook portraits. More examples of difference images can be found on the project webpage [web].

Please note how the vocabulary of parts (wheels, headlights or windshields for cars and hair, eyes or mouth for faces) as well as their correspondence across different instances is learnt implicitly by our model from the unpaired training data. The outcome is that the model is able to synthesize appearance variation over time for a specific new input object instance despite changes in viewpoint or specific new appearance of the input car. Note also that our model enables *pixel-level* analysis of temporal trends by just comparing the synthesized time-lapse image sequences.

Chapter 7

Discussion

In this chapter, we summarize the contributions made in this thesis in section 7.1 and present possible extensions of our work in secion 7.2.

7.1 Contributions of the thesis

In this thesis, we have developed models for temporal image translation and used them for visual discovery.

In chapter 4, we have introduced a trainable bilinear factorization module that allows us to isolate and transfer time dependent appearance variations. We have articulated the close relation between our bilinear factorization module and principal component analysis. We have compared the bilinear module to other approaches such as the standard concatenation module. Finally, we have shown how to implement the bilinear factorization module in an efficient way to speed up the computation of the bilinear operation.

In chapter 5, we have introduced two different architectures to perform temporal image translation. We have discussed the bottleneck auto-encoder model, which reduces the size of the vector representing the image in order to translate it to different time periods. We have discussed the different losses including a separation loss that can be used to train this model. We have also incorporated our bilinear module in an adversarial translation model to perform temporal image translation.

In chapter 6, we have shown how we can apply our models for temporal analysis of unpaired image collections. We have discussed how a combination of a reconstruction loss and a classification loss were applied to evaluate the different models. We have applied the bottleneck auto-encoder model to translate simple images of faces and to find typical and atypical images in the dataset. We have shown how the bilinear module performs in different

architectures, and how it compares to the concatenation module. Finally, we have shown how our models can be used to identify and highlight the differences between time periods and how they can be used to create history-lapse videos.

7.2 Future work

In this section we discuss different directions for future work. We first discuss two direct extensions of our work: training the model with more time periods or more factors of variation. We then discuss evaluation of translation models and ideas for that could be applied to our models.

7.2.1 Training the translation model with more time periods

We trained the bottleneck auto-encoder with a bilinear module using the whole cars dataset, representing time with a continuous variable. We experienced difficulties doing the same with the domain adversarial translation model, and stabilized the training by using three time periods only. With more than 3 time periods, our current discriminator does not learn to distinguish between the periods, and hence does not provide a good domain adversarial loss for the generator.

Below we discuss a possible change in the training loss in order to improve the training of the discriminator in the domain adversarial translation model and thus enable training the model for a larger number of time periods. The model currently uses for the discriminator the square loss described in equation (5.18), and similarly, the domain-adversarial loss for the generator is the square loss as described in equation (5.17). Instead, one possibility would be to replace the square loss of the discriminator by a cross-entropy loss, as it is commonly used for classification. With a cross-entropy loss, a single discriminator D could be used, which would take as input an image I or a translated image $M(I, y)$ and output a vector of dimension $T + 1$, where T would be the number of time periods. The discriminator would be trained to output the correct time period for images from the dataset and identify translated images. The cross-entropy loss, replacing equation (5.18), would be given by:

$$\mathcal{L}_{\text{dom-disc}}(D) = -\mathop{\mathbb{E}}_{I} \sum_{1 \leq i \leq T} \left[y_{Ii} \log \left(D(I)_i \right) \right] - \beta \mathop{\mathbb{E}}_{I,y} \left[\log \left(D(M(I,y))_0 \right) \right] , \quad (7.1)$$

where D is the discriminator, M the generator, $M(I, y)$ the output of the translation of image I to time y, $\mathbf{y}_I \in \mathbb{R}^T$ the one hot vector corresponding to the time period of image I, and β a hyper-parameter weighting the importance of images translated by M and images from the datasets to

