On Several Mathematical and Data-Driven Models for Image and Video Editing, Synthesis and Analysis

Alasdair Newson

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 20XXIPPAXXXX

Habilitation à diriger les recherches

TELECOM Paris

IP PARIS

On Several Mathematical and Data-Driven Models for Image and Video Editing, Synthesis and Analysis

Thèse d'habilitation à diriger les recherches de l'Institut Polytechnique de Paris
préparée à Télécom Paris

École doctorale n°626 École doctorale de l'Institut Polytechnique de Paris (EDIPP)
Spécialité : Informatique, Données et IA

Thèse présentée et soutenue à Palaiseau, le 15 Février 2023, par

ALASDAIR NEWSON

Composition du Jury :

Isabelle Bloch Professeure, Sorbonne Université	Examinatrice
Matthieu Cord Professeur, Sorbonne Université	Examinateur
Agnès Desolneux Directrice de Recherche, Centre Borelli	Rapporteuse
Olivier Lemeur Principal Scientist, Interdigital	Rapporteur
Pablo Musé Full Professor, Universidad de la República, Montévidéo (Uruguay)	Examinateur
Nicolas Papadakis Directeur de Recherche, Institut Mathématique de Bordeaux	Rapporteur

Contents

Acknowledgements :

I would first of all like to thank the members of the HDR commitee, the referees in particular, for reviewing this manuscript. This is a big job for people with extremely busy schedules, and I would like to express my sincere thanks.

I would like to thank some of the people who have been influential in my research career, starting with Yann Gousseau, Andrés Almansa and Patrick Pérez, my former PhD advisors. They have always supported me and given me advice, especially in the previous years of the Covid pandemic, when such help was so important. I cannot thank them enough for all their scientific and human qualities. My thanks also go to the many amazing people I have had the great fortune to work with in my career: Guillermo Saprio, Mariano Tepper, Qiang Qiu, Julie Delon, Bruno Galerne, Noura Faraj, Saïd Ladjal, Florence Tupin, Chi-Hieu Pham, Yann Traonmilin, Lara Raad and others. One of the main aspects of the research community which I like are the people who work there, so they are in no small part responsible for my being a part of it. I would like to thank the IMAGES team of the IDS department of Télécom Paris, who make work life a real pleasure. Thank you.

Chapter 1

Introduction

The objective of this manuscript is to give an overview of the research I have carried out since the end of my PhD thesis. The subject of this research is the restoration, editing and analysis of images and videos. I will note straight away that this does not represent all of my research work, as some subjects are starting/ongoing while others are further away from the central themes of this manuscript. I give an exhaustive list of all my publications at the beginning of the bibliography 6. Much of my work has been geared towards artistic or aesthetic goals, such as film post-production and image editing for personal purposes. However, what has interested me most of all during my research career has been the different image models that I have had the opportunity to use. Furthermore, my research can be roughly organised with respect to the model which was being used in each work. Given this twin motivation, I will place a particular emphasis on the notion of models in both the explanation and categorisation of the work in this manuscript.

I have a third reason for doing this. It is clear that we are at a very particular point in the development of image processing, which is due to the radically new paradigm that has turned the domain upside-down (I do not think I am exaggerating here) for the past nine years, since 2012 in fact. This paradigm is, of course, the explosion of deep learning, which has overtaken in terms of performance almost every other model in this research domain, and indeed in many others. The expression "turn upside-down" is quite relevant here. In the deep learning paradigm, rather than establishing a model a priori, that is to say using our reason to decide which properties an image should exhibit, we instead learn the model by observing examples of images in a database. In reality, there *is* an a priori in the deep learning approach, which is that a good model can be constructed using concatenations of simple operations such as matrix-vector multiplications, convolutions and non-linear functions (the structure of a neural network), but this gives rise to such a flexible class of models [55] that we can consider that the model is not a priori.

We have, therefore, at this point in time, an opposition between the "classical" techniques of image processing and computer vision, and deep learning, with the latter winning consistently in terms of performance. However, as everyone who has worked with deep learning knows, it can be extremely frustrating to use due its to lack of interpretability, and the immense difficulty in figuring out how the model will respond using theoretical analysis. Even my master's degree students first discovering deep learning point this out very quickly, showing that it is not simply researchers who find this state of affairs uncomfortable. This is to say nothing of its use in critical applications such as medical imagery or autonomous vehicles, which require a high degree of confidence in the performance. Thus, deep learning clearly has its limitations, and this is fundamentally due to the fact that there is *no a priori model*. Consequently, including models and some interpretability in deep learning approaches will obviously be a big goal of future research in image processing.

Part of my recent and curent works have been carried out with this in mind (see Section 4.2 and Section 4.6).

1.1 Image Processing Models

Since its advent in the 1980s, one of the main goals of image processing has been the search for a flexible, robust and powerful model to represent images. Some of the tools used to establish these models include partial differential equations, the total variation, patches, gaussian mixture models and deep neural networks. Please note that in the rest of the manuscript, I will refer to neural networks as models, even if I have just introduced the opposition between "model-based" and "data-based" approaches. Neural networks are indeed models, but they are models based on empirical observations of data.

As mentioned above, exploring the different models and their implications on how we consider images is the aspect of image processing which I find most interesting. In my research, I have used several of these models for the tasks of image restoration, editing and analysis. These are :

- Patch-based models, ie the model of image auto-similarity (during my PhD)
- Low-rank models
- Stochastic models
- Neural networks

These models are naturally more or less suitable for different applications. The main ones are, in the same order as the models above:

- Image and video inpainting
- Background estimation
- Analog film grain synthesis
- Image editing

I will use this organisation throughout the manuscript to present my work. The first model I have listed is the patch-based model. I will not discuss this in the present manuscript, since most of my work on them was done during my PhD. Nevertheless, I give a brief summary, as they will potentially play a role in future work.

Patch-based models Patch-based models are founded on the following idea: local squares or rectangles of image information are extremely useful as local descriptors of the image. Another way of putting this is that a good a priori of an image is the image itself. For example, in the case of the Non-Local Means denoising algorithm [39], a pixel is denoised by finding different "samples" of that pixel in different regions of the image, and averaging those versions. The average is weighted using the patch distance associated with each sample of the pixel. By averaging these weighted samples, we obtain a robust estimation of the pixel's value. In this sense, the image itself provides an a priori for denoising. Actually, with respect to the discussion of a priori models vs the empirical approach, patch-based methods are in a sense between these two paradigms. There is a model, which leads to an energy to minimise, however this is based on an observed database,

ie the patches of the image. Therefore, patches are still of great relevance to image processing and are indeed used in deep learning approaches (the patchGAN [93] for example). In fact, they potentially provide a way to reconcile deep learning with more traditional models. I will discuss this at the end of this manuscript (see Chapter 5).

Low-rank models The first model which I discuss in this manuscript is the low-rank model. I used it during a year's postdoc with Duke University in the team of Guillermo Sapiro. This model, introduced by Candés *et al*. [42], is designed for very specific tasks, the main two being face identification and background estimation. I employed it for the second goal, background estimation (also called background subtraction).

The main idea behind the low-rank model, in the case of background estimation, is that a video, put into matrix form, may be modelled as the sum of a low-rank matrix (the background), a sparse matrix (the foreground), and some noise. More precisely, let $\mathbf{X} \in \mathbb{R}^{m \times n}$ be an observed video, where m is the number of pixels in one frame, and n is the number of frames, which we want to separate into foreground and background. Each column of $\mathbf{X}$ contains the grey-level information of an image which has been linearised (or "flattened"). The low-rank model consists in decomposing $\mathbf{X}$ as:

$$\mathbf{X} \approx \mathbf{L} + \mathbf{S}, \tag{1.1}$$

where $\mathbf{L}$ is a low-rank matrix and $\mathbf{S}$ is a sparse matrix. We wish $\mathbf{L}$ to be low-rank because each column can be written as a linear combination of a small number of vectors, which represent components of the background. Ideally, we would have only one background image, which would lead to a rank 1 matrix $\mathbf{L}$, however the more general low-rank formulation gives a degree of flexibility to the model. The foreground $\mathbf{S}$ is considered to be sparse since it can be assumed that there is much more background than foreground. In cases where this is not true, it becomes difficult to properly define background. The noise appears in the difference $\mathbf{X} - (\mathbf{L} + \mathbf{S})$. The model is integrated into the following minimisation problem:

$$\min_{\mathbf{X},\mathbf{S}} \|\mathbf{X} - \mathbf{L} - \mathbf{S}\|_2^2 + \lambda_* \mathrm{rank}(\mathbf{L}) + \lambda \|\mathbf{S}\|_1. \tag{1.2}$$

Unfortunately, this problem is non-convex due to the presence of the rank of $\mathbf{L}$. Candès and Tao propose to reformulate this, giving rise to Robust Principal Component Analysis problem:

$$\min_{\mathbf{X},\mathbf{S}} \|\mathbf{X} - \mathbf{L} - \mathbf{S}\|_2^2 + \lambda_* \|\mathbf{L}\|_* + \lambda \|\mathbf{S}\|_1, \tag{1.3}$$

where $\|\cdot\|_*$ is the *nuclear norm* of a matrix, which is a convex surrogate of the rank.

In the work presented in this Chapter 2 we propose two algorithms which use the low-rank model:

- an algorithm to detect foregrounds at several timescales, meaning elements in a video which could be considered to be foreground or background depending on the timescale we are considering;
- a video segmentation algorithm to determine spatio-temporal regions where the low-rank model applies well.

Indeed, one of the weaknesses of the low-rank model is poor robustness to locally-changing lighting conditions, which happens quite often in videos, for example when a directed lamp is turned on. It is also not reasonable to apply the same low-rank background throughout a potentially long video. These algorithms were published in [7, 8].

Stochastic models Stochastic models, in a broad sense, correspond to any model which includes a random variable. This includes most models of degradation addressed by inverse problems (denoising etc.), but also texture models. In texture models, a pixel is considered to be a realisation of a random variable, and the goal is of course to determine the random distribution which leads to the best visual results. In this manuscript, I am interested in creating a specific type of texture, that of film grain. This is the texture due to the physical nature of the most common photographic process (silver halide photography) that existed prior to digital images. The grains are crystals of silver halide which solidify when light hits them and they are subsequently developed with a chemical compound. Thus, the final photographic image (once the negative image has been reversed) is consituted of light where the grains have solidified and dark where they have not. These grains give rise to the texture which is desired by photographers and film-makers for artistic purposes. Consequently, the goal of this work is to create an algorithm which can add synthetic film grain to an input digital image.

In the work I present here on the subject, we chose to do this using a model from the stochastic geometry literature, called the Boolean model. Stochastic geometry [164] is interested in describing the (geometric) properties of random processes which result from objects being randomly distributed in an n-dimensional space. For example, this could be balls of random sizes being randomly scattered on a 2D plane, and this indeed corresponds to the Boolean model. Another example would be the similar situation in 3D, which might correspond to the disposition of coffee grains in a coffee press. It is clear that the Boolean model corresponds quite nicely to the situation found in film grain. Thus we have used it here. More precisely, let $\Phi = \{x_i,\ i \in \mathbb{N}\}$ represent a Poisson point process on $\mathbb{R}^2$ with intensity λ. In other words, λ represents the average number of grains in a unit square. These x_i represent the centers of our silver halide grains. We also define a sequence of identically and independently distributed (i.i.d.) random compact sets in $\mathbb{R}^2$, $X_0, X_1, \ldots$, which will represent the grain shapes. Thus, Boolean model is the random set Z defined as the union of all the shapes X_i placed at the locations x_i, that is,

$$Z = \bigcup_{i \in \mathbb{N}} (X_i + x_i). \tag{1.4}$$

In our case, the grain shapes are balls, of possibly random radii. The key question, at this point, is how to fix the intensity λ of the Poisson point process. Indeed, the greater the intensity, the brighter the final image will be (remember, more grains means a lighter image). In this work, we set the intensity to respect the local grey-level of the input digital image. More precisely, given an input image u, and a pixel y, λ is set locally to:

$$\lambda(y) = \frac{1}{\mathbb{E}[A_1]} \log\left(\frac{1}{1-u(y)}\right), \tag{1.5}$$

where $\mathbb{E}[A_1]$ is the expected value of the area of a grain. This determines the Boolean model completely, and subsequently it can be sampled, meaning for each output pixel, we find if a grain covers it or not. However, we note that this would lead to a binary image (each pixel is covered or not), which is not the reality we percieve when viewing a silver halide image. In fact, there are two reasons we see a grey-level image and not a binary image:

- The film grain image gets blurred due to the process of inverting the image and potentially printing it onto paper;
- Our eyes cannot see with infinite precision.

In both cases, some sort of blurring process is at work. Thus, we define the output v of the *filtered* Boolean model, with filter ϕ as:

$$v(y) = (\phi * \mathbb{1}_Z)(y). \tag{1.6}$$

The filter ϕ is chosen to be a Gaussian filter. Now, even if the model is defined, it is not a simple task to evaluate it for a given sample of the Boolean model. Indeed, it involves the convolution of the indicator function of a random set with a filter. To evaluate it, we use a Monte Carlo simulation. Let us consider a series of iid random variables $\{\xi_i, i = 1 \dots N\}$ where each $\xi_i \sim \mathcal{N}(y, \sigma^2)$. Due to the law of large numbers, we have:

$$\frac{1}{N}\sum_{k=1}^{N} \mathbb{1}_Z(\xi_k) \xrightarrow[N\to+\infty]{} \mathbb{E}(\mathbb{1}_Z(\xi_1)) = \int_{\mathbb{R}^2} \mathbb{1}_Z(t)\phi(y-t)dt. \tag{1.7}$$

Thus, we can evaluate the filtered Boolean model approximately at any chosen position y. One interesting aspect of this model is that the output position y is completely arbitrary: we can evaluate the model with as high a resolution as we would like. This is important because the best film grain results are given with high resolutions. This approach was published in two journal publications [1, 2].

A drawback of the previous approach is that it is quite computationally expensive to evaluate. Thus, we also investigated a simplified version, where the film grain texture was considered to be a Gaussian texture. A Gaussian texture is a stochastic process where each pixel follows a multivariate Gaussian distribution, of average $\hat{\mathbf{u}}$ and of covariance $\mathbf{C}$. If we find the Cholesky decomposition of $\mathbf{C}$, which is positive semidefinite, such that $\mathbf{C} = \mathbf{L}\mathbf{L}^T$, then we can easily produce a realisation of this texture with the following equation:

$$\mathbf{Y} = \hat{\mathbf{u}} + \mathbf{L}\mathbf{X}, \tag{1.8}$$

where $\mathbf{X}$ is a multivariate normal random vector. Thus, the key question is how to determine $\hat{\mathbf{u}}$ and $\mathbf{C}$. We do this by a theoretical analysis of the Boolean model and the filtered Boolean model. Again, we turn to a Monte Carlo approach to evaluate both $\hat{\mathbf{u}}$ and $\mathbf{C}$. This was published in an international conference [9].

Deep learning models Deep learning models refer to *neural networks* which have many layers[1] to solve a wide variety of problems in statistics, signal and image processing and other domains. Neural networks are basically parametric functions that generally have a large number of parameters. Their power lies in the fact that they can approximate a very large class [55] of unknown, non-parametric functions, which are only observable through an annotated dataset. The parameters are determined by adjusting them during a *training* process to achieve some goal on the dataset. This training process involves an objective (or loss) function to minimise. The most basic example of such a goal is classification, but they have been modified to address a wide range of problems.

In this manuscript, we will be interested in a special kind of deep learning models, deep generative models. These are models which can be used to generate data of many different types, for example images of a certain type (faces, landscapes etc.). We will look at two types of generative models, the autoencoder and the Generative Adversarial Network. Strictly speaking, the autoencoder is not a generative model, however it can be modified to be one and its literature is closely

[1] we will define these terms more rigourously later on

linked with that of deep generative models, so we will make the abuse of nomenclature here. Autoencoders are neural networks which consist of two sub-networks: an encoder and a decoder. The first projects input data to a lower-dimensional space, known as the latent space, while the second projects the data back to the input data space. It is trained for the output to be as similar as possible to the input. Thus, the latent space is a more compact, and interesting/powerful, representation of the data. The underlying goal of the works presented in this Chapter 4 will be the understanding and, ultimately, editing of images using deep generative models.

The first work I will discuss will be an analysis of how an autoencoder can encode and decode simple shapes, in particular binary images of disks. The encoding process is achieved by extracting the area of the disk, while the decoding is more complicated. We show that in the case of the decoder with no biases, a general shape is learned by the decoder, and this is multiplied by the latent code to produce an image which can then be thresholded to achieve the disk. We also study how regularisation can address incomplete data, that is to say databases with certain disk radii missing. Finally, we study how position can be encoded in an autoencoder. This work was published in the Journal of Mathematical Imaging and Vision (JMIV) [3].

The next two works show how deep generative models can be used for image editing. The first proposes an autoencoder-type network which can take an input facial image and edit it to modify the age of the person. The age is encoded as a one-hot vector (in other words a delta impulse vector), so a range of ages can be obtained (not just age or de-age). This work was published in the International Conference of Pattern Recognition (ICPR) 2020 [10]. The second work takes another approach to image editing. In this approach, we use a Generative Adversarial Network (GAN) *after* it has been trained to synthesise images. GANs can be seen as networks with only the second half of the autoencoder architecture (the decoder), where the latent space is distributed following a multivariate normal distribution. Thus, synthesis of image data is simple: draw a random vector in the latent space and decode. We use this to our advantage for image editing; starting with an initial point corresponding to an image which we want to modify, we move around in the latent space until we have achieved our editing objective. Just how this moving around is done is the subject of this work. This approach was published in the International Conference of Computer Vision (ICCV) 2021 [11]. The two previous algorithms were part of the work of the CIFRE PhD of Xu Yao, with Interdigital.

The final work on deep learning models concerns a new autoencoder architecture and a loss function to imitate the behaviour of Principal Component Analysis (PCA). PCA is used very often to analyse and understand data. It is a linear transformation which changes the representation of data such that each axis is statistically independent and ordered according to decreasing variance of the data. We achieve this by:

- Progressively increasing and freezing the latent components of the autoencoder;
- Using a covariance loss function to ensure that the latent components are statistically independent.

We first show how this can be used to edit images of parametric shapes, and then how it can be applied to powerful Generative Adversarial Networks for more general image editing. This work was published in JMIV [4], and was part of the postdoc of Chi-Hieu Pham.

Chapter 2

Low Rank models for Background Estimation

2.1 Low-rank models for background estimation

In this Chapter, we discuss the low-rank model for the application of background estimation. Let us stop to note that background estimation is also known as background subtraction and foreground/background separation. In any case, the main goal is to find an image which represents the background, and potentially the residual which represents the foreground. For some context, the original motivation for this work, which I carried out during my postdoc at Duke University, was the detection of abandoned luggage in train stations. This has some bearing on the algorithms developped, as we shall see further on. My work in this Chapter was published in two international conferences [7, 8]. I would like to note that certain algorithms and results other than the ones presented here were produced in the context abandoned luggage detection, however due to confidentiality constraints (the work was carried out for the Department of Homeland Security in the United States) I could not share or publish them. Therefore, while the main scientific findings are shown here, the results in abandoned luggage detection are unfortunately not presented.

Let us start by showing a visual example of background estimation. In Figure 2.1, we see a frame from a video. This shows a person moving across an office.

This is a particularly simple example, however, generally speaking, there are several problems which may arise:

- Moving camera
- Dynamic background (water, leaves etc)
- Greatly varying lighting conditions
- Static objects

In this work, we will only be considering static cameras. Indeed, my work on this problem was geared toward video surveillance, where the cameras are all fixed. We do not consider dynamic background in particular; in all that follows, this will simply be considered as noise.

This Chapter will be concerned with the *low-rank* model and its application to background/foreground separation in videos, however, there are obviously many other techniques which exist for this purpose. We present a brief overview of these techniques now. Please note that these do not include current deep learning techniques, which were non-existant when this work was published.

Input frame Background Foreground

Figure 2.1: **Illustration of background/foreground separation.** The video frame is split into two images: the foreground and the background. Image taken from [112].

2.2 Previous work

Many early background estimation approaches used simple differences in greyscale to separate background from foreground, while trying to adapt to changes in lighting conditions, for example with the Kalman filter [149]. Wren et al. [176] use a single Gaussian distribution to model the background. A significant step forward was made by Stauffer and Grimson [162] who proposed to model the background as a mixture of Gaussian distributions, which are dynamically updated. This is still a very popular method due to its generality and flexibility.

The problem of segmentation is also a very old and important topic in the image processing and computer vision communities. An important early contribution was made by Mumford and Shah [130] who proposed a functional which basically evaluates how "good" a given piece-wise constant approximation of an input image is. The optimisation of this functional [29] provides a segmentation solution. Kass et al. [99] introduced another well-known segmentation model: active contours. This model evaluates a given segmentation curve with respect to the boundary smoothness and also to its proximity to object boundaries. Caselles et al. [43] extended this model, using tools from geometric curve evolution. Shi and Malik [157] recast the segmentation problem as a *graph clustering* problem, introducing the normalised cut criterion. Another common approach to segmentation is that of region merging [131] or splitting [134]. In this work, we draw inspiration from these ideas to achieve our goal of low-rank video segmentation.

More recently, Candès et al. [42] introduced a convex optimisation problem which can be applied to background/foreground estimation. The main idea is to separate an input matrix, which represents the video data, into two components: a low-rank component (the background) and a sparse component (the foreground). This will be the key tool in the rest of the Chapter. We present it now.

2.3 Low-rank models

The basic low-rank matrix approximation problem is the following. Let $\mathbf{X} \in \mathbb{R}^{m \times n}$ be a matrix. We we will try to approximate $\mathbf{X}$ with another, low-rank matrix, which we refer to as $\mathbf{L}$. This is given by:

$$\mathbf{L} = \arg\min_{\mathbf{L}} \|\mathbf{X} - \mathbf{L}\|_F^2, \qquad \text{such that } rank(\mathbf{L}) \leq r. \tag{2.1}$$

The first question is why would it be interesting to approximate a matrix with another that has lower-rank. To answer this, let us recall what the rank of a matrix means. If a matrix $\mathbf{L}$ is of rank

$r \leq n$, then we can write any column $\mathbf{L}_i$ as a linear combination of r vectors:

$$\mathbf{L}_i = \sum_{j=1}^{r} a_j \mathbf{c}_j. \tag{2.2}$$

This basically means that there is a sub-space with which we can describe the matrix $\mathbf{X}$. Another way of putting it is that the data contained in $\mathbf{X}$ live in the sub-space. Since we only need the vectors $\{\mathbf{c}_i\}$ to describe this data, they must represent it in some compact way. The significance of this is clear: it can be used for data analysis, dimensionality reduction etc.

It turns out that this problem is closely linked to Principal Component Analysis. Indeed, the solution to Equation (2.2) is given by the singular value decomposition (SVD) of $\mathbf{X}$. Let $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ be this SVD. The matrices $\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices, and $\mathbf{\Sigma} \in \mathbb{R}^{m \times n}$ is diagonal with diagonal values $\sigma_1, \ldots, \sigma_n$. Then the solution to Equation (2.2) is the matrix $\mathbf{L}$ such that $\mathbf{L} = \sum_{i=1}^{r} \sigma_i \mathbf{U}_i \mathbf{V}_i^{\mathrm{T}}$. This is known as the Eckart-Young-Mirsky theorem [60], and is equivalent to the solution given by Principal Component Analysis.

If we come back to background estimation in videos, the usefulness of the above discussion should start to become clear. Indeed, let us consider that our data $\mathbf{X}$ is a video, with each column X_j being a linearised (or "flattened") image, then the low-rank approximation is basically saying that each image in the video can be approximated by a weighted sum of a small set of images. For the case of background in videos, this hypothesis makes sense; indeed in ideal conditions, only one image would be needed to represent the background. In such a case, this image would represent the unchanging background, multiplied by a constant for each image, which would represent different global lighting conditions.

Let us take a visual example to illustrate this. Figure 2.2 shows how each frame of a video is converted into a column of a matrix. We have created a simple video by simply multiplying a given image by a new constant at each time step. In reality, videos are obviously much more complicated than this, with the following challenges:

- Foreground objects
- Local lighting conditions (spatially and temporally)

2.4 Robust Principal Component Analysis

To address the first issue, we can turn to the work of Candés *et al.* [42], who proposed "Robust Principal Component Analysis". We have seen that PCA is equivalent to low-rank approximation. However, PCA is known to be sensitive to outliers. More precisely, PCA works well when we have the following decomposition: $\mathbf{X} = \mathbf{L} + \mathbf{E}$, where $\mathbf{E}$ is some noise (or, seen another way, the approximation error). Unfortunately, this assumption does not hold at all when we have foreground in our video: there is no way to represent the foreground as either a linear combination of a few vectors or noise.

Candés *et al.* proposed to explicitly model the sparse outliers of a matrix, using the following model:

$$\mathbf{X} = \mathbf{L} + \mathbf{S} + \mathbf{E}, \tag{2.3}$$

where $\mathbf{S}$ is a *sparse* matrix, and, again $\mathbf{E}$ is some noise. This model makes sense for background estimation, since the foreground can be considered to not stay in one place for too long[1]. To find

[1] Although we will see that this hypothesis is only partially true

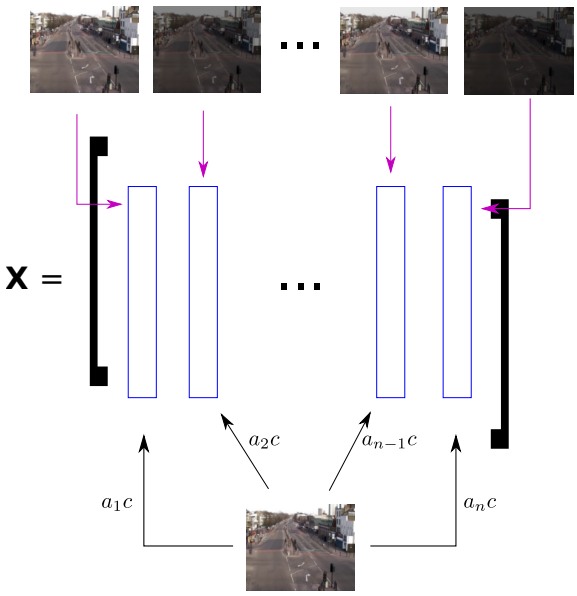


Figure 2.2: **Illustration of a low-rank representation of a video.** A video is converted into a matrix $\mathbf{X}$ by linearising (or flattening) each video frame into a column vector and then putting each vector into the columns of $\mathbf{X}$. A low-rank representation of this video is then given, for each column, by a linear combination of a few vectors. In this illustrative example we have created synthetic lighting changes. In such a simple case, we can approximate the video by a rank-1 matrix: each frame is simply a given vector $\mathbf{c}$ multiplied by a new constant.

the decomposition of $\mathbf{X}$, we can solve the following optimisation problem:

$$\mathbf{L}, \mathbf{S} = \arg\min_{\mathbf{L},\mathbf{S}} \tfrac{1}{2}\|\mathbf{X} - \mathbf{L} - \mathbf{S}\|_F^2 + \lambda\|\mathbf{S}\|_1, \qquad \text{such that } rank(\mathbf{L}) \leq r, \tag{2.4}$$

where $\|\mathbf{S}\|_1$ is the ℓ^1 norm, used to promote sparsity in $\mathbf{S}$. Unfortunately, the rank of a matrix is non-convex, and, contrary to the case of PCA in Equation (2.2), there is not a closed-form solution to the problem. Candés *et al*. proposed to replace the rank constraint with the *nuclear norm* (or trace norm) of $\mathbf{L}$, which is the best convex approximation of the rank of a matrix over the unit ball of matrices with norm less than one [145]. The nuclear norm, denoted with $\|\cdot\|_*$, is given by $\|\mathbf{X}\|_* = \sum_i \sigma_i$, where σ_i are the singular values of $\mathbf{X}$. This gives the following optimisation problem:

$$\mathbf{L}, \mathbf{S} = \arg\min_{\mathbf{L},\mathbf{S}} \tfrac{1}{2}\|\mathbf{X} - \mathbf{L} - \mathbf{S}\|_F^2 + \lambda_*\|\mathbf{L}\|_* + \lambda\|\mathbf{S}\|_1, \tag{2.5}$$

where λ_* and λ are two weighting scalars.

The significant advantage of this model is that the low-rank component and the sparse component are simultaneously and explicitly modelled, and retrieved, whereas in the basic low-rank approach, we could only first model the background and then hope that the residual $\mathbf{X} - \mathbf{L}$ would correspond to the foreground. Candés *et al*. proposed two main applications of this model: background estimation and facial recognition. We show an illustration of the kind of decomposition which this gives in Figure 2.3. For reference, in the original work of Candés *et al*. , the algorithm used to solve Equation (2.5) (they refer to this problem as Principal Component Pursuit) is an augmented Lagrange multiplier algorithm introduced in [115].

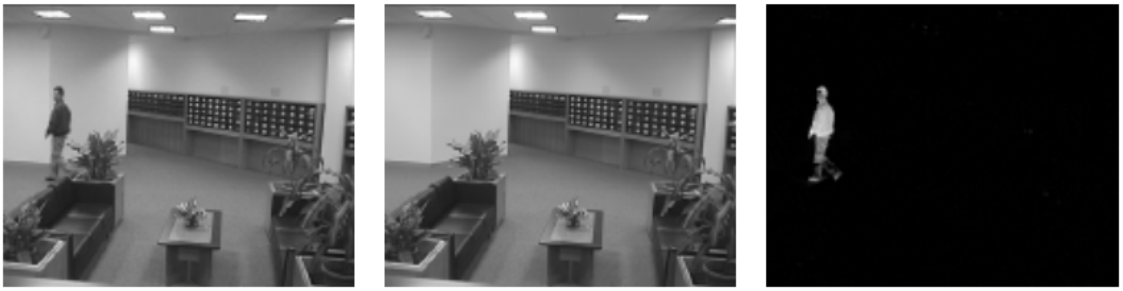


Figure 2.3: **Robust Principal Component Analysis (RPCA) applied to a video frame.** We see that the low-rank background and the sparse foreground are simultaneously retrieved using RPCA. Results taken from [42].

2.5 Multi-temporal detection in videos

We now recall that the original motivation for the work carried out during this postdoc was the detection of abandoned (and therefore potentially dangerous) luggage, in train stations in particular. In most realistic video surveillance settings, we do not have access to frames in the future, and we cannot wait until the end of the day, for example, to analyse a video. Indeed, if a suspicious package is dropped, then the authorities should know about it as soon as possible. Thus, we do not have access to the complete matrix $\mathbf{X}$, but only the frames in the past. Ideally, then, we would like to have an "online" version of RPCA to udpate the background and foreground estimations as we go along, and to not have to re-calculate them from scratch each time step. This is what is done in "Online RPCA", proposed by Sprechmann *et al*. [161].

2.5.1 Online RPCA

Online RPCA uses a very useful result by Recht *et al*. [145], which is that the nuclear norm of a matrix $\mathbf{L}$ of rank r may be reformulated as a penalty over all possible factorisations of the form

$$\|\mathbf{L}\|_* = \min_{\substack{\mathbf{U}\in\mathbb{R}^{m\times r}\\ \mathbf{V}\in\mathbb{R}^{r\times m}}} \tfrac{1}{2}\|\mathbf{U}\|_F^2 + \tfrac{1}{2}\|\mathbf{V}\|_F^2, \qquad \text{such that} \quad \mathbf{L}=\mathbf{U}\mathbf{V}. \tag{2.6}$$

In the context of video surveillance, it is reasonable to fix the maximum rank r. Please note that $\mathbf{U}$ and $\mathbf{V}$ are not orthogonal matrices this time, as they were in the context of the SVD. By combining Equations (2.5) and (2.6), we have the following minimisation problem:

$$\min_{\mathbf{U},\mathbf{V},\mathbf{S}} \tfrac{1}{2}\|\mathbf{X}-\mathbf{U}\mathbf{V}-\mathbf{S}\|_F^2 + \tfrac{\lambda_*}{2}\|\mathbf{U}\|_F^2 + \tfrac{\lambda_*}{2}\|\mathbf{V}\|_F^2 + \lambda\|\mathbf{S}\|_1 \tag{2.7}$$

Finally, this can be done in an *online* fashion [161], by determining the low-rank and sparse representations at each time step t. Let $\mathbf{X}_t$ represent the current image at time step t. Since the Frobenius norm and ℓ_1 norms are separable, and we can split the problem into two steps. First, given a fixed $\mathbf{U}^{t-1} \in \mathbb{R}^{m\times r}$, we wish to find the vectors $\mathbf{v}^t, \mathbf{s}^t \in \mathbb{R}^q$ as the solution of the following minimisation problem:

$$\min_{\mathbf{v},\mathbf{s}} \tfrac{1}{2}\|\mathbf{X}_t-\mathbf{U}^{t-1}\mathbf{v}-\mathbf{s}\|_2^2 + \tfrac{\lambda_*}{2}\|\mathbf{v}\|_2^2 + \lambda\|\mathbf{s}\|_1 \tag{2.8}$$

Indeed ,the minimisation over the term $\|\mathbf{U}\|_F^2$ in Equation (2.7) does not intervene if $\mathbf{U}$ is fixed to the value $\mathbf{U}^{t-1}$. We do this minimisation using the Iterative Soft Thresholding Algorithm (ISTA)-based approach described by Sprechmann *et al.* [161]. Once $\mathbf{v}, \mathbf{s}$ are found, they are fixed, so that $\mathbf{V}_t = \mathbf{v}^t$ and $\mathbf{S}_t = \mathbf{s}^t$.

Now, given fixed $\mathbf{V}, \mathbf{S}$, we wish to find $\mathbf{U}^t$ (this time we must minimise over the complete matrix). This is given by the following problem:

$$\min_{\mathbf{U}} \tfrac{1}{2}\|\mathbf{X} - \mathbf{U}\,\mathbf{V} - \mathbf{S}\|_2^2 + \lambda_* \|\mathbf{U}\|_F^2. \tag{2.9}$$

This is a Tikhonov-regularised least-squares problem which has the following solution:

$$\mathbf{U}^t = \mathbf{V}\,(\mathbf{X} - \mathbf{S})\left(\mathbf{V}\mathbf{V}^{\mathrm{T}} + \lambda_* \mathbf{I}\right)^{-1} \tag{2.10}$$

This summarises the online RPCA approach. For more details, see [161].

2.5.2 Detecting in multiple timescales

Recall that the initial motivation of this project was to detect abandoned objects in surveillance videos. Thus, we would like a map showing zones which may correspond to objects which have not moved for a long time. The proposed algorithm creates a heat map, showing foreground at several timescales. Longer timescales may indicate to a user that an abandoned object may be present. We show how this is done now.

Using the online RPCA method to decompose a video as each time frame comes, we establish a set of backgrounds and foregrounds, corresponding to different timescales. Consider the set of time intervals $\{[t - t_k]\}, k \in \{1, \ldots, T\}$, with $t_1 < t_2 < \cdots < t_T$, eg. time intervals organised from shortest to longest. We determine the set of low-rank and sparse representations $\{\mathbf{U}^{t_k}, \mathbf{v}^{t_k}, \mathbf{s}^{t_k}\}$ corresponding to each time interval. We also determine a long-term model $\mathbf{U}^\infty$, which should represent the "true" background. This is usually possible to in the setting of fixed video surveillance cameras. At each time step, we calculate the associated long-term representations $\{\mathbf{U}^\infty, \mathbf{v}^\infty, \mathbf{s}^\infty\}$.

Then, at each time-step t, we associate each pixel i with a characteristic time τ_i. This is defined as the shortest time interval in which the pixel is considered to be part of the foreground. More precisely, this is given by

$$\tau_i = \min_{\substack{k \in \{1,\ldots,T\} \\ \mathbf{s}_i^{t_k} \neq 0 \,\wedge\, \mathbf{s}_i^{\infty} \neq 0}} k. \tag{2.11}$$

An illustration of this can be seen in Figure 2.4.

To understand this more intuitively, consider a very short time interval $[t - t_k]$. In such an interval, almost everything will belong to the background, because there will not have been enough time for objects to move. However, as we increase the length of the interval, more and more pixels will switch from background to foreground. For pixels representing a moving person, this may only be a few seconds, which would mean that we do *not* want to detect these pixels, since suspicious objects stay in one place for longer. If the pixel switches from being in the background to the foreground at a very long time period, this means it has stayed in one place for a long time, and we should therefore detect it. Finally, if the pixel belongs to the long-term background, we do not detect it. Stricly speaking, with respect to Equation (2.11), this means that τ_i does not exist, but for simplicity we ignore this problem. In practice, we simply do not highlight the pixel in the output heatmap. Figure 2.5 shows the resulting map (image in the middle).

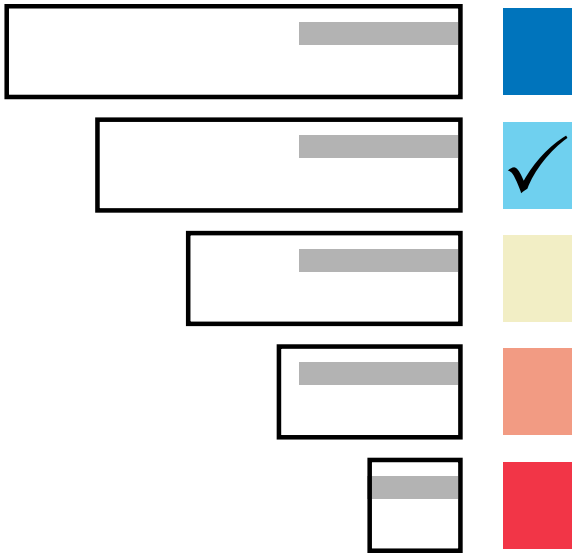


Figure 2.4: **Multiple timescales foreground/background detection.** We establish low-rank/sparse representations of the video at several time-scales. We associate a characteristic timescale to each pixel, which is defined as the smallest timestep at which the pixel belongs to the foreground. If the pixel belongs to the long-term background, then we do not highlight it (ie we keep it in grayscale). Longer timescales (in blue) represent longer scale foregrounds, which we want to detect.

Figure 2.5: **Map of characteristic timescales associated with each pixel, at time t, with spatial smoothing**. On the left, the original frame, in the middle the labelled timescale map τ. On the right the spatially smoothed map.

This map is quite noisy. Therefore, as a post-processing step, we perform a spatial smoothing on the labels τ_o, to improve their robustness and spatial coherence. We do this using an efficient discrete optimisation technique based on graph cuts [56]. Accordingly, the final labelling is given by

$$\arg\min_{\hat{\tau}} \sum_{i=1\ldots m} \Big[E_d(\hat{\tau}_i) + \sum_{j \in \mathcal{N}_i} E_s(\hat{\tau}_i, \hat{\tau}_j) \Big], \tag{2.12}$$

where $\mathcal{N}_i$ is the 4-neighbourhood of the pixel location i and E_d and E_s are the following data and smoothness terms

$$E_d(\hat{\tau}_i) = \begin{cases} \alpha & \text{if} \quad \hat{\tau}_i = \tau_i \\ 0 & \text{otherwise} \end{cases} \tag{2.13}$$

$$E_s(\hat{\tau}_i, \hat{\tau}_j) = \begin{cases} 1 & \text{if} \quad \hat{\tau}_i = \hat{\tau}_j \\ 0 & \text{otherwise.} \end{cases} \tag{2.14}$$

The scalar α is a constant penalty which we set to 1.5.

This final labelling represents the multi-temporal detection. We refer to the labelling for frame t as $\tau^{\mathbf{t}}$. The complete algorithm for the analysis of one incoming frame is presented in 1.

Algorithm 1: Online multi-temporal foreground detection algorithm

input : Data $\mathbf{X}$ at time t, long-term model $\mathbf{U}^{\infty}$, number of temporal windows T, length of each temporal window t_k, parameters λ_*, λ.
output : Multi-temporal detection map (τ_t).
`/* Detect foreground with different time windows */`
for $k \leftarrow 1$ **to** T **do**
  Find $\mathbf{s}_k^t, \mathbf{o}_k^t$, using $\mathbf{x}^t$, $\mathbf{U}_k^{t-1}$
  Find $\mathbf{U}_k^t$, using $\{\mathbf{x}^j, \mathbf{s}_k^j, \mathbf{o}_k^j\}_{j=t-t_k,\ldots,t}$
Associate a label τ_i to each pixel i in $\mathbf{x}^t$ using (2.11)
Smooth τ^t by solving (2.12)

The main fixed parameters of the algorithm are the maximum rank of the low-rank background and the parameters of the optimisation problem. We use $q = 3$ as the maximum rank, which is reasonable for most videos. Similarly to the work of Zhou *et al.* [188], we set the optimisation parameters to $\lambda = \sigma\sqrt{2}$ and $\lambda_* = \sigma\sqrt{2n}$, where σ is an estimation of the standard deviation of the noise in the video.

The main tunable parameters are the temporal window sizes t_k, and the total number of temporal windows T. These parameters are highly dependent on the target application and video. Together, they determine the temporal granularity and the maximum timescale of the analysis. For a finer analysis, the window sizes should vary slowly. We indicate the timescales for each experiment on a case-by-case basis.

Experimental Results We now discuss some of the applications of the multi-temporal foreground detection. Initially, the goal of this project was the detection of abandoned luggage in train stations. Unfortunately, we cannot show these images, since they are the property of the Department of Homeland Security (DHS). We proposed, apart from this application, others which we detail now.

One application is the automatic analysis of automobile traffic, in particular determining the fluidity of traffic. To test this application, we have used the "Qmul Junction" video from the work

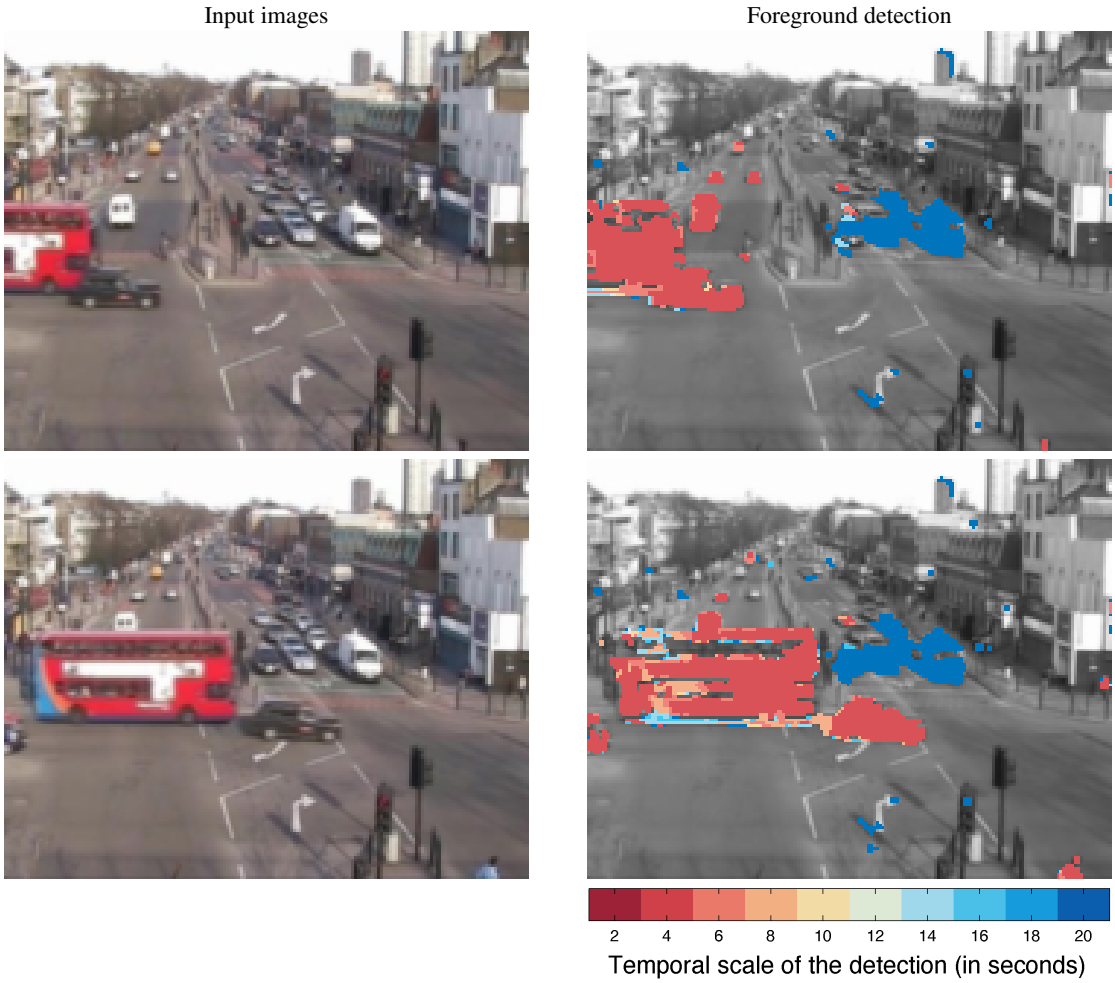


Figure 2.6: **Automatic detection of still and moving traffic.** The multi-temporal foreground is highlighted in colour in the images on the right. Blue corresponds to longer-term foreground, and red corresponds to shorter-term foreground. Cars stopped at the traffic lights are highlighted in blue, whereas those which are in movement are coloured in red.

of Loy et al. [121], which shows the traffic of a busy junction. The corresponding foreground detection can be seen in Figure 2.6. We highlight the detected foreground in colour, whereas the background remains in greyscale. The labelling τ^t is represented with colours ranging from blue to red. Blue corresponds to longer-term foreground and red to short-term foreground. Thus, we are able to distinguish between moving objects, static objects, and objects which have recently come to a halt.

We perform an additional analysis on this example, as illustrated in Figure 2.7. We have selected two regions in the video centred near one of the traffic lights, and plotted the ratio of long-term foreground pixels in each area. It is immediately apparent that the ratio related to the window indicated in green evolves in a cyclic manner due to the presence of traffic lights, whereas the one coloured in orange presents no clear pattern. It would be relatively easy for a user to analyse this information and detect if any anomalies are occurring based on prior knowledge of the traffic light periodicity.

A very useful application of the proposed algorithm is the detection of immobile people, for

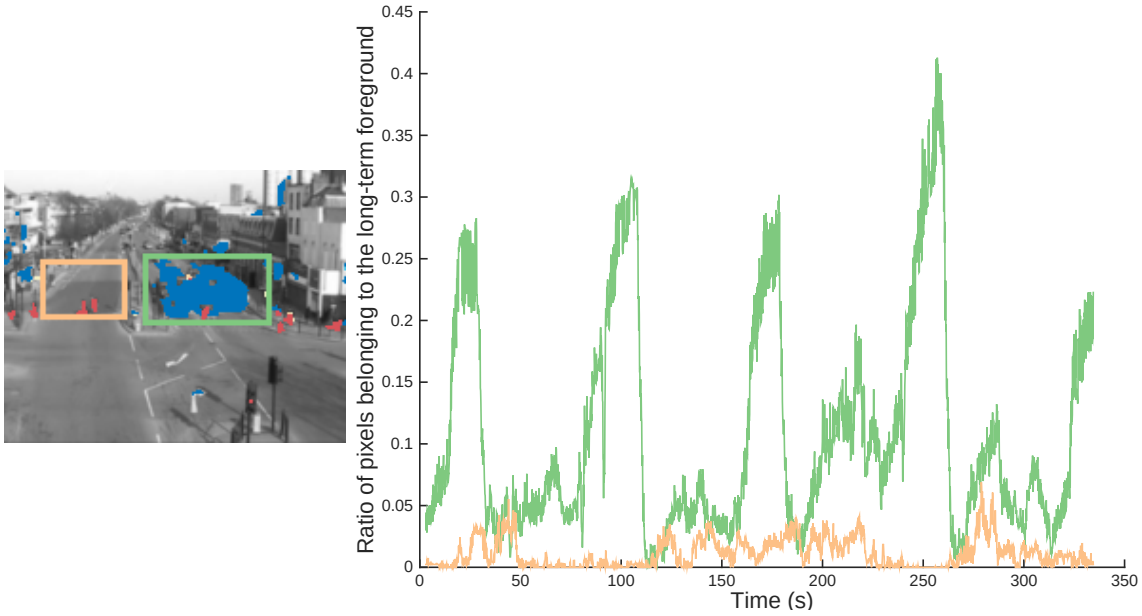


Figure 2.7: **Automatic analysis of traffic.** Evolution of the ratio of long-term foreground pixels in two regions of the video. The peaks of the line in green, correspond to a red traffic light.

example elderly persons who fall, or swimmers who are not moving. To illustrate this application, we have used the Fall Detection Dataset of Charfi et al. [45], which show videos of people falling and remaining still. Some results may be seen in Figure 2.8. We observe that the person is not detected while she is still moving, but is picked up in the long-term foreground when she remains still. By detecting objects in the correct timescale, a user could automatically detect people requiring help. An advantage of our approach is that we do not rely on human detection, which may not be robust to different poses and positions of humans.

Another interesting use for our algorithm is the analysis of time-lapse videos of Figure 2.9. In such cases, we are interested in changes over specific periods of time. This could be useful in particular for analysing certain events in natural videos, or agricultural time-lapse footage.

An important goal of this work is to provide an online algorithm. Therefore, we wish for as low time and memory requirements as possible. On the "Qmul junction" example (see Figure 2.7), our algorithm detects the foreground for ten timescales in 0.8 seconds on average, with no parallelisation. We used a machine with an Intel i7 3.40 GHz processor and 32GB of RAM.

2.6 Low-rank video segmentation

The low-rank model is quite robust, and easy to compute, however it has a fundamental drawback. In the presence of elements such as local lighting conditions, the rank of the background must be increased to be more flexible. However, by increasing this, we necessarily run the risk of incorporating more foreground elements into the background, especially if the foreground is immobile for a while. An illustration of this problem may be seen in Figure 2.10.

Thus, there is no solution to this trade-off problem in the normal RPCA framework. Therefore, we proposed an approach to automatically determine the different spatio-temporal regions in which the RPCA can be well-applied. This resulted in a video segmentation algorithm, based on the low-rank approximation. This was published in BMVC 2015 [7].

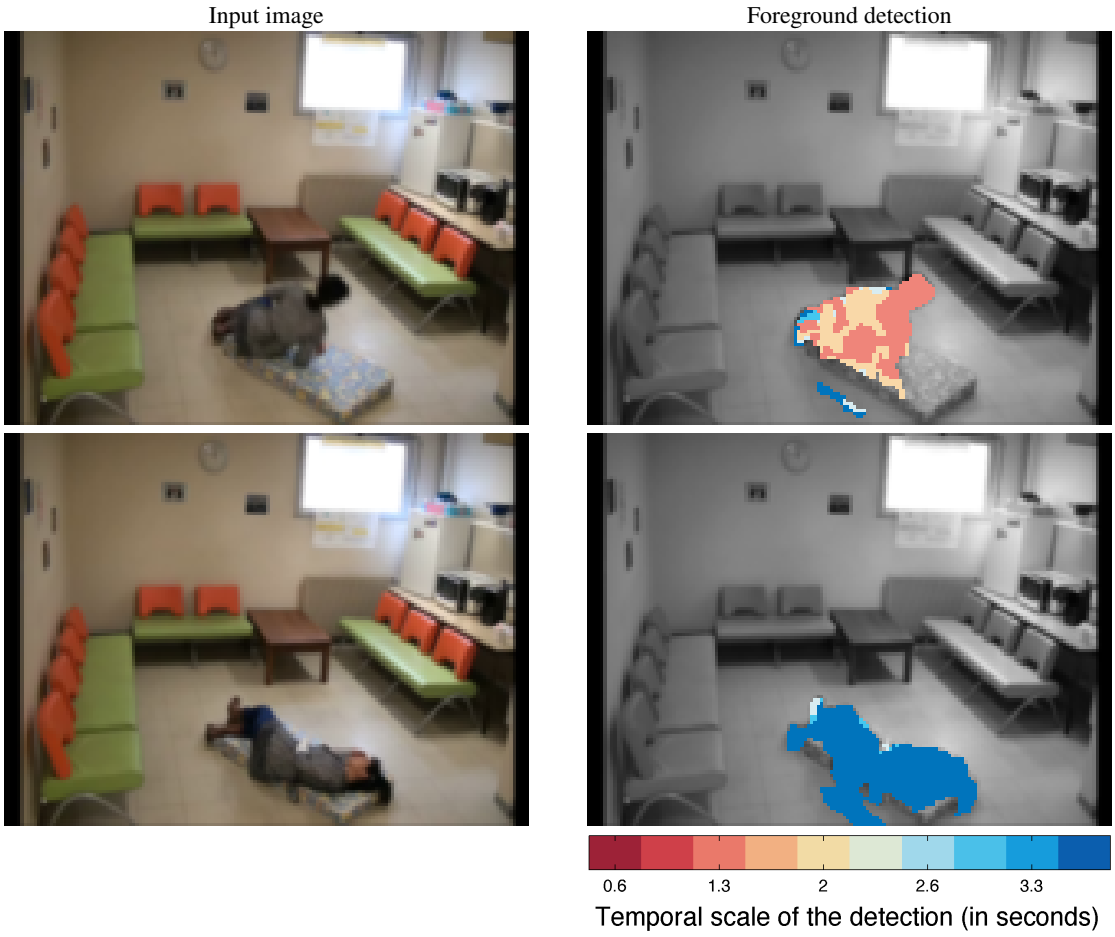


Figure 2.8: **An example of detecting still persons**. In this application, we detect people who are still and may need assistance, as they have fallen.

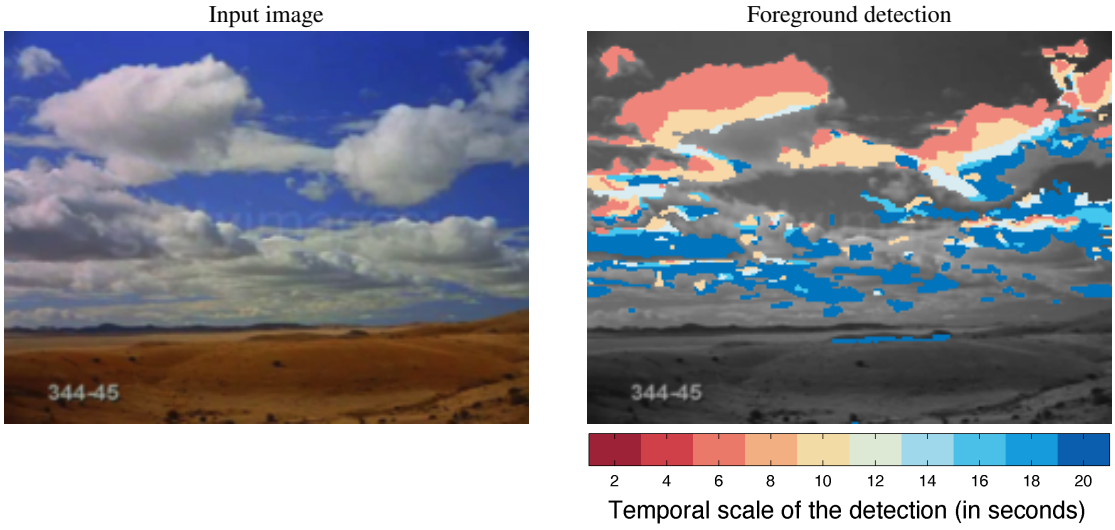


Figure 2.9: **A time-lapse example**. Our algorithm is able to detect clouds moving at different speeds in the sky. The closer the clouds are, the more they belong to short-term foreground.

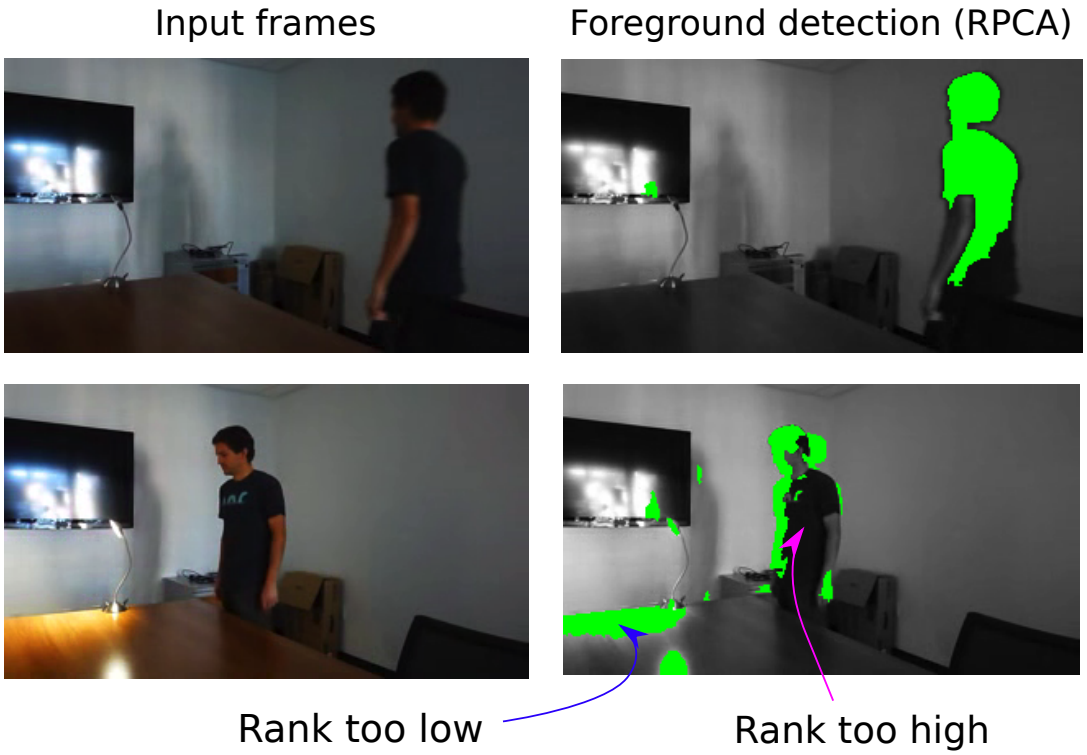


Figure 2.10: **Failure example of RPCA**. We see that it is impossible to find a satisfactory tradeoff in the rank of the background, due to the local lighting conditions.

To summarise our goal here, we wish to find a partitioning of the video which minimises the RPCA approximation error, when the RPCA is carried out locally in each partition. We denote the desired partitioning with $\mathcal{P} = \{\Omega_i\}_{i=1\ldots|\mathcal{P}|}$, where each Ω_i is a spatio-temporal region of the video. Thus, the desired partitioning is the solution of the following minimisation problem:

$$\min_{\mathcal{P}} \sum_{i=1}^{|\mathcal{P}|} \min_{\mathbf{L}^i,\mathbf{S}^i} \tfrac{1}{2}\|\mathbf{X}^i - \mathbf{L}^i - \mathbf{S}^i\|_F^2 + \lambda_*\|\mathbf{L}^i\|_* + \lambda\|\mathbf{S}^i\|_1, \tag{2.15}$$

where $\mathbf{X}^i$ corresponds to the video data in the region Ω_i, and $\mathbf{L}^i$ and $\mathbf{S}^i$ are the associated decomposition matrices, as previously.

Unfortunately, this is a very difficult combinatorial problem, and contains two optimisation problems. Therefore we propose a graph-based segmentation approach. The main idea of the algorithm is to establish a spatio-temporal graph whose nodes represent a spatio-temporal position, and whose edges have a weight which represents the cost of merging two adjacent regions into one. This cost should reflect how coherent the two regions are in terms of RPCA. If the cost is high, this should indicate that the two regions have different conditions which stop the RPCA being effective, and therefore we should have separate models. If it is low, then this means that we can safely represent both regions with the same low-rank model.

2.6.1 Creating the spatio-temporal graph

We start by creating an initial partition of the spatio-temporal domain of the video. We do this by defining a regular grid of regions, where each region is denoted with Ω_i. Each Ω_i is a spatio-

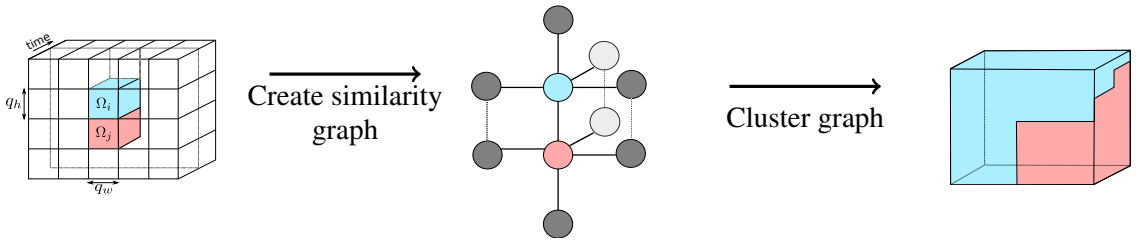


Figure 2.11: **Low-rank video segmentation algorithm's workflow.** We first calculate the cost of merging each adjacent region, using a low-rank approximation. From these costs, we build and then cluster a weighted graph, which produces the desired segmentation. The similarity $s(\Omega_i, \Omega_j)$ is defined in Equation (2.16).

temporal block of size $q_w \times q_h \times t$. We will try to merge each spatio-temporally adjacent region into a larger, coherent region. For an illustration, see Figure 2.11.

From this initial grid, we create an undirected, weighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of all the vertices of the graph, and $\mathcal{E}$ are the edges between these vertices. Each vertex corresponds to a single region in the initial grid. We refer to a vertex in the graph with Ω_i, in the same manner as a region of the video domain, even though this is an abuse of notation, strictly speaking. We choose a six-connectivity for our graph: each region is connected to the region directly left, right, above, below, before and after itself. This is obviously the minimum requirement for regions of any size to be produced; each vertex in the graph is connected to every other one by at least one path. We now need to define the most important element of our graph: the weights.

In the graph clustering approach that we use, based on the work of Zelnik-Manor and Perona [185], the weights of the graph are defined via a *similarity matrix*. This similarity should be the contrary of the cost: the greater the cost the smaller the similarity, and vice versa. We define the similarity in a classical manner::

$$s(\Omega_i, \Omega_j) = \exp(-d(\Omega_i, \Omega_j)^2/(2\beta^2)) + \varepsilon, \tag{2.16}$$

where $d(\Omega_i, \Omega_j)$ is a distance between two regions, β is a kernel width and ε is a very small scalar which avoids two adjacent regions being disconnected in the similarity matrix.

Obviously, this just puts the definition of similarity on the shoulders of the *distance* function. We define in detail in the next Subsection 2.6.2. We reiterate that given this similarity matrix, we find the optimal partitioning using the spectral clustering algorithm of Zelnik-Manor and Perona [185].

2.6.2 A reliable criterion for video region merging

We wish to determine a distance function d that indicates how coherent two regions are in terms of their backgrounds. Let us consider two regions, Ω_i and Ω_j whose choerence we wish to assess. We propose to use the RPCA decomposition itself as an indicator of the coherence of these regions. The most direct way to proceed would be to apply the RPCA to Ω_i, Ω_j, and $\Omega_{i\cup j}$, and observe the ranks of the background components of each decomposition. Unfortunately, the rank is a relatively unstable criterion, and quite sensitive to changes in the optimisation parameters λ and λ_*. Alternatively, we could use the nuclear norm as an indicator, given that this is the closest convex function to the rank. Again, there is no obvious way to compare $\|\mathbf{L}^i\|_*$, $\|\mathbf{L}^j\|_*$ and $\|\mathbf{L}^{i\cup j}\|_*$ since the nuclear norm is non-separable. For example, if two matrices $\mathbf{A}$ and $\mathbf{B}$ have the same

number of rows, then

$$\begin{aligned} \|[\mathbf{A},\ \mathbf{B}]\|_* &= \|[\mathbf{A},\ \mathbf{0}] + [\mathbf{0},\ \mathbf{B}]\|_* && (2.17)\\ &\leq \|[\mathbf{A},\ \mathbf{0}]\|_* + \|[\mathbf{0},\ \mathbf{B}]\|_* && (2.18)\\ &= \|\mathbf{A}\|_* + \|\mathbf{B}\|_*. && (2.19) \end{aligned}$$

The upshot of all this is that an algorithm using this criterion may have a tendency to over-merge regions, that is, prefer the concatenation $[\mathbf{A},\ \mathbf{B}]$.

To design a more reliable merging criterion, we propose to modify the RPCA decomposition of each region, by returning to the "non-relaxed" formulation based on the *rank* rather than the nuclear norm. We redefine the low-rank decomposition of a region Ω_i as:

$$\begin{aligned} \left\{\mathbf{L}^i, \mathbf{S}^i\right\} = \underset{\mathbf{L},\mathbf{S}}{\arg\min}\ & \tfrac{1}{2}\|\mathbf{X}^i - \mathbf{L} - \mathbf{S}\|_F^2 + \lambda\|\mathbf{S}\|_1 \qquad (2.20)\\ \text{subject to}\quad & \operatorname{rank}(\mathbf{L}) \leq r. \end{aligned}$$

This formulation has two major advantages. Firstly, we have fixed the maximum rank of $\mathbf{L}$ in the decomposition. This means, as mentioned above, that the nuclear norm does not play a role in the energy of the decomposition, making comparisons more reliable. Secondly, it avoids having to set the parameter λ_*. This is useful since the rank-constraint r is more easily interpretable than the nuclear norm weight λ_*.

The new rank-constrained problem is non-convex; giving up convexity is the price to pay for having a decomposition which leads to a reliable merging criterion. Nevertheless we can address it using an alternating approach. Accordingly, we perform a minimisation firstly over $\mathbf{L}$, and then over $\mathbf{S}$. The first problem can be addressed [145] by decomposing the low-rank matrix into the product of two submatrices $\mathbf{L}^i = \mathbf{U}^i\mathbf{V}^i$, with $\mathbf{U}^i \in \mathbb{R}^{m\times r}$, $\mathbf{V}^i \in \mathbb{R}^{r\times n}$, and alternately minimising the Frobenius norms of the submatrices. The second may be solved with soft thresholding. We give details of these minimisation processes in Algorithm 2.

Let e_i denote the quadratic error of the low-rank/sparse approximation:

$$e_i = \|\mathbf{X}^i - \mathbf{L}^i - \mathbf{S}^i\|_F^2. \qquad (2.21)$$

We propose to use this error for the cost of merging two adjacent regions. Assuming that the sparse foreground elements appear equally in $\mathbf{S}^i$, $\mathbf{S}^j$, and $\mathbf{S}^{i\cup j}$, logically the energy due to the term $\|\mathbf{S}^i\|_1$ will have no influence on the merging decision. For these reasons we argue that if the two sub-regions of $\Omega_{i\cup j}$ are coherent, then this cost will be very low, which is our goal. Indeed, the Frobenius norm is separable, which means that for two adjacent, coherent regions (two regions where the low-rank assumption is accurate) we should have $e_i + e_j = e_{i\cup j}$. Thus the quadratic error provides a meaningful comparison of the coherence of two regions.

We now give the formal definition of the cost of merging two regions:

$$d(\Omega_i, \Omega_j) = \frac{|e_i + e_j - e_{i\cup j}|}{\phi_{i\cup j}}, \qquad (2.22)$$

where $\phi_{i\cup j}$ is the following scaling factor:

$$\phi_{i\cup j} = \begin{cases} 1 & \text{if } \Omega_i \text{ and } \Omega_j \text{ are spatially adjacent} \\ q_w q_h \sigma^2 & \text{if } \Omega_i \text{ and } \Omega_j \text{ are temporally adjacent.} \end{cases} \qquad (2.23)$$

We discuss this factor $\phi_{i\cup j}$ in detail in the next section. We recall that q_w and q_h are the spatial width and height of the initial blocks.

Algorithm 2: Alternating minimization scheme for solving Equation (2.20), with $r = 1$

input : Data $\mathbf{X}^i$ to decompose, parameter λ, step size τ.
output : Sparse matrix $\mathbf{S}^i$, rank one background matrix $\mathbf{L}^i$
$\mathbf{S} \leftarrow \mathbf{0}$
Initialise $\mathbf{u}$ as the temporal median of $\mathbf{X}^i$
repeat
  $\mathbf{v} \leftarrow \left(\mathbf{u}^{\mathrm{T}}\mathbf{u}\right)^{-1}\left(\mathbf{u}^{\mathrm{T}}(\mathbf{X}^i - \mathbf{S})\right)$
  $\mathbf{u} \leftarrow \left((\mathbf{X}^i - \mathbf{S})\mathbf{v}^{\mathrm{T}}\right)\left(\mathbf{v}\mathbf{v}^{\mathrm{T}}\right)^{-1}$
  $\mathbf{S} \leftarrow \mathrm{shrink}_\lambda(\mathbf{S} + \tau(\mathbf{X}^i - \mathbf{uv}))$ // $\left(\mathrm{shrink}_\lambda(\mathbf{A})\right)_{p,q} = \begin{cases} 0 & \texttt{if } (\mathbf{A})_{p,q} < \lambda \\ (\mathbf{A})_{p,q} - \lambda & \texttt{otherwise} \end{cases}$
until *convergence*
$\mathbf{S}^i \leftarrow \mathbf{S}$; $\mathbf{L}^i \leftarrow \mathbf{uv}$

2.6.3 Comparing spatial and temporal merging fairly

In Equation (2.22), we have assumed that when we try to merge two regions which are coherent, the quadratic errors of the two regions will be similar, irrespective of whether they are spatially or temporally adjactent, so that our merging cost is reliable. Unfortunately, this assumption is not quite correct, since the RPCA decomposition itself will give different errors if we merge spatially or temporally. Note, this does not contradict the fact that the Frobenius norm is separable, it just means that the decompositions themselves will be different. Let us see why now.

Consider two adjacent regions Ω_i and Ω_j which contain the same static background and no lighting changes, plus some Gaussian noise of (estimated) variance σ. Let us also suppose that $r = 1$, which is reasonable in this case. In such a setting, the expected value of the merging cost is very different if we merge spatially or temporally. For spatially adjacent regions, on average the cost will be zero. However, when we merge two temporally adjacent regions, we have:

$$\mathbb{E}(|e_i + e_j - e_{i\cup j}|) \approx q_w q_h \sigma^2. \tag{2.24}$$

These results are proven in Appendix A.1. Intuitively, the costs in the spatial and temporal merging situations are dissimilar for the following reason. Two temporally adjacent, coherent regions, contain different (noisy) observations of the same variables/pixels. In the case of spatial adjacency, we have twice the number of variables, without increasing the number of observations. This means that by merging temporally adjacent regions, we will significantly decrease the approximation error, while merging spatial regions will not give any difference in approximation error. In summary, temporal merging gives a larger value of $d(\Omega_i, \Omega_j)$ than spatial merging.

In order to counter this effect, and make sure that merging is not favored in either the spatial or temporal directions, we need to set the scaling factor $\phi_{i\cup j}$ of Equation 2.23 correctly. Given the previous reasoning, we scale the temporal merging with $\phi_{i\cup j} = q_w q_h \sigma^2$. In the spatial merging case, we do not scale the cost function, i.e., we set $\phi_{i\cup j} = 1$.

Minimisation of Equation (2.20) and choice of r

It is clear that the most expensive operations of our algorithm are the local RPCA decompositions in each spatio-temporal region. To speed this up, we propose to choose $r = 1$ in Equation (2.20). In fact, this restriction makes sense; in one coherent region there should logically be only one "true" background. We distinguish this special case by denoting the matrices $\mathbf{U}^i$ and $\mathbf{V}^i$ with lowercase letters, since they are now vectors, so that $\mathbf{L}^i = \mathbf{u}^i\mathbf{v}^i$. We have used this speedup in all

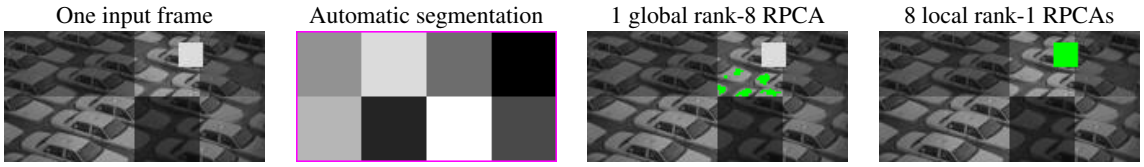


Figure 2.12: **Foreground estimation in a challenging, synthetic video using the proposed segmentation. The estimated foreground is highlighted in green.** In this case, the background contains several regions whose contrast varies independently, and a foreground component (the white square). We detect the coherent regions, indicated in the second image with the grey squares, and use them to carry out localised RPCA decompositions.

of our experiments, with good results. The minimisation algorithm to decompose $\mathbf{X}^i$ in this case is shown in Algorithm 2.

Segmentation overlap

The choice of a non-overlapping initial grid, imposes a lower limit on the granularity of the segmentation, which can be problematic, especially in the temporal direction. The main goal of our segmentation is to carry out background/foreground estimation locally in each region. For this purpose, we do not need the segmentation to be pixel-precision. Instead, we dilate each segmented region to a half of the initial grid precision and perform the final low-rank/sparse decomposition in these dilated regions. Then, for the pixels in overlapping regions, we choose the model which best fits the data for that pixel.

2.6.4 Experimental results

We now show some results of our segmentation algorithm on synthetic and real data. In particular, we show that commonly used background/foreground estimation algorithms and the standard RPCA fail when faced with difficult situations including both variable lighting and foreground which may be static for a while. The proposed algorithm exhibits significant improvement over these other approaches in such scenarios.

Firstly, in Figure 2.12, we show a synthetic example where it is impossible to correctly separate the foreground and background with a standard, global RPCA. The video contains eight regions which are each illuminated independently, and a sparse component which moves around before finally staying in one position for a short while. In this example, the initial blocks are chosen to span the entire temporal extent of the video. Our algorithm finds the correct segmentation. The main point here is that *whatever* the rank of the global approximation, the classical RPCA will not be able to recover the background and foreground correctly. Let us further illustrate this issue with a real example.

In general, the background subtraction literature uses examples which are simple in terms of varying lighting conditions. Either no lighting changes happen, or they are relatively global. In Figure 2.13 we provide a more complex example. A person is walking in and out of the video, while different lights are turned on (a lamp, and then an overhead light). We annotated the foreground of each frame of this video by hand to provide a ground truth. We segmented the video with our algorithm, which was able to locate the different points in time and space of the lighting changes. We then carried out a local rank-one RPCA in each region.

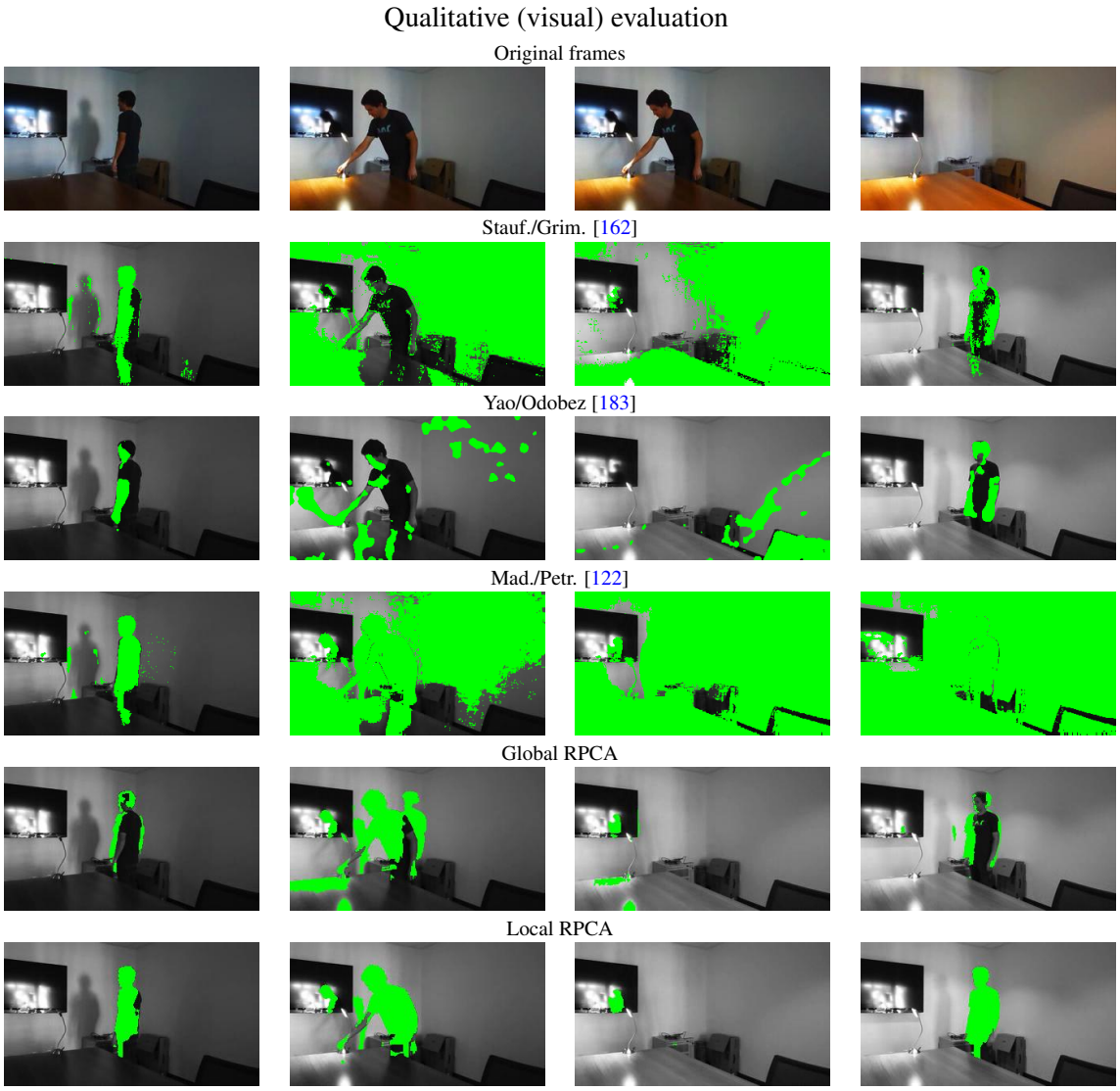


Figure 2.13: **Foreground estimation in variable lighting conditions.**. We segment the video using the proposed algorithm, and carry out a local, rank one RPCA in each region. We compare with three other well-known background subtraction algorithms [162, 183, 122].

	Stauffer/Grimson [162]	Yao/Odobez [183]	Maddalena/Petrosino [122]	Global RPCA	Local RPCA
Recall	70.83	67.88	61.69	50.27	**74.97**
Precision	39.35	68.16	05.97	60.57	**81.26**
f1-score	50.60	68.02	09.97	54.94	**77.99**

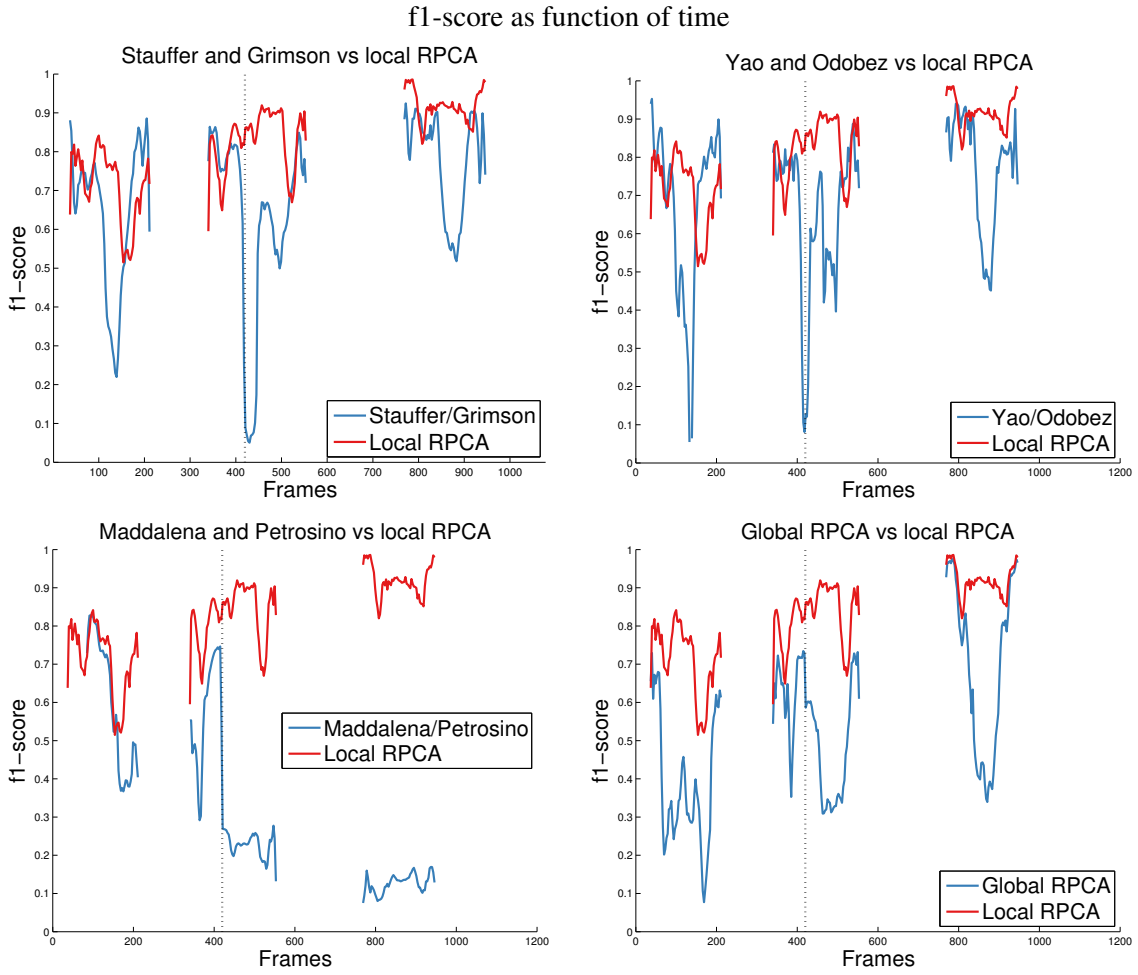


Figure 2.14: **Quantitative evaluation of foreground estimation in variable lighting conditions.**. We compare our algorithm with those of [162, 183, 122], evaluating the recall, precision and f1 score. The video example is the same one as chosen for Figure 2.13. The black dotted vertical line in the f1 score plots indicates a local lighting change when the foreground person is present.

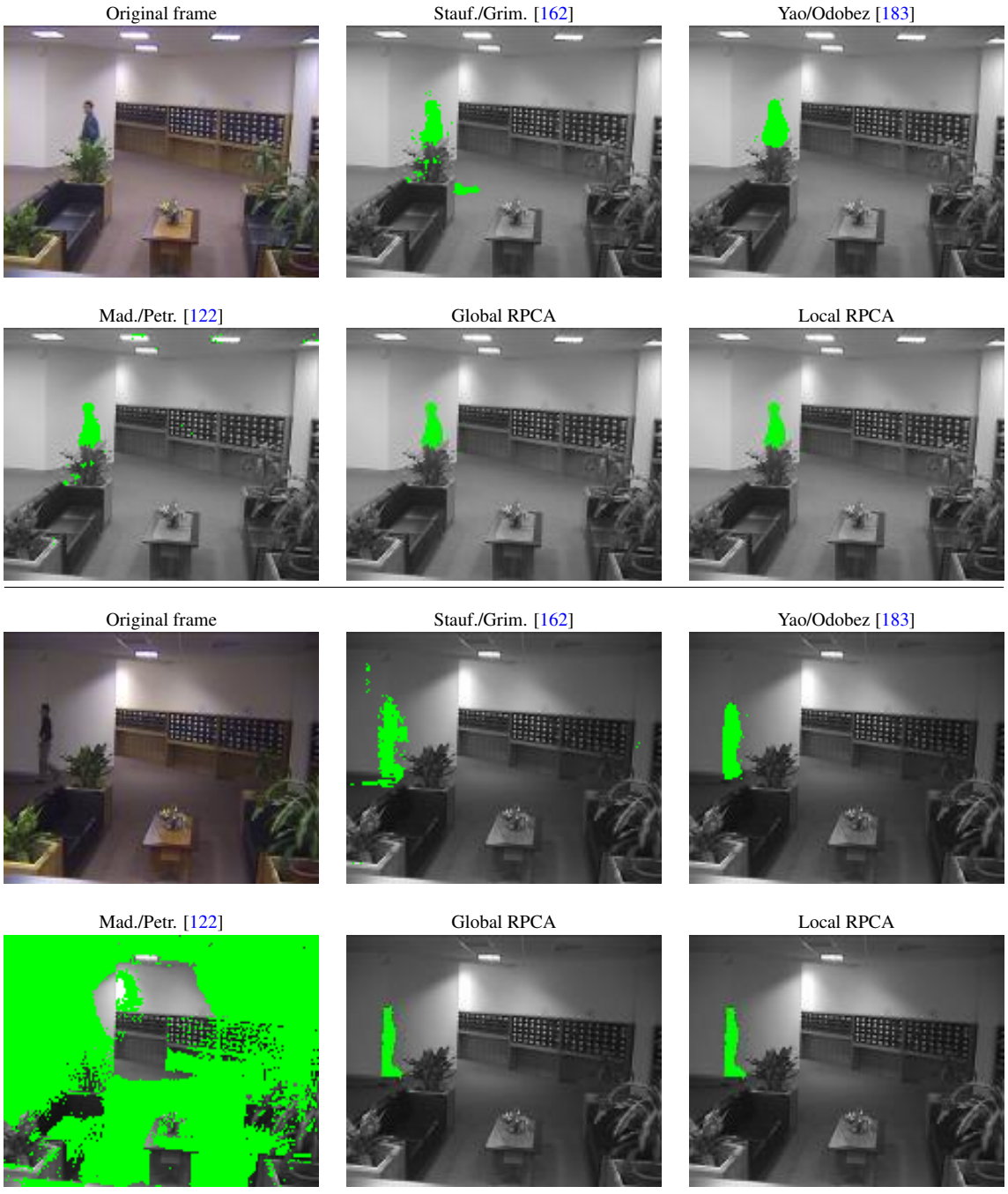


Figure 2.15: **Detection in an example from the database of [111].** In this case, the lighting variations are global, so the proposed algorithm performs similarly to the standard RPCA.

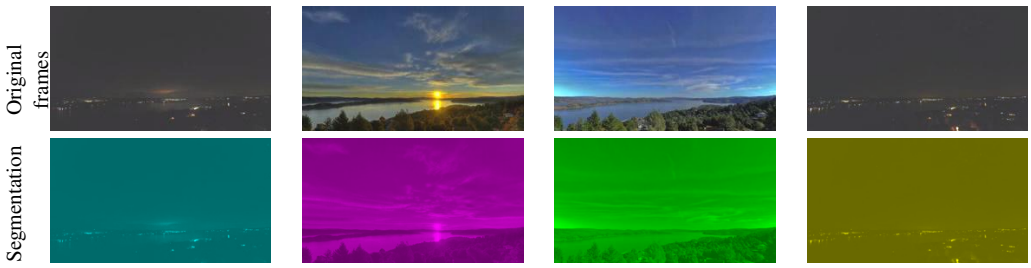


Figure 2.16: **Segmentation in a time-lapse example. The segmentation is indicated by varying colours.** Here, the background changes dramatically depending on the time of day. Our algorithm picks up these changes, and this is reflected in the segmentation. The standard RPCA considers that there is only one background, which does not make sense here.