train the discriminator. Please note that the first term in equation (7.1) is encouraging the discriminator to classify the time period of real images, while the second term encourages the discriminator to identify images translated with the model M. As the cross-entropy loss is commonly used to train classifiers, it may help for training the discriminator which we found difficult to train when using many time periods. Similarly, the domain-adversarial loss would also change to the cross-entropy loss:

$$\mathcal{L}_{\text{dom-adv}}(M) = \mathop{\mathbb{E}}_{I,y} \sum_{1 \leq i \leq T} \left[y_{Ii} \log \left(D(I)_i \right) \right]. \tag{7.2}$$

The translation model would try to output images that the discriminator classifies as the target time period.

7.2.2 Multiple factors of variation

In this thesis we have focused on the evolution of images through time and introduced a trainable bilinear module to model relations between time, annotated in the datasets we use, and other factors of variations. A possible extension is to introduce other annotated factors of variation. For example, how do building facades vary with time across different locations? The bilinear module can be extended to a multilinear module, with multiplicative interactions between the different factors. This task would be challenging because the size of the parameter tensor would require the computation to be efficient in both memory and speed.

7.2.3 Metrics for quantitative evaluation

In section 6.1, we have discussed metrics to evaluate our models for image translation and proposed to use a combination of a classification loss to check if the image was correctly translated and a reconstruction loss to evaluate if identity can be preserved when the model does not perform translation. These metrics are imperfect as they do not evaluate if the identity is preserved when the model performs translation. Other approaches for evaluation can be considered.

A possible approach is to test our methods on a dataset for which other metrics are available. In [Dosovitskiy et al., 2016], images of cars and chairs are rendered from 3D models. They evaluate their methods of translating from one viewpoint to another by comparing their translations with the ground truth renders that are available for different views of the same object. The advantage of this method is that we could use a square loss to compare directly the translated image with our methods and a generated image of what the object looks like with a different attribute that serves as ground truth. While this metric can only be used on generated datasets, it would

allow us a direct evaluation what we want to achieve with our translation model. However, the square loss on pixels is not a direct metric for how humans evaluate proximity between images.

Another possible approach is to use human evaulators. Human perceptual studies have been used to assess the visual quality of generated or translated images. For example, in [Isola et al., 2017], Amazon Mechanical Turk (AMT) participants are asked to discern generated images from real images. In [Choi et al., 2018], AMT participants rank images from different methods for transferring attributes such as the hair color or the gender to images of faces. Similarly, human perceptual studies would help us evaluate and compare the different methods we discuss for temporal image translation. To achieve this, a possible solution would to ask annotators to rank translations of the same image at the same time period by different models, ranking best the translations that convincingly change the time period of the image while preserving its identity. It would also be possible to evaluate the quality of the image by presenting to participants sets of two images of the time period, one translated and one from the dataset, and ask them to identify them.

Bibliography

Pierre Baldi and Kurt Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural networks*, 2(1):53–58, 1989. 38

B. Cheung, J. Livezey, A. Bansal, and B. Olshausen. Discovering hidden factors of variation in deep networks. *arXiv preprint arXiv:1412.6583*, 2014. 25, 26

Yunjey Choi, Minje Choi, Munyoung Kim, Jung-Woo Ha, Sunghun Kim, and Jaegul Choo. Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8789–8797, 2018. 74

R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011. 38

E. J. Crowley and A. Zisserman. Of gods and goats: Weakly supervised learning of figurative art. In *Proceedings of the British Machine Vision Conference*, 2013. 9

N. Dalal and B Triggs. Histogram of Oriented Gradients for Human Detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 886–893, 2005. 28

Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977. 22

E. Denton, S. Chintala, R. Fergus, et al. Deep generative image models using a laplacian pyramid of adversarial networks. In *Advances in Neural Information Processing Systems*, 2015. 26