We compared the resulting local RPCA foreground detection with three other popular background subtraction methods from the literature [162, 183, 122], and also with the results of the global RPCA. Our local background models are all of rank-one. We used the implementations of the background subtraction library from [158]. Figure 2.13 analyses four frames of the video, which clearly illustrate the advantages of the local RPCA. At the moment of sudden lighting changes (second and third columns), two of the algorithms from the literature strongly over-detect, whereas both the global and the local RPCAs are robust to this effect. However, the global RPCA achieves this at a cost: to represent the locally varying lighting, the low-rank model must also include the person who remains static for a short while in the background. The local rank-one models are robust to the temporarily static person; we detect the person well whenever he is present.

Our quantitative evaluation, shown in Figure 2.14 is carried out in terms of recall, precision and f1-score. The f1-score is defined as $\mathrm{f1} = 2\frac{\mathrm{recall}\cdot\mathrm{precision}}{\mathrm{recall}+\mathrm{precision}}$. A high f1-score implies both a high recall and precision, and is thus a better way to evaluate algorithms than using recall or precision alone. We show the recall, precision and f1-score taken over the whole video in the table of Tab 2.13. The local, rank-one RPCA has an f1-score of 77.99%, compared with 68.02% of the best other method [183]. As a complement to this, we also show the f1-score per frame. The goal of this is to illustrate the lack of robustness which other methods exhibit. We do not show the f1-score on frames where few or no pixels are labelled as foreground in the ground truth, as the scores are quite unstable or meaningless for all the algorithms in this case. It is clear that the other approaches suffer from a lack of robustness either to strong lighting changes, or to the foreground person which is incorporated into the background. The local RPCA maintains a good f1-score during in these challenging situations.

In Figure 2.15, we see results from a standard video from the literature [111]. In this example, our local RPCA performs similarly to the global one, since the illumination changes are global. On this video (160×128, 50 seconds), the segmentation took 7m, and the subsequent local RPCA took 6m24s, on a machine with an Intel Core i5 processor. Interestingly, the graph clustering only took 0.5s, meaning that most of the work goes into calculating the edge weights. This process could obviously be carried out completely in parallel, which would greatly decrease execution times.

In Figure 2.16, we show another interesting segmentation result on a timelapse video. In this example, our segmentation algorithm is able to identify the different temporal segments which contain coherent lighting. This lighting varies throughout the video, as the timelapse goes from sunrise to sunset.

We have introduced the problem of segmenting videos into regions which are well-represented

by a low-rank background model plus a sparse foreground. We address this problem by creating and clustering a graph from an initial grid of spatio-temporal regions. We carefully design a cost function to determine whether two adjacent regions are coherent, in terms of their low-rank approximations. With this clustering/segmentation, we can carry out several *local* RPCAs instead of one global one. Using quantitative and qualitative comparisons with the standard RPCA and several state-of-the-art algorithms from the background subtraction literature, we show that the new piece-wise low-rank model produces significantly better background/foreground estimation in challenging situations.

One weakness of this approach is the imprecision of the segmentation due to the resolution of the initial grid to be clustered. It is not possible for this grid to become too fine for computational reasons. While this is not much of a problem for applications in video surveillance, for other purposes it could potentially be problematic. One option would be to create several overlapping grids and choose/merge the best results.

2.6.5 Conclusion on the low-rank model for video analysis

In this Chapter, we have concentrated on the *low-rank* model for the analysis of videos. We considered two problems:

- The separation of videos into foreground and background, over several timescales;
- The segmentation of videos into regions where the low-rank assumption holds well.

The main advantages of the low-rank approach are that both foreground and background are modelled in the optimisation problem (Equation (2.5)) proposed by Candés *et al*. [42], and by using the nuclear norm as a convex substitute for the rank constraint [145], they produce a convex optimisation problem to solve.

One of the drawbacks of the model is its inability to deal with moving/random backgrounds. While the camera is mostly considered static in background estimation, the background itself can sometimes move in a way which does not qualify it to become foreground. A common example of such a motion is that of leaves in a tree or forest; the leaves do indeed belong to the background, but exhibit random motion with the wind. In this case, the ideal solution is to put the residual into the error of the optimisation problem, however the motion is still not modelled. This is not the case with other simple models such a Gaussian Mixture Models [162], which can introduce some sort of uncertainty in the background. In this sense, the low-rank model is quite rigid.

As mentioned previously, since this work was carried out, there have been many papers using deep neural networks for background/foreground estimation. This Chapter will not delve into these approaches, however we note that a rich literature in segmentation of photographic images/videos has come about which uses them [70, 69, 147, 79]. These approaches display exceptional performances, meaning that for the purposes of video surveillance (ie detecting bags, humans), it may be easier to simply rely on them rather than solving a foreground/background separation problem. Nevertheless, as always in the case of deep learning, this requires labelled databases, which may not be easy to obtain.

Chapter 3

Film grain synthesis

In this Chapter, we present a model for film grain synthesis. Film grain is the texture which appears in images that are taken with analog cameras. This texture is due to the fact that such images are made up of small crystals which are photo-sensitive. The most common example of this process is the silver-halide film process. It is this process which we imitate with our model. The work presented in this Chapter was published in [1, 2] and [9].

Before continuing to describe this process, we note that the goal of this work is purely *artistic*. There are two main cases in which the synthesis is useful:

- Adding grain to give a certain look to a digital image
- Re-graining an image. This is necessary when restoring old films; first, the film grain is removed, the film is then restored (scratches, lines etc. are removed), and finally, the grain is put back (otherwise the film does not look right)

Now that we have set out the uses of film grain synthesis, let us move on to a brief description of the silver-halide photographic process.

3.1 The silver-halide analog film process

Silver-halide films are made up of several layers: a flexible plastic base, the actual film emulsion, and finally a protective upper layer. The emulsion is a transparent gelatin in which silver-halide crystals are suspended. These silver halides are often silver bromide, Ag^+Br^-. As mentioned above, these crystals are light-sensitive. When light is shone on the emulsion, sometimes a photon hits a silver bromide molecule, giving a reduction reaction: $Ag^+Br^- \rightarrow Ag^0 + Br^0$. This creates a tiny amount of solid silver in the crystal, only about 4-8 atoms. This step, called the *sensitisation* step, takes place when the photograph is taken by the photographer, when the aperture opens. At this point, there is only a *latent* image, which is barely visible. Then the photographer takes the film to be developed. This is done by introducing a chemical component which completes the reduction reaction on any crystals which have been sensitised. This gives a *negative* image: where there was a lot of light, there are now opaque crystals, which means the image blocks light (is dark). Similarly, where there was no light, the emulsion remains transparent. The negative image is then inverted and projected onto photosensitive paper to give the final photograph. This physical model of the silver-halide film process is due to Gurney and Mott [77], and is widely accepted. A diagram of the process can be seen in Figure 3.1. We now present the model whose goal is to replicate this process.

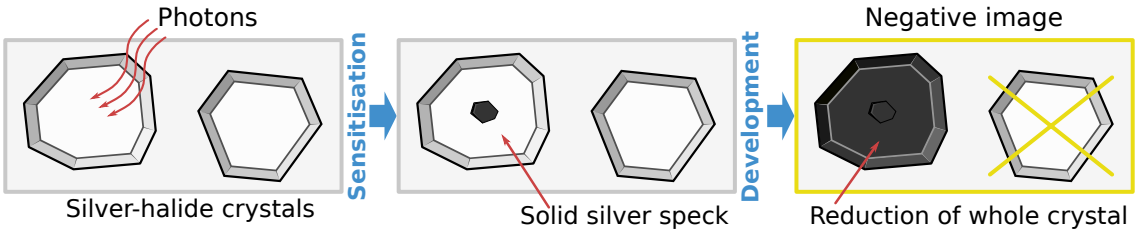


Figure 3.1: **Illustration of the silver-halide photographic process**.

3.2 Previous work

There is surprisingly little previous work on film grain synthesis. Nutting [132] was the first to study the statistical properties of the so-called "random dot model" for film grain, and proposed the Nutting formula which links the optical density of a film emulsion to the average number of grains present and to the size of the inspected region. An important quantity studied in the analog literature is *granularity* or root-mean-square (rms) granularity. This is experimentally measured using a microdensitometer on any given film emulsion after development, and corresponds to the standard deviation of the optical density of the emulsion. Another useful connected quantity is that of Selwyn granularity [174], which is basically the granularity defined in such a fashion that it is independent of the aperture size. A good summary of these basic notions may be found in the paper of Bayer [34] in the context of the random dot model. A particularly hot topic concerning film grain is that of grain "clumping". This corresponds to the perceived clustering of film grain. Much of the subsequent analog literature is concerned with proposing mathematical models which imitate this effect [44, 108, 165].

Many approaches use actual scanned examples of film grain for the purposes of synthesis. This appears to be the most popular approach in the industrial environment as well as in some academic approaches. Film grain synthesis products such as DxO's "FilmPack"[58] and Grubba Software's "TrueGrain"[76] tools take this approach. More precisely, a single grain image is saved for each film type. In the same philosophy, Schallauer and Mörzinger [154] extract the grain pattern from real images of grain and synthesise a new grain image from these examples, which they then apply to the image in an additive fashion. Unfortunately, they do not go into detail as to how this synthesis is carried about. A similar approach is used in the Film Emulation feature of the G'MIC free software [71] using the random phase texture algorithm [64] to synthesise large grain textures from small stored samples.

Bae et al. [33] use the classical Heeger-Bergen texture synthesis [84] approach on a constant region in an example grainy image to produce film grain. However, they do not specify how this is applied to a given input image. Stephenson and Saunders [163] filter white noise in the Fourier domain. Yan et al. [180] proposed an additive film grain model with signal-dependent noise. A drawback is that their approach supposes that film grain noise is spatially uncorrelated, which is clearly unrealistic for film grain noise. Oh et al. [133] propose an auto-regressive model for film grain removal and synthesis. They point out that spatial correlation is crucial for producing realistic film grain. However, they consider that an input grainy image is available, and that the characteristics of the grain may be extracted. We also note that other works have looked at simulating various photographic processes [68, 59], but these are not concerned with simulating physical film-grain.

Strangely enough, there are *no works on grain synthesis using deep learning methods*. I think

that this is the only domain I know where this is the case: a very exceptional outlier!

A common drawback of the approaches of the digital literature is that no model based on the physical reality of film grain is proposed. Either a digital example of film grain, with fixed resolution, is considered to be available (which may not always be possible), or spatial correlation of the film grain texture is not considered. Furthermore, even if a good example of film grain is available, it is not obvious how to blend this grain with an input image. In this work, we propose a realistic film grain model based on physical considerations which requires no example for synthesis, and which does not need any such blending process.

3.3 A stochastic film grain model

In order to imitate film grain, we use a model from the literature of stochastic geometry [52], known as the *Boolean* model. This consists of a set of balls whose centres are randomly distributed in a given space. The Boolean model is one of the simplest and most straightforward from stochastic geometry. Obviously, the balls will represent the silver-halide crystals.

3.3.1 A Boolean model for film grain

Let us define the Boolean model more precisely. Let $\Phi = \{x_i,\ i \in \mathbb{N}\}$ represent a Poisson process on $\mathbb{R}^2$ with intensity λ. These x_i represent the centers of our grains. We also define a sequence of identically and independently distributed (i.i.d.) random compact sets in $\mathbb{R}^2$, $X_0, X_1, \ldots$, which will represent the grain shapes. The Boolean model is the random set Z defined as the union of all the shapes X_i placed at the locations x_i, that is,

$$Z = \bigcup_{i \in \mathbb{N}} (X_i + x_i). \tag{3.1}$$

Finally let us also define the indicator function of the Boolean model Z as the function $\mathbb{1}_Z(y)$, which for all $y \in \mathbb{R}^2$, equals 1 if $y \in Z$ and 0 otherwise. This is a particularly flexible model, and allows for random grain radii or even different shapes. In all that follows, we will consider the shapes X_i to be *balls* of possibly random radii r_i: $X_i = \mathcal{B}\left((,x)_i, r_i\right)$. However, note that the model works with any type of shape. We denote with $A_i = \pi r_i^2$ the area of X_i. To summarise, the Boolean model consists of "white" balls on a "black" background, and we use this model to represent the physical reality of film grain. To see an illustration of this, see Figure 3.2.

The main question now is how to choose λ, which represents on average how many grains are contained per unit square of $\mathbb{R}^2$. If we wanted to replicate the complete analog photographic process, we would need to know the amount of light originally present in the scene which created a digital image when the photo was taken, then imitate the sensitisation probability of a grain. Also, we would have to know the density of grains in the original emulsion. Unfortunateley, much of this information is unknown. Therefore, we opt for a more pragmatic approach.

To set λ, we first present one of the fundamental properties of the Boolean model, the volume fraction occupied by Z. This is equivalent to the probability of a given point y being covered by a ball:

$$\forall y \in \mathbb{R}^2, \quad \mathbb{P}\left(\mathbb{1}_Z(y) = 1\right) = 1 - \exp(-\lambda \mathbb{E}\left[A_1\right]), \tag{3.2}$$

where $\mathbb{E}\left[A_1\right] = \pi \mathbb{E}[r_1^2]$ is the common mean area of the i.i.d. balls.

Given this property, we set λ to respect the local grey-level of the input image. This is in fact the only sensible option to maintain the contrast of the image. However, setting a different λ for each pixel requires a slight modification of our model. Indeed, the model described above

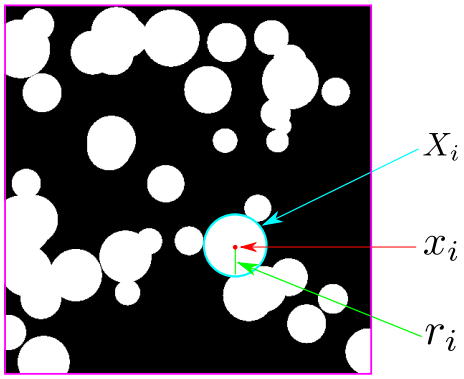


Figure 3.2: **Illustration of the Boolean model.** The Boolean model consists of randomly positioned, overlapping, shapes $X_i + x_i$. In this work, we consider balls of possibly random radii r_i.

is the *homogeneous* Boolean model, where λ is constant. For λ to vary locally, we turn to the *inhomogeneous* Boolean model.

3.3.2 Inhomogeneous Boolean model for film grain

As in the homogeneous case, the inhomogeneous Boolean model is built upon a sequence of random positions $\Phi = \{x_i, i \in \mathbb{N}\}$ given by a Poisson process. However, the intensity λ of the Poisson process Φ is no longer constant. It is given by a function $\lambda(y)$ which varies spatially with $y \in \mathbb{R}^2$. We choose λ to be piece-wise constant, with each λ being determined locally for the underlying input pixel.

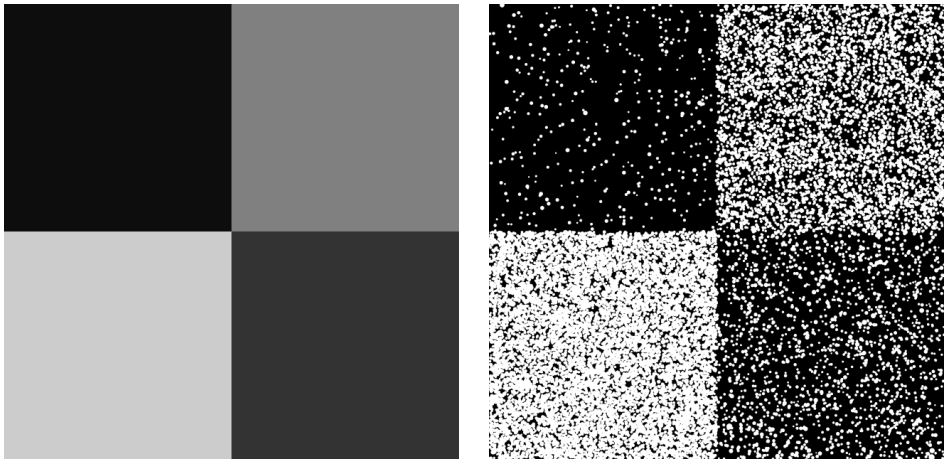
Let $u : \{0, \dots, m-1\} \times \{0, \dots, n-1\} \subset \mathbb{N}^2 \to [0, u_{max}]$ be the input, digital, image, of size $m \times n$. We wish to synthesise film grain on this image. We start by normalising the input image u to the interval $[0, 1)$ by defining $\tilde{u}(y) = \frac{u(y)}{u_{max}+\varepsilon}$, where ε is a small parameter, and u_{max} is the maximum possible grey-level value. We restrict the image to $[0, 1)$, as a Boolean model with $\mathbb{P}(\mathbb{1}_Z(y) = 1) = 1$ would require a degenerate infinite intensity λ.

The average grey-level is equivalent to the probability of being covered by a ball, shown in Equation (3.2). Thus, to maintain the local average grey-level, we set λ as:

$$\lambda(y) = \frac{1}{\mathbb{E}[A_1]} \log\left(\frac{1}{1 - \tilde{u}(\lfloor y \rfloor)}\right), \tag{3.3}$$

where $\lfloor \cdot \rfloor$ is the whole part (or "floor") function, which is extended to $\mathbb{R}^2$ by taking the whole part of each components independently. As mentioned above, we need to restrict the input image to the interval $[0, 1)$, and Equation (3.3) clearly shows why this is necessary: if $\tilde{u}(y) = 1$, then we have $\lambda(y) = \infty$. This makes sense, since we need an infinite density of balls to be sure of covering absolutely any point y. An illustration of the final inhomogeneous Boolean model for film grain can be seen in Figure 3.3.

We now have a new representation of the image; instead of a series of discrete 2D samples, ie. a digital image, we have a *continuous* representation via the model Z. Unfortunately, the output of our algorithm is supposed to be a digital output with grain. The question is then, how do we evaluate our Boolean model on a discrete grid? This clearly depends on the resolution at which we want to display the output image, with respect to that of the input image. For example, if we



Input pixels (close-up) Inhomogeneous Boolean model

Figure 3.3: **Illustration of the inhomogeneous Boolean model for film grain**. Each pixel is replaced by a continuous representation consisting of the inhomogeneous Boolean model Z, where the intensity λ of the underlying Poisson process is determined locally with respect to the grey-level of each pixel.

were to take a film emulsion and project its image onto a cinema screen, we might start to be able to see individual grains[1]. Therefore, the grainy quality of the output image clearly depends on this output resolution. We need to carefully consider what it means to *observe* the output image.

3.3.3 Evaluating the Boolean model

When the human eye observes a silver-halide film, we do not observe with infinite resolution, otherwise we would see all such images as binary images! In reality, due to the optical processes of going from the negative to the positive image, as well as the functioning of the human eye itself, we see a kind of locally averaged value, which gives the observed grey-level. To model this, we consider two filters, ψ and ψ', with $\int_{-\infty}^{\infty} \psi(y)dy = 1$ (no image energy is lost), and similarly for ψ'. The filter ψ represents the optical apparatus, which acts on the negative image $1 - \mathbb{1}_Z$, while ψ' represents the human visual apparatus, which acts on the positive image.

Let v be the output image. To model the complete process from negative to observed positive, we have:

$$\forall y \in \mathbb{R}^2, \quad v(y) = \left(\psi' * [1 - \psi * (1 - \mathbb{1}_Z)]\right)(y) = (\psi' * \psi * \mathbb{1}_Z)(y). \tag{3.4}$$

The upshot of this is that we can model the process simply, with one filter. This is particularly practical, as we can separate the creation of the Boolean model, which represents the physical film grain, from the application of the filter, which represents the observation process. In this work, we simply apply a single Gaussian low-pass filter to represent the combined blurring steps from the negative image to the perceived image. We shall refer to this filter as ϕ in all that follows, and put aside the complete model. An illustration of this is shown in Figure3.4

[1]The size of a grain may reach 10^{-6}m, while the smallest point visible by a human is around 10^{-4}m. This zoom of 100 times is roughly the order of magnitude when going from a film emulsion (24mm) to a cinema screen (9.1m)

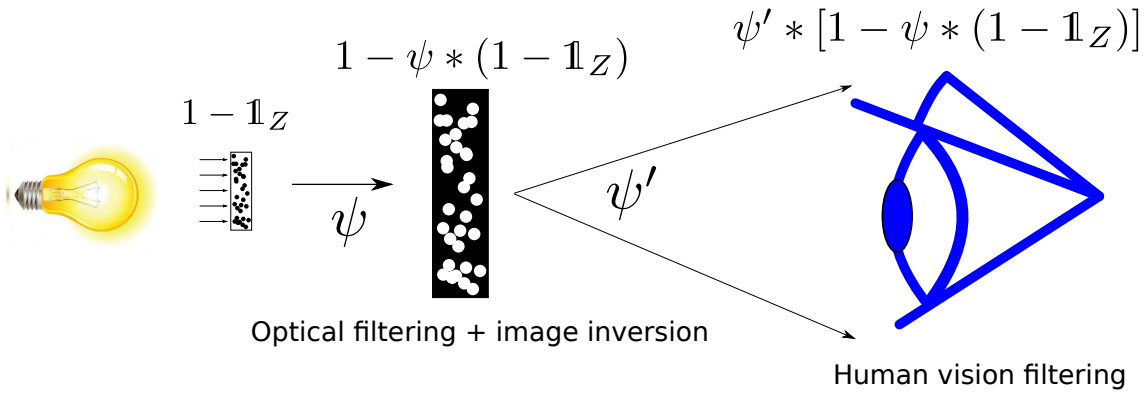


Figure 3.4: **Illustration of observation process of a silver-halide film**. The initial emulsion contains the negative image $1 - \mathbb{1}_Z$: where there was light in the scene, crystals block light, and where there was no light, light can pass through. This is inverted into the positive image $1 - \psi * (1 - \mathbb{1}_Z)$, with an optical apparatus corresponding to the filter ψ. Finally, the human eye observes the positive image with its own filter: $\psi' * [1 - \psi * (1 - \mathbb{1}_Z)]$.

To summarise, our final observed film grain model is simply a filtered indicator function of the inhomogeneous Boolean model. This is a conveniant, compact model. It is important to note that this can be evaluated on any desired discrete output grid, so we could indeed project it onto a cinema screen, and the graininess would be meaningful. If we only considered grain at a single resolution, its visual aspect would always be the same, irrespective of the size of the output image. We will see in our experiments that the best visual quality for film grain requires quite high-resolution images. Now that we have the complete model, we describe the algorithm which will produce the final film grain output.

3.3.4 Film grain rendering algorithm

The algorithm which we propose can be split up into two steps:

- sampling the inhomogeneous Boolean model;
- evaluation of the filtered inhomogeneous Boolean model.

Let us first describe the sampling step.

Sampling the inhomogeneous Boolean model

Firstly, we wish to produce a *realisation* of the inhomogeneous Boolean model, in other words we wish to sample the centers and radii of the grains throughout the image. We consider that the unit square is defined by the input image grid $\{0, \dots, n-1\} \times \{0, \dots, m-1\}$.

Since the intensity function λ (see Equation (3.3) of our inhomogeneous Boolean model is constant on each pixel, represented by unit squares $[i, i+1) \times [j, j+1)$, the Poisson process of the centers of our inhomogeneous model can be partitioned into the disjoint union of $m \times n$ Poisson processes having their points in their respective pixel square $[i, i+1) \times [j, j+1)$. Then, within a pixel square $[i, i+1) \times [j, j+1)$, the intensity is constant and given by (3.3) with $y = (i, j)$, and one can simulate the centers using the standard Poisson process simulation. This consists in drawing

Algorithm 1 Sampling of the inhomogeneous Boolean model from an input image.

Data: $u : \{0, 1, \dots, m-1\} \times \{0, 1, \dots, n-1\} \to [0, u_{max}]$ (input image)
Parameters:
$\mathcal{D}(\mu_r, \sigma_r^2)$: distribution of grain radii
Result:
x: List of grain centers
r: List of grain radii

Sample Boolean model within the whole image domain:
$x \leftarrow \emptyset$, $r \leftarrow \emptyset$
foreach $(i, j) \in \{0, \dots, m-1\} \times \{0, \dots, m-1\}$ **do**
  Convert grey-level to the interval $[0, 1)$:
  $\tilde{u}(i, j) = \frac{u(i,j)}{u_{max}+\varepsilon}$
  Compute local value of intensity λ:
  $\lambda = \frac{1}{\pi(\mu_r^2+\sigma_r^2)} \log \frac{1}{(1-\tilde{u}(i,j))}$
  Draw the number of grains Q *in the square* $[i, i+1) \times [j, j+1)$:
  $Q \leftarrow$ Poisson(λ)
  Sample $x_{i=1\dots Q}$ from $\mathcal{U}\left([i, i+1) \times [j, j+1)\right)$
  Sample grain radii $r_{i=1\dots Q} \sim \mathcal{D}(\mu_r, \sigma_r^2)$
  Add the new points to the list:
  $x \leftarrow x \cup x_{i=1\dots Q}$; $r \leftarrow r \cup r_{i=1\dots Q}$

the number of points Q according to a Poisson distribution with parameter $\lambda(y)$ and then drawing Q grain centers x_i from a uniform distribution $\mathcal{U}([i, i+1) \times [j, j+1))$ and Q independent radii r_i from the radius distribution. In practice, for these radii we choose a constant distribution or a log-normal distribution. This method is described in Algorithm 1. Note that this pseudo-code describes our algorithm in the case where the grains are balls of possibly random size, but it can be modified to include arbitrary shapes, as we shall describe in Section 3.3.5.

Evaluation of the inhomogeneous Boolean model

Equation (3.4) gives us a method to evaluate the Boolean model on any given grid. Unfortunately, in practice this is not done so straightforwardly. The reason is that evaluating the convolution of ψ with a continuous, random, function ($\mathbb{1}_Z$) is not easy task. We have no closed-form method to do this. However, it is possible to approximate the integral required by the convolution using a Monte Carlo simulation.

We first define a scalar N representing the number of samples in the Monte Carlo simulation. For each desired output position y, we draw a list of offsets $\{\xi_i, i = 1 \dots N\}$ whose row-column coordinates follow a Gaussian distribution $\mathcal{N}(y, \sigma^2)$. This corresponds to the Gaussian filter ψ that we used to model the Boolean model observation process. We produce the output pixel value using

$$v(y) = \frac{1}{N} \sum_{k=1}^{N} \mathbb{1}_Z(\xi_k). \tag{3.5}$$

As N increases, according to the law of large numbers :

$$\frac{1}{N}\sum_{k=1}^{N}\mathbb{1}_Z(\xi_k) \xrightarrow[N\to+\infty]{} \mathbb{E}(\mathbb{1}_Z(\xi_1)) = \int_{\mathbb{R}^2}\mathbb{1}_Z(t)\phi(y-t)dt, \tag{3.6}$$

where ϕ is the pdf of the Gaussian distribution $\mathcal{N}(0,\sigma^2 I_2)$, that is, the targeted Gaussian blur kernel. I_2 represents the identity matrix of size 2×2.

We denote with s the *zoom* factor of the output image, such that the dimensions of the latter is $sn \times sm$. Thus $\frac{1}{s}$ represents the output image grid discretization step, with respect to the unit square of the input. In simple terms, s represents the "zoom" of the output image resolution with respect to the input image. The pseudo-code for the Monte Carlo simulation is shown in Algorithm 2. We use the same set of random offsets for each pixel, which avoids the repeated use of random number generation, which can be slow.

Algorithm 2 Evaluation of an inhomogeneous Boolean model with Monte Carlo simulation.

Data: $x_{i=1\dots Q}$, $r_{i=1\dots Q}$ sampled inhomogeneous Boolean model, with a total of Q grains
Parameters:
s : output zoom
σ : standard deviation of the Gaussian low-pass filter
N : number of iterations in the Monte Carlo method
Result: v : Rendered film grain image

Initialize the output image to 0:
$v = 0$
for $k = 1$ **to** N **do**
  Draw a random offset from a centered Gaussian distribution of variance σ^2:
  $\xi_k \leftarrow \mathcal{N}(0,\sigma^2 I_2)$
  for $\ell = 1$ **to** Q **do**
    $y = sx_\ell + \xi_k$
    foreach $(a,b) \in \{0,\dots,sm-1\}\times\{0,\dots,sn-1\}$ *s. t.* $\|y-(a,b)\|_2 \le sr_\ell$ **do**
      $v(a,b) = v(a,b)+1$

Average the contributions:
$v = \frac{1}{N}v$
return(v)

3.3.5 Algorithmic details and implementation

In Section 3.3.4, we presented the film grain rendering algorithm in two separate parts: the sampling of the inhomogeneous Boolean model (Algorithm 1), and the evaluation of the filtered model (Algorithm 2). A disadvantage of this method is that all the grain positions must be stored in memory and then processed. For example, if we suppose a grain radius $r = \frac{1}{40}$, with a high resolution image (2048×2048) with constant grey-level values of 128 everywhere, 35 GB of memory is needed to store the grain positions and radii, if the information is stored with single precision floating point. It is clear that this naïve approach is not satisfactory and will not scale to large images.

We propose two algorithms which address this problem of storing the grain information. The first generates each grain once only, determines the effect of this grain on the output image, and then erases the grain information. We refer to this as the "grain-wise" approach (see Algorithm 3). Unfortunately, this algorithm is not well-adapted to parallelization on the GPU, due to excessive memory accesses. Therefore, we propose a second approach (Algorithm 4) which we refer to as the "pixel-wise" algorithm, which is suitable for GPU parallelization. We describe these approaches now.

Grain-wise algorithm

As previously mentioned, this approach samples the grains sequentially for each pixel, and evaluates the effect of each grain on each Monte Carlo iteration. Once a grain has been generated and processed, its information (position and radius) are erased from memory. Instead of saving the grain information, we store a sequence of N *binary* images $v_k,\ k \in \{1 \dots N\}$, with each image corresponding to the result of one Monte Carlo iteration. A Monte Carlo iteration consists in evaluating the Boolean model on the randomly shifted grids $\xi_k + \{0, \dots, sm-1\} \times \{0, \dots, sn-1\}$, with $\xi_k \sim \mathcal{N}(0, \sigma^2 I_2)$. Finally, the result of our algorithm is simply the average of all the temporary images v_k. An advantage of this approach is that the memory requirement is independent of the grain size (since we do not store the grains), and only depends on the output image size and the number of Monte Carlo iterations N. This algorithm can be seen in Algorithm 3.

Pixel-wise algorithm

The second algorithm we present here, which we refer to as the "pixel-wise" approach, also avoids storing the grain information, but in quite a different manner from the grain-wise approach. The main reason for proposing another algorithm is that memory accesses should be limited when using the GPU. Therefore, we cannot save a large number of intermediate images required by the Monte Carlo simulation of Algorithm 3. Instead, we employ an on-the-fly Poisson process generation often used in the procedural noise literature [105]. This type of approach consists in using a grid partition of the space $\mathbb{R}^2$ and generating the Poisson process on-the-fly within each partition cell using a local pseudo-random number generator [175, 106]. Given a coordinate pair of a given partition cell, the pseudo-random number generator can generate the number of grains whose centers belong to this cell, as well as the positions of these centers. The numbers given by the pseudo-random number generator are completely reproducible, so there is no need to store the information pertaining to the grains. Note that the cell size δ must be a fraction of the input image pixel size.

The main task is to evaluate $\mathbb{1}_Z(y)$ at $y \in \mathbb{R}^2$ for each Monte Carlo sample. This is equal to 1 if there is a point x_i such that $y \in \mathcal{B}(x_i, r)$. Therefore, one only needs to simulate the Poisson process in cells intersecting the ball of radius r centered at y to evaluate $\mathbb{1}_Z(y)$. Unfortunately, this approach is not valid when using random radii given by a distribution producing unbounded variates, such as the log-normal distribution. In this case, we specify a maximal value of the radii, r_m. Therefore, we need to check more cells in order to evaluate $\mathbb{1}_Z(y)$, and this number of cells obviously increases quadratically with a linearly increasing maximum radius. This approach is described in detail in the pseudo-code of Algorithm 4.

Algorithm 3 The proposed "grain-wise" film grain rendering algorithm. The loop coloured in blue is parallelized.

Data: $u : \{0, 1, \dots m-1\} \times \{0, 1, \dots, n-1\} \to [0, u_{max}]$: input image
Parameters:
$\mathcal{D}(\mu_r, \sigma_r^2)$: distribution of grain radii
s: output zoom
σ: standard deviation of the Gaussian low-pass filter
N: number of iterations in the Monte Carlo method
Result: v: Synthesised, film grain image

Set up N binary images of size $ms \times ns$ and draw N random offsets:
for $k = 1$ **to** N **do**
    $v_k = 0$
    $\xi_k \leftarrow \mathcal{N}(0, \sigma^2 I_2)$
foreach $(i, j) \in \{0, \dots, m-1\} \times \{0, \dots, n-1\}$ **do**
    $\tilde{u}(i, j) = \frac{u(i,j)}{u_{max}+\varepsilon}$
    $\lambda = \frac{1}{\pi(\mu_r^2+\sigma_r^2)} \log \frac{1}{(1-\tilde{u}(i,j))}$
    $Q \leftarrow$ Poisson(λ)
    Sample $x_{i=1\dots N}$ from $\mathcal{U}\left([i, i+1) \times [j, j+1) \right)$
    Sample grain radii $r_{i=1\dots Q} \sim \mathcal{D}(\mu_r, \sigma_r^2)$
    for $k = 1$ **to** N **do**
        for $\ell = 1$ **to** Q **do**
            $y = sx_\ell + \xi_k$
            foreach $(a, b) \in \{0, \dots, sm-1\} \times \{0, \dots, sn-1\}$ *s. t.* $\|y - (a, b)\|_2 \leq sr_\ell$ **do**
                $v_k(a, b) = 1$
foreach $(i, j) \in \{0, \dots, sm-1\} \times \{0, \dots, sn-1\}$ **do**
    $v(i, j) = 0$
    for $k = 1$ **to** N **do**
        $v(i, j) = v(i, j) + v_k(i, j)$
    $v(i, j) = \frac{1}{N}\, v(i, j)$
return(v)

Algorithm 4 The proposed "pixel-wise" film grain rendering algorithm. The loop colored in blue is parallelised.

Data: $u : \{0, 1, \dots m-1\} \times \{0, 1, \dots, n-1\} \to [0, u_{max}]$: input image
Parameters:
$\mathcal{D}(\mu_r, \sigma_r^2)$: distribution of grain radii
r_m: maximum radius allowed
s: output zoom
σ: standard deviation of the Gaussian low-pass filter
N: number of iterations in the Monte Carlo method
Result: v: Image rendered with film grain

$\delta = \frac{1}{\lceil \frac{1}{r_m} \rceil}$
foreach $(i, j) \in \{0, \dots, sm-1\} \times \{0, \dots, sn-1\}$ **do**
  $v(i, j) = 0$
  for $k = 1$ **to** N **do**
    $\xi_k \leftarrow \mathcal{N}(0, \sigma^2 I_2)$
    $(i_g, j_g) = \frac{1}{s}((i, j) + \xi_k)$
    Get the list of cells which might contain the balls covering (i_g, j_g) *:*
    foreach $(i_\delta, j_\delta) \in \{\lfloor \frac{i_g - r_m}{\delta} \rfloor, \dots, \lfloor \frac{i_g + r_m}{\delta} \rfloor\} \times \{\lfloor \frac{j_g - r_m}{\delta} \rfloor, \dots, \lfloor \frac{j_g + r_m}{\delta} \rfloor\}$ **do**
      $\tilde{u} = \frac{u(\delta.i_\delta, \delta.j_\delta)}{u_{max} + \varepsilon}$
      $\lambda = \frac{1}{\pi(\mu_r^2 + \sigma_r^2)} \log \frac{1}{(1 - \tilde{u})}$
      $Q \leftarrow$ Poisson(λ)
      for $\ell = 1$ **to** Q **do**
        $x \leftarrow \mathcal{U}\left([i_\delta, i_\delta + 1) \times [j_\delta, j_\delta + 1) \right)$
        $y = (\delta.i_g, \delta.j_g) + \delta.s$
        $r = \min(\mathcal{D}(\mu_r, \sigma_r^2), r_m)$
        if $||y - x||_2 < r$ **then**
          $v(i, j) = v(i, j) + 1$
          Break: go to next Monte Carlo iteration
  $v(i, j) = \frac{1}{N} v(i, j)$
return(v)

3.3.6 Performance comparisons

It is a difficult task to automatically determine which alogrithm, the pixel-wise or the grain-wise, is more efficient in different situations. This depends in a complex manner on the size of the grains, the number of Monte Carlo iterations N, the average grey-level etc. We have extensively tested both the grain-wise and pixel-wise algorithms in different situations in order to identify the speedups which are achieved. We have tested the following implementations:

- Grain-wise, no parallelisation;
- Grain-wise, with parallelisation (OpenMP) on a multi-core CPU;

	Image size			
	256×256	512×512	1024×1024	2048×2048
Grain-wise, non-para.	168.815 s	676.502 s	2718.92 s	10843.0 s
Grain-wise, para. (CPU)	10.749 s	40.992 s	165.433 s	654.696 s
Pixel-wise, non-para.	9.207 s	36.567 s	147.687 s	584.496 s
Pixel-wise, para. (CPU)	0.732 s	2.430 s	9.499 s	37.786 s
Pixel-wise, para. (GPU)	0.137 s	0.429 s	1.275 s	4.534 s
# grains processed	3.58×10^6	14.3×10^6	52.3×10^6	229×10^6

Table 3.1: **Algorithm execution times**. In this Table, we show the execution times for our algorithms for different image sizes. We also note the total number of grains which need to be processed for each image. The images used are of increasing sizes, with a constant grey-level of 128. The grain radius is set to $r = 0.05$ pixels for all grains, and we use $N = 800$ Monte Carlo samples.

- Pixel-wise, no parallelisation;
- Pixel-wise, with parallelisation (OpenMP) on a multi-core CPU;
- Pixel-wise, with parallelisation on a GPU.

The machine used for these tests had four Intel Xeon 2.00 GHz processors, each with ten cores (for the purposes of parallelisation on the CPU). The pixel-wise algorithm was implemented on a GPU in CUDA using an Nvidia Tesla T10 graphics card. This GPU implementation is based on the publicly available source code of [66].

Table 3.1 shows the execution times for our algorithms. For these experiments, we rendered film grain on images of increasing sizes, whose grey-level values are equal to 128 everywhere. We set the grain radius to $r = 0.05$ pixel. We have shown the execution times of the parallelised versions of our code, and show our execution times with and without this acceleration. It can be seen that, for a fixed constant grey-level, the complexity of our algorithm is linear with respect to the number of pixels in the input image. We observe that it is possible to achieve interactive execution times with the GPU implementation of our algorithm in the fixed-radius case.

In Figure 3.5, we analyse the execution times of our algorithms when the grain radii are variable. As in the rest of this work, the distribution of the radii is a log-normal distribution. The standard deviation of the grain radii are set to a certain fraction $a \in [0, 1)$ of the average grain radius. Naturally, the fastest results are achieved with the pixel-wise algorithm on the GPU. This is several order of magnitudes faster than the grain-based approach without parallelization. Another observation is that, with equal processing power, the pixel-based approach is preferable to the grain-based one when the grain radii are fixed. However, it becomes slower as the standard

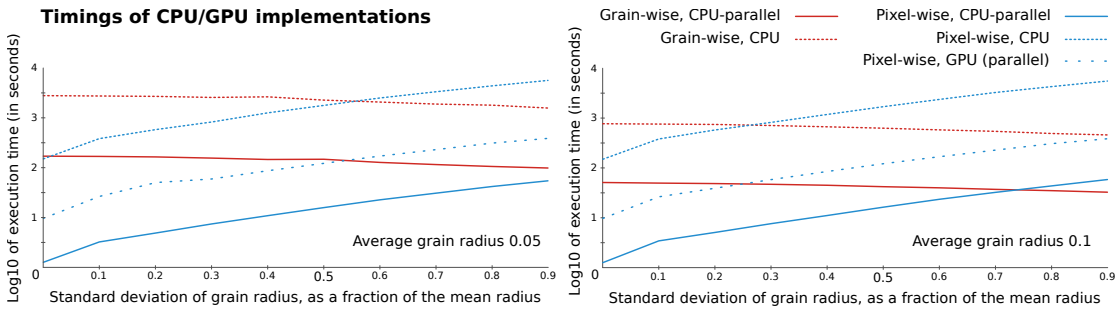


Figure 3.5: **Execution times of the proposed algorithms with increasing radius standard deviation.** The "grain-wise" approach is implemented with and without parallelization on the CPU, and the "pixel-wise" approach is implemented with and without parallelization on the CPU, and with parallelization on the GPU. The size of the image used is 1024x1024, with a constant grey-level of 128.

deviation of the radii increases, since each evaluation requires that more and more cells be visited. Therefore, these two algorithmic approaches both have strengths and weaknesses in different situations. Nevertheless, the conclusion is that the considerable processing power of the GPU leads to a faster pixel-wise algorithm, and so this implementation is preferable in all situations apart from when we use a very large standard deviation, which is rarely required.

An important parameter which has a great impact on the quality of the output is the number of iterations of the Monte Carlo approach N. The trade-off here is obviously execution time versus accuracy. From Equation (3.6), we know that the approach converges to the correct convolution. Therefore, we control the standard deviation of $\frac{1}{N}\sum_{i=1}^{N} \mathbb{1}_Z(\xi_i)$. This standard deviation is bounded by $\sqrt{\frac{1}{4N}}$. In our experiments, we set $N = 800$, giving a standard deviation of 1.77% around the average, which is an acceptable error. For faster results, the parameter N can be reduced to around 100, which gives an error of 5%. The visual quality with this parameter is still quite high. For readers who wish to reproduce similar results to those shown in this work, the following parameters are advised : $\mu_r = 0.1$, $\sigma_r = 0$ or $\sigma_r = 0.02$ (for increased graininess), $\sigma = 0.8$ and $N = 800$. The parameter s should be set on a case-by-case basis, usually $s \in \{1, 2, 3, 4\}$.

3.3.7 Results

In this section, we present the results of our film grain rendering method. Firstly, we illustrate the advantages of having a model for film grain rendering, in particular the ability to handle any given resolution. We illustrate the influence of each of our model's parameters on the visual output, and show that using these parameters, it is possible to approximate real examples of film grain emulsions. We also clearly demonstrate that independently distributed random variables are insufficient to produce realistic film grain. The C++ implementation of our algorithm is freely available online [2].

In Figures 3.6 and 3.7, we show results of our film grain rendering on some vintage images which will benefit artistically from added film grain. One of the main claims of this work is that our algorithm is resolution-free. To demonstrate this we show the capacity of our algorithm to render film grain on any digital image and at any desired resolution. In these experiments, we have used a constant grain radius $r = 0.1$ pixels. We have shown an extreme close-up of the eyes of the subjects in the images, so that the individual grains can be seen. To the best of our knowledge this

Full resolution Zoom $4\times$ Zoom $8\times$ Zoom $20\times$

Figure 3.6: **Film grain rendering results with several zoom factors**. We propose a stochastic film grain model and a film grain rendering algorithm which can render film grain on a digital image at any chosen resolution.

Full resolution Zoom $4\times$ Zoom $30\times$

Full resolution Zoom $4\times$ Zoom $30\times$

Figure 3.7: **Film grain rendering results on vintage images**. Our model and rendering algorithm allow us to create film grain at any desired resolution, which is not possible with any other grain synthesis approach. In these examples, we have used a constant grain radius $r = 0.1$.

capacity to chose any resolution is not proposed by any other grain synthesis algorithm.

Variability of graininess

A key advantage of having a model is the possibility of tuning parameters to change the visual grain aspect. Therefore, an important question is what are the parameters in our model which will allow us to imitate different types of grain? Note that what we are interested in here is

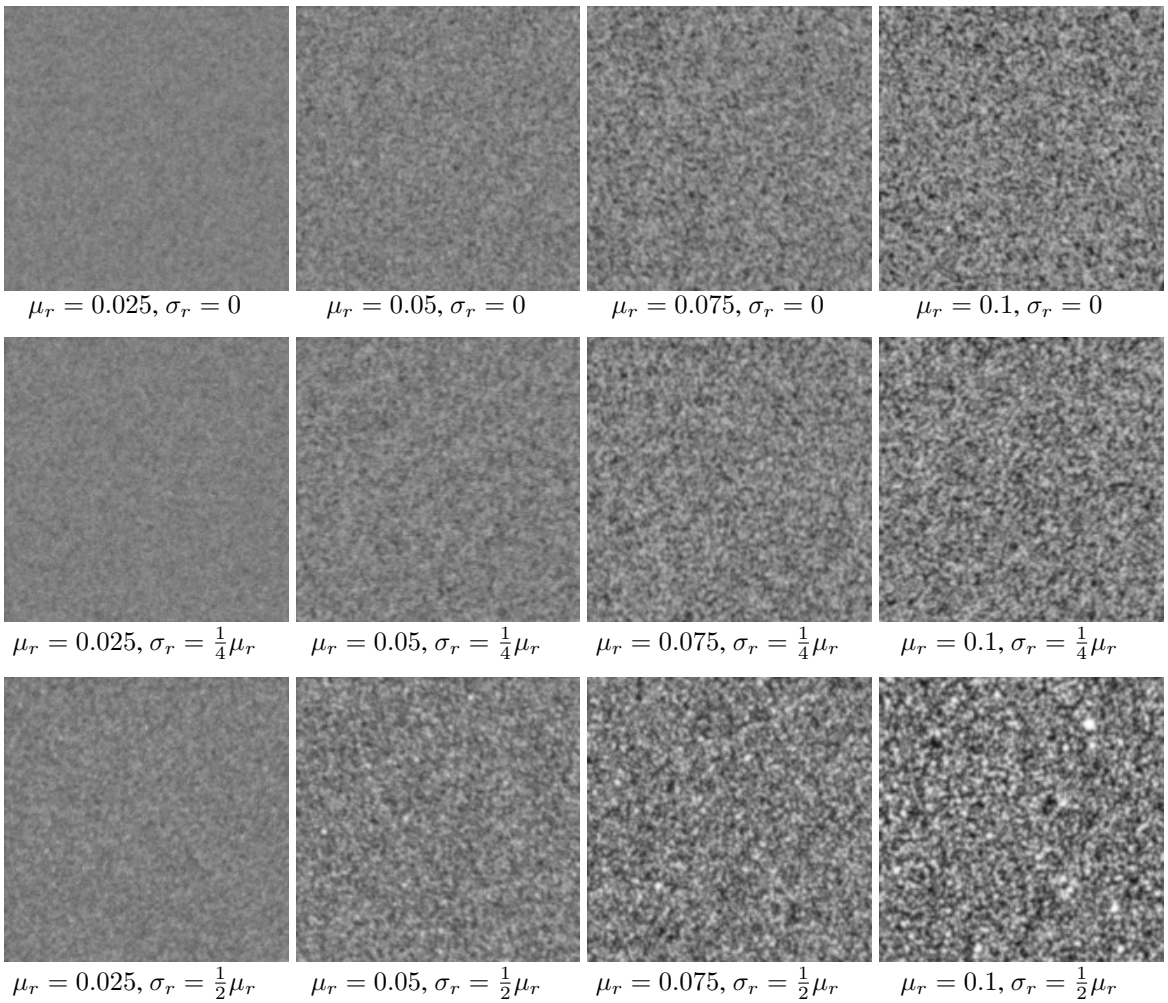


Figure 3.8: **Film grain texture with varying parameters.** In this Figure, we show the effect of varying grain size on the results of our grain synthesis. We vary the average size of the grains as well as the standard deviation of a log-normal grain distribution. It can be seen that using either constant ($\sigma_r = 0$) or random grain sizes has a significant impact on the rendering results.

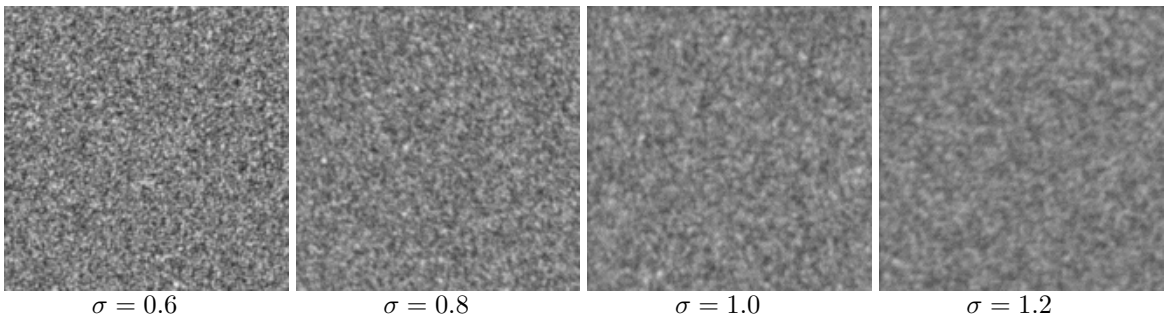


Figure 3.9: **Film grain texture with varying Gaussian blur parameter** σ. Increasing the standard deviation of the Gaussian filter results in a less pronounced graininess. In these experiments, $\mu_r = 0.05$ pixels and $\sigma_r = \frac{1}{2}r$.

the subjective visual aspect of film grain "graininess" as opposed to "granularity", which is an objective measurement of the optical density of the developed film emulsion [120].

The most important parameter which we can use to change the visual aspect of the film grain is the grain size. Since our algorithm is resolution-free we can provide accurate rendering for any grain size. Furthermore, the flexibility of our model means that we can take into account grain radii with any probability distribution (with finite variance). Let us recall that if the radii r_i are distributed according to a distribution with a mean μ_r and a variance σ_r^2 then $\mathbb{E}[A_1] = \pi(\mu_r^2 + \sigma_r^2)$ in Equation (3.3). In Figure 3.8, we show the effect of varying the grain radius. We choose a log-normal distribution, as indicated in the results reported in [120]. The mean grain radius is increased, and the standard deviation is increased as a fraction of μ_r. The use of random grain sizes as opposed to fixed sizes changes the visual aspect of the grain considerably. The log-normal distribution means that very large grains have a non-negligible chance of appearing. This distribution is defined in the following manner. Suppose that a random variable X is normally distributed with mean μ and variance σ^2, then $Y = \exp(X)$ is distributed with a log-normal distribution. If we wish to specify the effective mean μ_r and standard deviation σ_r of the grain radii, then we specify the mean and standard deviation of the underlying Gaussian distribution as:

$$\mu = \log(\mu_r) - \frac{\mu_r^2 + \sigma_r^2}{2\mu_r^2} \quad ; \quad \sigma^2 = \log\left(\frac{\mu_r^2 + \sigma_r^2}{\mu_r^2}\right). \tag{3.7}$$

Another parameter which we can tune in order to change the visual result of our algorithm is the variance of the Gaussian filter ϕ used to represent the blurring processes due to the creation and perception of the positive image (see Section 3.3.3). Figure 3.9 shows the effect of this parameter on the film grain texture, for a fixed mean radius and standard deviation. The parameter can be tuned to produce more or less sharp grain. Finally, given the flexibility of our model, we can also use different grain shapes, such as triangles. The visual effect of different shapes is not very significant, so we choose to use balls in practice. Nevertheless, we show the results with different shapes in Appendix B.2.

Grain "dithering"

In the Figure 3.10 we show a closeup comparing our film grain rendering with a compressed input image. In the input image, compression block artifacts are clearly visible. However, in the output with film grain, we have the subjective impression that the quality and resolution of the image are improved. This is linked to the well-known effect called dithering, where noise is added to a signal in order to avoid problems due to quantization. Thus, our grain rendering has the added advantage of giving a subjective impression of improved quality.

Film grain comparisons

We compare our results with those of independently distributed noise and to those of a commercially available product based on scans of real film grain. In Figure 3.11, we show the comparison of our film grain rendering with additive independently distributed noise, that is to say where each pixel acts as a "grain" that has no spatial correlation with other pixels. In this experiment, we use Gaussian noise for which the variance is signal-dependent. We learned the variance from the result of our algorithm on a series of constant images of grey level values increasing from 0 to 255. We demonstrate that the covariance of the film grain texture is one of its defining characteristics, and any model which lacks this covariance [180] will not look realistic.

Input

Film grain rendering, zoom : $\times 1$

Film grain rendering, zoom : $\times 8$

Figure 3.10: **Example of the "dithering" effect of film grain**. We show the subjective impression of increased "resolution" with a low-resolution example. This example shows that the only resolution limitation of our algorithm comes from the pixel grid of the input.

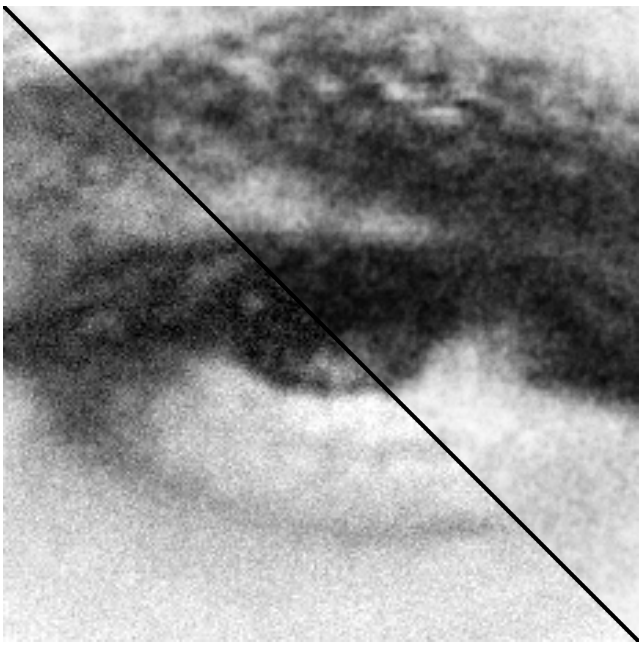


Figure 3.11: **Comparison of our film grain synthesis with signal dependent noise**. We show two closeups of images with our film grain synthesis approach vs. signal-dependent noise. The latter is clearly insufficient for realistic film grain synthesis.

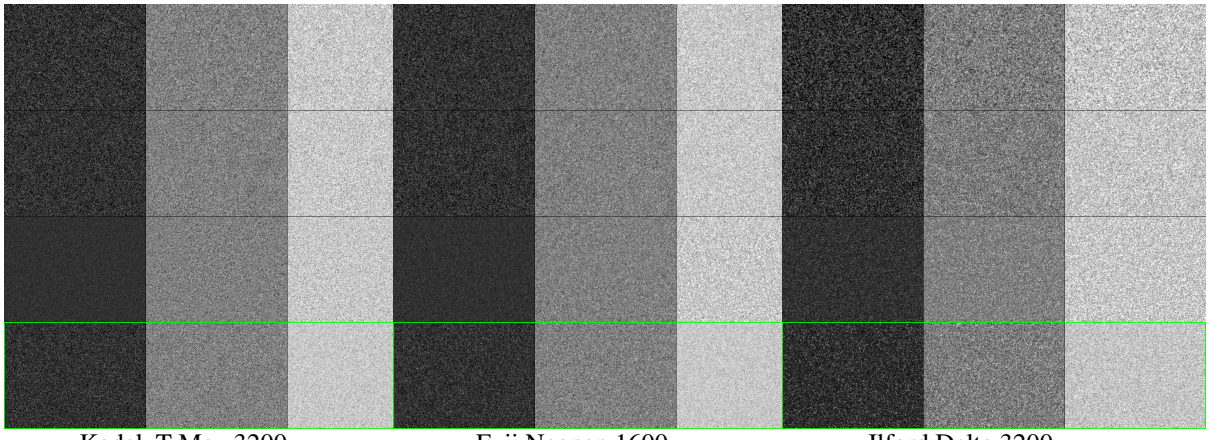


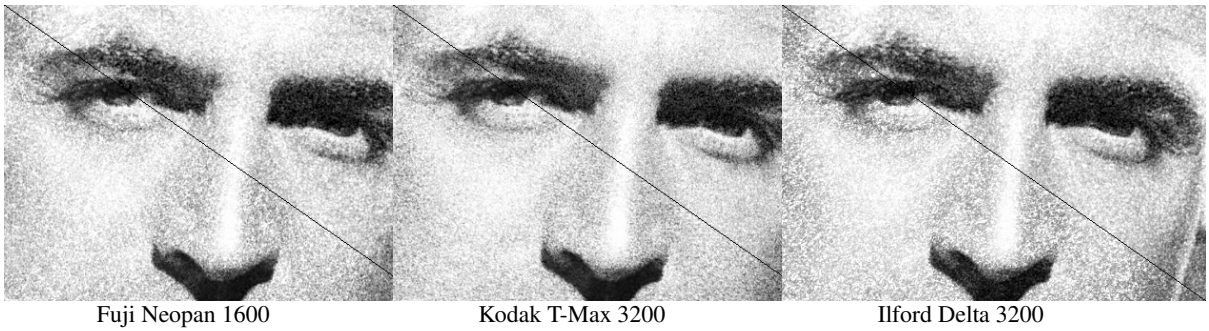
Figure 3.12: **Comparisons of film grain.** We have compared our work with three other approaches: From top to bottom: that of Stephenson and Saunders [163], that of Bae et al. [33], the result of the DxO grain tool [58] and finally the proposed approach, outlined in green. From left to right, we show images with the following constant grey-levels: 50, 128, 200. The parameters used for our results are, from left to right, ($\mu_r = 0.06$, $\sigma_r = 0.024$, $s = 4$), ($\mu_r = 0.05$, $\sigma_r = 0.024$, $s = 4$) and ($\mu_r = 0.06$, $\sigma_r = 0$, $s = 4$). The result of the DxO tool can be considered to be a reference, since they are scanned examples of film grain.

Figure 3.12 shows the results of film grain synthesis on constant images of increasing grey-levels with three other approaches: that of Stephenson and Saunders [163], that of Bae et al. [33] and using the FilmPack software of Dxo [58]. The grey-levels used are 50, 128 and 200. The first two algorithms are based on well-known approaches to texture synthesis. That of Stephenson and Saunders filters the spectrum of an input white noise, and the second employs the Heeger-Bergen [84] film grain synthesis algorithm. We used the implementation of Briand et al. [37] of the Heeger-Bergen approach. The FilmPack grain was used as an approximation of "ground truth" grain, since their algorithm is scan based. We have tuned the parameters of the other algorithms to ensure a similar visual aspect at an average grey-level of 128. In the case of the first two approaches [33, 163], neither of the papers specify how their texture is applied to an image, or either an unspecified multiplicative parameter is included to control the variance of the texture. Therefore, we set this parameter to a constant which ensured that the two methods had a similar visual aspect for the average grey-level 128. We observe that the visual grain aspect in the two first approaches is quite similar, and that their behaviour with different average grey-levels is also consistent with one other. An extremely important point to notice is that the proposed approach shows very different behavior when confronted with dark or light backgrounds. The grain is much more striking and visible in dark areas than in light ones. This behavior is the result of our physical modeling of film grain, and is a strong argument in favor of our algorithm.

In Figure 3.13, we display approximations of our algorithm of several real film emulsion types, available with the FilmPack software of DxO [58]. In these experiments, we have used an approach similar to that of DxO, in order to have meaningful comparisons. More precisely, we produce a grain image from an input image where the grey-level is equal to 128 everywhere. Then, we apply this texture additively to the input image, modulating the variance of the grain so that it attains a maximum value when the input image grey-level is 128, decreasing to zero at both grey-level extremities (0 and 255), which seems to be a similar procedure to that of FilmPack. We tune our parameters so that the result closely resembles each emulsion type. Interestingly, for the Kodak T-Max 3200 we find that constant grain sizes are most adequate, whereas for the Fuji Neopan 1600 the log-normal distribution is more visually accurate. This may reflect the fact that the size and shape of T-Max crystals are carefully controlled. These examples show that our model is realistic enough to approximate real film grain types, by tuning its physically meaningful parameters. On the other hand, in the case of "Ilford Delta 3200", it is difficult to get a precise imitation. Indeed, the white grain seems to be strangely "connected", and there does not seem to be the symmetry between black and white grain that one has with the Boolean model. This could be because of the simplicity of the Boolean model.

Colour photography

Naturally, film grain is also present in colour photography, and so we wish to synthesise this as well. Colour photographic films are made of several layers of emulsions of silver halide crystals. Silver halide crystals are naturally sensitive to blue light. This sensitivity can be extended to other wavelengths via chemical treatment. Therefore, the top layer of the emulsion consists of normal silver halide, followed by a yellow filter. This is necessary since only a fraction of the blue light is actually absorbed by the grains, and most of it would get through otherwise. The next two layers consist of crystals which have been sensitized to green and red light. Note that these layers are also sensitive to blue light, which explains the need for the yellow filter. It is a good approximation to consider that the layers of colour film interact independently with light, as each colour is only absorbed by one layer. Thus, we add film grain to a colour image by running our method independently on each colour channel. Figure 3.14 shows an example of colour film



Fuji Neopan 1600 Kodak T-Max 3200 Ilford Delta 3200

Figure 3.13: **Comparison with the DxO FilmPack software**. In this figure, we show three closeups of comparisons of our film grain rendering with different films types available in the FilmPack software of DxO. With this figure, we illustrate that our model is capable of producing film grain which closely resembles scanned images of grain. On each image, our result is shown on the upper right half, and the result of FilmPack is shown bottom left half. Please zoom on the electronic version of the paper for the best visual results.

$r = 0.05$ $r = 0.1$

$r = 0.05$, close-up $r = 0.1$, close-up

Figure 3.14: **Film grain on color images**.

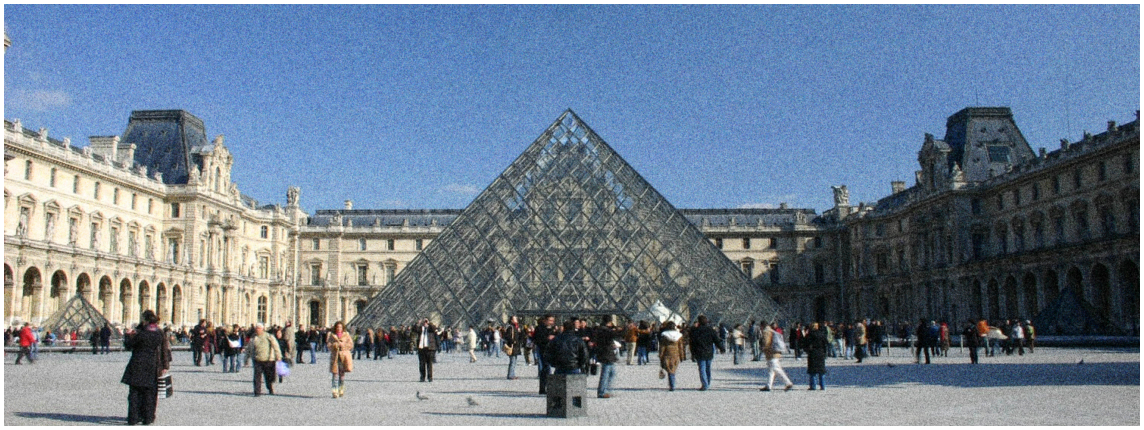


Figure 3.15: **Colour film grain on a modern image**. We show the result of our film grain rendering on a modern image, which shows that our film grain method can be used for personal artistic purposes to give photographs a certain look and feel.

grain rendering with two different grain radii. Figure 3.15 shows a modern photo which has been rendered to give it a more vintage look.

3.3.8 Conclusion

This Section has presented a film grain synthesis algorithm for digital images. This algorithm is based on the use of the Boolean model, taken from the stochastic geometry literature. This model imitates the physical reality of film grain emulsions. The density of the grains is chosen, for each pixel, to reflect the grey-level of this pixel, using the properties of the Boolean model. We have also considered how to evaluate this Boolean model on a discrete output grid. This evaluation takes the form of the convolution of a Gaussian blurring filter with the Boolean model. This process replicates what happens when we actually view film grain. Finally, we have used a Monte Carlo method to approximate this convolution, which would otherwise be difficult to determine. We present two variations on our algorithm to increase its speed, since it can be quite slow with a naïve implementation.

There are considerable advantages of this method:

- We have a physical model of film grain, which gives us meaningful parameters to tune (grain size and size distribution)
- We can easily change the shape of the grains (and have shown the result with triangles in Appendix B.2), as long as there is a function to evaluate whether a point is inside a grain or not
- The size of the output image is arbitrary, meaning we can render an image as large as we wish. This is important, since the best results are obtained at high resolutions

However, the main disadvantage is that the algorithm is quite slow, if a GPU is not used (an example from Table 3.1: around 2 minutes for an image of size 1024×1024). In order to try and simplify the algorithm, we investigated another approach, based on the Gaussian texture model, which we look at now.

3.4 A Gaussian model approximation of the stochastic film grain model

This Section looks at an *approximation* of the full, stochastic film grain model presented in Section 3.3. The goal of this approximation is to simplify and accelerate the process of film grain synthesis, while still employing a physically realistic model. This work was published in SSVM 2017 [9].

The approximation is achieved using a *Gaussian* model, that is to say we consider that an image with film grain can be modelled as a multi-variate Gaussian vector, whose parameters are the expectation vector, which we denote $\hat{\mathbf{u}}$ and covariance matrix, denoted by $\mathbf{C}$. We will manipulate the visual appearance of the image via these parameters. Once these parameters are chosen, it is straightforward to simulate a new realisation of the film grain texture. Indeed, if we denote with $\mathbf{L}$ the Cholesky decomposition of $\mathbf{C}$, such that $\mathbf{L}$ is lower-triangular and $\mathbf{L}\mathbf{L}^{\mathrm{T}} = \mathbf{C}$, then the following equation gives such a realisation:

$$\mathbf{Y} = \hat{\mathbf{u}} + \mathbf{L}\mathbf{X}, \tag{3.8}$$

where $\mathbf{X}$ is a multi-dimensional normally-distributed vector (0 average and identity covariance matrix). Indeed, the vector $\mathbf{Y}$ will have $\hat{\mathbf{u}}$ and $\mathbf{C}$ as its average and covariance, respectively. Thus, sampling is easy and fast, once the parameters are known.

We have chosen the Gaussian model for two reasons:

- Given the expected value vector and covariance matrix of the model, it is very easy to synthesise a random example of this film grain image;
- The Gaussian model is often used to create *micro-textures* [65], whose properties ressemble film grain.

To summarise, in order to establish the Gaussian model, the main task is the theoretical analysis of the grain model from a statistical point of view, more precisely, how to determine the *expected value* and *covariance matrix*. Please note that in this Section, we deal with balls of *fixed* radii. This is a limitation, but makes the analysis easier.

3.4.1 Expected Value and Covariance of the Filtered Boolean Model

Proposition 1 (Expected Value and Covariance of the Filtered Boolean Model)**.** *Consider a Boolean model Z with underlying Poisson process $\mathcal{P}$ having intensity measure $\mu : A \mapsto \int_A \lambda(t)dt$. Let ϕ represent a blurring filter. Then for all $x, y \in \mathbb{R}^2$,*

$$\mathbb{E}\left[\mathbb{1}_Z(x)\right] = 1 - \mathbb{E}\left[\mathbb{1}_{Z^c}(x)\right] = 1 - \exp\left(-\mu(\mathcal{B}_r(x))\right) \tag{3.9}$$

$$\mathrm{Cov}(\mathbb{1}_Z)(x,y) = \exp(-\mu(\mathcal{B}_r(x)) - \mu(\mathcal{B}_r(y)))\Big(\exp(\mu(\mathcal{B}_r(x) \cap \mathcal{B}_r(y))) - 1\Big). \tag{3.10}$$

Hence, due to the linearity of the convolution with filter ϕ, the expected value and covariance of the filtered Boolean model are given by

$$\mathbb{E}\ [\phi * \mathbb{1}_Z(x)] = \phi * \mathbb{E}\left[\mathbb{1}_Z\right](x) = 1 - \int_{\mathbb{R}2} \exp(-\mu(\mathcal{B}_r(x-t)))\phi(t)dt \tag{3.11}$$

$$\mathrm{Cov}(\phi * \mathbb{1}_Z)(x,y) = \int_{\mathbb{R}2}\int_{\mathbb{R}2} \mathrm{Cov}(\mathbb{1}_Z)(x-s, y-t)\phi(s)\phi(t)dsdt. \tag{3.12}$$

Proof. The second part of the proposition is straightforward, so here we give a detailed proof of Equation (3.9) and (3.10). Let us first consider the expectation. Clearly, $\mathbb{E}\left[\mathbb{1}_Z(x)\right] = 1 - \mathbb{E}\left[\mathbb{1}_{Z^c}(x)\right]$ since $\mathbb{1}_Z(x) = 1 - \mathbb{1}_{Z^c}(x)$.

Note that for any point x, $\mathbb{1}_{Z^c}(x)$ is only equal to 1 if *no* balls cover x, that is,

$$\mathbb{1}_{Z^c}(x) = \prod_{z_i \in \mathcal{P}} \mathbb{1}_{\mathcal{B}_r^c(z_i)}(x).$$

Hence one can compute $\mathbb{E}\left[\mathbb{1}_{Z^c}(x)\right]$ by invoking the following general formula. In general, for any Poisson process $\mathcal{P}$ with intensity measure Θ, and for any measurable function $f : E \to [0, 1]$, one has [155, page 65]

$$\mathbb{E}\left[\prod_{z_i \in \mathcal{P}} f(z_i)\right] = \exp\left(\int_{\mathbb{R}2} (f - 1) d\Theta\right). \tag{3.13}$$

In our case, we have $\Theta = \mu$ and $f(z) = \mathbb{1}_{\mathcal{B}_r^c(z)}(x) = \mathbb{1}_{\mathcal{B}_r^c(x)}(z)$, thus,

$$\mathbb{E}\left[\mathbb{1}_{Z^c}(x)\right] = \exp\left(\int_{\mathbb{R}2} \left(\mathbb{1}_{\mathcal{B}_r^c(x)}(z) - 1\right) \lambda(z) dz\right) = \exp\left(-\mu(\mathcal{B}_r(x))\right),$$

which proves Equation (3.9). Let us now turn to the computation of the covariance. Since $\mathbb{1}_{Z^c} = 1 - \mathbb{1}_Z$ and the covariance is invariant by the multiplication by -1 and the addition of a constant, one has $\mathrm{Cov}(\mathbb{1}_Z)(x, y) = \mathrm{Cov}(\mathbb{1}_{Z^c})(x, y)$. Now,

$$\mathrm{Cov}(\mathbb{1}_{Z^c})(x, y) = \mathbb{E}\left[\mathbb{1}_{Z^c}(x)\ \mathbb{1}_{Z^c}(y)\right] - \mathbb{E}\ \left[\mathbb{1}_{Z^c}(x)\right] \mathbb{E}\ \left[\mathbb{1}_{Z^c}(y)\right],$$

and we need to evaluate $\mathbb{E}\left[\mathbb{1}_{Z^c}(x)\ \mathbb{1}_{Z^c}(y)\right]$.

Using the above expression of $\mathbb{1}_{Z^c}(x)$

$$\mathbb{1}_{Z^c}(x)\mathbb{1}_{Z^c}(y) = \prod_{z_j \in \mathcal{P}} \mathbb{1}_{\mathcal{B}_r^c(x) \cap \mathcal{B}_r^c(y)}(z_j)$$

Using again Equation (3.13) with $f(z) = \mathbb{1}_{\mathcal{B}_r^c(x) \cap \mathcal{B}_r^c(y)}(z)$ one has

$$\begin{aligned}\mathbb{E}\left[\mathbb{1}_{Z^c}(x)\mathbb{1}_{Z^c}(y)\right] &= \exp\left(\int_{\mathbb{R}^2} (\mathbb{1}_{\mathcal{B}_r^c(x) \cap \mathcal{B}_r^c(y)}(z) - 1)\ \lambda(z)\ dz\right) \\ &= \exp(-\mu(\mathcal{B}_r(x) \cup \mathcal{B}_r(y))).\end{aligned}$$

Hence,

$$\begin{aligned}\mathrm{Cov}(\mathbb{1}_Z)(x, y) &= \exp(-\mu(\mathcal{B}_r(x) \cup \mathcal{B}_r(y))) - \exp\left(-\mu(\mathcal{B}_r(x))\right) \exp\left(-\mu(\mathcal{B}_r(y))\right) \\ &= \exp(-\mu(\mathcal{B}_r(x)) - \mu(\mathcal{B}_r(y)))\Big(\exp(\mu(\mathcal{B}_r(x) \cap \mathcal{B}_r(y))) - 1\Big).\end{aligned}$$

□

Before continuing, there are a few interesting theoretical points concerning the model which we summarise now:

- In terms of covariance, the "positive" and "negative" Boolean grain models are equivalent. In other words, the covariance of the texture produced in dark regions or light regions will be symmetric with respect to the "middle" gray-level;

- The covariance is dependent on the input image gray-level, which means that methods that use scanned examples of film grain (the case for some commercially available tools) are inherently incorrect;
- In the case of the *unfiltered* Boolean model, when $||x-y|| \geq 2r$, we have $\mathrm{Cov}(\mathbb{1}_Z(x), \mathbb{1}_Z(y)) = 0$. This is coherent with what we expect from the Boolean model, and will be useful further on.

There is another interesting observation to make here. If we calculate the variance of the Boolean model, we find:

$$\begin{aligned}
\mathrm{Var}(\mathbb{1}_Z)(x) &= \mathrm{Cov}(\mathbb{1}_Z)(x,x) \\
&= \exp(-\mu(\mathcal{B}_r(x)) - \mu(\mathcal{B}_r(x)))\Big(\exp(\mu(\mathcal{B}_r(x) \cap \mathcal{B}_r(x))) - 1\Big) \\
&= \exp(-2\mu(\mathcal{B}_r(x)))\Big(\exp(\mu(\mathcal{B}_r(x))) - 1\Big) \\
&= \exp\left(-2\frac{1}{\pi r^2}\pi r^2 \log\left(\frac{1}{1-u(x)}\right)\right)\left(\exp\left(\frac{1}{\pi r^2}\pi r^2 \log\left(\frac{1}{1-u(x)}\right)\right) - 1\right) \\
&= \exp\left(-2log\left(\frac{1}{1-u(x)}\right)\right)\left(\exp\left(log\left(\frac{1}{1-u(x)}\right)\right) - 1\right) \\
&= 1 - u(x) - \big((1-u(x))^2\big) \\
&= (1-u(x))u(x).
\end{aligned} \tag{3.14}$$

The derivative of this variance is equal to 0 when $u(x) = \frac{1}{2}$. Therefore, the variablility of the grain is maximal for grey levels precisely in the middle of the intensity range. This is coherent, and a useful property to know. In particular, it means that if we did decide to use a method based on scanning an example of film grain, the best approach would be to take a photo of a constant grey-level scene, with a middle grey-level.

3.4.2 Gaussian approximation of the filtered Boolean model

The second main objective of this work is to propose an approximation of the filtered Boolean model using Gaussian textures. This requires the evaluation of the expected value and the covariance of the model for all pixels on a grid. Unfortunately, the expressions given in Equation (3.11) and Equation (3.12) cannot be evaluated exactly. However, we can approximate them using a Monte Carlo integration.

Monte Carlo Integration for Approximating the Expected Value and Covariance of the Filtered Boolean Model

We will carry out two Monte Carlo integrations, one for the expected value, and one for the covariance. Let M and N be the number of samples for these Monte Carlo integrations, and $\{\xi_1 \dots \xi_M\}$ and $\{\xi'_1 \dots \xi'_N\}$ be two sequences of independently and identically distributed (i.i.d.) standard normal variables. Using the law of large numbers, we have

$$\frac{1}{M}\sum_{k=1}^{M} \exp[-\mu(\mathcal{B}_r(x - \xi_k))] \xrightarrow[N\to+\infty]{} \mathbb{E}[\phi * \mathbb{1}^c_Z(x)], \tag{3.15}$$

almost surely. This gives us a straightforward method to estimate $\mathbb{E}[\phi * \mathbb{1}_Z(x)]$. We now consider the approximation of the covariance function. Recall that the final goal of this is to create a covariance matrix which will be used to produce an output image with the same covariance as a filtered Boolean grain model.

Definition 1. *We define the approximate covariance function* $\mathrm{Cov}_N(x, y)$ *as the approximation of* $\mathrm{Cov}(\phi * \mathbb{1}_Z)$ *evaluated at the couple of positions* (x, y)

$$\mathrm{Cov}_N(x, y) = \frac{1}{N^2} \sum_{k,\ell=1}^{N} \mathrm{Cov}(\mathbb{1}_Z)(x - \xi'_k, y - \xi'_\ell). \tag{3.16}$$

Proposition 2. *The function* Cov_N *is symmetric, positive semidefinite, and* $\mathrm{Cov}_N(x, y)$ *converges almost surely towards* $\mathrm{Cov}(\phi * \mathbb{1}_Z)(x, y)$ *when* $N \to +\infty$.

Proof. The proof of symmetry is direct. For the positivity, we have to check that for every integer d, every $(\alpha_1, \ldots, \alpha_d) \in \mathbb{R}^d$ and every $(x_1, \ldots, x_d) \in (\mathbb{R}^2)^d$, $\sum_{i,j=1}^{d} \alpha_i \alpha_j \mathrm{Cov}_N(x_i, x_j) \geq 0$. Now, using Bienaymé's identity, we have for fixed values of $\xi'_1, \ldots, \xi'_N$,

$$\begin{aligned}
\sum_{i,j=1}^{d} \alpha_i \alpha_j \mathrm{Cov}_N(x_i, x_j) &= \sum_{i,j=1}^{d} \alpha_i \alpha_j \frac{1}{N^2} \sum_{k,\ell=1}^{N} \mathrm{Cov}(\mathbb{1}_Z)(x_i - \xi'_k, x_j - \xi'_\ell) \\
&= \frac{1}{N^2} \sum_{k,\ell=1}^{N} \sum_{i,j=1}^{d} \alpha_i \alpha_j \mathrm{Cov}(\mathbb{1}_Z)(x_i - \xi'_k, x_j - \xi'_\ell) \\
&= \frac{1}{N^2} \sum_{k,\ell=1}^{N} \mathrm{Var}\left(\sum_{i=1}^{d} \alpha_i \mathbb{1}_Z(x_i - \xi'_k) \right) \geq 0.
\end{aligned} \tag{3.17}$$

As for the convergence, a direct application of the strong law of large numbers for u-statistics [86] shows that the part of this sum containing only couples (k, l) of distinct integers ($k \neq l$) converges almost surely towards its expectation $\mathrm{Cov}(\phi * \mathbb{1}_Z)(x, y)$ when $N \to +\infty$. Since the part of the sum composed of couples (k, k) is bounded by $\frac{N}{N^2}$, the whole sum converges almost surely towards the desired covariance. □

3.4.3 Gaussian Texture Approximation for Grain on an Input Image

As previously mentioned, we propose to approximate analog film grain with a Gaussian texture, the latter being especially good at modelling "micro-textures" [64], of which film grain is a very good example. Recall that u denotes the input image defined over the image grid $\{0, \ldots, m-1\} \times \{0, \ldots, n-1\}$ and its associated filtered Boolean model $\phi * \mathbb{1}_Z$. By computing approximations of the expected value and covariance of this model on the grid, we can produce Gaussian vectors which approximate the filtered Boolean model. These Gaussian vectors will be the output images of our algorithm. In the following, we list the pixel coordinates as $\{p_i\}$ with $i \in \{0, \ldots, mn-1\}$, and $p_i \in \mathbb{R}^2$. Vectors and matrices will be denoted with bold font. The approximation of the expectation $\mathbb{E}[\phi * \mathbb{1}_Z(p_i)]$ on a pixel p_i of the image grid is denoted by $\hat{\mathbf{u}}_\mathbf{i}$ and computed thanks to the Monte Carlo integration (3.15)

$$\hat{\mathbf{u}}_\mathbf{i} = 1 - \frac{1}{M} \sum_{k=1}^{M} \exp[-\mu(\mathcal{B}_r(p_i - \xi_k))].$$

In order to compute this sum, we consider first of all the following vectors:

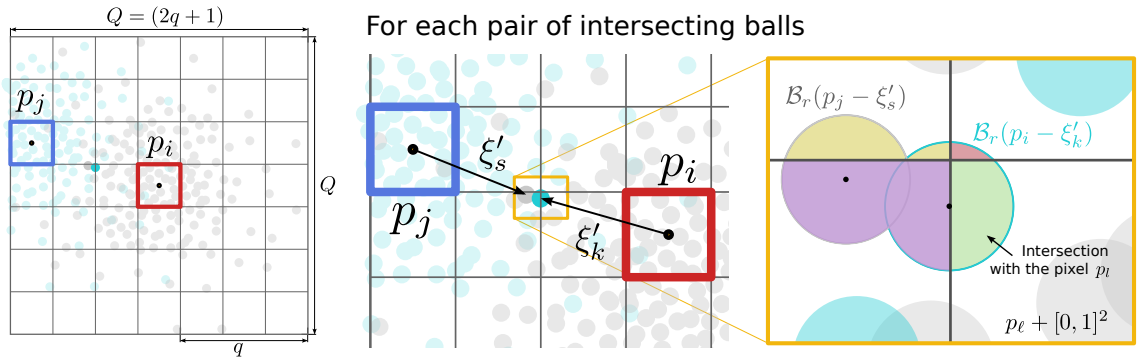


Figure 3.16: **Illustration of the approximation of covariance of filtered Boolean model**. In this Figure, we illustrate the manner in which the covariance is approximated, using a Monte Carlo simulation.

- $\boldsymbol{\lambda} \in \mathbb{R}^{mn}$ such that $\boldsymbol{\lambda}_i = \frac{1}{\pi r^2} \log(\frac{1}{1-u(p_i)})$;
- $\mathbf{1}$: a vector of ones.

Next, we define the matrix $\mathbf{A}^{p_i} \in \mathbb{R}^{M,mn}$, with $p_i \in \mathbb{R}^2$ such that

$$\mathbf{A}^{p_i}_{k,\ell} = \mathcal{A}(\mathcal{B}_r(p_i - \xi_k) \cap (p_\ell + [0,1[^2)), \tag{3.18}$$

where $\mathcal{A}$ is the Lebesgue measure in $\mathbb{R}^2$. In other words, $\mathbf{A}^{p_i}_{k,\ell}$ is the area of the part of the disk $\mathcal{B}_r(p_i - \xi_k)$ which is contained in the pixel region $p_\ell + [0,1[^2$. Using this matrix, one has $\mu(\mathcal{B}_r(p_i - \xi_k)) = \mathbf{A}^{p_i}_{k,\cdot}\boldsymbol{\lambda}$, that is, computing the intensity measure of the ball $\mathcal{B}_r(p_i - \xi_k)$ boils down to a matrix-vector multiplication. Thus, the vector $\hat{\mathbf{u}}$ which approximates the expected value of the filtered Boolean model can be written

$$\hat{\mathbf{u}}_i = 1 - \frac{1}{M}\mathbf{1}^T \exp[-\mathbf{A}^{p_i}\boldsymbol{\lambda}]. \tag{3.19}$$

We now turn to the covariance matrix $\mathbf{C}$. The entry (i,j) of $\mathbf{C}$ is defined as the approximate covariance function evaluated at the points (p_i, p_j). In short, $\mathbf{C}_{i,j} = \mathrm{Cov}_N(p_i, p_j)$. Similarly to the case of the expectation, we define the matrices $\mathbf{B}^{p_i}$, $\mathbf{D}^{p_j}$ and $\mathbf{D}^{p_i \cap p_j}$ (this time in $\mathbb{R}^{N^2 \times mn}$), such that, for $(k,s) \in \{0,\dots,N-1\} \times \{0,\dots,N-1\}$

$$\begin{aligned}
\mathbf{B}^{p_i}_{k+Ns,\ell} &= \mathcal{A}(\mathcal{B}_r(p_i - \xi'_k) \cap (p_\ell + [0,1[^2)) \\
\mathbf{D}^{p_j}_{k+Ns,\ell} &= \mathcal{A}(\mathcal{B}_r(p_i - \xi'_s) \cap (p_\ell + [0,1[^2)) \\
\mathbf{D}^{p_i \cap p_j}_{k+Ns,\ell} &= \mathcal{A}(\mathcal{B}_r(p_i - \xi'_k) \cap \mathcal{B}_r(p_i - \xi'_s) \cap (p_\ell + [0,1[^2)).
\end{aligned} \tag{3.20}$$

Finally, given these matrices, we can define the entry (i,j) of $\mathbf{C}$ as

$$\mathbf{C}_{i,j} = \frac{1}{N^2}\mathbf{1}^T \exp\left(-\left(\mathbf{B}^{p_i} + \mathbf{D}^{p_j}\right)\boldsymbol{\lambda}\right) \odot \left[\exp\left(\mathbf{D}^{p_i \cap p_j}\boldsymbol{\lambda}\right) - \mathbf{1}\right], \tag{3.21}$$

where $\odot$ represents the Hadamard (element-wise) vector product. Proposition 2 ensures that $\mathbf{C}$ is symmetric semi-definite positive. The covariance approximation process is illustrated in Figure 3.16. An interesting feature is that we can precompute these area matrices for a given parameter set, since they are independent of the image u. Furthermore, it seems reasonable to assume that

Algorithm 5 Film grain rendering algorithm with Gaussian texture.