Chris Dijkshoorn, Lizzy Jongma, Lora Aroyo, Jacco Van Ossenbruggen, Guus Schreiber, Wesley ter Weele, and Jan Wielemaker. The rijksmuseum collection as linked data. *Semantic Web*, 9(2):221–230, 2018. 9

C. Doersch, S. Singh, A. Gupta, J. Sivic, and A. A. Efros. What makes paris look like paris? In *Proceedings of the ACM SIGGRAPH Conference on Computer Graphics*, 2012. 28, 29

C. Doersch, A. Gupta, and A. Efros. Mid-level visual element discovery as discriminative mode seeking. In *Advances in Neural Information Processing Systems*, 2013. 9

A. Dosovitskiy and T. Brox. Inverting convolutional networks with convolutional networks. *CoRR*, abs/1506.02753, 2015. URL `http://arxiv.org/abs/1506.02753`. 43

A. Dosovitskiy and T. Brox. Generating images with perceptual similarity metrics based on deep networks. *CoRR*, abs/1602.02644, 2016. URL `http://arxiv.org/abs/1602.02644`. 26

Alexey Dosovitskiy, Jost Springenberg, Maxim Tatarchenko, and Thomas Brox. Learning to generate chairs, tables and cars with convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2016. 36, 73

Akira Fukui, Dong Huk Park, Daylen Yang, Anna Rohrbach, Trevor Darrell, and Marcus Rohrbach. Multimodal compact bilinear pooling for visual question answering and visual grounding. *arXiv preprint arXiv:1606.01847*, 2016. 23

S. Ginosar, K. Rakelly, S. Sachs, B. Yin, and A. Efros. A century of portraits: A visual historical record of american high school yearbooks. *arXiv preprint arXiv:1511.02575*, 2015. 16, 52, 55, 64

Nicolas Gonthier, Yann Gousseau, Said Ladjal, and Olivier Bonfait. Weakly supervised object detection in artworks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 0–0, 2018. 9

I. Goodfellow, 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*, 2015. 26

K. Grauman and T. Darrell. Unsupervised learning of categories from sets of partially matching image features. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2006. 28

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 27, 48

G. E. Hinton, S. Osindero, and Y. W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006. 24

Geoffrey E Hinton, Peter Dayan, and Michael Revow. Modeling the manifolds of images of handwritten digits. *IEEE transactions on Neural Networks*, 8 (1):65–74, 1997. 21

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. URL `http://arxiv.org/abs/1502.03167`. 43

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017. 25, 52, 74

Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*, 2016. 26, 27, 48

L. Karlinsky, M. Dinerstein, and S. Ullman. Unsupervised feature optimization (ufo): Simultaneous selection of multiple features with their detection parameters. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2009. 28

D. P. Kingma and M. Welling. Auto-encoding variational Bayes. In *Proceedings of the International Conference on Learning Representations*, 2015. 24

Michael Kirby and Lawrence Sirovich. Application of the karhunen-loeve procedure for the characterization of human faces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(1):103–108, 1990. 20

A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1106–1114, 2012. 43

T. Kulkarni, W. Whitney, P. Kohli, and J. Tenenbaum. Deep convolutional inverse graphics network. In *Advances in Neural Information Processing Systems*, 2015. 25

N. Le Roux, N. Heess, J. Shotton, and J. Winn. Learning a generative model of images by factoring appearance and shape. *Neural Comput.*, 23(3): 593–650, March 2011. ISSN 0899-7667. doi: 10.1162/NECO_a_00086. URL `http://dx.doi.org/10.1162/NECO_a_00086`. 24

H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 609–616. ACM, 2009. 28

S. Lee, N. Maisonneuve, D. Crandall, A. Efros, and J. Sivic. Linking past to present: Discovering style in two centuries of architecture. In *International Conference on Computational Photography*, 2015. 9

Y. Lee, A. Efros, and M. Hebert. Style-aware mid-level representation for discovering visual connections in space and time. In *Proceedings of the International Conference on Computer Vision*, 2013. 16, 30, 52, 64