Data: $u : \{0, 1, \dots m-1\} \times \{0, 1, \dots, n-1\} \to [0, u_{max}]$: input image
Parameters:
σ: standard deviation of the Gaussian low-pass filter
P_α: Gaussian $(1-\alpha)$th quantile
N: number of iterations in the Monte Carlo method
Result: Image rendered with film grain

$\mathbf{X} \sim \mathcal{N}(0, \mathbf{I}^{mn,mn})$
$q = \lfloor 2(P_\alpha + r) \rfloor$
$\xi, \xi' \leftarrow$ i.i.d. r.v. with Gaussian density truncated at P_α
$\Psi_0 \leftarrow$ computeLocalNeighbourhood(q)
Load or compute the area matrices for this parameter set
$\{\mathbf{A}^0, \mathbf{B}^0, \dots, \mathbf{D}^{\max(\Psi_0)}, \dots, \mathbf{D}^{0 \cap \max(\Psi_0)}\} \leftarrow \text{AreaMatrices}(\xi, \xi', r, \sigma)$
foreach $(i,j) \in \{0, \dots, mn-1\} \times \{0, \dots, mn-1\}$ *s.t.* $\text{Cov}(\phi * \mathbb{1}_Z)(p_i, p_j) \neq 0$ **do**
  $\boldsymbol{\lambda}^{p_i} \leftarrow u(\Psi_{x_i})$
  $\hat{\mathbf{u}}_i \leftarrow \frac{1}{M} \mathbf{A}^0 \boldsymbol{\lambda}^{p_i}$
  $\mathbf{C}_{p_i,p_j} = \frac{1}{N^2} \mathbf{1}^T \exp\left(-\left(\mathbf{B}^0 + \mathbf{D}^{p_j - p_i}\right) \boldsymbol{\lambda}^{p_i}\right) \odot \left[\exp\left(\mathbf{D}^{0 \cap (p_j - p_i)} \boldsymbol{\lambda}^{p_i}\right) - \mathbf{1}\right]$
$\mathbf{L} = \text{Chol}(\mathbf{C})$
return($\hat{\mathbf{u}} + \mathbf{L}\mathbf{X}$)

the covariance will be zero for couples (p_i, p_j) which are further apart than a certain distance, by choosing a blurring kernel ϕ with compact support. In practice, we choose a truncated Gaussian for ϕ , which is truncated at the value P_α such that, for $\xi \sim \mathcal{N}(0,1)$,

$$\mathbb{P}\left[\xi \in [-P_\alpha, P_\alpha]^2\right] = 1 - \alpha, \tag{3.22}$$

for some small parameter α. This is the $(1 - \frac{\alpha}{2})$*th* quantile of the Gaussian distribution. Now, recall that for any couple (p_i, p_j) we have

$$\mathcal{B}_r(p_i) \cap \mathcal{B}_r(p_j) = \emptyset \implies \text{Cov}(\mathbb{1}_Z(p_i), \mathbb{1}_Z(p_j)) = 0. \tag{3.23}$$

This equation, combined with the fact that our Gaussians are truncated at P_α implies that for a couple (p_i, p_j) we have

$$||p_i - p_j||_2 > 2(P_\alpha + r) \implies \mathbf{C}_{i,j} = 0. \tag{3.24}$$

Let us denote with q the maximum output pixel distance for which the covariance is non-zero. This distance is

$$q = \lfloor 2(P_\alpha + r) \rfloor. \tag{3.25}$$

Let $Q = (2q + 1)$. For any pixel p_i, the (non-zero) covariance values are therefore limited to a square neighbourhood Ψ_{p_i} of size Q^2.

Now that we have limited the extent of the covariance function, we can drastically decrease the size of the area matrices. Furthermore, these matrices only depend on the relative position of p_j with respect to p_i. Therefore, we can set p_i to be the "origin" 0 and $p_j - p_i \in \Psi_0$. In this case, we only need to calculate the matrices $\mathbf{A}^0$, $\mathbf{B}^0$, $\mathbf{D}^{p_j - p_i}$, and $\mathbf{D}^{0 \cap p_j - p_i}$. Let $\boldsymbol{\lambda}^{p_i}$ represent the values of $\boldsymbol{\lambda}$ in the neighbourhood Ψ_{p_i}. We can now rewrite the expected value and covariance using this reduced number of vectors and matrices

$$\mathbf{C}_{i,j} = \frac{1}{N^2} \mathbf{1}^T \exp\left(-\left(\mathbf{B}^0 + \mathbf{D}^{p_j - p_i}\right) \boldsymbol{\lambda}^{p_i}\right) \odot \left[\exp\left(\mathbf{D}^{0 \cap (p_j - p_i)} \boldsymbol{\lambda}^{p_i}\right) - \mathbf{1}\right]. \tag{3.26}$$

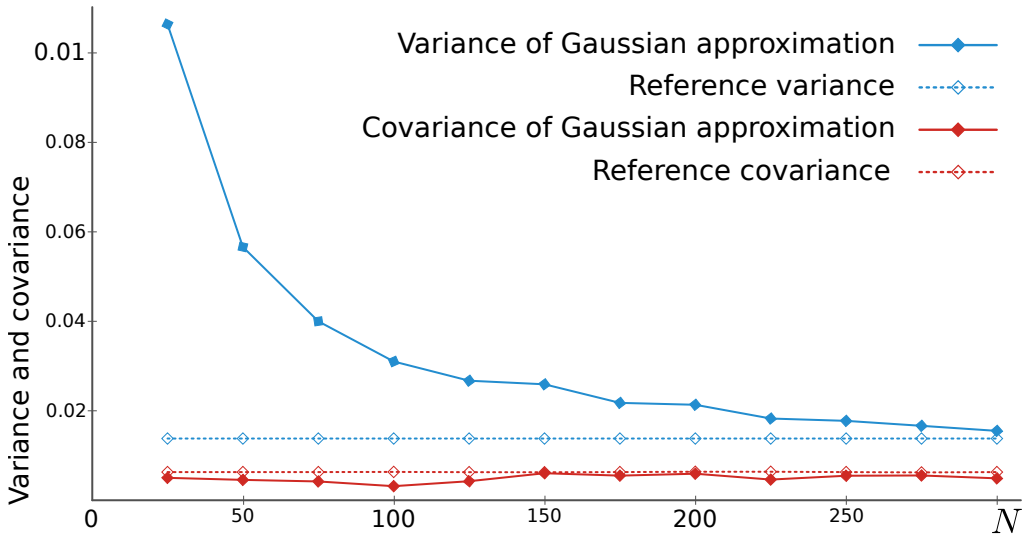


Figure 3.17: **Analysis of variance and covariance of the Gaussian approximation of the Boolean model**. In this Figure, we analyse the evolution of the variance and covariance of the Gaussian approximation of the Boolean model, as N increases. As predicted, for small values of N, the approximation is biased, due to the couples (ξ'_k, ξ'_k) in the Monte Carlo simulation. This effect diminishes as N increases. The "reference" variance and covariance are determined empirically using the original model and algorithm proposed in Section 3.3.

This is the final expression of the covariance matrix used in our algorithm. Once the positive semi-definite covariance matrix $\mathbf{C}$ and expected value $\hat{\mathbf{u}}$ are computed, we can easily produce Gaussian vectors with these specific expected value and covariance matrix. Indeed, consider the lower triangular matrix $\mathbf{L}$ resulting from the Cholesky decomposition of $\mathbf{C}$, such that $\mathbf{C} = \mathbf{L}\mathbf{L}^T$. For any $\mathbf{X} \in \mathbb{R}^{nm}$ following a standard Gaussian white noise, the vector $\hat{\mathbf{u}} + \mathbf{L}\mathbf{X}$ has expected value $\hat{\mathbf{u}}$ and covariance matrix $\mathbf{C}$.

Algorithm summary and parameters We now recap the full film grain synthesis algorithm. This consists of two stages: firstly the computation of the area matrices. These matrices can be pre-computed (for a given parameter set), and then stored in memory. This, in turn, requires the areas of disks and intersections of disks in the ranges of the pixels of the image grid. These areas are calculated with the geometry software CGAL [167]. The second part of the full method calculates the non-zero elements of the matrix $\mathbf{C}$, carries out the Cholesky decomposition on the latter, and then produce the output image. The complete algorithm is presented in Algorithm 5. In the experiments shown in the next section, we use the following parameters: $\sigma = 0.8$, $M = 800$, $N = 200$. The radius parameter r is varied to show different grain qualities.

3.4.4 Results

In this Section, we show some visual and numerical results of our algorithm. The first step is to verify experimentally that our Monte Carlo approach converges to the correct statistics of the Boolean model. One particular drawback of our approach is that, since we must use the same Gaussian offsets ξ'_i (in order to ensure symmetry and positive-definiteness), there is a list of couples (ξ'_k, ξ'_k) which are not i.i.d, but whose influence on the approximation diminishes as $N \to \infty$. With

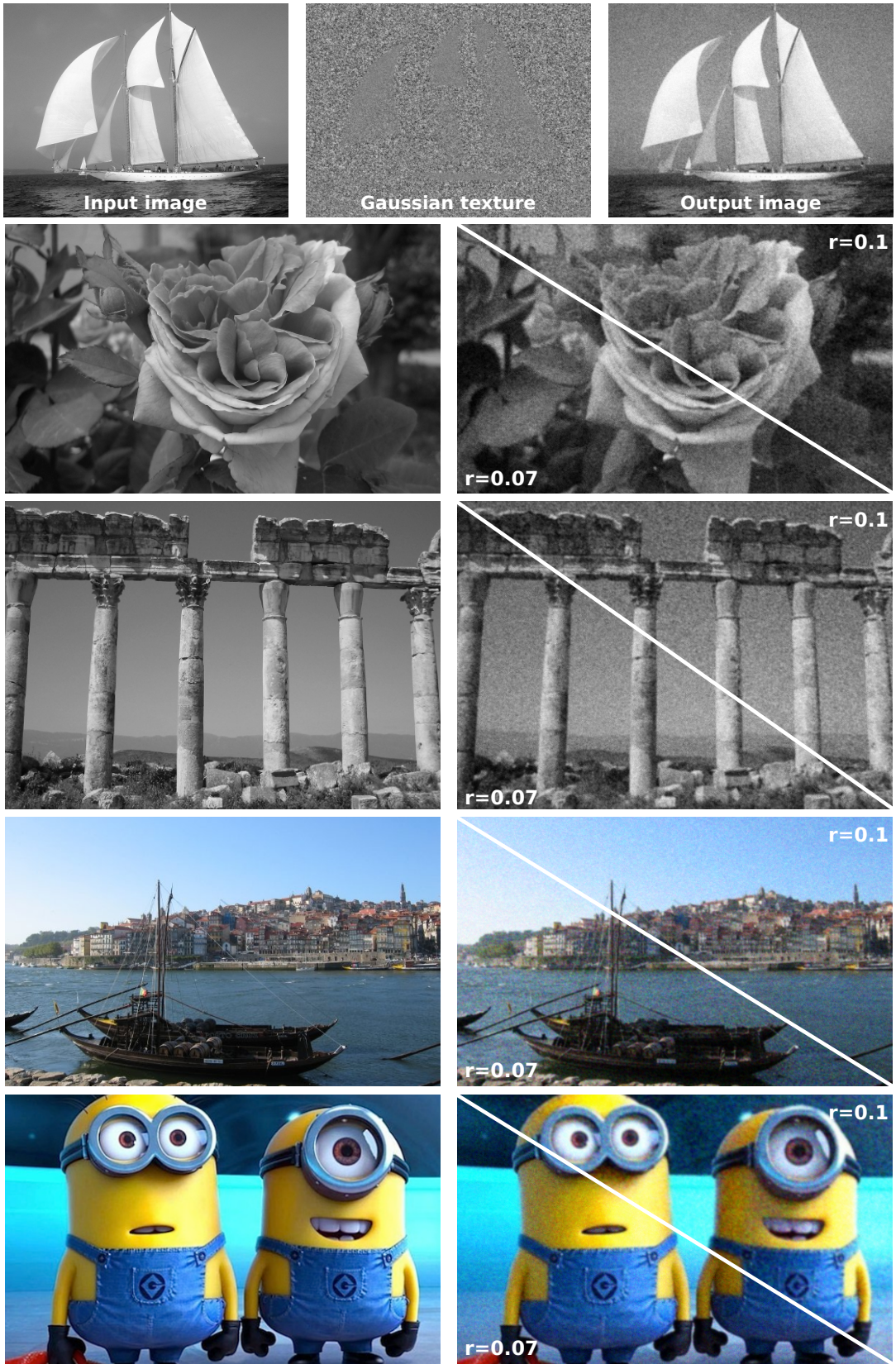


Figure 3.18: **Gaussian approximation of film grain.** In this Figure, we show a result of our film grain algorithm on several input images, in grey-scale and color. In the first example of the boat, we show the "pure" texture $\mathbf{LX}$ which is added to the image. This texture has a variance which is maximized in the areas of middling grey-level (the sky), and is minimal in the areas of extreme gray-level (the boat's sails, for example). This is coherent with the result of Equation (3.14).

small values of N, this influence is significant, since the quantity $\mathrm{Cov}(\mathbb{1}_Z(p_i - \xi'_k), \mathbb{1}_Z(p_j - \xi'_k))$ (see Equation (3.16)) is maximized precisely when $i = j$, and with small N there are not enough samples to "rectify" this bias. We note, however, that this problem is mostly restricted to the case of the *variance*. Indeed, for (x, y), s.t. $||p_i - p_j||_2 > 2r$, the quantity $\mathrm{Cov}(\mathbb{1}_Z(p_i - \xi'_k), \mathbb{1}_Z(p_j - \xi'_k))$ is *necessarily* equal to 0. Only in the case of large zooms and/or large radii, will we have a non-zero influence of the couples (ξ'_k, ξ'_k). Thus, the convergence of the covariance is much faster, and indeed changes very little as N increases. This is confirmed by numerical experiments shown in Figure 3.17, where we analyse the evolution of the values of the variance and covariance of the Monte Carlo approach, as N increases. These values are determined on a constant image, equal to 0.5 everywhere, and we compare the values to a "reference" value determined empirically with the result of the original film grain synthesis presented in Section 3.3. The covariance shown is that of two vertically adjacent pixels. This gives us an idea of how large N should be, and also serves as a strong sanity check that our approach is indeed correct. In Figure 3.18, we show a result of our algorithm on several input images. In order to provide some means of "objective" validation of the proposed grain, we show the result of $\mathbf{LX}$ in the middle of the top row. This represents the "pure" texture (the variance and covariance of the Gaussian approximation). This texture is coherent with what we expect, since the variance is maximal in the areas of medium gray-level (see Equation (3.14)), such as the sky. In areas with more extreme gray-levels, the variance is lower (the texture is smoother), which is coherent with the Boolean model. This serves as a verification that the Gaussian approximation indeed displays the characteristics which we are looking for. Finally, we have shown an example of film grain on an animation image to illustrate the kind of visual style which can be achieved with our approach even on modern images.

Now, in this part of the work, we would naturally like to compare this algorithm to the full Boolean model presented in Section 3.3. Unfortunately, while the Gaussian algorithm produces good visual results, it is as yet (at the time of publishing) limited to medium-sized images (maximum 512×512) due to memory limitations. We noted that large zooms or grain radii induce a much heavier computational load, since we need to pre-compute much larger area matrices to use in Equation (3.26). This is problematic, since much of the visual power of the original algorithm is precisely in the case of zooms. To take a concrete example, let us consider the following parameters: $r = 0.1$, $s = 1$ (no zoom), image size 512×512, $\alpha = 0.01$, $\sigma = 0.8$. This gives a maximum non-zero covariance area of 11×11 ($Q = 11$). Thus, if $\mathbf{C}$ is stored as a sparse matrix, we need $512 \times 512 \times 121$ entries, which requires 32 MB of memory. However, if we increase the zoom to $10\times$, this leads to a non-zero covariance area of 110×110, giving a final covariance matrix of $512 \times 512 \times 12100$ entries, requiring 3 GB of memory. While this is manageable, it is clearly not a particularly efficient solution. In actual fact, I tried this on my (five-years) old laptop with just a zoom of $5\times$, and the process was killed. So, again, this is not a perennial solution. I will discuss potential solutions to this in the next Section.

3.5 Film grain synthesis conclusion

Film grain synthesis is a much more complicated problem than it seems at first glance. Indeed, the first approach is simply to use blurred Gaussian noise [163]. Unfortunately, film grain is dependent on the underlying image. This is shown by our Boolean model, where theoretical analysis shows that the covariance between two pixels is indeed linked to the grey-levels of these pixels. We also found that, under the Boolean model, the highest variance (and thus the highest graininess), is found when the grey-level is in the middle of the contrast range.

It is clear that the main disadvantage of both approaches is computational complexity. In the

case of the full Boolean model, both the pixel-wise and the grain-wise algorithms are limited, mostly in terms of execution time. Parallelisation of the algorithms alleviate this to some extent, but it seems rather extreme to require a GPU for the apparently simple task of grain synthesis.

Thus, to carry on this work, the main future goal will be to propose faster, versions of these algorithms. One option would be to continue to the Gaussian texture approach. For this, instead of calculating the mean and covariance using a Monte Carlo approach, we could train a neural network to predict them given a local image patch. This makes sense, because the covariance of the filtered Boolean model can be considered to be zero at a certain distance. Another approach, would be to take the initial film grain algorithm, and use it to create a database of grainy images. Again, we could use a neural network to produce an image which contains the same type of texture as the reference images, using an approach similar to that of Gatys *et al.* [67] or a variant thereof, for example that of Ulyanov *et al.* [168]. These approaches posit that the texture of an image can be described by the correlations between channels of their representation inside of a deep neural network. They perform extremely well, so it is likely that good results could be obtained.

The main challenge with the aforementioned possibilities is how to integrate the different parameters present in the original Boolean film grain model. Indeed, these parameters (zoom and grain size especially), are crucial to the quality of the final result. It is not immediately clear how the tuning of these parameters could be achieved by a neural network. The approach of Gatys would require a reference image for each set of parameters, and this is clearly not possible. However, it may be possible to modify that of Ulyanov, which is a feed-forward network, to include these parameters as inputs. Nevertheless, for extreme zooms, where the balls can start to be seen, it might be a difficult task for even a neural network.

Chapter 4

Deep learning

In this Chapter, we turn to models based on *neural networks*. These are the basis for what is known today as *deep learning*, which has revolutionised many different domains. My work on the subject has revolved around two specific types of neural networks:

- Autoencoders
- Generative Adversarial Networks

These are examples of *unsupervised* methods in deep learning. The goal of my work has been to either understand them or use them for image editing/synthesis purposes. I will explain these further on in the manuscrpit, but before doing this I will (very) briefly present neural networks in a general context.

4.1 Neural networks

4.1.1 A brief history of neural networks

Nowadays, neural networks are ubiquitous, and represent the state-of-the-art in a vast majority of image processing and computer vision tasks (among others). Neural networks, in the context of computer science, are a very old idea, dating back to the work of McCulloch and Pitts [125], who first propsed a computational model of neurons in the human brain. In the human brain, neurons, which contain some kind of electrical signal, are connected to other neurons in the brain through synapses. In the brain, these neurons only transmit a signal if the information which they have recieved (from other neurons), reaches some threshold. This corresponds to a non-linear function, applied to a linear combination of input signals. This was the model of McCulloch and Pitts, and have formed the basis of neural networks to today. Neural networks are, therefore, nothing more than a series of affine transformations and non-linear functions applied in a cascaded fashion to some input data. Cybenko [55] showed that by using this framework, it was possible to approximate a very large class of functions to any precision, given enough neurons.

A further step forward was made with the introduction of *convolutional neural networks* (CNNs). Again, this idea was based on the work in the domain of biology of Hubel and Wiesel [90], who studied the *receptive fields* of animals' cortexes. This showed that small regions in the brain corresponded to regions of the visual field. Fukushima [63] imited this by using convolutions instead of general affine transformations in a neural network, giving birth to the neocognitron. Yann LeCun *et al*. [109] first used the backpropagation algorithm to carry out automatic parameter

setting, paving the way for the deep learning revolution. This was delayed until two decades later, since the computational capacities of computers were not great enough. Once this was the case, in 2012 Krizhevsky *et al*. [103] produced the first deep convolutional neural network, which vastly improved classification scores on the ImageNet Large Scale Visual Recognition Challenge [152]. This ushered in a new era of deep learning architectures, which have taken over a wide array of domains as the leading algorithms, with image processing and computer vision being among them.

4.1.2 Notation and jargon of neural networks

As mentioned above, neural networks are simply cascades of affine transformations and non-linearities. Different architectures have various details and specificities, but the underlying principles always remain the same. We will use the following notation:

- $x \in \mathbb{R}^{m \times n}$: an input image, of size $m \times n$. Generally speaking, this can be considered as a vector, but since we deal with images here, we specify that x is an image;
- $y \in \mathbb{R}^q$: the output of the network. In classification problems, this is a vector of size the number of classes. In other problems, such as super-resolution or restoration, this is an image;
- Non-linearities: we will use the functions f, g and h for general non-linearities. For more specific ones, we detail this further on.

Please note that contrary to the previous Chapter 3, the variable x is *no longer a spatial position in an image*, rather it is itself an image. While this change of notation is regrettable, it is necessary in order to be coherent with the rest of the deep learning literature.

Neural networks are organised into *layers*. In general, a layer contains an affine transformation and a non-linearity function. There are five specific types of non-linearities which are commonly used in neural networks. These are:

- "Sigmoid" function:

$$\begin{aligned} \sigma \colon \mathbb{R} &\to (0,1) \\ x &\mapsto \frac{1}{1+e^{-x}}. \end{aligned} \tag{4.1}$$

  This is used at the end of networks for binary classification problems, since it maps a real number to the open interval $(0,1)$, ie, a probability.

- "Tanh" function, or hyperbolic tangent function:

$$\begin{aligned} \mathrm{Tanh} \colon \mathbb{R} &\to (0,1) \\ x &\mapsto \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}. \end{aligned} \tag{4.2}$$

  This is often used in the inner layers of neural networks, but is now less used.

- "Softmax" function

$$\begin{aligned} \mathrm{Softmax} \colon \mathbb{R}^K &\to (0,1)^K \\ x &\mapsto \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}. \end{aligned} \tag{4.3}$$

This is used for multi-class classification problems, where K is the number of classes, and where each class is mutually exclusive. The output $\mathrm{Softmax}(x)$ is a discrete probability distribution, since we have $\mathrm{Softmax}(x)_j \in (0,1)$ and $\sum_{j=1}^{K} \mathrm{Softmax}(x)_j = 1$.

- "ReLU" function, ie. the Rectified Linear Unit.

$$\begin{aligned} \mathrm{ReLU} \colon \mathbb{R} &\to \mathbb{R}^+ \\ x &\mapsto \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases} . \end{aligned} \tag{4.4}$$

This is commonly used in the inner layers of a neural network (ie. not at the end), and is preferred to Tanh because it avoids the vanishing gradient problem.

- "Leaky ReLU" function, ie. the Leaky Rectified Linear Unit.

$$\begin{aligned} \mathrm{LeakyReLU}_\alpha \colon \mathbb{R} &\to \mathbb{R} \\ x &\mapsto \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{otherwise} \end{cases} . \end{aligned} \tag{4.5}$$

This is commonly used in the inner layers of a neural network (ie. not at the end), and is often preferred now to the standard ReLU, because the gradient is non-zero below zero.

In the deep learning jargon, a layer containing an affine transform plus a non-linearity is known as a *fully-connected* or *dense* layer. There is a bit of ambiguity as to whether fully-connected/dense includes the non-linearity, but most often it does.

Finally, most modern deep learning methods for images use *convolutions* in their architectures. Since these are also linear transformations, this does not change anything fundamentally in the formalism of neural networks. The precise implementation of these is somewhat particular to deep learning, but we will consider it to be known here.

Let us now introduce the two architectures which I have used in my work on deep learning.

4.1.3 Autoencoders and Generative Adversarial Networks

As mentioned at the beginning of this Chapter, this part of my work is connected with image editing and synthesis. At this point in time, there are two main types of neural network architecture connected to these goals, and these are the *autoencoder* (AE) and the Generative Adversarial Network (GAN) [75]. These are two of the most popular networks today, around which much of the current research is based. Since my work in the field of deep learning also revolves around them, I will explain how they work now.

The autoencoder

The AE is a neural network consisting of two sub-networks, an encoder and a decoder. AEs have been around in various forms for some time, going back to the 1980s [27, 36, 61]. The core idea is quite simple: the encoder projects a data point x to an intermediate space, called the latent space, and the decoder projects this back to the data space, giving the output y. The network is trained so that the output ressembles the input as much as possible. In general the error $\|x - y\|_2^2$ is minimised. At first glance, it seems odd that a network would be trained to simply produce the input as its output. In fact, a simple solution would be just to set both the encoder and decoder to

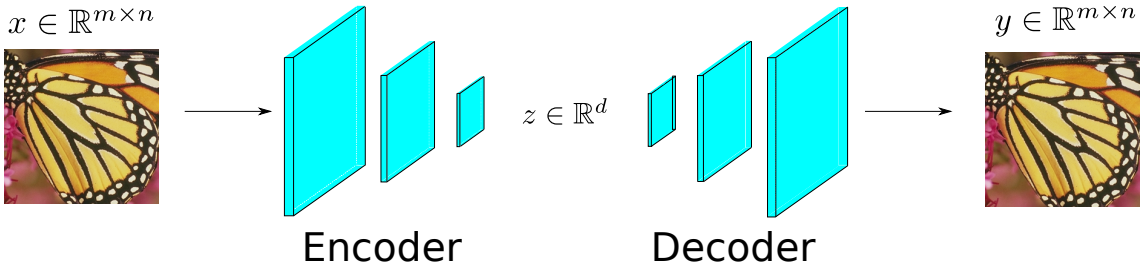


Figure 4.1: **Illustration of an autoencoder**. The encoder projects an input data point x to the, smaller, latent space, while the decoder projects the latent code z back to the data space.

the identity function. However, in general the size of the latent space is much smaller than that of the data space, meaning that the data point has in some sense been *compressed* while going through this latent space, and has been therefore distilled into a much more compact and potentially useful representation. Let us now explain this network more precisely.

We denote the data space with $\mathcal{X} = \mathbb{R}^{m \times n}$ and the latent space with $\mathcal{Z} = \mathbb{R}^d$, where d is the dimensionality of the latent space. In general, an element of the latent space will be denoted with z. We denote the encoder with

$$\begin{aligned} E\colon \mathcal{X} &\to \mathcal{Z} \\ x &\mapsto E(x). \end{aligned} \tag{4.6}$$

We denote the decoder with

$$\begin{aligned} D\colon \mathcal{Z} &\to \mathcal{X} \\ z &\mapsto D(z). \end{aligned} \tag{4.7}$$

In general, the loss applied for training an AE is the ℓ^2 norm of the difference between the input and the output:

$$\mathcal{L}_{AE} = \|x - D(E(x))\|_2^2. \tag{4.8}$$

We will refer to the parameters of the encoder with Θ_E and to those of the decoder with Θ_D. This summarises the AE. An illustration of this architecture can be seen in Figure 4.1.

The Generative Adversarial Network

The GAN is an incredibly popular architecture, proposed by Goodfellow *et al.* [75] in 2014, whose goal is to synthesise images. Contrary to texture synthesis, for example, GANs can produce a wide variety of different types of images. It is quite a generic and flexible approach.

Structurally, a GAN contains only the second half of an AE, the decoder. In the GAN jargon, it is referred to as the *generator*, so we will denote it with G, but bear in mind that it has the same definition as D. The GAN takes as input a random latent code z, almost always drawn from a multivariate normal distribution, and outputs a synthesised example of the type of image that we want. The main question is how is the GAN able to synthesise realistic-looking images if there is no AE loss to make sure that the output looks correct? The answer to this question is the main, revolutionary, innovation of the GAN: another network is trained to learn how to recognise a realistic-looking image.

More precisely, we introduce another network, the *adversarial network*, which we will denote with $\mathcal{D}$. This network is also known as the *discriminator* network, since it is trained to discriminate between real images from the database and fake images produced by G. Note that this network is most often denoted with D, but since we have already used this for the decoder, we prefer to use the caligraphical $\mathcal{D}$ to avoid confusion. Formally, we have

$$\begin{aligned} \mathcal{D}\colon \mathcal{X} &\to (0,1) \\ x &\mapsto \mathcal{D}(x). \end{aligned} \tag{4.9}$$

The generator and discriminator are trained in tandem to solve the following min/max optimisation problem

$$\min_G \max_{\mathcal{D}} \mathbb{E}_{x\in p_x}\left[\log\left(\mathcal{D}(x)\right)\right] + \mathbb{E}_{z\in p_z}\left[\log\left(1-\mathcal{D}(G(x))\right)\right], \tag{4.10}$$

where p_x and p_z are the probability distribution functions of the data and the latent code. Note that while the latent probability distribution function p_z is known and fixed, we never actually manipulate that of the data, p_x, since this data is high dimensional. We simply suppose that p_x exists.

The StyleGAN [98] architecture

In the last parts of the work on image editing in Section 4.5, we make extensive use of a specific GAN, the StyleGAN [98]. This is the basis for a family of GANs (StyleGAN2 [95], StyleGAN3 [97]) that present the best results to date (in 2022). Since their model is central to the aforementioned work, and quite a step forward in GAN architectures, we present it in detail.

The key modification of StyleGAN with respect to standard GANs is that the latent code is fed into the generator *in a parallel manner* to different resolutions. For a visual illustration of this, see Figure 4.2. More precisely, the authors of StyleGAN introduced a new, intermediate, latent space, denoted $\mathcal{W} \subset \mathbb{R}^{512}$, and an element of this space $w \in \mathcal{W}$ is fed into the generator via "Adaptive Instance Normalisation (AdaIN)" layers [88]. These operations are very similar to Batch Normalisation layers [92], and allow the latent code to control the mean and variance of the tensors within the generator. Since they are fed in parallel to different resolutions, they control attributes with different granularities in the final image.

Finally, random noise is input to each resolution in order to generate stochastic elements of the image such as hair and skin texture. These elements are difficult to integrate into the latent space, since intuitively it would require that a small displacement in $\mathcal{W}$ result in a completely new random sample of the texture, which would mean that the generator is not continuous. This goes against the nature of neural networks, which are fundamentally piece-wise linear functions.

We now turn to the different works I have carried out on the subject of deep generative models.

4.2 Understanding how autoencoders process simple geometric shapes

This work was carried out during my postdoc at Télécom Paris, in collaboration with Andrés Almansa, Saïd Ladjal and Yann Gousseau [3]. The goal of this work was to understand the precise mechanisms with which autoencoders can encode and decode simple geometric shapes. Indeed, in the AE literature, as is the case in much of deep learning, there is very little understanding of how the architectures actually work. Here, we decided to use an AE to encode and decode the simplest

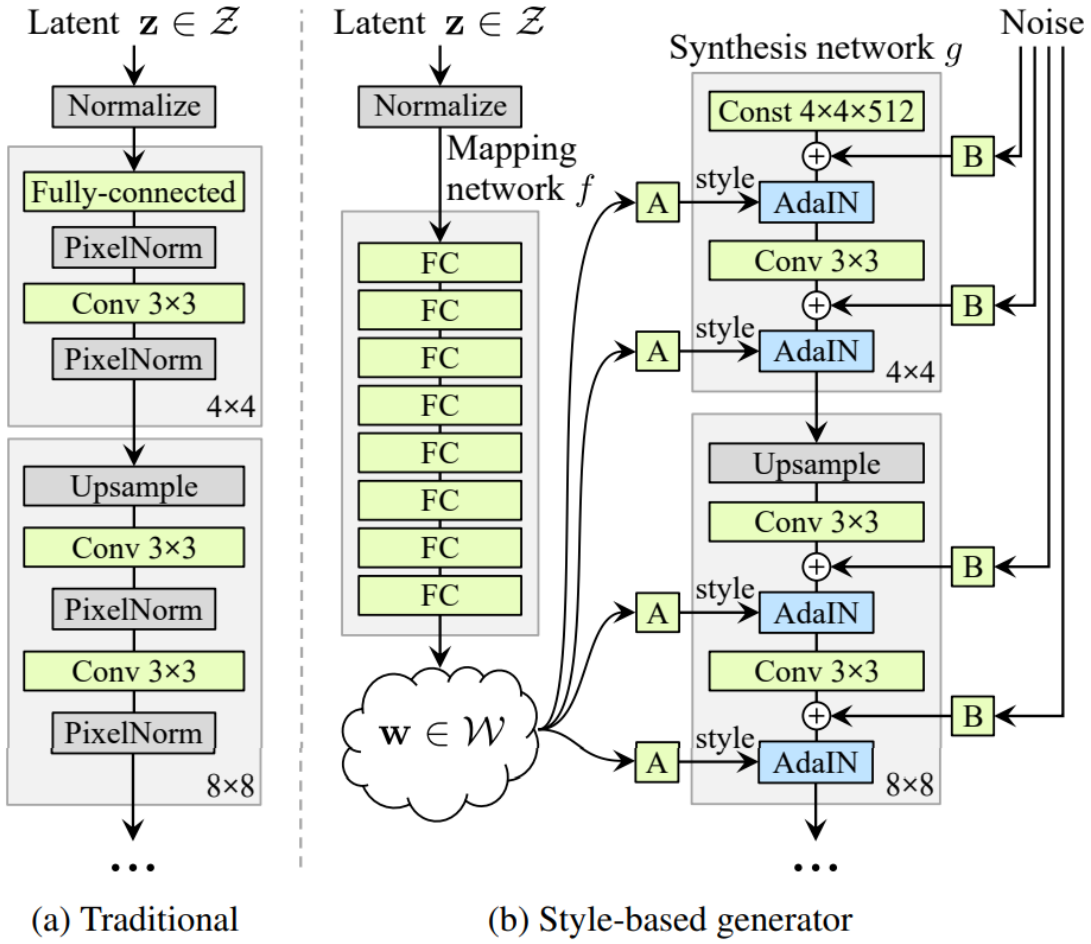


Figure 4.2: **Illustration of the StyleGAN [98] model**. On the left, a standard, sequential, GAN architecture. On the right, the StyleGAN model. Image from [98].

shape possible, a centred *disk*, and see if we could understand what goes on inside. The advantage of this situation is that we know exactly what the size of the latent space should be, since we are using parametric shapes. In this case, we have $d = 1$, since the disks are parametrised by one scalar (the radius). We chose to use a purely convolutional architecture, that is to say with no fully-connected layers.

4.2.1 General autoencoder architecture

Our autoencoder consists of a series of convolutions with filters of small compact support, sub-sampling/up-sampling, biases and non-linearities. The values of the filters are termed the weights of the network, and we denote the encoding filters with $w_{\ell,i}$, where ℓ is the layer and i the index of the filter. Similarly, we denote the decoding filters $w'_{\ell,i}$. Since we use *strided convolutions*, the subsampling is carried out just after the convolution. The encoding and decoding biases are

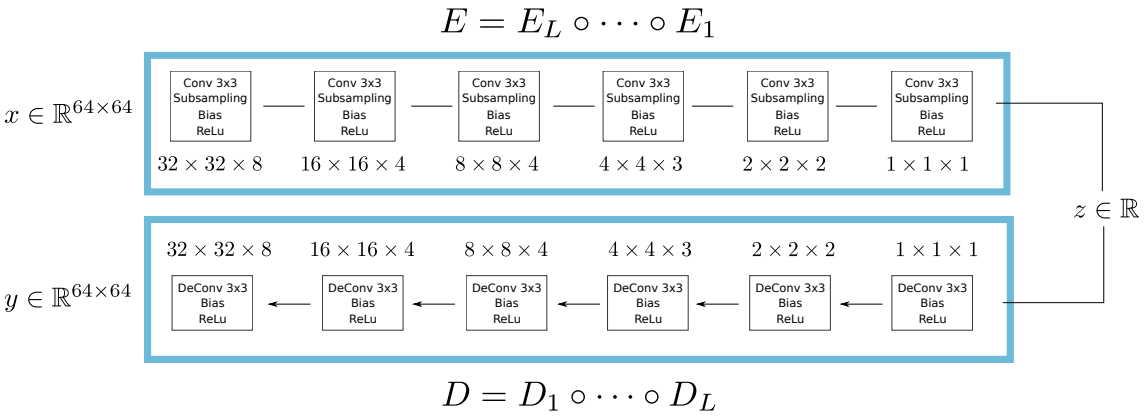


Figure 4.3: Generic autoencoder architecture used in the experiments of Section 4.2.2.

Layer	Input	Hidden layers				Code (z)
Depths	1	8	4	4	3	2	1
Parameter	Spatial filter size	Non-linearity	Learning rate	Learning algorithm	Batch size	
Value	3×3	Leaky ReLU ($\alpha = 0.2$, see Eq. (4.11))	0.001	Adam	300	

Table 4.1: Parameters of autoencoder designed for processing centred disks of random radii.

denoted with $b_{\ell,i}$ and $b'_{\ell,i}$, and we choose leaky ReLUs for the non-linearities :

$$\phi_\alpha(x) = \begin{cases} x, & \text{for } x \geq 0 \\ \alpha x, & \text{for } x < 0 \end{cases}, \tag{4.11}$$

with parameter $\alpha = 0.2$. Leaky ReLU non-linearities are commonly used in the literature [143, 96].

Thus, the output of a given encoding layer is given by

$$E_i^{l+1} = \phi_\alpha(E^l * w_{\ell,i} + b_{\ell,i}), \tag{4.12}$$

and similarly for the decoding layers (except for zero-padding upsampling prior to the convolution), with weights and biases w' and b', respectively. We have used an abuse of notation by not indicating the subsampling here, as this is carried out with the strided convolution.

We consider that the spatial support of the image $\Omega = [0, m-1] \times [0, m-1]$ is fixed throughout this work with $m = 64$, and also that the subsampling rate s is fixed to 2. In the encoder, subsampling is carried out until z achieves the size defined by the problem at hand. In the case of disks with varying radii, it is reasonable to assume that z will be a scalar. Thus, the number of layers in our encoder and decoder is not a free parameter. We set the support of all the convolutional filters in our network to 3×3. The architecture of our autoencoder remains the same throughout this Section, and is shown in Figure 4.3. We summarise our parameters in Table 4.1. We now investigate the inner mechanics of autoencoders in the case of a simple geometric shape: the disk.

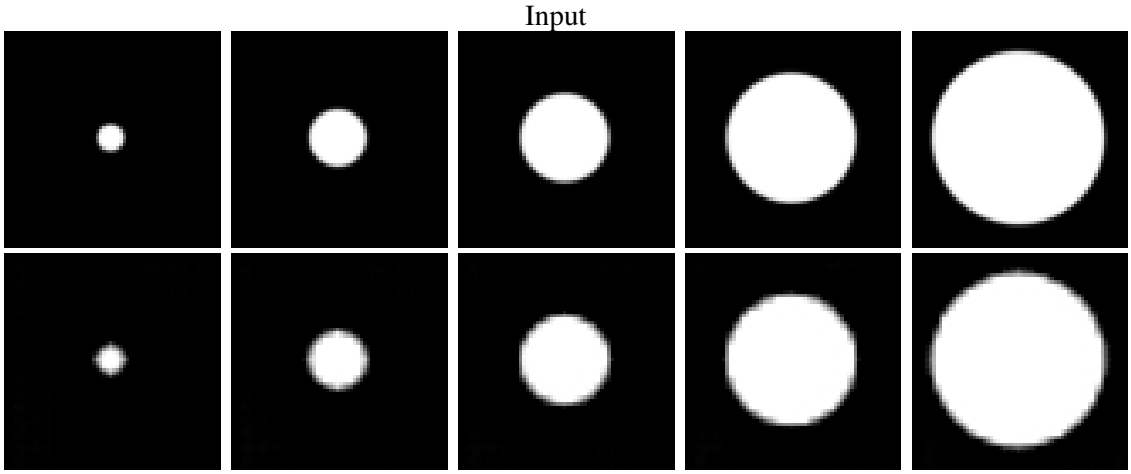


Figure 4.4: **Result of autoencoding disks, with a latent space dimension of size** $d = 1$

4.2.2 Autoencoding disks

Training dataset and preliminary autoencoder results Our training set consists of grey-level images of centred disks. The radii of the disks are sampled following a uniform distribution $\mathcal{U}\left((0, \frac{m}{2})\right)$. We generate 3000 disks in the training set, so that the radius distribution is quite densely sampled. In order to create a continuous dataset, we slightly blur the disks with a Gaussian filter g_σ, so that $x_r = g_\sigma * \mathbb{1}_{B_r}$, where $\mathbb{1}_{B_r}$ is the indicator function of the ball of radius r. This is done in the same manner as in the previous Chapter on film grain, using a Monte Carlo simulation. More details are given in Appendix C.1.

Theoretically, an optimal encoder would only need one scalar to represent the image. Therefore the architecture in Figure 4.3 is set up to ensure a code size $d = 1$. After training, we observe experimentally that the network indeed learns to encode/decode disks correctly with a latent space size of $d = 1$. This can be seen in Figure 4.4.

We now proceed to see how the autoencoder actually works on a detailed level, starting with the encoding step.

4.2.3 Encoding a disk

Encoding a centred disk of a certain radius to a scalar z can be done in several ways, for example calculating the *area* or the *perimeter* of the disk. The empirical evidence given by our experiments points towards the first option, since z seems to represent the area and not the radius of the input disks (see Figure 4.5). This can be carried out in a variety of ways, the most obvious being a simple integration over the image.

Now, while we have described a simple solution which exists to the encoding network with disks (integration over the image), there is no guarantee that this is the *only* solution. Indeed there are probably many other valid representations of the disk, such as encoding the radius. However, these solutions are likely to be more complicated in terms of their filters, and we may also ask if encoding the area may be more desirable than other solutions. Another way of putting this is to find the simplest encoder network out of all the possible valid encoders. In deep learning, this is most often done using network *regularisation* [74]. The simplest regularisation technique is to minimise a norm of the filter weights, which aims to reduce the model complexity. Another regu-

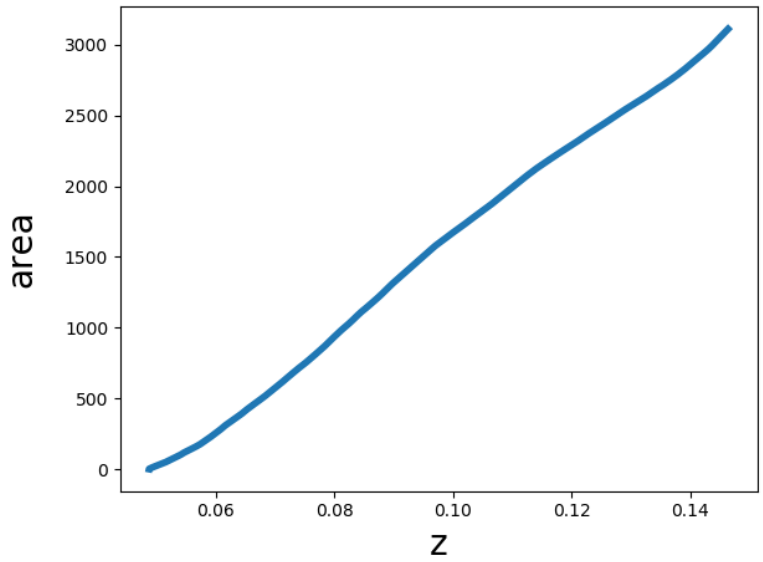


Figure 4.5: **Investigating the latent space of disks**. We have plotted the areas (y-axis) of the input disks against their codes $z \in \mathbb{R}$ (x-axis) in the latent space. The code is linearly proportional to the disks' area. This behaviour is formalised in Proposition3.

larisation approach of Rifai et al [150] is the "contractive autoencoder". This adds a penalisation of the Jacobian of the code z w.r.t the image x : $\|\nabla_x z\|_2^2$. Basically, this specifies that a small perturbation in the input image should result in a small perturbation in the code. It turns out that this regularisation leads to an encoder that indeed extracts the area of the disk. We formalise this result now.

Proposition 3. *[The contractive encoder encodes the area of a disk]*

Consider an encoder $E : \mathbb{R}^{m\times m} \to \mathbb{R}$, which has been presented with images of centred disks having radii uniformly distributed between 0 and 1 during training. Let R_{max} represent the largest radius observed in the dataset. The encoder which has minimal contractive loss $\|\nabla_x E(x)\|_2^2$ and is non-constant is given by $E(x_r) = \gamma r^2$, where γ is a constant. In other words, the contractive encoder represents the image of a disk with its area.

Proof. Proof : See Appendix C.2 for the proof.

We also show in Appendix C.2 further experimental evidence that the area is indeed extracted by the contractive encoder.

Before moving on, we highlight again that the encoder can learn any representation as long as there is an unambiguous link between each point in the data space and in the latent space. The contractive loss allows us to carry out the previous calculations, but this does not lead to the only valid representation.