Tsung-Yu Lin, Aruni RoyChowdhury, and Subhransu Maji. Bilinear cnn models for fine-grained visual recognition. In *Proceedings of the International Conference on Computer Vision*, pages 1449–1457, 2015. 23

D. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the 7th International Conference on Computer Vision, Kerkyra, Greece*, pages 1150–1157, September 1999. 28

Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2794–2802, 2017. 44

D. Marcus, A. Fotenos, J. Csernansky, Morris J., and R. Buckner. Open access series of imaging studies (oasis): Longitudinal mri data in nondemented and demented older adults. *Journal of Cognitive Neuroscience*, 22(12): 2677–2684, 2010. 9

R. Martin-Brualla, D. Gallup, and S. Seitz. Time-lapse mining from internet photos. *ACM Trans. Graph.*, 34(4), 2015. 27

K. Matzen and N. Snavely. Scene chronology. In *Proceedings of the European Conference on Computer Vision*, 2014. 28

Kevin Matzen and Noah Snavely. Bubblenet: Foveated imaging for visual discovery. In *Proceedings of the International Conference on Computer Vision*, 2015. 29, 30

J.-B. Michel, Y. Shen, A. Aiden, A. Veres, M. Gray, et al. Quantitative analysis of culture using millions of digitized books. *Science*, 331(6014), 2011. 9

S. Osindero and G. E. Hinton. Modeling image patches with a directed hierarchy of Markov random fields. In *Advances in Neural Information Processing Systems*, 2008. 24

Karl Pearson. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901. 19

T. Quack, B. Leibe, and L. Van Gool. World-scale mining of objects and events from community photo collections. In *Proceedings of the ACM International Conference on Image and Video Retrieval*, 2008. 27

A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *Proceedings of the International Conference on Learning Representations*, 2016. 26

S. Reed, K. Sohn, Y. Zhang, and H. Lee. Learning to disentangle factors of variation with manifold interaction. In *Proceedings of the International Conference on Machine Learning*, 2014. 25, 52

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015. 52

D. Roy et al. The human speechome project. *Symbol Grounding and Beyond*, pages 192–196, 2006. 9

O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, S. Huang, A. Karpathy, A. Khosla, M. Bernstein, A.C. Berg, and F.F. Li. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 2015. 43, 53

R. Salakhutdinov and G. E. Hinton. Deep Boltzmann machines. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2008. 24

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training GANs. In *Advances in Neural Information Processing Systems*, 2016. 53

Terence D Sanger. Optimal unsupervised learning in a single-layer linear feedforward neural network. *Neural networks*, 2(6):459–473, 1989. 38

Xi Shen, Alexei A. Efros, and Mathieu Aubry. Discovering visual patterns in art collections with spatially-consistent feature learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 9

S. Singh, A. Gupta, and A. A. Efros. Unsupervised discovery of mid-level discriminative patches. In *Proceedings of the European Conference on Computer Vision*, 2012. 28

J. Sivic and A. Zisserman. Video data mining using configurations of viewpoint invariant regions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Washington, DC*, 2004. 28

J. Sivic, B. C. Russell, A. A. Efros, A. Zisserman, and W. T. Freeman. Discovering objects and their location in images. In *Proceedings of the 10th International Conference on Computer Vision, Beijing, China*, 2005. 28

An Tang and et al. Canadian association of radiologists white paper on artificial intelligence in radiology. *Canadian Association of Radiologists Journal*, 69(2):120 – 135, 2018. ISSN 0846-5371. doi: https://doi.org/10.1016/j.carj.2018.02.002. URL `http://www.sciencedirect.com/science/article/pii/S0846537118300305`. 9

Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. Multi-view 3d models from single images with a convolutional network. In *Proceedings of the European Conference on Computer Vision*, pages 322–337. Springer, 2016. 24, 25