4.2.4 Decoding a disk

A more difficult question is how does the autoencoder convert a scalar, z, to an output disk of a certain size (the decoding process) ? One approach to understanding the inner workings of autoencoders, and indeed any neural network, is to remove certain elements of the network and to see how it responds, otherwise known as an *ablation* study. We found that removing the *biases*

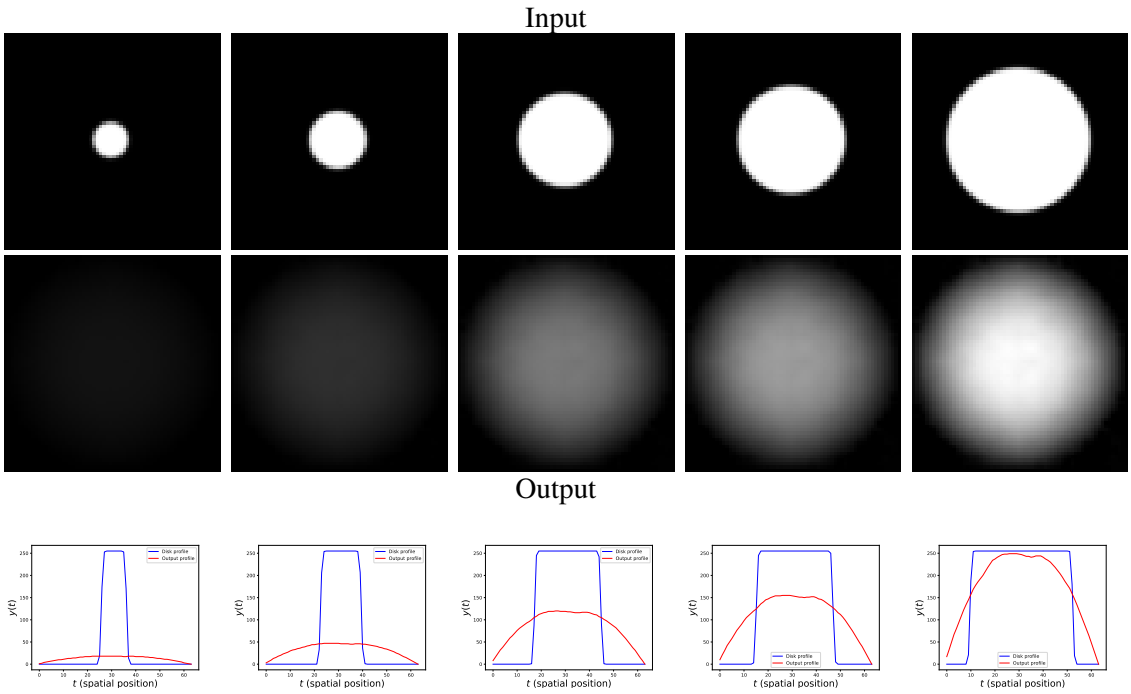


Figure 4.6: **Autoencoding of disks when the autoencoder is trained with no bias.** The first two rows are the input and output of disks when no bias is included in the network. The third row represents the 1D cross-section of these radially symmetric images. The autoencoder learns a function f which is multiplied by a constant scalar, $h(r)$, for each radius. This behaviour is formalised in Equation (4.14).

of the autoencoder leads to very interesting observations. While the encoder is perfectly able to function without these biases (see previous section), this is not the case for the decoder. Figure 4.6 shows the results of this ablation. The decoder learns to spread the energy of z in the output according to a certain function g. Thus, the goal of the biases is to shift the intermediary (hidden layer) images such that a cut-off can be carried out to create a satisfactory decoding.

In order to analyse the inner mechanism of the decoder in more depth, we have investigated the behaviour of the decoder in this ablated case (without biases), where it is possible to describe the decoding process with great precision. In particular, we will derive an explicit form for the energy minimized by the network, for which a closed form solution can be found (see Appendix C.3), and we will show experimentally that the network indeed finds this solution. We first make a general observation about this configuration (without biases).

Proposition 4. *[Positive Multiplicative Action of the Decoder Without Bias]*

Consider a decoder, without biases $D(z) = D^L \circ \cdots \circ D^1(z)$*, with* $D^{\ell+1} = \phi_\alpha \left(U(D^\ell) * w'_{\ell_i} \right)$*, where* U *stands for upsampling with zero-padding. In this case, the decoder acts multiplicatively on* z*, meaning that*

$$\forall z,\, \forall \lambda \in \mathbb{R}^+,\; D(\lambda z) = \lambda D(z).$$

Proof. Proof : For a fixed z and for any $\lambda > 0$. We have

$$\begin{aligned} D^1(\lambda z) &= \phi_\alpha\left(U(\lambda z) * w'_\ell\right) \\ &= \max\left(\lambda(U(z) * w'_\ell), 0\right) + \alpha \min\left(\lambda(U(z) * w'_\ell), 0\right) \\ &= \lambda \max\left(U(z) * w'_\ell, 0\right) + \lambda\alpha \min\left(U(z) * w'_\ell, 0\right) \\ &= \lambda\phi_\alpha\left(U(z) * w'_\ell\right) = \lambda D^1(z). \end{aligned} \tag{4.13}$$

This reasoning can be applied successively to each layer up to the output y. When the code z is one dimensional, the decoder can be summarized as two linear functions, one for positive codes and a second one for the negative codes. However, in all our experiments, the autoencoder without bias has chosen to use only one possible sign for the code, resulting in a linear decoder. □

The profiles in Figure 4.6 suggest that a single function is learned, and that this function is multiplied by a factor depending on the radius. In light of Proposition 4, this means that the decoder has chosen a fixed sign for the code and that the decoder is linear. This can be expressed as

$$D(E(\mathbb{1}_{B_r}))(t) = h(r)f(t), \tag{4.14}$$

where t is a spatial variable and $r \in (0, \frac{m}{2}]$ is the radius of the disk. This is checked experimentally in Figure C.3 in Appendix C.3. In this case, we can write the optimisation problem of the decoder as

$$\hat{f}, \hat{h} = \arg\min_{f,h} \int_0^R \int_\Omega \left(h(r)f(t) - \mathbb{1}_{B_r}(t)\right)^2 dt\ dr, \tag{4.15}$$

where R is the maximum radius observed in the training set, $\Omega = [0, m-1] \times [0, m-1]$ is the image domain, and B_r is the disk of radius r. Note that we have expressed the minimisation problem for continuous functions f. In this case, we have the following proposition.

Proposition 5 (Decoding Energy for an autoencoder without Biases)**.** *The decoding training problem of the autoencoder without biases has an optimal solution $\hat{f}$ that is radially symmetric and maximises the following energy:*

$$J(f) := \int_0^R \left(\int_0^r f(\rho)\ \rho\ d\rho\right)^2 dr, \tag{4.16}$$

under the (arbitrary) normalization $\|f\|_2^2 = 1$.

Proof. Proof : When f is fixed, the optimal h for Equation (4.15) is given by

$$\hat{h}(r) = \frac{\langle f, \mathbb{1}_{B_r}\rangle}{\|f\|_2^2}, \tag{4.17}$$

where $\langle f, \mathbb{1}_{B_r}\rangle = \int_\Omega f(t)\mathbb{1}_{B_r}(t)\ dt$. After replacing this in Equation (4.15), we find that

$$\hat{f} = \arg\min_f \int_0^R -\frac{\langle f, \mathbb{1}_{B_r}\rangle^2}{\|f\|^2} dr = \arg\min_f \int_0^R -\langle f, \mathbb{1}_{B_r}\rangle^2\, dr, \tag{4.18}$$

where we have chosen the arbitrary normalisation $\|f\|_2^2 = 1$. The form of the last equation shows that the optimal solution is obviously radially symmetric[1]. Therefore, after a change of variables,

[1] If not, then consider its mean on every circle, which decreases the L^2 norm of f while maintaining the scalar product with any disk. We then can increase back the energy by dividing by this smaller L^2 norm according to $\|f\|_2 = 1$.

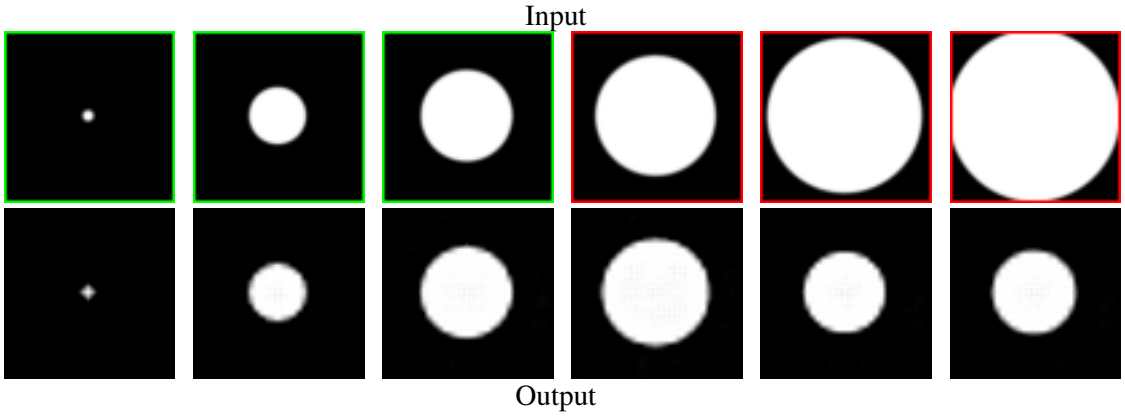


Figure 4.7: **Autoencoding of disks with a database with limited radii.** The autoencoder is not able to extrapolate further than the largest observed radius. The images with a green border represent disks whose radii have been observed during training, while those in red have not been observed.

the energy maximised by the decoder can be written as

$$\int_0^R \left(\int_0^r f(\rho)\ \rho\ d\rho \right)^2 dr =: J(f), \tag{4.19}$$

such that $\|f\|_2^2 = 1$. □

In Appendix C.3, we compare the numerical solution of this problem with the actual profile learned by the network, yielding a very close match. This result is enlightening, since it shows that the training process has achieved the optimal solution, in spite of the fact that the loss is non convex.

4.2.5 Generalisation and regularisation

Many works have recently investigated the generative capacity of autoencoders or GANs. Nevertheless, it is not clear that these architectures truly invent or generalize some visual content. A simpler question is : to what extent is the network able to generalise in the case of the simple geometric notion of size ? In this section, we address this issue in our restricted but interpretable case.

For this, we study the behaviour of our autoencoder when examples are removed from the training dataset. In Figure 4.7, we show the autoencoder result when the disks with radii above a certain threshold R are removed. The radii of the left three images (with a green border) are present in the training database, whereas the radii of the right three (red border) have not been observed. It is clear that the network lacks the capacity to extrapolate further than the radius R. Indeed, the autoencoder seems to project these disks onto smaller, observed, disks, rather than learning the abstraction of a disk.

Again by removing the biases from the network, we may explain why the autoencoder fails to extrapolate when a maximum radius R is imposed. In Appendix C.4, we show experimental evidence that in this situation, the autoencoder learns a function f whose support is restricted by the value of R, leading to the autoencoder's failure. However, a fair criticism of the previous experiment is simply that the network (and deep learning in general) is not designed to work on data which lie outside of the domain observed in the training data set. Nevertheless, it is reasonable

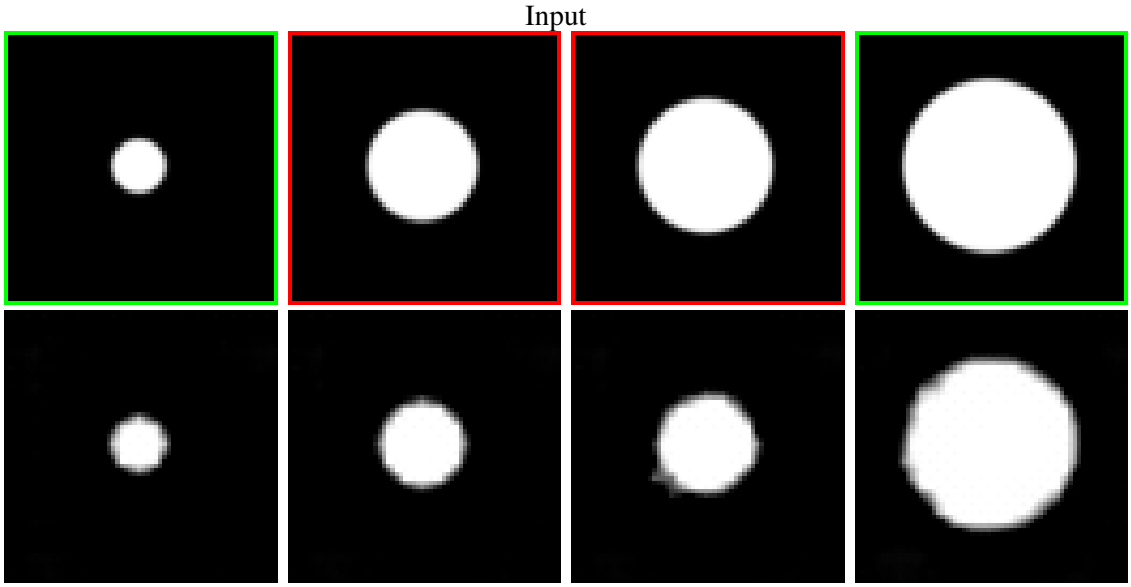


Figure 4.8: **Input and output of our network when autoencoding examples of disks when the database contains a "hole".** Disks of radii between 11 and 18 pixels (out of 32) were not observed in the database. In green, the disks whose radii have been observed in the database, in red those which have not.

to expect the network to be robust to holes *inside* the domain. Therefore, we have also analysed the behaviour of the autoencoder when we removed training datapoints whose disks' radii lie within a certain range, between 11 and 18 pixels (out of a total of 32). We then attempt to reconstruct these points in the test data. Figure 4.8 shows the results of this experiment failure. Once again, in the unknown regions the network is unable to recreate the input disks. Several explanations in the deep learning literature of this phenomenon, such as a high curvature of the underlying data manifold [74] (see page 521, or end of Section 14.6), noisy data or high intrinsic dimensionality of the data [35]. In our setting, *none of these explanations is sufficient*. Thus we conclude that, even in the simple setting of disks, the classic autoencoder cannot generalise correctly when a database contains holes.

Consequently, this effect is clearly due to the gap between two different formulations of the loss of an autoencoder :

$$\mathcal{L}_1 = \mathbb{E}_{x\sim p_x}\|x - D(E(x))\|^2 \tag{4.20}$$

$$\mathcal{L}_2 = \mathbb{E}_{x\in \text{dataset}}\|x - D(E(x))\|^2. \tag{4.21}$$

The latter supposes that the dataset faithfully reflects the distribution p_x of images and is the empirical loss actually used in most of the literature. In our setting we are able to faithfully sample the true distribution p_x and study what happens when a certain part of the distribution is not well observed.

This behaviour is potentially problematic for applications which deal with more complex natural images, lying on a high-dimensional manifold, as these are very likely to contain such holes. We have therefore carried out the same experiments using the recent DCGAN approach of [143]. The visual results of their algorithm are displayed in Appendix C.5. We trained their network using a code size of $d = 1$ in order to ensure fair comparisons. The network fails to correctly autoencode the disks belonging to the unobserved region. Indeed, *GAN-type networks may not be*

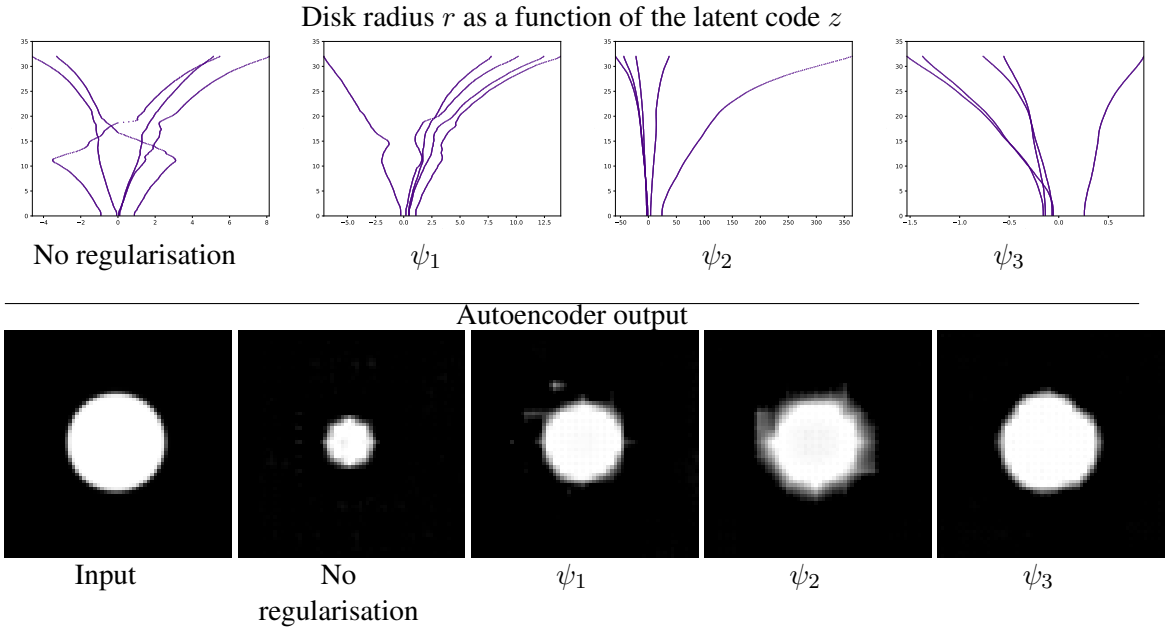


Figure 4.9: **Result of different types of regularisation on autoencoding in an "unknown region" of the training database.** We have encoded/decoded a disk which was not observed in the training dataset. We show the results of four experiments: no regularisation, ℓ_2 regularisation in the latent space (ψ_1), ℓ_2 weight penalisation of the encoder and decoder (ψ_2) and ℓ_2 weight penalisation of the encoder only (ψ_3). In order to highlight the instability of the autoencoder without regularisation, we have carried out the same experiment five times, and shown the resulting latent spaces for each experiment. The latent spaces produced by a regularised autoencoder, and in particular types 2-3, are consistently smoother than the unregularised version, which can produce incoherent latent spaces, and thus incorrect outputs.

very good at generalising data, since their goal is to find a way to map the observed data to some predefined distribution, therefore there is no way to modify the latent space itself. This shows that the generalisation problem is likely to be ubiquitous, and indeed observed in more sophisticated networks, designed to learn natural images manifolds, even in the simple case of disks. We therefore believe that this issue deserves careful attention. Actually this experiment suggets that the capacity to generate new and simple geometrical shapes could be taken as a minimal requirement for a given architecture.

In order to address the problem, we now investigate several regularisation techniques whose goal is to aid the generalisation capacity of neural networks.

Regularisation

We would like to impose some structure on the latent space in order to interpolate correctly in the case of missing datapoints. This is often achieved via some sort of regularisation. This regularisation can come in many forms, such as imposing a certain distribution in the latent space, as in variational autoencoders [102], or by encouraging z to be sparse, as in sparse auto-encoders [144, 123]. In the present case, the former is not particularly useful, since a probabilistic approach will not encourage the latent space to correctly interpolate. The latter regularisation does not apply, since we already have $d = 1$. Another commonly used approach is to impose an ℓ_2 penalisation

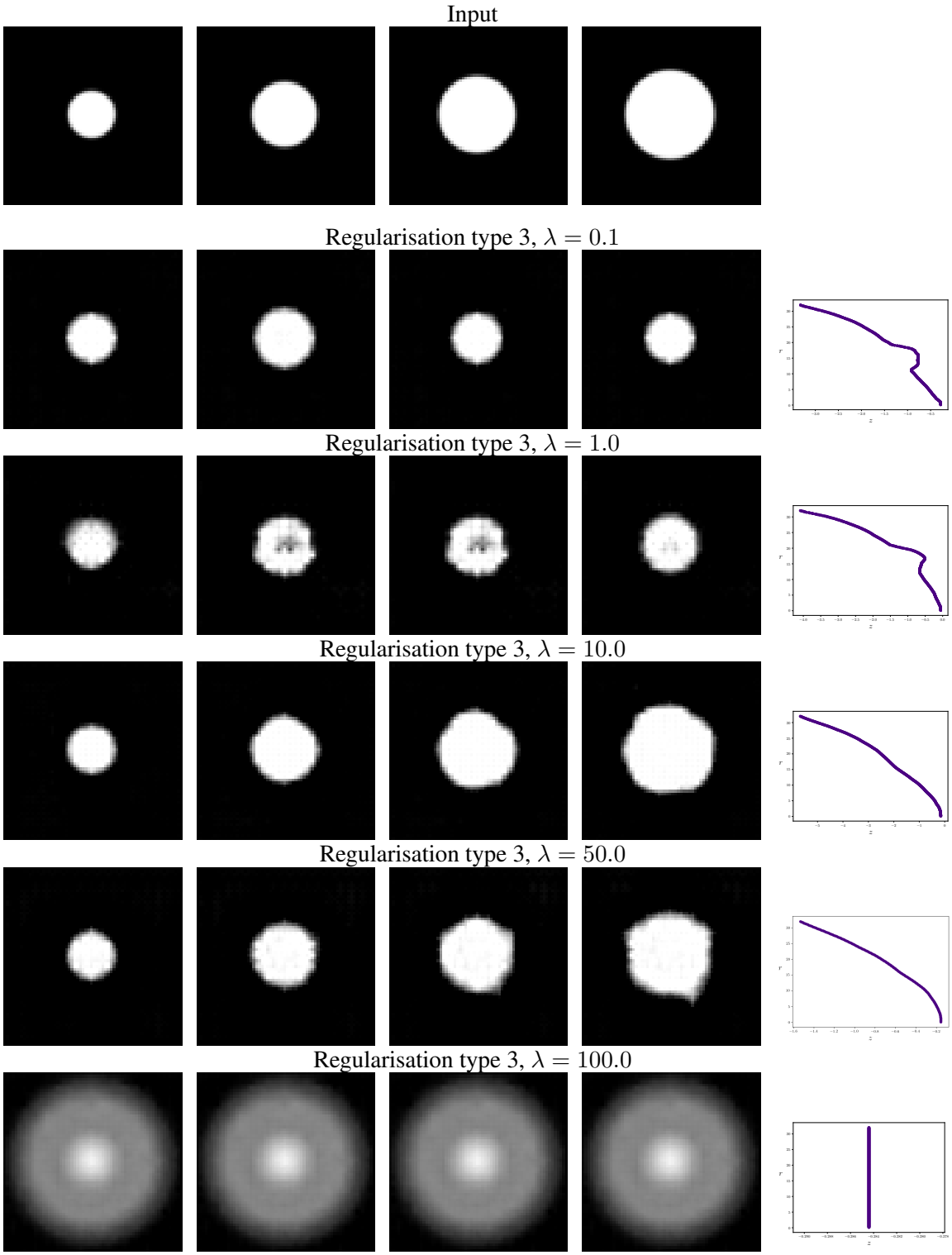


Figure 4.10: **Effect of encoder regularisation on the generalisation capacity of the network**. Regularisation of the network with a varying value of λ, using the regularisation ψ_3 (encoder regularisation) described in Section 4.2.5

of the weights of the filters in the network. The idea behind this bears some similarity to sparse regularisation; we wish for the latent space to be as simple as possible, and therefore hope to avoid over-fitting.

We have implemented several regularisation techniques on our network. Firstly, we attempt a simple regularisation of the latent space by requiring a locality-preservation property as suggested in [78, 28, 114], namely that the ℓ_2 distance between two images (x,x') be maintained in the latent space. This is done by randomly selecting a neighbour of each element in the training batch. Secondly, we regularise the weights of the encoder and/or the decoder (also known as weight decay).

Our training attempts to minimise the sum of the data term, $\|x - D(E(x))\|_2^2$, and a regularisation term $\lambda\psi(x, \theta)$, which can take one of the following forms:

- Type 1 : $\psi_1(x, x') = (\|x - x'\|_2^2 - \|E(x) - E(x')\|_2^2)^2$
- Type 2 : $\psi_2(\Theta_E, \Theta_D) = \sum_{\ell=1}^{L} \|w_{\cdot,\ell}\|_2^2 + \|w'_{\cdot,\ell}\|_2^2$
- Type 3 : $\psi_3(\Theta_E) = \sum_{\ell=1}^{L} \|w_{\cdot,\ell}\|_2^2$.

In Section 4.2.3, we used the contractive loss to derive Proposition 3 which showed that this (contractive) loss encouraged the encoder to extract the area. We have not shown this loss here, however, because it gives quite similar results to the Type 3 regularisation. This is coherent with the results from Rifai et al [150], who showed a formal link between the contractive regularisation and weight regularisation in the case of one hidden layer. Since the weight regularisation is a more practical alternative to the contractive regularisation, we have only experimented with the former here. Finally, we note that, given the very strong bottleneck of our architecture, the dropout regularisation technique does not make much sense here.

Figure 4.9 shows the results of these experiments. First of all, we observe that ψ_1 does not work satisfactorily. One interpretation of this is that the manifold in the training data is "discontinuous", and therefore there are no close neighbours for the disks on the edge of the unobserved region. The second type of regularisation, minimising the ℓ_2 norm of the encoder and decoder weights, produces a latent space which appears smooth, however the final result is not of great quality. Finally, we observe that regularising the weights of the encoder (ψ_3) works particularly well, and that the resulting manifold is smooth and correctly represents the area of the disks. Consequently, this asymmetrical regularisation approach is to be encouraged in other applications of autoencoders. We show further results of this regularisation approach in Figure 4.10, when the regularisation parameter is varied. We see that increasing this parameter smooths the latent space, until λ becomes too great and the training fails.

At this point, we take the opportunity to note that the clear, marked effects seen with the different regularisation approaches are consistently observed in different training runs. This is due in large part to the controlled, simple setting of autoencoding with disks. Indeed, many other more sophisticated networks, especially GANs, are known to be very difficult to train [153], leading to unstable results or poor reproducibility. We hope that our approach can be of use to more high-level applications, and possibly serve as a sanity check to which these complicated networks should be submitted. Indeed, it is reasonable to assume that such networks should be able to perform well in simple situations before moving onto complicated data.

x	$[1,0,0,0,0,0,0,0]$	$[0,1,0,0,0,0,0,0]$	$[0,0,1,0,0,0,0,0]$	$[0,0,0,1,0,0,0,0]$
$u^{(1)}$	$[2,0,0,0]$	$[1,1,0,0]$	$[0,2,0,0]$	$[0,1,1,0]$
$u^{(2)}$	$[4,0]$	$[3,1]$	$[2,2]$	$[1,3]$
$u^{(3)}$	$[8]$	$[7]$	$[6]$	$[5]$
x	$[0,0,0,0,1,0,0,0]$	$[0,0,0,0,0,1,0,0]$	$[0,0,0,0,0,0,1,0]$	$[0,0,0,0,0,0,0,1]$
$u^{(1)}$	$[0,0,2,0]$	$[0,0,1,1]$	$[0,0,0,2]$	$[0,0,0,1]$
$u^{(2)}$	$[0,4]$	$[0,3]$	$[0,2]$	$[0,1]$
$u^{(3)}$	$[4]$	$[3]$	$[2]$	$[1]$

Table 4.2: Results of all possible one-hot vectors of size eight in the simple linear neural network described in Section 4.2.6

4.2.6 Encoding position in an autoencoder

We now move on to the analysis of our second geometric property : position. For this, we ask the following question : is it possible to encode the position of a simple one-hot vector (a discretised Dirac in other words) to a scalar, and if so, how ? A similar situation was investigated concurrently to our work by Liu et al. [117], who studied a network which projected images of randomly positioned squares to a position (a vector in $\mathbb{R}^2$), and then back again to the pixel space, with as small a loss as possible. Their opinion was that this was not possible, at least to a satisfactory degree, by training neural networks, which led them to propose the CoordConv network layer.

In the following, we hand-craft a simple neural network which can achieve this in the forward direction : from a one-hot vector to the position. To simplify, we will analyse the 1-D case, that is to say the input lives in a one dimensional space.

Firstly, let us define some notation. We denote $x \in \mathbb{R}^n$ the input to the network, where n is the input dimension. We shall denote with $u^{(\ell)}$ the output of the ℓ-th layer of the neural network. We shall denote with φ the filter of our network. We shall consider the following hand-crafted filter:

$$\varphi = [1, 2, 1]\,. \tag{4.22}$$

Let us also suppose that subsampling factor is $s = 2$, and that it takes place at every even position (0, 2, 4 etc). We denote with $\mathcal{S}$ the subsampling operator. We do not use any non-linearities or biases in the network. Finally, we denote with E the whole linear neural network. Table 4.2, shows some examples of the results of inputting these one-hot vectors into this network. We can conjecture that it indeed extracts the position. Let us show this now.

Consider $x \in \mathbb{R}^n$ with $n = 2^L$, where L is the total number of layers. The output of each layer $u^{(\ell)}$ can be written in terms of the convolution with the previous layer:

$$u^{(\ell)}(t) = \sum_{i \in \mathcal{A}} \varphi(i) u^{(\ell-1)}(st - i), \tag{4.23}$$

where $\mathcal{A}$ is defined as the support of the filter φ. In our case, $\mathcal{A} = \{-1, 0, 1\}$. Using an induction argument, we can show that the network E indeed extracts the position of the one-hot input vector. More precisely, as we have seen, the network extracts the position in an inverted order, that is to say $n - a$, if a is the position of the non-zero element of x and if we number the elements of x from x_0 to x_{2^L-1}.

Proposition 6 (The linear neural network E extracts the position of a Dirac input)**.** *Consider the neural network E described earlier in this section, and a one-hot input vector $x \in \mathbb{R}^n$, with $n = 2^L$ and where $(x_i), i \in [0, \ldots, n-1]$ denotes the i^{th} element of x. If a is the position of the*

non-zero element of x, then $E(x) = n - a$. In other words, the network E extracts the (inverted) position of the non-zero element.

Proof. We prove this by induction over the number of layers in the network.

One hidden layer This is easy to verify for a network with one hidden layer. Indeed, if the input $x \in \mathbb{R}^2$ contains a 1 at the first (0^{th}) position, then the network output is $2 * 1 = 2$. If x contains a 1 at the second position, then the network output is $1 * 1 = 1$. Thus, the property is true for the case of one hidden layer.

L hidden layers Let us suppose that the network contains L hidden layers, and extracts the non-zero position in reverse order, that is to say $u^{(L)} = 2^L - a$, where a is the non-zero position in x. Since the output of the network is a positive linear combination of the input vector with fixed coefficients, and the property holds for any a, we can rewrite the output as

$$E(x) = \sum_{i=0}^{2^L-1} (2^L - i)x_i. \tag{4.24}$$

Now let us suppose that we add a layer above the input layer, so that the network now has $L + 1$ hidden layers and the input x now belongs to $\mathbb{R}^{2^{L+1}}$, and the previous x is now $u^{(1)}$. We can determine the output of the network using Equation (4.24). There are three cases to distinguish between.

Suppose first that a is an even position, so that $\exists k \in \mathbb{N}, a = 2k$. Thus, using Equation (4.24), we have that

$$\begin{aligned} E(x) &= \sum_{i=0}^{2^L-1} (2^L - i)u^{(1)}(i) \\ &= (2^L - k).2 \\ &= 2^{(L+1)} - 2k. \end{aligned} \tag{4.25}$$

Thus, we find that the network extracts the correct "inverted-order" position, with $a = 2k$.

Let us suppose now that $a = 2k + 1$. In this case, we have

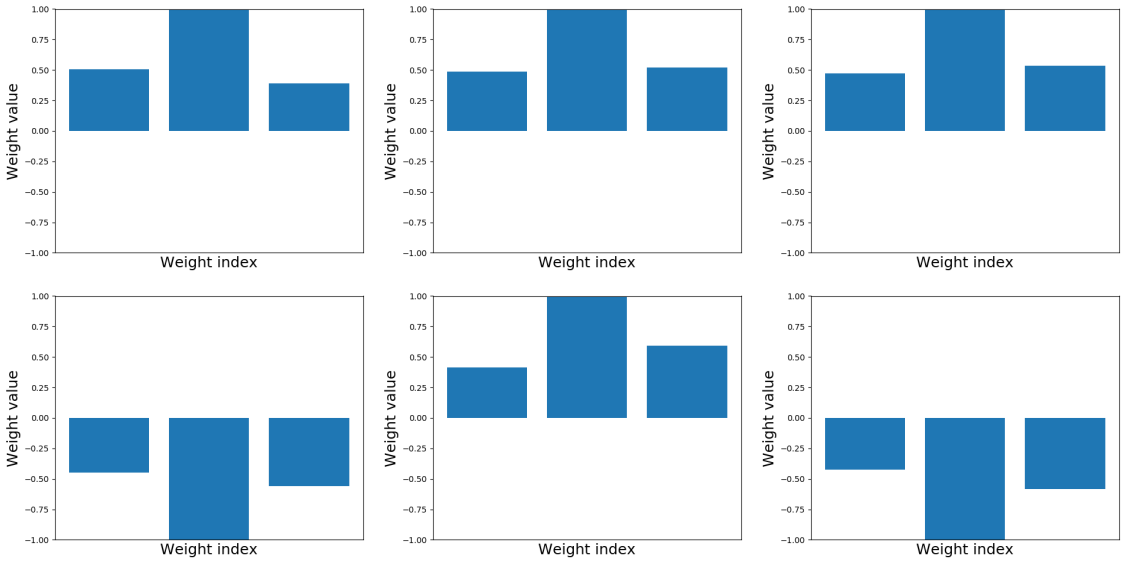
$$\begin{aligned} E(x) &= (2^L - k).1 + (2^L - (k + 1)).1 \\ &= 2^{(L+1)} - (2k + 1). \end{aligned} \tag{4.26}$$

Again, the network correctly identifies the position $a = 2k + 1$.

Finally, there is a special case, where $a = 2^{(L+1)} - 1 = 2k + 1$, with $k = 2^L - 1$ (at the end of the vector x). In this case, we have

$$\begin{aligned} E(x) &= (2^L - k).1 \\ &= 2^L - (2^L - 1) \\ &= 1. \end{aligned} \tag{4.27}$$

Thus, in the extreme case of $a = 2^{(L+1)} - 1$, E still extracts the inverted-order position. Thus, we have proved that the network E extracts the position k of the non-zero element of a one-hot input vector. □



Weights for each layer of the encoder network E

Figure 4.11: **Weights of position encoder network**. We show the weights found by the encoder network E, trained to extract the position. These weights agree with our theoretical prediction in Section 4.2.6.

Furthermore, obviously any variant $b\varphi$, with $b \in \mathbb{R}^*$ also extracts the position multiplied by b^L, since the encoder described in Proposition 6 is a linear function. Finally, we note that our proof relies on the fact that the subsampling factor is $s = 2$. While this proof only directly applies to the example of one-hot pixels, it can provide a useful rule-of-thumb for designing networks which need to deal with position.

4.2.7 Experimental results

We now present experimental evidence that training a neural network with the above encoder leads to the previously exhibited hand-crafted weights in practice. Please note that our goal here is to confirm that the weights which we have constructed are indeed correct. Therefore, in this experiment, we have imposed two main restrictions. Firstly, we construct our encoder to have one filter per layer. We do not allow for many filters, since they would become uninterpretable due to the various possible combinations of these filters. Secondly, we train the encoder to predict the position a of the one-hot vector. The loss function is therefore simply the mean squared error loss between the predicted position and the true position. Indeed, while we have described a mechanism whereby a convolutional network can extract the position, we have no guarantee that this is the *only* solution. Therefore we use this restricted experimental setting in order to improve interpretability.

In Figure 4.11, we show the weights found by stochastic gradient descent training. They fit the handcrafted weights in Equation (4.22) remarkably well. At every layer, we have the handcrafted weights, multiplied by $+1$ or -1. This obviously makes no difference to the final result of the network, since it can flip the sign at any point before the latent code.

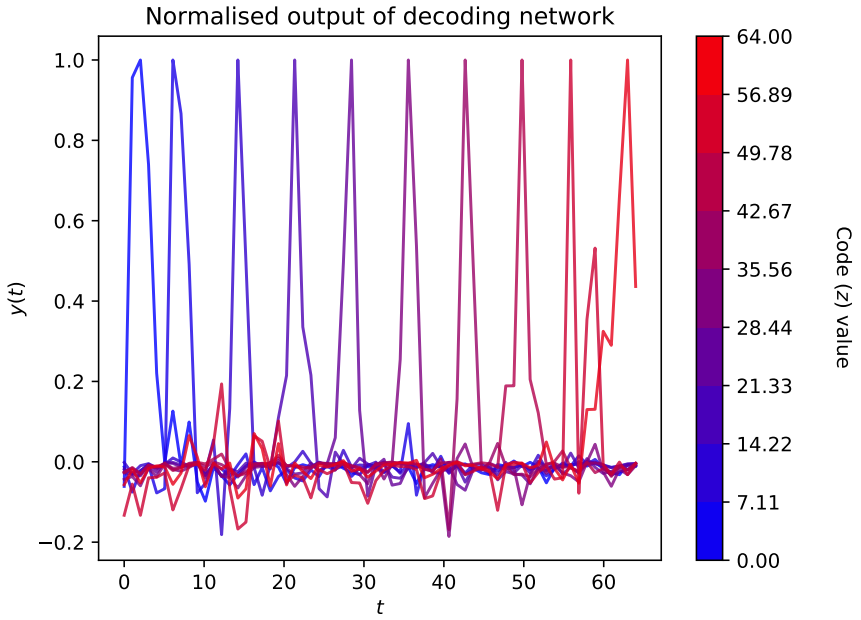


Figure 4.12: **Normalised output of decoding of a postion to a 1D Dirac**. We show the decoding of increasing values of z. We have normalised each output y, to highlight the position of the Dirac.

Decoding position

We now show that it is also possible to perform this inverse operation, in other words, starting from a position z, output a 1-D signal which approximates a delta at position z. To do this, we use a triangular approximation of the Dirac delta. For a Dirac positioned at $a \in [0, n]$, this approximation is:

$$y_a(t) = \begin{cases} 1 - |t - a|, \ if \ |t - a| < 1 \\ 0 \text{ otherwise} \end{cases} . \tag{4.28}$$

It is important to note that the continuous sampling of the parameter space (the position of the Dirac) and subsequent discretisation is crucial to obtaining successful decoding (as it was in the case of disks). Indeed, we also tried to use the approach described in the case of the encoder, that is to say that the Dirac is a one-hot vector at the position a, similar to the experiments described in the "CoordConv" network [117]. In this case, the database is limited, and the decoding is not successful. In particular, interpolating between known datapoints is quite unstable. Sampling a continuous parameter a and choosing an appropriate discretisation solves this problem.

The decoding network was chosen in a similar manner to the case of disks 4.2.4, with 1D convolutions of size 3, biases and leaky ReLU non-linearities. The filter depths chosen were the same as in the case of disks (see Table 4.1), with an output signal size of $n = 64$. The results of the decoding can be seen in Figure 4.12.

In this Subection, we have described a hand-crafted filter which, when coupled with subsampling, can achieve perfect encoding of the position of a Dirac input signal. We show that a network with an appropriate architecture indeed finds this filter during training. Secondly, we have shown

experimentally that decoding is also possible as long as the latent space is sampled in a continuous manner and the corresponding signals are appropriately discretised. This highlights the necessity of correctly sampling the input data.

4.2.8 Conclusion and future work

We have investigated in detail the specific mechanisms which allow autoencoders to encode and decode two fundamental image properties : size and position. The first property is studied via the specific case of binary images containing disks. We first showed that the architecture we proposed was indeed capable of projecting to and from a latent space of size 1. We have shown that the encoder works by integrating over disk, and so the code z represents the area of the disk. In the case where the autoencoder is trained with no bias, the decoder learns a single function which is multiplied by a scalar that is dependent on the size of the disk. Furthermore, we have shown that the optimal function is indeed learned by our network during training. This indicates that the decoder works by multiplying and thresholding this function to produce a final binary image of a disk. We have also illustrated certain limitations of the autoencoder with respect to generalisation when datapoints are missing in the training set. This is potentially problematic for higher-level applications, whose data have higher intrinsic dimensionality and therefore are more likely to include such holes. We identify a regularisation approach which is able to overcome this problem particularly well. This regularisation is asymmetrical as it consists in regularising the encoder while leaving more freedom to the decoder.

Secondly, we have analysed how an autoencoder is able to process position in input data. We do this by studying the case of vectors containing Dirac delta functions (or "one-hot vectors"). We identify a hand-crafted convolutional filter and prove that by using convolutions with this filter and subsampling operations, an encoding network is able to perfectly encode the position of the Dirac delta function. Furthermore, we show experimentally that this filter is indeed learned by an encoding network during training. Finally, we show that a decoding network is able to decode a scalar position and produce the desired Dirac delta function.

We believe that it is important to study generative networks in simple cases in order to properly understand how they work, so that, *in fine*, we can propose architectures that are able to produce increasingly high-level and complex images in a reliable manner and with fine control over the results (for example interpolating in the latent space). An important future goal is to extend the theoretical analyses obtained to increasingly complex visual objects, in order to understand whether the same mechanisms remain in place. We have experimented with other simple geometric objects such as squares and ellipses, with similar results in an optimal code size. Another question is how the decoder works with the biases included. This requires a careful study of the different non-linearity activations as the radius increases. Finally, we are obviously interested in how these networks process other fundamental image properties, such as rotation or colour.

4.3 Image Editing with Deep Generative Models: introduction and previous work

In the previous Section, we looked at how autoencoders could encode and decode simple geometric attributes. However, in practice there are rarely situations where we can completely parametrise images with such (known or unknown) attributes. Indeed, if this were true, there would be no need for neural networks, since we could describe the images exactly! Deep generative models create latent spaces where these attributes are encoded efficiently, thus it is tempting to carry out image editing tasks directly in these spaces. Unfortunately, in their native state with a standard autoencoder or GAN loss, these spaces are usually not be easy to understand, which hinders image editing. Thus, organisation, understanding and navigation of latent spaces is an extremely hot topic currently, precisely because such understanding can lead to very powerful image analysis and editing possibilities.

Prior to the advent of deep learning, image editing methods were based on standard filtering operations (sharpening etc.) or energy minimisation algorithms. In these approaches, a model or prior concerning images was embedded into the method. For example, in the work of Pritch et al. [139], a discrete energy was minimised which tried to respect the manual editing, while maintaining smoothness of the image (similar to the total variation prior). Since around 2017, an entirely new philosophy has appeared, which consists in using pre-trained, powerful generative models. This means projecting an image to the latent space and modifying the subsequent code z_0 to achieve a certain editing objective. The final image is given by generating from the modified latent code. An illustration of this approach can be seen in Figure 4.13. Finding the code z_0 is a challenge in itself, and is the subject of much current research (in essence, finding an encoder for a GAN). Once this is done, we must find a way to move around in the latent space to achieve an editing objective. Again, there is a very vast and recent literature devoted towards this problem. This is the main problem with which my recent research has been concerned.

In the following Sections of this Chapter, we are interested in **image editing using generative models**. When we refer to generative models in this document, we are including autoencoders. This is, strictly speaking, not quite correct, since a standard autoencoder is not a generative model; it does not synthesise any data. However, autoencoders are often modified to be generative models (e.g. variational autoencoders [102]), therefore we will make the abuse of language and include them in generative models. We will discuss three of my works on the subject:

- A feed-forward network approach to image face age editing. This is a supervised approach which creates a network that takes as input a face image and a target age, and outputs the person in the image with the target age. This was part of the PhD of Xu Yao, carried out in collaboration with Yann Gousseau (Télécom Paris) and Pierre Hellier (Interdigital), and published in ICPR 2020 [10];

- A supervised method to navigate in the latent space of GANs to achieve a face image editing goal. This creates a neural network which gives a direction in the latent space to modify a certain facial attribute, such as smile or hair style (although it can also be applied to other types of images). This was part of the PhD of Xu Yao, carried out in collaboration with Yann Gousseau (Télécom Paris) and Pierre Hellier (Interdigital), and published in ICCV 2021 [11];

- A Principal Component Analysis Autoencoder. This is an unsupverised method to create latent components in an autoencoder which correspond to visual image attributes. It is

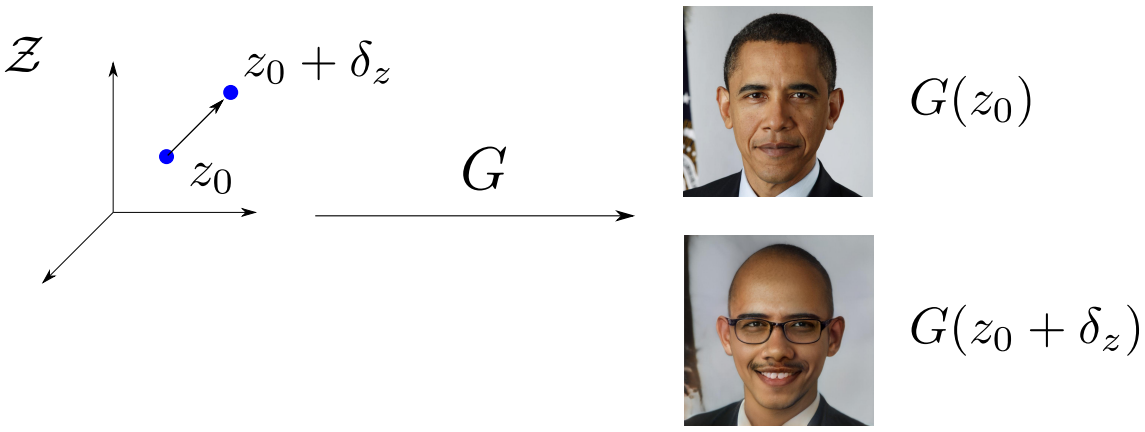


Figure 4.13: **Illustration of editing via the latent space of a deep generative model**. We start out with a latent code z_0 and move in a direction δ_z. The final edited image is given by $G(z_0+\delta_z)$. The problems of finding the starting point z_0 and the direction δ_z are the two main challenges of editing in the latent space.

heavily inspired by Principal Component Analysis. This work was the subject of the postdoc of Chi-Hieu Pham, and carried out in collaboration with Saïd Ladjal, published in JMIV, 2022 [4];

Let us stop to note that this general latent space editing approach is also at the core goal of a project for which I received an ANR Jeunes Chercheurs/Jeunes Chercheuses grant in the academic year 2021. This project is called "IDeGeN": **I**mage Editing with **De**ep **G**enerative **N**etworks. Since the project was only recently started, I do not present any results here, but it constiutes the centre of my current research.

Before presenting my works on image editing with deep generative models, I am going to present some of the previous work already carried out on this subject. This is a domain which has expanded rapidly in the past years, in large part due to the increasing power of the generative models. Thus, it is likely that this is not an exhasutive list, but most of the well-known algorithms are discussed.

Image Editing with Deep Learning : Previous Work

Face aging The survey work [62] gives an exhaustive overview of the pre-deep learning synthesis algorithms. Conditional generative models [128] were first introduced for face aging task by [31, 187, 135, 186]. Their approaches encode a face image to a latent code, which is further manipulated and decoded to an aged face. However, the identity information is damaged during this process. This is further improved by [173, 182], by adding an identity preserving term to the objective. Despite the improvement, their results are over-smoothed compared with the input images. To capture texture details, wavelet-based generative models are introduced by [113, 118]. Their complex models increase the training difficulty and still yield strong artifacts. All the aforementioned models only enable face aging from one age group to another, e.g., from 20s to 40s, lacking flexibility. He *et al.* . [82] proposed an encoder-decoder network, in which a personalised aging basis is synthesised, and an age-specific transform is applied. Their method also uses age groups at training time.

Image-to-image translation Face aging can be considered as an image-to-image translation problem, i.e. translating images between young age and old age domains. An optimization based method is proposed by [169], showing the possibility to use linear interpolation of deep features from pretrained convnets to transform images. GAN based methods [93, 190, 89] further enable real-time translation, by training a feed forward generator. Existing image-to-image translation studies [50, 53, 107, 140, 142, 179] on face images also yield impressive results in manipulating facial attributes. Lample *et al.* [107] design an autoencoder architecture to reconstruct images, and isolate single image characteristics in a latent component via a discriminator. These characteristics can then be modified directly in the latent space. Choi *et al.* [53] propose a method to perform image-to-image translation for multiple domains using only a single model. Pumarola *et al.* [140] introduce an attention based model, which enables face animation by simple interpolation.

Facial Attribute Editing. The following related work concerns the next Section 4.5, however in the interest of clarity of exposition, we will include this in the present related work. Previous works regarding facial attributes mainly focussed on images of limited resolution. As mentioned above, Upchurch *et al.* [169] showed that it is possible to achieve semantic transformations such as aging or adding facial hair by interpolating deep features in a pre-trained feature space. Another type of approach trains feed-forward models for the attribute editing task. Attribute2image [181] proposed to train a Conditional Variational Auto-Encoder to generate attribute-conditioned images. With the success of generative networks in image synthesis, a number of studies [53, 83, 107, 116, 177] explored the possibility of training auto-encoders using adversarial learning. FaderNet [107] and StarGAN [53] proposed to disentangle different attributes in the latent space of auto-encoder and generate the output image conditioned on the target attributes. AttGAN [83] and STGAN [116] enhanced the flexible translation of attributes to improve the image quality by relaxing the strict constraints on the target attributes. Several studies investigate different possibilities to tackle high resolution images. CooGAN [49] proposed a patch-based local-global framework to process HR images in patches. Observing the great progress of generative networks in high quality image synthesis, Viazovetskyi *et al.* [171] trained the pix2pixHD model [172] for single attribute editing with the synthetic images generated by StyleGAN2 [95].

Disentangled Representations. Much of the deep learning based image editing literature is concerned with encouraging the latent space of generative models to be *disentangled*. As explained in Section 4.6, this means that each axis corresponds to editing a single attribute, or alternatively, finding a way to move in the latent space which modifies one and only one attribute. One optimization-based method, Image2StyleGAN++ [25], carried out local editing along with global semantic edits on images by applying masked interpolation on the activation features of StyleGAN. Collins *et al.* [54] performed a k-means clustering on the activations of StyleGAN and detected a disentanglement of semantic objects, which enables further local semantic editing on the generated image. For high level semantic edits, Ganalyze [72] learned a manifold in the latent space of BigGAN [38] to generate images of different memorability. InterFaceGAN [156] proposed to learn a hyper-plane for a binary classification in the latent space, which one can use to manipulate the target facial attribute by simple interpolation. Following their work, StyleSpace [178] carried out a quantitative study on the latent spaces of StyleGAN [95] and realized a highly localized and disentangled control of the visual attributes. StyleFlow [26] achieved conditional exploration of the latent space by training conditional normalizing flows. StyleRig [166] introduced a method to provide a face rig-like control over a pretrained and fixed StyleGAN via a 3D morphable face model. GANSpace [91] performed PCA in the latent space of generative networks, explored the principal directions and discovered interpretable controls. The above-mentioned methods gen-

erally focus on manipulations of synthetic images, as it remains a challenge to project real images to the latent space of StyleGAN. Image2StyleGAN used and optimisation method to project real images to an extended latent space of StyleGAN, but whose characteristics are not suitable for manipulation. Some recent works [136, 138, 148, 189] try to train an encoder together with the StyleGAN model. Although the images cannot be perfectly reconstructed, we see the possibility of carrying out attribute editing on real images using the disentanglement characteristics of the StyleGAN latent space.

High-resolution image synthesis Recently, deep generative models show significant progress in high resolution image generation [96, 98, 95]. Shen *et al*. [156] propose an effective way to interpret latent spaces learned by generators and achieve high quality face manipulation on synthesized images. In spite of the considerable progress of recent methods, editing *existing* images of high resolution is still a difficult problem. Viazovetskyi [171] build a dataset based on manipulated synthesized images and uses it to train image-to-image translation models [172]. However, as the attributes are not well disentangled in the synthesized images, their results inherit the same problem. The authors of [25] propose a method to encode a natural image to the latent space of StyleGAN [98] and further enables local edits on reconstructed images. However, according to our experiments, only a fraction of natural images can be accurately reconstructed with a latent code, which makes this type of method impractical.

4.4 High Resolution Face Age Editing

The goal of this first work is to modify/edit the visual age of a face in a digital image, using a deep neural network. This is a key task in the movie post-production industry, where many actors are retouched in some way, either for beautification or texture editing. Synthetic aging or de-aging effects are usually generated by makeup or costly special visual effects. Although impressive results can be obtained digitally, the underlying processes are extremely time consuming. Thus, robust, high-quality algorithms for performing automatic age modification are highly desirable. Nevertheless, this is an intrinsically difficult task. Indeed, the human brain is particularly good at perceiving faces' attributes in order to detect, recognize or analyse them, for instance to infer identity or emotions. Consequently, even small artifacts are immediately perceived and ruin the perception of results. For this reason, the goal here is to produce artifact-free, sharp and photorealistic results on high-resolution face images.

This is the work of part of the PhD of Xu Yao, in collaboration with Yann Gousseau and Pierre Hellier (Interdigital). It was published in ICPR 2020 [10].

We propose a face age editing method which uses a neural network architecture, allowing for fine-grained (continuous) choice of the target age. This favours learning of age-related/invariant features valid for any target age, instead of across arbitrarily defined coarse age groups. We use an encoder-decoder architecture, where an input image is encoded to a set of age-invariant features and a fine-grained target age is encoded to a modulation vector. We then combine these two elements and decode the modulated features to a realistic image. The *feature modulation layer* is thus an important component of our model as it acts directly on the age-related features and enables both a *fine-grained age control* and enforces the model to learn a disentanglement of age-related/age-invariant features. In addition, our approach can perform both aging and de-aging within a single network. Our second key ingredient is to use an unconditioned discriminator which concentrates solely on the photo-realism of the output images to reduce editing artifacts. Note that this in contrast with competing methods where the discriminator is conditioned on age classes.

We show experimental results on multiple datasets with qualitative and quantitative evaluations. These experiments provide clear evidence that the visual quality achieved by our results outperforms state-of-the-art methods. Our work is the first to present face aging/de-aging results at 1024×1024 resolution.

4.4.1 Notation

Let x_0 be an image drawn randomly from a face dataset. We denote by α_0 the age of the person in x_0. Our goal is to transform x_0 so that the person in this image looks like someone at α_1 years old. We want the aged version of x_0 to share many age-unrelated characteristics with x_0: identity, emotion, haircut, background, etc. That is to say: the facial attributes not relevant to age, as well as the background, need to be preserved during age transformation. Therefore, we assume that a face aging model and a face de-aging model can share most of their parameters. In this setting, we consider a single age editing network G and assume that G can transform any face image to any target age. Note that this is **not** a generator in the sense of GANs, that is it does not take as input a latent code and output an image. It is an end-to-end network which takes as input an image and a target age and outputs the edited image. Nevertheless, we shall keep the notation G since it generates the desired image in the latter sense.

The inputs of our model are the face image x_0 and the target age α_1. The output is denoted by $G(x_0, \alpha_1)$, which depicts x_0 at the target age α_1.

4.4.2 Age Editing Network

The proposed age editing network shown in Figure 4.14 employs an autoencoder architecture and is made of an encoder, a feature modulation block and a decoder. The encoder consists of three strided convolutional layers (the first one of stride 1, the other two of stride 2) and four residual blocks [80], while the decoder contains two nearest-neighbour upsampling layers and three convolutional layers, similar to the architecture used in [94, 190]. The main difference compared to these works is our feature modulation block, in which the output features of the encoder are modulated by an age-specific vector (see details below). This idea is inspired by recent works on style transfer [57, 88] which show the possibility to represent different styles using the parameters of normalization layers. The architecture's components are described as follows:

- **Encoder** The face image x_0 is the input of the encoder. The output features are denoted by $\mathbf{C} \in \mathbb{R}^{n \times c}$, where $c = 128$ is the number of channels and n is the product of the two spatial dimensions.

- **Feature modulation for age selection** The target age α_1 is encoded as a one-hot vector, denoted by z_1, and passed to the modulation network. This network consists of a single fully connected layer with a sigmoid activation. It outputs a modulation vector $w \in [0, 1]^c$, which is used to re-weight the features $\mathbf{C}$ before passing them into the decoder and obtaining the face image at the desired age. The modulated features are $\mathbf{C}\,\mathrm{diag}(w)$, where $\mathrm{diag}(w)$ is the diagonal matrix with diagonal w.

- **Decoder** The decoder takes the modulated features $\mathbf{C}\,\mathrm{diag}(w)$ as input and two skip connections, used to preserve the finer details of the input image. The final output is denoted by $G(x_0, \alpha_1)$.

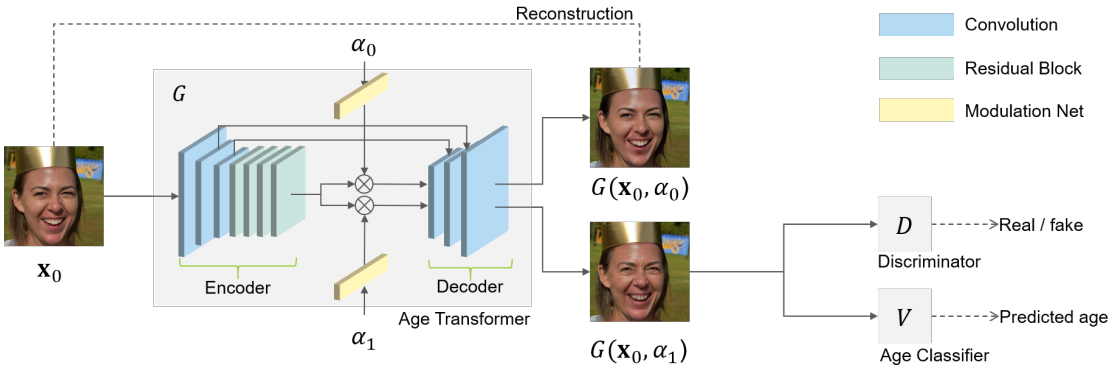


Figure 4.14: **Training process**: input image x_0 is edited by the age transformer G using the initial age α_0 (reconstruction task) and the target age α_1 (editing task). The reconstructed image $G(x_0, \alpha_0)$ should be identical to the input image. The edited image $G(x_0, \alpha_1)$ is further passed in a discriminator $\mathcal{D}$ that ensures photorealism of the transformed image, and an age-classifier V that ensures age-accurate transformation. **The age editing network** G: contains three sub networks: an encoder, a modulating network and a decoder. The encoder maps the input image x_0 to an age-invariant deep feature space. The modulating network maps a target age α to a 128-dimensional modulation vector. This vector is used to modulate each channel of the encoded features, hence applying the desired age transformation. The modulated features are passed in the decoder to obtain the transformed image. Two skip connections are used between the encoder and the decoder in order to better preserve the age irrelevant details.

4.4.3 Training

As illustrated in Figure 4.14, we train our age editing network with an age classifier that ensures age-accurate editing and a discriminator that preserves photo-realism.

The initial age α_0 of x_0 is easy to estimate using a pretrained age classifier, e.g. [151]. The original age range of the training dataset is denoted by $\mathcal{Q} \subset \mathbb{N}$. At test time, the target age can be chosen as any age in $\mathcal{Q}$. At training time, it would seem reasonable to choose any value in $\mathcal{Q}$ uniformly at random. However, we noticed that the artifacts appearing during large age editing were better corrected when selecting a target age α_1 far enough from α_0 during training. We propose to sample α_1 from the set $\mathcal{Q}_{\alpha_0} = \{\alpha \in \mathcal{Q} \ : |\alpha - \alpha_0| \geq \alpha_*\}$ at training time, where α_* is a predefined constant representing the minimum age editing interval. We denote by $q(\alpha|\alpha_0)$ the uniform distribution over $\mathcal{Q}_{\alpha_0}$.

Classification loss

To measure the age of $G(x_0, \alpha_1)$, we use a pretrained age classifier [151]. The classifier, denoted by V, takes $G(x_0, \alpha_1)$ as input and generates a discrete probability distribution over the set of ages $\{0, 1, \ldots, 100\}$. The classification loss is

$$\mathcal{L}_{\text{class}} = \mathbb{E}_{x_0 \sim p(x)} \mathbb{E}_{\alpha_1 \sim q(\alpha|\alpha_0)} \left[\ell(z_1, V(G(x_0, \alpha_1))) \right], \tag{4.29}$$

where $p(x)$ denotes the training image distribution over $\mathcal{X}$, ℓ denotes the categorical cross-entropy loss, and z_1 is the one-hot vector encoding α_1.

Adversarial loss

To enforce better photorealism of $G(x_0, \alpha_1)$, we adopt an adversarial loss built using PatchGAN [93] with the LSGAN objective [124]. Unlike the latest works on face aging [82, 118, 159, 173, 187], our unconditioned discriminator is used to distinguish between real and manipulated images. The aging effects are obtained solely with the age classification loss. The discriminator can be considered as a regularizer which imposes photorealism other than a conditional discriminator trying to match age distributions.

The discriminator is denoted by $\mathcal{D}$. The architecture of $\mathcal{D}$ is the same as proposed in [93]. We use a patch size 142×142 for 1024×1024 images. The modified image $G(x_0, \alpha_1)$ should be indistinguishable from real samples. Therefore, the losses we use are:

$$\mathcal{L}_{\text{GAN}}(G) = \mathbb{E}_{x_0 \sim p(x)} \mathbb{E}_{\alpha_1 \sim q(\alpha|\alpha_0)} [(\mathcal{D}(G(x_0, \alpha_1)) - 1)^2], \tag{4.30}$$

when training G, and

$$\begin{aligned} \mathcal{L}_{\text{GAN}}(\mathcal{D}) = \mathbb{E}_{x_0 \sim p(x)} \mathbb{E}_{\alpha_1 \sim q(\alpha|\alpha_0)} [(\mathcal{D}(G(x_0, \alpha_1)))^2] + \\ \mathbb{E}_{y \sim p(x)} [(\mathcal{D}(y) - 1)^2] \end{aligned} \tag{4.31}$$

when training $\mathcal{D}$. We apply R_1 regularization [127] with $\gamma = 10$ on the discriminator.

Reconstruction loss

When the age editing network receives x_0 and α_0 as inputs, the generated output image $G(x_0, \alpha_0)$ should be identical to the input image. Hence, we minimise the following reconstruction loss:

$$\mathcal{L}_{\text{recon}} = Expect_{x_0 \sim p(x)} \left[\|G(x_0, \alpha_0) - x_0\|_1 \right]. \tag{4.32}$$

Full loss

We train the age editing network and the discriminator by minimising the full loss function:

$$\mathcal{L} = \lambda_{\text{recon}} \mathcal{L}_{\text{recon}} + \lambda_{\text{class}} \mathcal{L}_{\text{class}} + \mathcal{L}_{\text{GAN}} \tag{4.33}$$

where λ_{recon} and λ_{class} are weights balancing the influence of each loss.

4.4.4 Results

We present our training setup and present the experimental results. We further evaluate the quality of our results using quantitative metrics. Our training dataset is built upon FFHQ [98], a high resolution dataset which contains $70,000$ face images at 1024×1024 resolution. The dataset includes large variations in age, ethnicity, pose, lighting, and image background. However, the dataset contains only unlabeled raw images collected from Flickr.

To obtain the age information, we use an age classifier pretrained on IMDB-WIKI [151]. We observe that FFHQ contains much more samples of young faces than of old ones. This data imbalance is challenging since the aging and de-aging tasks would not be treated equally during training: most of faces being young, the age transformer would be trained to perform aging much more often than de-aging, failing to yield satisfying de-aging results. To compensate this imbalance in the age distribution, we propose to perform data augmentation using StyleGAN [98]. We use the StyleGAN model pretrained on FFHQ to generate $300,000$ synthetic images. A quick

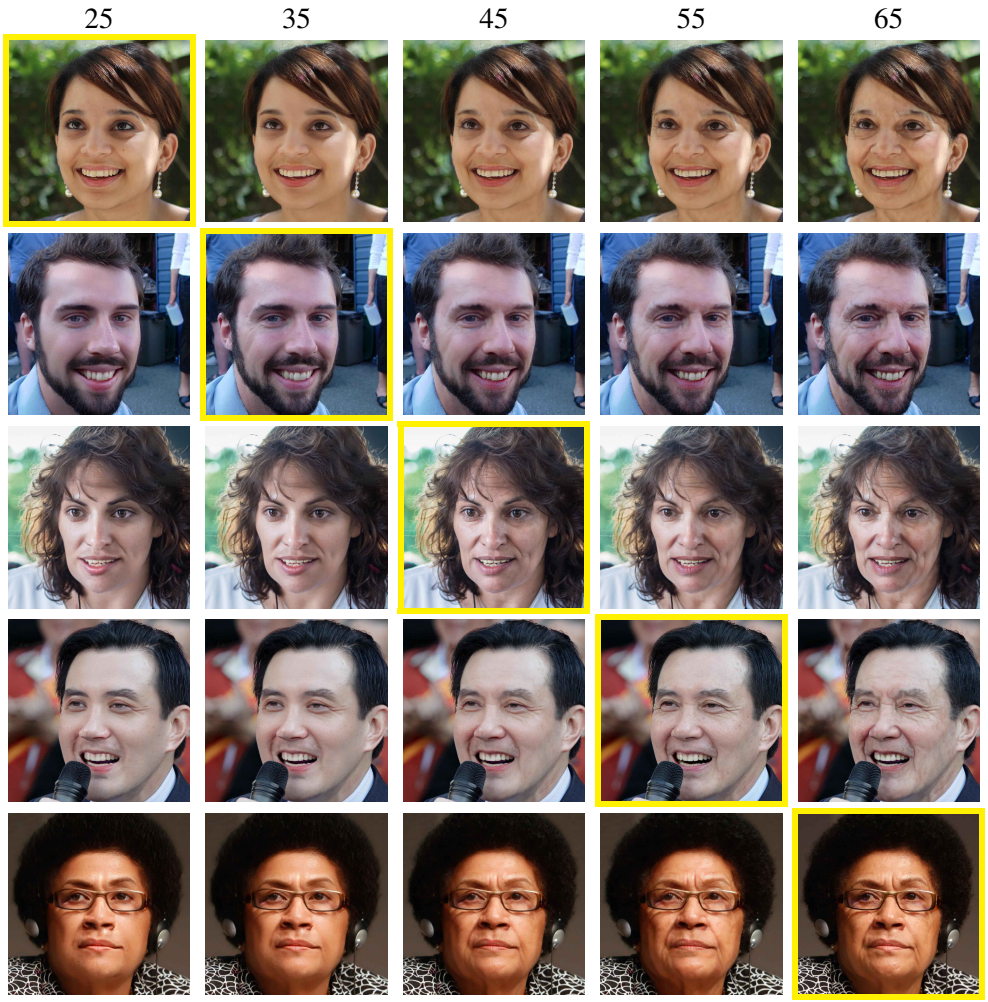


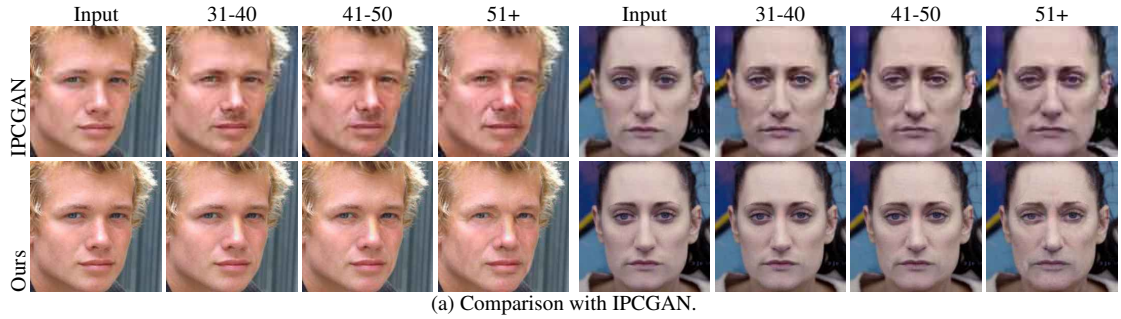
Figure 4.15: **Age editing results on** 1024×1024 **images on FFHQ [98]**. On each row, the yellow frame indicates the original image. Each column corresponds to a target age of: 25, 35, 45, 55, 65. Our approach yields visually satisfying results without introducing significant artifacts. Only age relevant features are modified, while the identity, haircut, emotion and background are perfectly preserved.

visual inspection shows that most of the generated images have no significant artifacts and are nearly indistinguishable from real images by a human. Therefore, we use them for data augmentation to obtain a quasi-uniform age distribution over $\mathcal{Q}$: for any age bin with less than $1,000$ samples in the original FFHQ dataset, we complete this bin with some of the generated synthetic face images; for any age bin with more than $1,000$ samples, we select randomly $1,000$ face images from the original FFHQ dataset. The age-equalized dataset contains $47,990$ images over the range $\mathcal{Q} = \{20, \ldots, 69\}$.

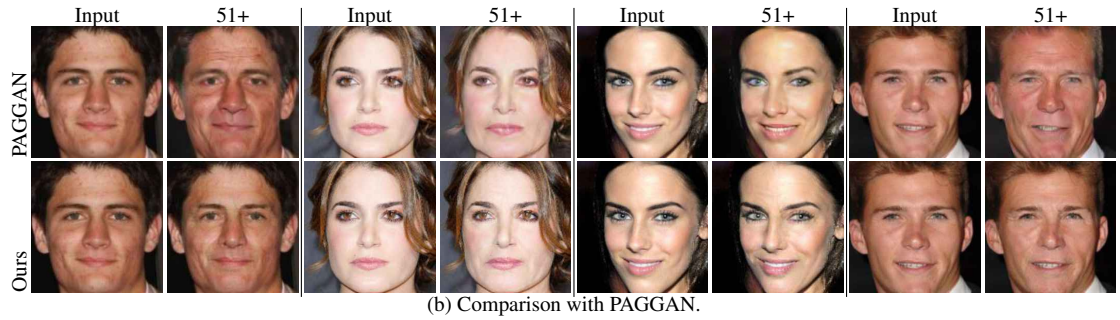
We take 95% of the equalized dataset as our training set and the rest as test set. For the age transformer and the discriminator, spectral normalisation [129] is applied on all the convolution layers except the last one of the age transformer. All the activation layers use Leaky ReLU with a negative slope of 0.2.

We consider age transformation only in the age range $\mathcal{Q} = \{20, \ldots, 69\}$. The constant α_*

Figure 4.16: **Continuous face age editing results on FFHQ [98].** As can be observed, the difference between two adjacent results is nearly invisible, which demonstrates the smoothness of the aging process.



(a) Comparison with IPCGAN.



(b) Comparison with PAGGAN.

Figure 4.17: **Comparison with IPCGAN [173] and PAGGAN [182] on CACD [46].** For each subfigure in (a), the top row corresponds to the aging results of IPCGAN. The second row shows the images generated by our method. For each subfigure in (b), the top row corresponds to the aging results of PAGGAN. The second row shows the images generated by our method.

is set to 25. We have observed that the most significant artifacts appear when the gap between the source and target age is large. By choosing α_* large enough, we force the discriminator $\mathcal{D}$ to suppress these artifacts during adversarial training. The weights λ_{recon} and λ_{class} are set to 10 and 0.1, respectively. We use Adam optimizer with a learning rate of 10^{-4}. The age transformer G is updated once after each discriminator update. Our model is trained for 20 epochs to achieve face age editing on high resolution images. The first 10 epochs are trained on 512×512 images with a batch size of 4. The next 10 epochs are trained on 1024×1024 images, for which we reduce the batch size to 2, learning rate to 10^{-5} and λ_{recon} to 1.

Figure 4.15 presents age editing results on 1024×1024 images in different age groups. Our approach yields visually satisfying results with sharp details (best viewed when zooming on the results) and without introducing significant artifacts. Only the age relevant facial features are modified, while the identity, haircut, emotion and background are well preserved. This is all

Method	Predicted Age	Blur	Gender Preservation(%)	Smiling Preservation(%)	Emotion Preservation(%) Neutral	Happiness
FaderNet [107]	44.34 ± 11.40	9.15	**97.60**	95.20	90.60	92.40
PAGGAN [182]	49.07 ± 11.22	3.68	95.10	93.10	90.20	91.70
IPCGAN [173]	49.72 ± 10.95	9.73	96.70	93.60	89.50	91.10
Ours	54.77 ± 8.40	**2.15**	97.10	**96.30**	**91.30**	**92.70**

Table 4.3: **Quantitative evaluation using online face recognition API [126]**. We compare our method against three methods: Fader Network [107], PAGGAN [182] and IPCGAN [173]. Images are transferred to the oldest age group ($50+$) for all the methods. The 2nd column presents the average predicted age. The 3rd column indicates the blurriness of the results (lower means less blurry). The 4th column is the gender preservation rate, meaning to which percentage the original gender is preserved. The 5th column refers to smiling preservation rate. The last two columns indicate the emotion preservation rate.

the more satisfying that no mask has been used to isolate the face from the rest of the image. Figure 4.16 presents age editing results with a smooth evolution of the target age. The difference between two adjacent results is nearly invisible, which illustrates the smoothness of the aging process.

We compare our method to the two most recent state-of-the-art methods, at this time (2020), on face aging for which the official codes are released - IPCGAN [173] and PAGGAN [182]. We also compare our results to those obtained with FaderNet [107], which allows one to manipulate several facial attributes including the age.

Figure 4.17 presents the face aging results of IPCGAN, PAGGAN and our method on CACD [46]. The output size of each method is: 128×128 for IPCGAN, 224×224 for PAGGAN, 256×256 for our method. IPCGAN generates satisfying aging results and preserves well the identity of input images. However, as can be seen e.g. in Figure 4.17(a) row 1, column 4, the generated image presents noticeable artifacts. PAGGAN generates impressive aging effects but also introduce colored artifacts as shown in Figure 4.17(b) row 1 column 2. IPCGAN and PAGGAN both degrade the quality of the input images. Our method is able to generate consistent aging effects, and preserve well the fine details of the input images.

Quantitative evaluation

Quantitative evaluation of image-to-image translation tasks is still an open question and there is no universal metric to measure photo-realism or quantify artifacts in an image. The recent works [82, 116] on face aging use an online face recognition API to estimate the age and the identity preservation accuracy of the modified images. We thus employ a similar evaluation process.

In our evaluation, the first $1,000$ images with true "Young" label of the CelebA-HQ dataset are extracted as test images. Using this test set, we make a quantitative comparison with FaderNet [107], IPCGAN [173] and PAGGAN [182]. Each image is transferred to the oldest age group using their official released models. For IPCGAN and PAGGAN, the oldest age group refer to $50+$ and $[51, 60]$ respectively. For FaderNet, the old attribute is set to be the default largest value for aging in their official code. To have a fair comparison with group-wise methods, and since $50+$ is considered as the oldest age group, we choose a target age of 60 (the mean of the age range $\{51, \ldots, 69\} \subset \mathcal{Q}$) for our age transformer.

Thus, we get 1000 modified images for each method. We further evaluate these output images using the online face recognition API of Face++ [126]. From the detect API, we obtain the

following interesting metrics: age, gender, blurriness (whether the face is blurry or not, larger values means blurrier), smiling and emotion estimation. The emotion estimation contains a series of emotions: sadness, neutral, disgust, anger, surprise, fear and happiness. With a preliminary analysis on the results, 94.20% of the input images are classified as neutral or happiness. Thus, we just keep these two terms for emotion preservation comparison. We have also compared the identity preservation rate by comparing the modified images with the original inputs. However, since all methods achieve a nearly 100% accuracy, this metric is not reported here.

Table 4.3 shows the quantitative evaluation results. All the methods are given the oldest age group as aging target, and we notice that our method has the highest average predicted age. The gender preservation rate is calculated by comparing the estimated gender with the original CelebA annotations. Using this metric, FaderNet achieves the best performance, followed by our method. For expression preservation (smiling) and emotion preservation (neutral, happiness), our approach yields the best results. It is to be noted however that all methods have similar results. For the blur evaluation, results are much more contrasted. Our method performs much better in generating sharper images, which agrees with the visual comparisons.

4.4.5 Conclusion on deep face age editing

The algorithm proposed in this Section relies on an encoder-decoder type architecture. There are two main advantages:

- Realistic looking results (no artefacts);
- Possibility to choose the age α_1 explicitly, instead of choosing either old/young or copying the age of another photo.

It will be noted, however, that α_1 is a positive integer, and not a positive real number, so it is not continuous, strictly speaking. For most purposes, this is probably not an issue.

The downside of the algorithm is that it can tend to be too conservative, taking too few risks. Indeed, in avoiding artefacts, it can sometimes have not enough modifications. Whether this is a serious problem depends on the final application. Indeed, in the film post-production industry, it may be much more desirable to have a small, reliable, effect rather than a striking one which induces errors every four or five frames, ultimately increasing the work of the editor.

In the next Section, we will see a different sort of approach to editing general face attributes (not just age) with deep models. This approach, which has become more and more popular since the previous approach (on face age editing) was proposed, consists in taking a powerful pre-trained GAN, and editing images in the latent space of the latter. Fundamentally, this means that there is a separation of the editing task into two sub-tasks:

- Learn a good representation for the space of images which interest us;
- Learn how to navigate in this representation's space, in order to obtain an editing objective.

We look at this approach now.

4.5 A Latent Transformer for Disentangled Face Editing in Images and Videos

We now look at a method to edit general labelled attributes in an image, and not just age. In this work we aim to train a neural network, which we call a *latent transformer* T, which will produce

the editing direction δ_z that will edit a single attribute a of the generated image, for any given latent code z. This is a very direct approach, that does not assume any sort of structure of the latent space. Please note that **this is not a transformer in the sense of Transformer networks** [170], rather just a network which transforms one latent code into another according to an editing task.

We choose to use the latent space of StyleGAN [98], due to its impressive performance. To see an explanation of StyleGAN, please refer back to Section 4.1.3. In this section, therefore, when we talk about the latent space, we will be referring to the StyleGAN latent space $\mathcal{W}$, unless otherwise specified. A code of this space will be referred to as $w \in \mathcal{W}$.

The work in this Section was part of the PhD of Xu Yao (with Interdigital), in collaboration with Yann Gousseau and Pierre Hellier, published in [11].

4.5.1 Latent transformer

Our goal is to train a latent transformer T which will edit face images in the latent space of StyleGAN [98]. For a given image x, we assume that we can compute a latent representation $w \in \mathcal{W}$, so that $G(w) \approx x$. Let $\{a_1, a_2, ..., a_K\}$ be a set of image attributes, where K is the total number of considered attributes. In reality, we could just index the attribute numbers $1, \ldots, K$ but a_k seems a clearer way of referring to them. For each attribute a_k, a different T_k is trained. Each T_k is trained using information solely in the latent space.

For this, we first train an attribute classifier in the latent space, which we denote $\mathcal{C} : \mathcal{W} \rightarrow (0, 1)^K$. This consists of three fully connected layers, with ReLU activations between them. The classifier is trained using latent codes associated with labels. The latent code-label pairs are obtained by taking images from the CelebA dataset (which already have associated labels), and projecting the images to the latent space with an encoder trained for StyleGAN. We use the encoder proposed by Richardson *et al.* [148], named Image2StyleGAN. The latter method employs (yet) another, extended latent space, originally introduced by Abdal *et al.* [24]. This space is referred to as $\mathcal{W}^+$. To introduce this space $\mathcal{W}^+$, we recall that StyleGAN inserts the code w in parallel at each resolution of the generator. Let us call L the number of resolutions of this generator. The extended latent space $\mathcal{W}^+$ is created by copying the code w L times and allowing each copy to be modified independently, whereas in the original $\mathcal{W}$, w is the same for all resolutions. This allows for better encodings of images into StyleGAN's latent space. Therefore, given that we use the approach of Richardson *et al.* [148] which uses $\mathcal{W}^+$, we also allow w to move around in $\mathcal{W}^+$.

This approach avoids having to pass latent codes through the StyleGAN generator during the training of T; both training and inference are carried out in the latent space. Finally, given a input latent code w_0, the output edited latent code w_1 is given by:

$$w_1 = w_0 + \alpha T(w_0), \tag{4.34}$$

and the final edited image is obviously $G(w_1)$. The scalar α controls how much editing we want, and can be modified during test time, but in practice we limit it to the range $[-1, 1]$. We explain how it is set during training further on.

Loss functions

Each T_k is trained with the following three loss functions:

- To ensure that T_k manipulates attribute a_k effectively, we minimize the binary classification loss:

$$\mathcal{L}_{\text{cls}}(w) = -y_k \log(p_k) - (1 - y_k) \log(1 - p_k), \tag{4.35}$$

where $p_k = C(T_k(w))_k$ is the probability of the target attribute and $y_k \in \{0, 1\}$ is the desired label.

- To ensure that other attributes $a_i, i \neq k$ remain the same, that is to say disentangled from a_k, we apply an attribute regularisation term:

$$\mathcal{L}_{\text{attr}}(w) = \sum_{i \neq k} (1 - \gamma_{i,k}) \|p_i - C(w)_i\|_2^2, \tag{4.36}$$

where $\gamma_{i,k}$ is the absolute correlation value between a_i and the target attribute a_k, measured on the training dataset. The regularisation term is weighted based on this correlation to avoid over-constraining the attributes which are naturally correlated with the target, i.e "chubby" and "double chin", which would make the task of T impossible.

- Finally, to ensure that the identity of the person is preserved, we further apply a latent code regularization:

$$\mathcal{L}_{\text{rec}}(w) = \|T(w) - w\|_2^2. \tag{4.37}$$

The full loss function can be described as:

$$\mathcal{L} = \mathbb{E}_w \left[\mathcal{L}_{\text{cls}}(w) + \lambda_{\text{attr}} \mathcal{L}_{\text{attr}}(w) + \lambda_{\text{rec}} \mathcal{L}_{\text{rec}}(w) \right], \tag{4.38}$$

where λ_{attr} and λ_{rec} are weights balancing each loss.

Training

We now give training details of our architecture. During training, the scalar α, which controls how much editing we want, is set to:

$$\alpha = \begin{cases} 1 - C(w)_k & \text{if} \quad C(w)_k \in (0, 0.5) \\ -C(w)_k & \text{if} \quad C(w)_k \in (0.5, 1) \end{cases} \tag{4.39}$$

This ensures that we force a significant editing during training, although we could make other choices. The dataset used for training is the CelebA-HQ dataset [96]. The classifier is trained using a binary cross entropy loss for each output component $C(w)_k$, seperately. We do not use a categorical cross-entropy since more than one attribute can be present in each image. For the training of the latent transformer T, we use 90% of the prepared data for training set and train the model for $100K$ iterations, with a batch size of 32. The weights balancing each loss are set to $\lambda_{\text{attr}} = 1$ and $\lambda_{\text{rec}} = 10$. We use Adam optimizer [101] with a learning rate of 0.001, $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

4.5.2 Results

Firstly, Figure 4.18 shows an example of the power of our approach. We can see that on two well-known celebrities (Emma Watson and Barack Obama), attributes are modified progressively until a very remarkable and convincing editing effect has been achieved. There are hardly any artefacts, and the identity of the person is in each case well maintained (except possibly when Emma Watson is aged).

We compare our results with two state-of-the-art methods: InterFaceGAN [156] and GANSpace [91]. For a fair comparison, we follow the methodology of InterFaceGAN and train their model on

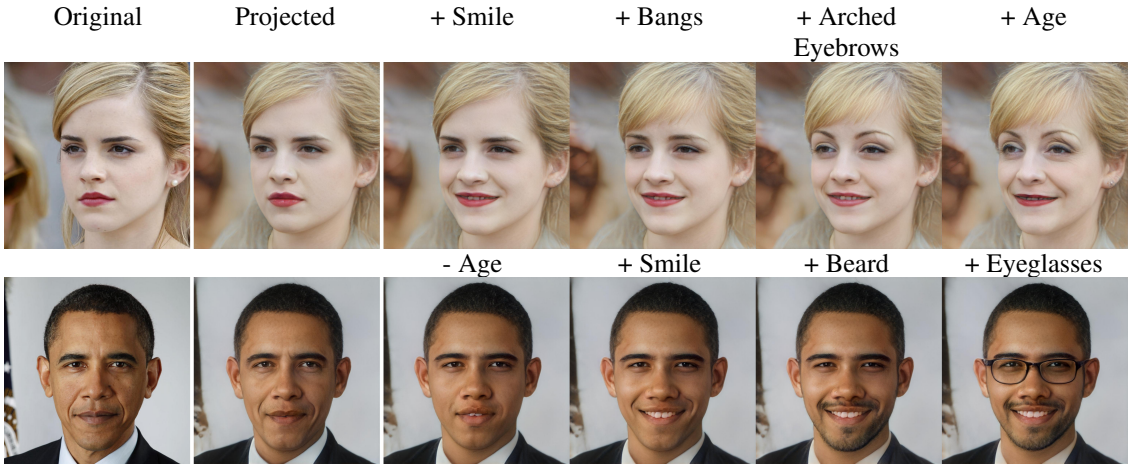


Figure 4.18: **Face attribute editing using pre-trained deep generative model (StyleGAN2 [95]).** We project real images to the latent space of a StyleGAN generator and achieve sequential disentangled attribute editing on the encoded latent codes. From the original and the projected image, we can edit sequentially a list of attributes such as: 'smile', 'bangs', 'arched eyebrows', 'age', 'beard' and 'eyeglasses'. All results are obtained at resolution 1024×1024.

StyleGAN2 for the attributes of CelebA-HQ using their official code. For the evaluation data we use FFHQ, independent from the training of all methods. We project the real images of FFHQ to the latent space $\mathcal{W}^+$ of StyleGAN using the pre-trained encoder [148], and manipulate the latent codes using each method with the suggested magnitude of edits (3 for InterFaceGAN, specified range based on attributes for GANSpace and 1 for our method). Figure 4.19 compares the manipulation results on the attributes which are available for all methods ('gender', 'age', 'beard' and 'makeup'). Our method achieves better disentangled manipulations. For example, when changing 'gender', both GANSpace and InterFaceGAN modify the hairstyle, and when changing 'age', GANSpace adds eyeglasses and InterFaceGAN affects smile. In contrast, our method succeeds to separate hairstyle from 'gender' and disentangle 'eyeglasses' from 'age', thanks to the attribute and latent code regularization terms. The directions of GANSpace are discovered from PCA so that they may control several attributes simultaneously. For InterFaceGAN, no attribute preservation is applied when searching the semantic boundary. Compared with their methods, our editing results are of better visual quality and preserve the original facial identities better.

Quantative evaluation

We compare our method quantitatively with GANSpace and InterFaceGAN using three metrics: target attribute change rate, attribute preservation rate and identity preservation score. Given a set of manipulated samples, the target attribute change rate refers to the percentage of the samples where the target attribute has changed. We consider an attribute has changed when it goes from being "active" to "inactive" or vice versa. An attribute is considered to be active if its probability is greater than 0.5, otherwise it is considerd inactive. The attribute preservation rate indicates the proportion of unchanged samples on the other attributes apart from the target. Finally, the identity preservation score refers to the average cosine similarity between the VGG-Face [137] embeddings of the original projected images and the manipulated results.