Joshua B Tenenbaum and William T Freeman. Separating style and content with bilinear models. *Neural computation*, 12(6):1247–1283, 2000. 22, 23, 38, 41

The Visual Arts Data Service. the visual arts data service. `https://www.vads.ac.uk/index.php`. 9

The Watermark Project. The watermark project. `https://filigranes.hypotheses.org/the-project`. 9

Matthew A Turk and Alex P Pentland. Face recognition using eigenfaces. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 586–591. IEEE, 1991. 21

M Alex O Vasilescu and Demetri Terzopoulos. Multilinear analysis of image ensembles: Tensorfaces. In *Proceedings of the European Conference on Computer Vision*, pages 447–460. Springer, 2002. 22

P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the International Conference on Machine Learning*, 2008. 24

Xiaolong Wang and Abhinav Gupta. Generative image modeling using style and structure adversarial networks. In *Proceedings of the European Conference on Computer Vision*, 2016. 26, 53

web. The project webpage. `https://www.di.ens.fr/willow/research/bilineartranslation`. 61, 64, 65, 66, 67, 69

X. Yan, J. Yang, K. Sohn, and H. Lee. Attribute2Image: Conditional image generation from visual attributes. *Proceedings of the European Conference on Computer Vision*, 2016. URL `http://arxiv.org/abs/1512.00570`. 24, 52

M-H Yang, Narendra Ahuja, and David Kriegman. Face detection using a mixture of factor analyzers. In *Image Processing, 1999. ICIP 99. Proceedings. 1999 International Conference on*, volume 3, pages 612–616. IEEE, 1999. 22

Richard Zhang, Phillip Isola, and Alexei A Efros. Colorful image colorization. In *Proceedings of the European Conference on Computer Vision*, 2016. 53

J.-Y. Zhu, P. Krahenbuhl, E. Shechtman, and A. A. Efros. Generative visual manipulation on the natural image manifold. In *Proceedings of the European Conference on Computer Vision*, 2016. 26

J.-Y. Zhu, T. Park, P. Isola, and A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the International Conference on Computer Vision*, 2017. 27, 41, 45, 47, 48, 49, 50, 60, 61, 64, 69

RÉSUMÉ

L'objectif de cette thèse est de développer des outils pour analyser les collections d'images temporelles afin d'identifier et de mettre en évidence les tendances visuelles à travers le temps. Cette thèse propose une approche pour l'analyse de données visuelles non appariées annotées avec le temps en générant à quoi auraient ressemblé les images si elles avaient été d'époques différentes. Pour isoler et transférer les variations d'apparence dépendantes du temps, nous introduisons un nouveau module bilinéaire de séparation de facteurs qui peut être entraîné. Nous analysons sa relation avec les représentations factorisées classiques et les auto-encodeurs basés sur la concaténation. Nous montrons que ce nouveau module présente des avantages par rapport à un module standard de concaténation lorsqu'il est utilisé dans une architecture de réseau de neurones convolutionnel encodeur-décodeur à goulot. Nous montrons également qu'il peut être inséré dans une architecture récente de traduction d'images à adversaire, permettant la transformation d'images à différentes périodes de temps cibles en utilisant un seul réseau.

MOTS CLÉS

Apprentissage, Découverte Visuelle, Vision par Ordinateur

ABSTRACT

This thesis proposes an approach for analyzing unpaired visual data annotated with time stamps by generating how images would have looked like if they were from different times. To isolate and transfer time dependent appearance variations, we introduce a new trainable bilinear factor separation module. We analyze its relation to classical factored representations and concatenation-based auto-encoders. We demonstrate this new module has clear advantages compared to standard concatenation when used in a bottleneck encoder-decoder convolutional neural network architecture. We also show that it can be inserted in a recent adversarial image translation architecture, enabling the image transformation to multiple different target time periods using a single network.

KEYWORDS

Machine Learning, Computer Vision, Visual Discovery