For the evaluation data, we project the first $1K$ images of FFHQ into the the latent space $\mathcal{W}^+$ of StyleGAN using the pre-trained encoder [148]. For each input image and each method, we

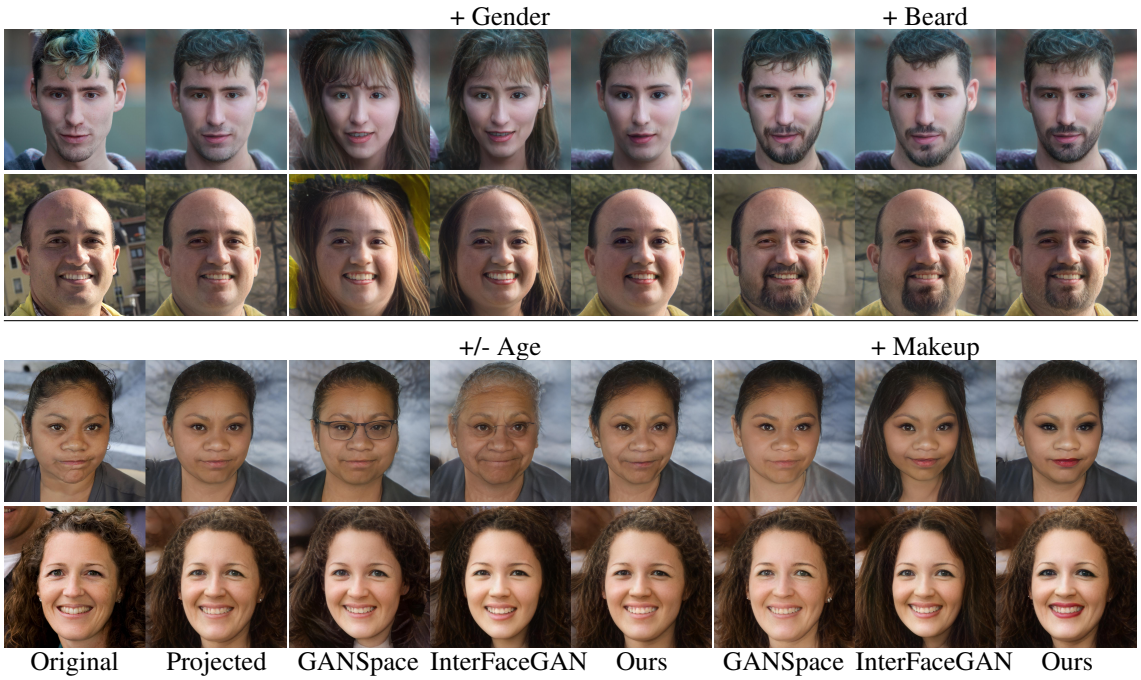


Figure 4.19: **Disentangled facial attribute editing on real images.** The first two columns show the original image and the projected image reconstructed with the encoded latent code in StyleGAN. From the third column in each subfigure, from left to right are the manipulation result of GANSpace [91], that of InterFaceGAN [156] and our result. Compared to these approaches, our method achieves a controllable, disentangled and realistic editing, where the person's identity is preserved.

edit each attribute with α set to ten different scaling factors, $\{0.2 \cdot d, 0.4 \cdot d, ..., 2 \cdot d\}$, where d is the magnitude of change suggested by each method, and generate the corresponding images. To predict the attributes on the modified images, we use a state-of-the-art facial attribute classifier [81], independent from all methods. For each scaling factor, we compute the target attribute change rate, and the attribute preservation rate averaged on the other attributes. To check the identity preservation, we compute the average identity preservation score. Figure 4.20 presents the attribute and identity preservation w.r.t. the target attribute change on the attributes detected by all methods. For attributes like 'beard', 'gender' and 'smile', all the methods handle well. For other attributes, we observe that for the same amount of change on the target attribute, our approach has a higher attribute preservation rate while achieving a comparable or better identity preservation score. Overall our method achieves better disentanglement and better identity preservation than existing methods

We note that this method of evaluation reflects the different goals of editing well (disentanglement, identity preservation), and had not been previously proposed in the literature. We feel it is useful, and hope that it will be used in the future.

4.5.3 Latent transformer sequential editing

Thanks to the disentanglement property of our approach, it achieves sequential modifications of several attributes on real images. We project real images of FFHQ to the latent space $\mathcal{W}^+$ of StyleGAN using the pre-trained encoder [148], and apply manipulations on a list of attributes se-

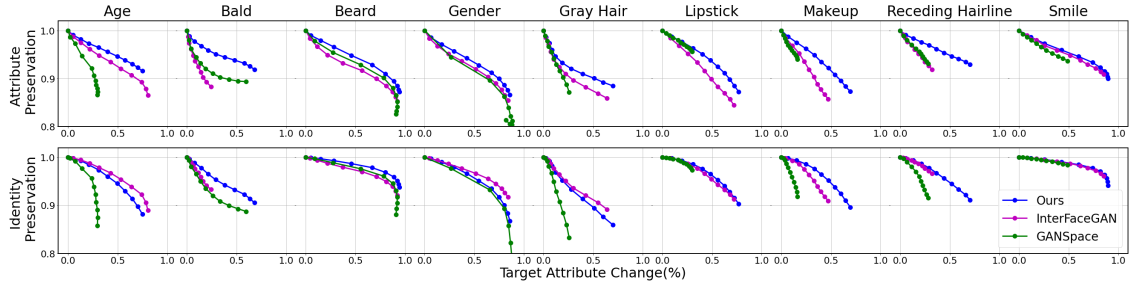


Figure 4.20: **Attribute and identity preservation vs. target attribute change.** For each method, we edit each target attribute with 10 different scaling factors ($\{0.2 \cdot d, 0.4 \cdot d, ..., 2 \cdot d\}$, where d is the magnitude of change suggested in each method, and generate the modified images. Attribute preservation rate and identity preservation score are measured on the output images. An attribute is considered to be changed if it changes from being "active" to "inactive" or vice versa after the editing. An attribute is considered to be active if its probability, according to an independent facial attribute classifier [81], is above 0.5 and it is inactive if this probability is below 0.5. In the figure, each point corresponds to a scaling factor, where the position x indicates the target attribute change rate (the fraction of the samples with target attribute successfully changed among all the manipulations). In the upper sub-figure, the position y indicates the average attribute preservation rate on the other attributes. In the bottom sub-figure, the position y indicates the average identity preservation score. Ideally, we want higher attribute and identity preservation for the same amount of change on the target attribute (higher curve is better).

quentially. As shown in Figure 4.21, our method achieves disentangled and realistic modifications, and is not limited to a defined order of attributes. We note that the order of the manipulation can change the results.

4.5.4 Latent transformer for video editing

We also propose a pipeline which applies the image editing method described above to the case of videos. The encoding process ensures that the encoded latent codes of two consecutive frames are similar to each other. Therefore, we can reconstruct a face video using the frames projected to the latent space of StyleGAN, which provides the basics for the next manipulation step. Thanks to the stability of our proposed latent transformer, the manipulation does not affect the consistency between the latent codes and generates stable edits on the projected frames. An overview of our proposed pipeline is presented in Figure 4.22. The pipeline consists of three steps: pre-processing, image editing and seamless cloning.

Pre-processing. In order to edit the video in the latent space of StyleGAN, we first extract face images from the frames, according to the StyleGAN setting. We crop and align each frame around the face, following the pre-processing step of FFHQ dataset [95], on which the StyleGAN is pretrained. For face alignment we detect landmarks independently on each frame using a state-of-the-art method [40]. To avoid jitter, we further process the landmarks using optical flow between two consecutive frames and a Gaussian filtering along the whole sequence. All frames are cropped and aligned to have eyes at the center and resized to 1024×1024.

Image editing. In this step, we apply our manipulation method on the processed face images. Each frame is encoded to the latent space of StyleGAN using the pre-trained encoder [148]. The encoded latent codes are processed by the proposed latent transformer to realize the attribute edit-

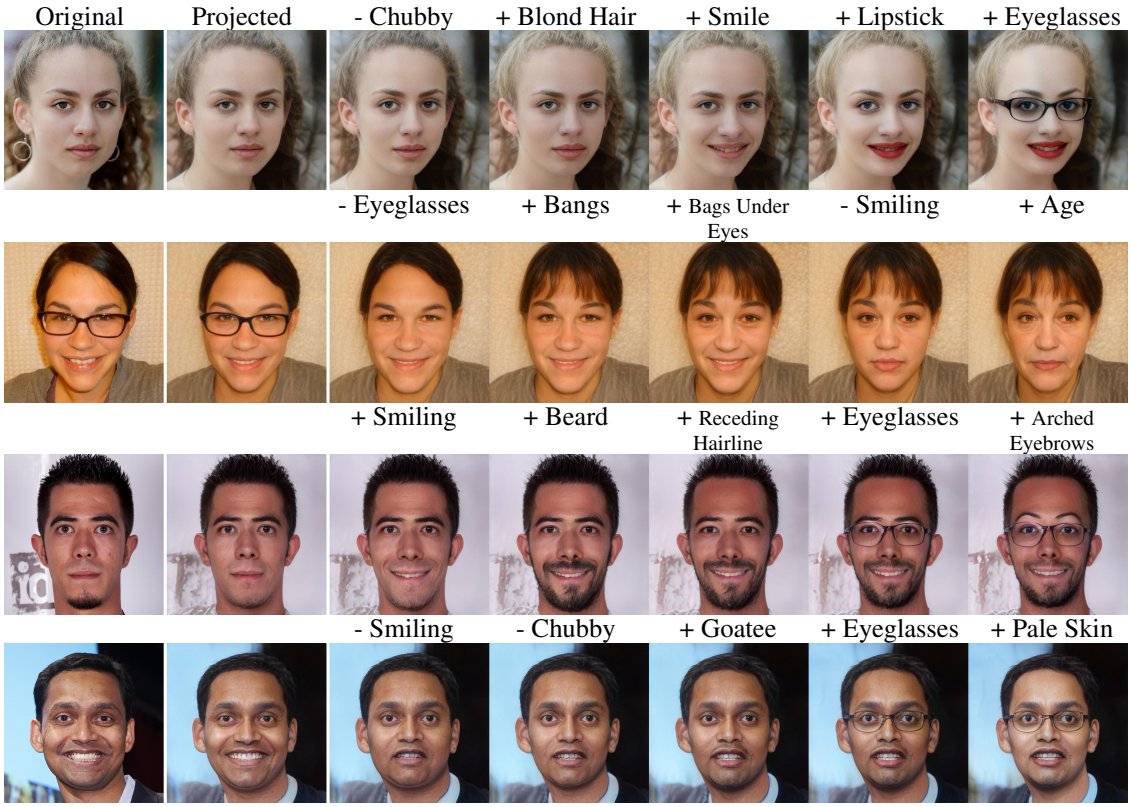


Figure 4.21: **Sequential facial attribute editing on real images**. Given an input image, we manipulate a list of attributes sequentially, where each time a single attribute is modified from the previous latent representation.

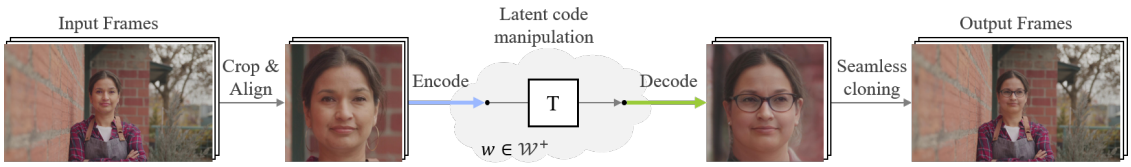


Figure 4.22: **Video manipulation pipeline.** Each input frame is cropped and aligned to a face image individually. A pretrained encoder [148] is used to encode the face images to the latent space $\mathcal{W}^+$ of StyleGAN [95]. The obtained latent codes are processed by the proposed latent transformer T to realize the attribute editing. The manipulated latent codes are further decoded by StyleGAN to generate the manipulated face images, which are blended with the original input frames to get the output frames.

ing. The manipulated latent codes are further decoded by StyleGAN to generate the manipulated face images.

Seamless cloning. We use Poisson image editing method [141] to blend the modified faces with the original input frames. In order to blend only the face area, we use the segmentation mask obtained from the detected facial landmarks.

We apply our manipulation method on real-world videos collected from FILMPAC library [23]. Figure 4.23 shows the qualitative results of facial attribute editing on videos obtained from

Figure 4.23: **Facial attribute editing on videos.** In each sub-figure, the top row shows the input frames, the bottom row shows the output frames obtained from our proposed video manipulation pipeline. A face image is cropped and aligned from each frame, and encoded to latent space of StyleGAN. The encoded latent code is passed into the latent transformer to get the target attribute varied, then decoded to an output face and blended with the input frame.

our proposed pipeline. From each input frame, we crop and align a face image and encode it to the latent space of StyleGAN with a pre-trained encoder [148]. The encoded latent code is processed by the latent transformer to vary the target attribute, then decoded to an output face image and blended with the input frame. As can be seen from the results, our proposed method succeeds in removing the facial hair or adding the makeup, without influencing the consistency between the frames. Nevertheless, we also observe that the proposed method has more difficulty handling extreme pose (side face), which may be due to the limitation of the generation capacity of the StyleGAN model. Please see the Appendix C.6 to see more video results.

4.5.5 Conclusion

In this work, we proposed a latent transformation network to perform facial attribute editing in real images and videos via the latent space of StyleGAN. Our method generates realistic manipulations with better disentanglement and identity preservation than other approaches. We have also extended the method to the case of videos, achieving stable and consistent modifications.

There are some drawbacks to this approach, the main one being that it is necessary to train a different network for each attribute that we want to modify. Secondly, it is a supervised approach, meaning that to modify attributes, we have to have an annotated database at our disposal, which is obviously time-consuming. We can contrast this with the PCA-AE approach, which is unsupervised, and discovers these attributes automatically.

Another limitation of the algorithm is its difficulty in managing side poses, which is a particular issue when dealing with videos, where people tend to turn their heads. The best approach to this

would be to enrich the training data, which would require either new images or a data augmentation method which can handle 3D rotations of faces.

4.5.6 Conclusion on supervised face editing with deep generative models

The two previous approaches represented two general methods of editing images with deep generative models. They follow the chronological evolution of such methods; previously feed-forward autoencoders seemed the obvious way to achieve such editing, but it is becoming increasingly clear that using powerful pre-trained GANs (which corresponds to the second approach) produces high-quality results, with fewer artefacts while not being too conservative in the editing. Another reason why this is such a favoured approach is that it "outsources" the problem of creating the generative model. For GANs in particular, it is well-known that training and choosing an architecture is a difficult process, and one which requires large computational resources, eg. GPUs. This is why NVIDIA is so present in this domain [96, 98, 95]. However, the approach of learning the generative model first and then editing in the latent space poses several questions which have not been satisfactorily addressed as of yet. The first one is how to move in the latent space to modify one attribute and no others. This is known as *disentanglement* in the literature, and while it has not been solved, there are indeed many people working on it. I have started to work on this subject with a PhD student, Gwilherm Lesné, in the context of my ANR Jeune Chercheurs Jeunes Chercheuses project. Another question is how do we edit images which remaining in the original, photo-realistic, latent space of the generative model. Indeed, this photo-realistic latent space is a sub-space of the whole latent space, simply from the fact that images have a finite pixel value range. Thus, when navigating in the latent space for editing, it is entirely possible to move away from this photo-realistic space, which would lead to undesirable images. On the other hand, it may be the case that the generative model cannot take into account certain edits well enough for our purposes. Thus, there may be a tradeoff to obtain here. However, this requires that we can describe the photo-realistic sub-space in the first place, which is not trivial. This is also part of my ANR project.

These editing approaches have supposed that attribute annotations were available to guide the editing. However, this is not always possible. For example, imagine a database of shapes which we have difficulty parametrising, or even an unknown database. In such a situtation, we might want to achieve editing (or understanding) without annotation, that is to say in an unsupervised manner. This is the subject of the next work on deep generative models.

The final work on the subject of image editing with deep generative models is an unsupervised method to create an autoencoder whose latent space controls different image attributes. Let us highlight that, contrary to the previous two methods, *we no longer consider that we know beforehand which attributes describe the images*. This algorithm will try to discover these attributes during the training process. In other words, it is an *unsupervised* image editing algorithm.

This work is the result of the postdoc of Chi-Hieu Pham, and has resulted in a journal article in the Journal of Mathematical Imaging and Vision [4].

4.6 PCA-Autoencoder

In this work we proposed a method whose goal is to organise the latent space in a manner which makes it understandable and navigable, for the purposes of editing. We refer to this method as the "Principal Component Analysis Autoencoder" (PCA-AE). By navigable, we mean that we have

a method to move in the latent space such that some visual attributes of the output image are modified in a controlled manner, *i.e.* without changing all attributes at once or randomly.

We propose to achieve the goals presented above by creating an autoencoder which shares some of the desirable characteristics of the PCA. The classical PCA is a linear transformation to a space with two main properties. Firstly, the axes are organised in order of decreasing variability. So, along the first axis lies the greatest variability of the data, along the second orthogonal axis lies the second-greatest variability, and so on and so forth. Secondly, the axes are orthogonal to each other, which is necessary for interpretation and manipulation. Ideally, we would like to have the best of both worlds, *i.e.* the power of a non-linear transformation (a neural network here) with the aforementioned properties of PCA. This is the objective of this work. More precisely, our goal is to propose an autoencoder with the following two properties: i) the latent space components (axes) are ordered in terms of decreasing "importance" (this is defined shortly afterwards) and ii) each component of a code is statistically independent from the other components.

To achieve this, we start by training an autoencoder with a latent space of size 1. Once this is trained, we fix the values of this first element in the latent space, and train an autoencoder with a latent space of size 2, where only the second component is trained. At each step, the decoder is discarded, and a new one is trained from scratch. This continues until we reach the required latent space size (see Figure 4.24 for an illustration of this approach). Therefore, **"importance" in this context refers to the ℓ_2 reconstruction error** : the first element is the one which has the most impact on this reconstruction error.

Secondly, we add a latent space covariance loss term to the autoencoder loss **to ensure that each latent component is statistically independent from the others**. If the intrinsic characteristics of the data are distributed independently throughout the dataset, then this will be reflected in the PCA-AE latent space. The final objective is to create an autoencoder whose latent space efficiently separates independent characteristics of the data being considered. This is known as **disentanglement** in the generative model literature, and is one of the core goals of such methods. To give an example, this could refer to separating properties such as size, shape or colour, or more high-level characteristics such as gender or hair colour in the case of images of faces. In this work, we achieve this without any reference to labels relative to these characteristics. Instead, we aim to discover the latter in a completely unsupervised fashion, through the data itself.

To summarise, in this work we propose the following contributions:

- An algorithm to create a autoencoder with a latent space where the components of the latent code are ordered in terms of decreasing importance to the data. This importance refers to the ℓ_2 reconstruction error;
- We use a covariance loss term to encourage the components of the latent space to be statistically independent to increase disentanglement;
- We show how the PCA-AE can be used to organise and disentangle the latent space of a pre-trained generative network such as a GAN. An illustration of this can be seen in Figure 4.25.

We demonstrate the efficiency of our autoencoder on synthetic examples of images of geometric shapes as well as on the more complex data of the CelebA dataset. In the first case, we show that the resulting autoencoder retrieves meaningful axes that can be manipulated to change different geometric characteristics (size, rotation) of the shapes. In the second, we automatically discover properties such as hair colour, gender and pose. We emphasise that this is done in a **completely unsupervised manner, without any access to the labels of these characteristics**. In this

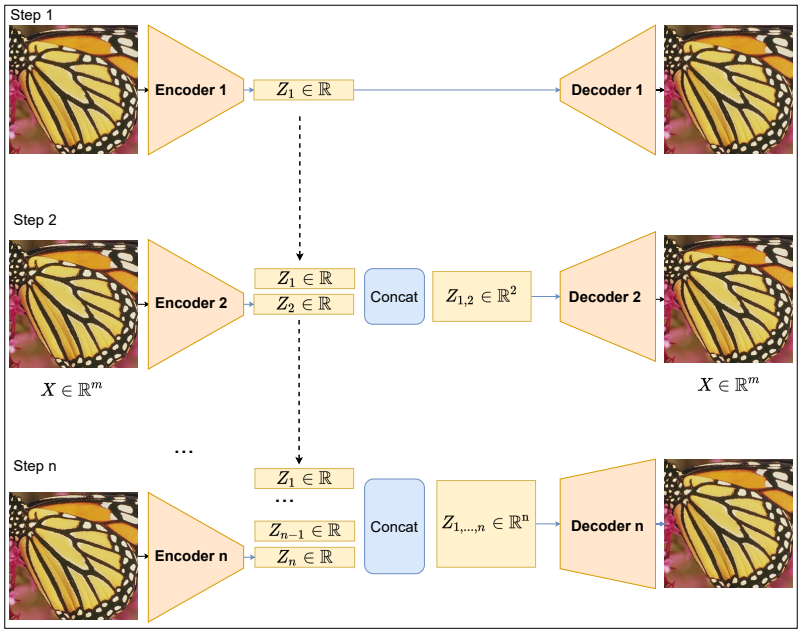


Figure 4.24: Architecture of our PCA-AE. At the n^{th} step, the PCA-AE takes all previous pre-trained encoders, while the decoders are discarded. The parameters of these encoders are fixed. Their output are concatenated with those of the n^{th} encoder and trained with the new n^{th} decoder.

work, we wish to discover these underlying properties automatically, by letting the data indicate its different variable characteristics.

4.6.1 Previous work

As mentioned above, one of the key goals of such editing methods is **disentangling the latent space**. Several previous works exist on this task. Rifai et al [150] employ contractive autoencoders to learn locally invariant features at multiple resolutions, which is then given to a "contractive discriminant analysis" block for the purpose of emotion prediction. Reed et al [146] propose a Boltzmann machine to discover underlying variation in data with two strategies. Firstly, they include the data labels in their cost function for the Boltzmann machine, and secondly, they "clamp" (impose) a code for two data points which are known to share some characteristics. The work of Cheung et al [51], Kumar et al [104] and Lezama [110] are the most similar previous works to ours, in certain aspects. In particular, these works employ some form of covariance loss. Cheung et al use a semi-supervised autoencoder to output an image and at the same time predict a class. Kumar et al propose the covariance loss for the latent space to decorrelate its dimensions, leading to match the moments of the distributions of data and the latent space. Lezama et al use a loss on the Jacobian of an autoencoder output with respect to the latent code, to encourage the code to follow the desired class, as well as a prediction loss using binary classes. Lample et al [107] proposed Fader networks, which try to isolate a single image characteristic in a single latent component, with an innovative use of a discriminator network. This produces a network where the characteristic can be effectively controlled with a slider. In the case of the work of β-VAE$_B$ [85], β-VAE$_H$ [41], FactorVAE [100] and β-TCVAE [48], propose frameworks or regularisations to disentangle VAE by modelling and weighting the Kullback-Leibler divergence term to encourage factorised representations in the latent space.

Algorithm 6 PCA-AE algorithm. Note, we have described the algorithm with a simple gradient descent, but any descent-based optimisation can be used (Adam, Adagrad etc)

Require:
Regularisation coefficient $\lambda_{cov} > 0$. Maximum latent space size n. Initialise the parameters θ_i of the encoders E_i and the decoders D_i.

while θ_1 *not converged* **do**
  Sample $\{x_1, \dots, x_N\}$ from the training set.
  Update E_1 and D_1 by minimising:

$$\mathcal{L}(\theta_1) = \frac{1}{mN} \sum_{i=1}^{N} \|x_i - D_1 \circ E_1(x_i)\|_2^2$$

for $k = 2, \dots, n$ **do**
  while θ_k *not converged* **do**
    Sample $\{x_1, \dots, x_N\}$ from the training set.
    Update E_i and D_i by minimising:

$$\mathcal{L}(\theta_k) = \frac{1}{N} \sum_{i=1}^{N} \left(\frac{1}{m} \|x_i - D_k \circ (E_1(x_i), \dots, E_k(x_i))\|_2^2 \right) + \lambda_{cov} \mathcal{L}_{cov}(\theta_k)$$

4.6.2 Principal Component Analysis Autoencoder

Before describing the PCA-AE, we recall out some notation. Let $\mathcal{X}$ be the data space (images), and $\mathcal{Z} = \mathbb{R}^n$ the latent space, d being the dimensionality of this latent space. We denote the encoder with $E : \mathcal{X} \to \mathcal{Z}$, and the decoder with $D : \mathcal{Z} \to \mathcal{X}$. We denote with z_i the i^{th} *component* of z. We will refer to this as a *latent component*. Let $y = D \circ E(x)$ be the output of the autoencoder. As in the previous Section, we have the standard reconstruction loss which is the ℓ^2 norm of the difference between the input and the output:

$$\|x - D \circ E(x)\|_2^2. \tag{4.40}$$

Now, we describe the core idea and algorithm of PCA-AE. As we explained above, we wish to organise the latent space according to two principles:

- Decreasing order of "importance"
- Statistical independence of the components

If we consider importance to mean variability, then in the case where the data is drawn from a multi-dimensional Gaussian distribution, the PCA achieves these two goals. Let us consider the importance first.

Decreasing importance of latent space codes In the PCA, the first component contains the greatest variability, so in some sense the greatest energy or importance of the data. In other words, if we project our data onto one dimension, then the best direction to choose is precisely the first component of the PCA. To imitate this, we learn our PCA-AE by progressively increasing the size of the latent space, and at each step freezing the latent code which was just learned. At each step, we use the ℓ_2 reconstruction norm. This is illustrated in Figure 4.24.

Statistical independence of latent space codes Another property of the PCA is that the different components are linearly decorrelated. This is useful since, from the point of view of an AE, we would like independent attributes along each component (smile, hair colour etc). In the PCA-AE, we imitate this behaviour by **requiring that the covariance matrix of the vector z to be as close as possible to the identity matrix**. In other words, we minimise the correlations between the latent components. Recall that for any two codes z_i, z_j, the covariance between the two is defined as:

$$\text{Cov}(z_i, z_j) = \mathbb{E}\left[z_i z_j\right] - \mathbb{E}[z_i]\mathbb{E}[z_j]. \tag{4.41}$$

Given this equation, to simplify our task, we can, without loss of generality, impose a *Batch Normalisation* [92] (BN) layer to the latent vector z to control the expected value of our latent codes. A Batch Normalisation normalises the data tensors at a given layer of a neural network, such that the expectation is α and the standard deviation is β. Therefore, we use Batch Normalisation, such that the mean of each latent component z_i of z is 0 ($\alpha = 0$) and the standard deviation is 1 ($\beta = 1$).

The magnitude of the off-diagonal entries of the covariance matrix can then be simply expressed as $\sum_{i \neq j} \left(\mathbb{E}(z_i z_j)\right)^2$ where i and j range through the dimensions of z. We recall that we are adding a new dimension to our latent space while freezing the first dimensions. Therefore, imposing the independence between the components of the vector z boils down to minimising:

$$\sum_{i<k} \left(\mathbb{E}[z_i z_k]\right)^2 \tag{4.42}$$

where k is the current dimension being added. This, in turn, can be translated into a loss term, by replacing the expectation by a mean over the whole dataset, giving our final covariance loss:

$$\mathcal{L}_{\text{cov}}(\theta_k) = \sum_{j=1}^{k-1} \left(\sum_{i=1}^{N} E_j(x_i) E_k(x_i) \right)^2 \tag{4.43}$$

In practice the sum over the dataset is replaced by a sum over the mini-batch, similarly to what is done in Batch Normalisation.

This gives the following loss to minimise at step k:

$$\mathcal{L}(\theta_k) = \sum_{i=1}^{N} \left(\|x_i - D_k \circ (E_1(x_i), \ldots, E_k(x_i))\|_2^2 \right) + \lambda_{cov} \mathcal{L}_{\text{cov}}(\theta_k), \tag{4.44}$$

where the parameters θ_k are the parameters of the new 1-dimensional encoder E_k and the completely new decoder D_k, and where λ_{cov} is a weighting factor. We chose to discard the previous decoders because we wish to give the highest degree of freedom possible to the reconstruction part, while we keep the first computed dimensions of our latent space, since they were determined as being the most effective to reconstruct the input.

The pseudo-code for our algorithm can be seen in Algorithm 6. Note that we use the mean squared error (MSE), since it is the default setting for neural network packages (we used Pytorch), so we have added the normalisation factor m. In this pseudo-code, we do not specify the minimisation scheme, but any gradient-descent based algorithm can be used (we used the Adam optimiser [102]).

4.6.3 PCA-AE for GAN

The objective of the generator of GANs is to find a mapping from the latent distribution p_z into the image data distribution p_{data}. Ideally, we would like each latent component to correspond to one

factor of variation in the data. In practice, the latent representations of GANs are entangled. In Appendix C.7, Figure C.10, we show several examples of interpolation in the original latent space of Progressive Growing of GANs (PGAN) [96], which is a GAN-based approach for generating high quality images, before applying our PCA-AE. It is clear that this latent space is heavily entangled, with several characteristics modified by changing one component. This makes it difficult to understand, navigate and manipulate the latent space. Addressing these problems is precisely the goal of the present work. In order to organise and disentangle this latent representation, we apply the PCA-AE to the latent space of a pre-trained GAN. Indeed, we do not intend to create a new GAN architecture which can compete with state-of-the-art generators such as PGAN, rather we propose to use our PCA-AE to better understand and organise the latent space of a high quality, pre-trained GAN. In other words, since the problem of simultaneously learning and organising the latent space is too difficult, we propose to learn first and organise afterwards. The learning part is done during the training of the high-quality GAN. This difficulty may arise from the fact that more complex data, such as faces, may need larger increments of latent space size to achieve good results. In this case, it is easier to rely on the pre-trained GAN.

Let us highlight that the strategy we propose can be easily adapted to analyse any GAN, and we have chosen PGAN in the existing work due to its impressive performances. The input sample to the PGAN lives in $\mathbb{R}^{512}$ (the PGAN latent space, $d = 512$), or more precisely is a random sample from the normal Gaussian distribution in dimension 512. Since the input is normalised in the first operation of PGAN's generator during testing, we can assume that the latent codes are drawn uniformly from a sphere, which is not convex. To make the job of the autoencoder easier, and since the latent space is not convex, we will apply our tool locally around a given point from the latent space. More precisely, let $\overline{\eta}$ be a fixed point of this sphere (see Figure 4.25). Let G be the generator of PGAN and η be a small perturbation vector (drawn randomly). Our goal is to design a low dimensional autoencoder E, D that minimises the following loss :

$$\begin{aligned}\mathcal{L}(\theta) =& \|G(\eta + \overline{\eta}) - G(D \circ E(\eta) + \overline{\eta})\|_2^2 \\ &+ \lambda_{cov}\mathcal{L}_{\text{cov}}(\theta)\end{aligned} \tag{4.45}$$

where θ are the parameters of the PCA-AE.

In other words, the autoencoder's goal is to produce a vector $D \circ E(\eta) + \overline{\eta}$ which leads to an image that is as close as possible to $G(\eta + \overline{\eta})$. For this, we have created a new latent space, which is in fact a latent space of a latent space, is of dimension d'.

The vector $D \circ E(\eta)$ will have passed through the low dimensional internal representation of the autoencoder, which is well-organised, since E maps $\mathbb{R}^d$ onto $\mathbb{R}^{d'}$ with $d' \ll d$. The covariance loss $\mathcal{L}_{\text{cov}}$ in Equation (4.45) is defined as in (4.43) and will encourage disentanglement of the latent space of the pair E, D. We apply the same training strategy that consists in iteratively increasing the number of latent components, while freezing the first components. This training process is illustrated in Figure 4.25.

4.6.4 Results

In this section, we present the results of our PCA-AE, and we compare with those of VAE [102], β-VAE$_B$ [85], β-VAE$_H$ [41], FactorVAE [100] and β-TCVAE [47] [2]. Note that other approaches to disentangling the latent space use data labels, which we wish to avoid here: our goal is to discover the variability of the data in an unsupervised fashion.

[2] https://github.com/YannDubs/disentangling-vae

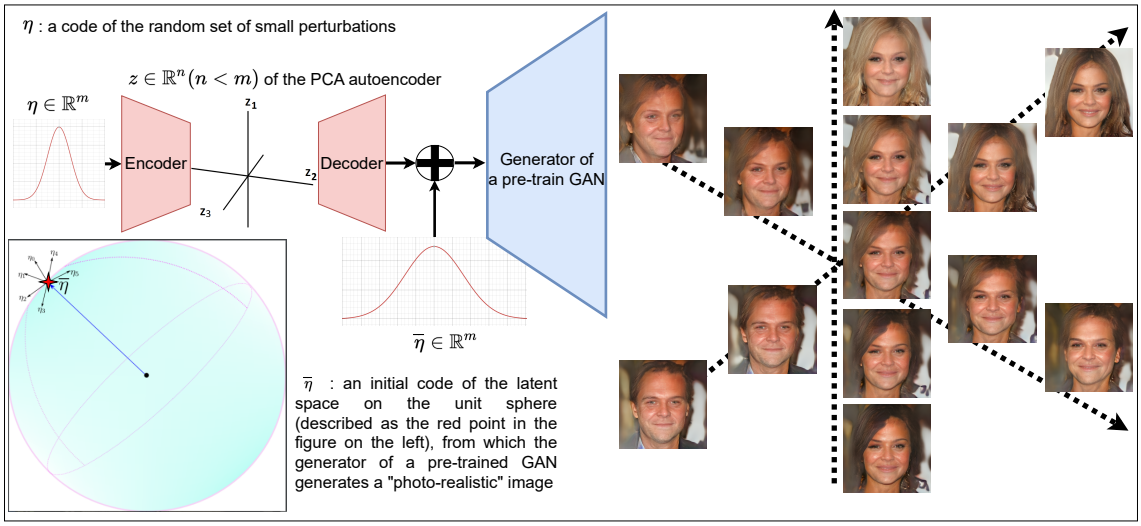


Figure 4.25: **PCA-AE is applied for navigating in the latent space of a pre-trained GAN**. Each component of the PCA-AE attempts to control one attribute of generated images.

	PCA-AE with respect to three considered attributes		
	A	R1	R2
z_1	**0.99**	0.15	0.16
z_2	0.00	0.06	**0.61**
z_3	0.00	**0.72**	0.06

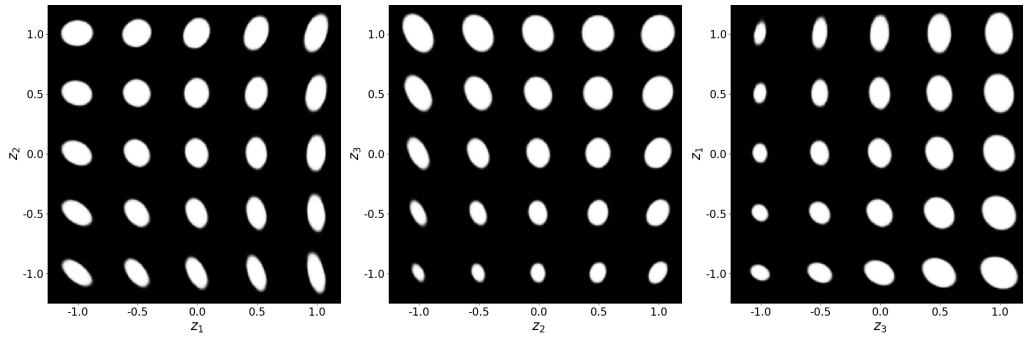
	Area of ellipses (A)						
	AE	VAE	β-VAE$_B$	FactorVAE	β-TCVAE	PCA-AE ($\lambda_{cov} = 0$)	PCA-AE
z_1	0.52	0.00	0.15	0.01	0.10	**0.99**	**0.99**
z_2	0.00	0.33	0.90	0.07	0.14	0.01	0.00
z_3	0.64	0.83	0.06	0.89	0.86	0.55	0.00

Table 4.4: **Evaluation of the absolute PCC between the attributes of ellipses with respect to three components (z_1, z_2 and z_3) of the trained latent space**. We consider three attributes: the area (A), the ratio of two diameters towards vertical and horizontal directions (R1), the ratio of two diameters towards diagonal directions (R2). In the top table, bold font denotes the largest value among the components. In the bottom table, the strongest correlation (the PCA-AE's) is in bold font. We can see that each component of PCA-AE is strongly correlated with only one ellipse attribute.

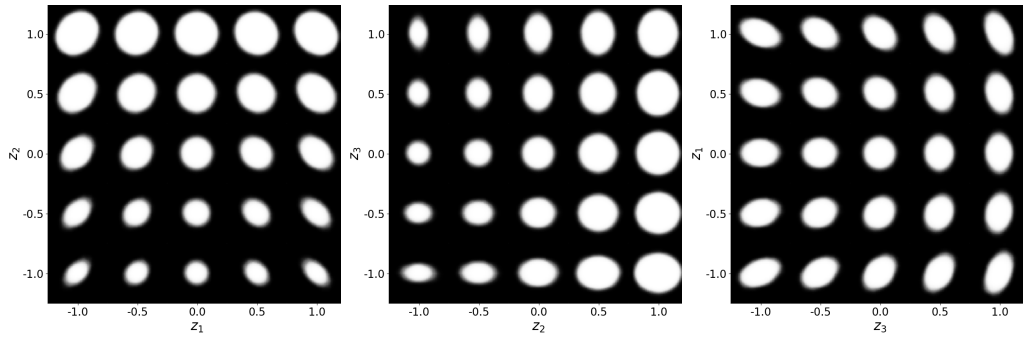
Disentanglement evaluation

We use the absolute Pearson correlation coefficient (PCC) as a disentanglement evaluation to verify the relationship between the attributes of image data and the components of the trained latent space. Given a pair of random variables $(Attr(x), z_i)$ where $Attr(x)$ is the attribute of image x and z_i denotes the i^{th} component of the latent space z, the absolute PCC $\rho(Attr(x), z_i)$ is computed as:

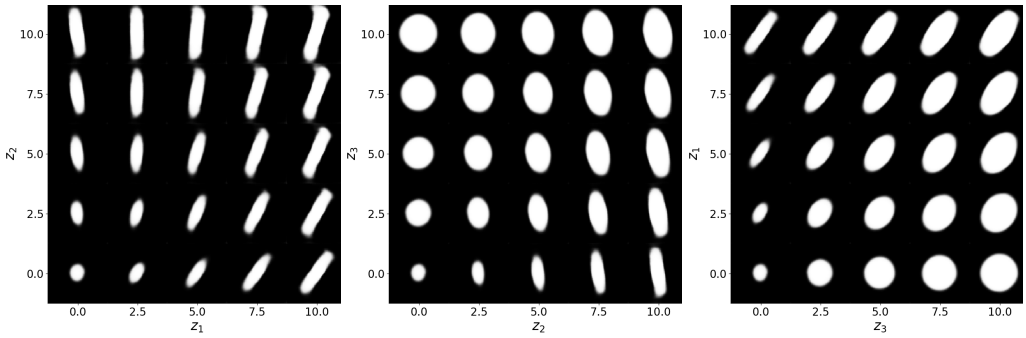
$$\begin{aligned} \rho(Attr(x), z_i) &= \left| \frac{cov(Attr(x), z_i)}{\sigma_{Attr(x)} \sigma_{z_i}} \right| \\ &= \left| \frac{\mathbb{E}[(Attr(x) - \mu_{Attr(x)})(z_i - \mu_{z_i})]}{\sigma_{Attr(x)} \sigma_{z_i}} \right| \end{aligned} \tag{4.46}$$



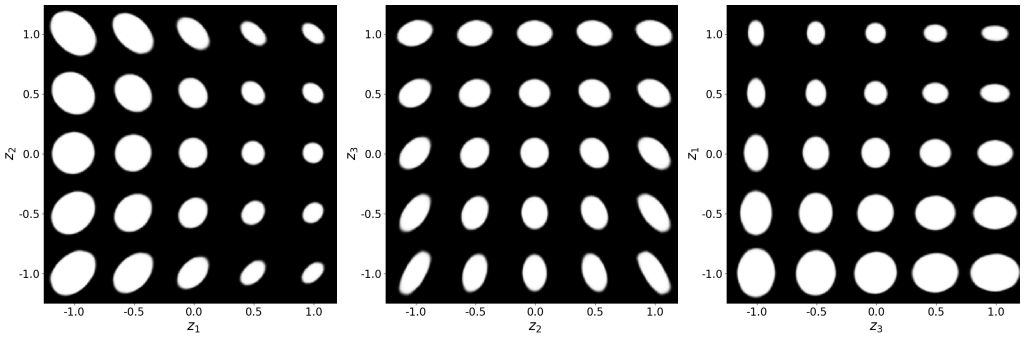
(a) VAE



(b) β-VAE



(c) β-TCVAE



(d) PCA-AE

Figure 4.26: **Interpolation in latent space w.r.t image reconstruction, ellipses with rotation (three parameters) of VAE, β-VAE, β-TCVAE and our proposed method**. The PCA-AE can create a meaningful latent space where different geometric attributes are separated (*i.e.* the 1^{st} component corresponds to the area and the next two parameters are the ratios of the ellipses' axes in different directions).

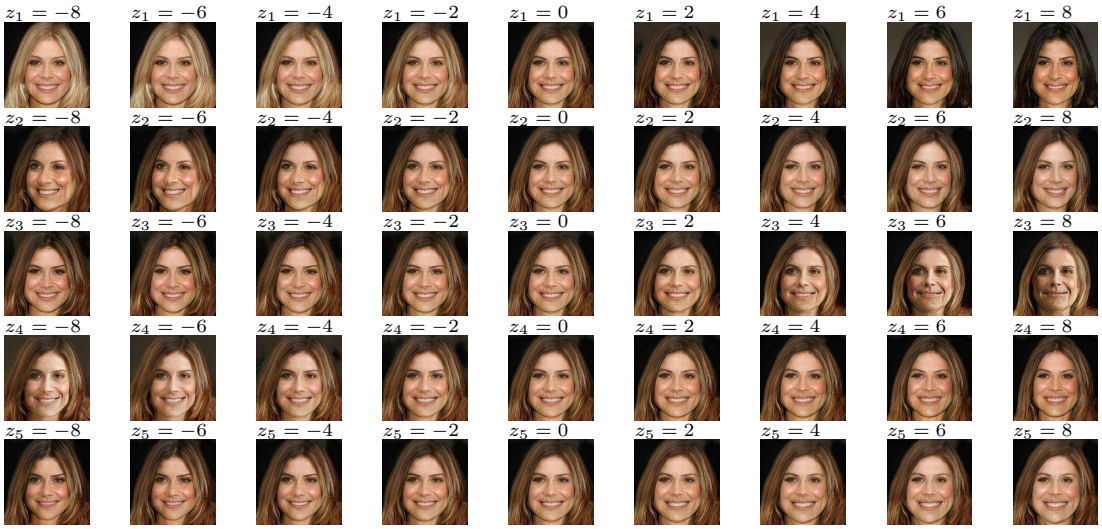


Figure 4.27: **Results of navigation in the latent space of the PCA-AE for a pre-trained PGAN**. We trained this PCA-AE around the code $\overline{\eta}$ corresponding to the middle column. On each row, we have modified a single component (the other components are set to 0). We see that the component z_1 of the latent space z of the PCA-AE represents hair colour, while z_2 corresponds head poses, and in this case z_3 seems to correspond to gender and z_5 to the mouth posture.

where $\sigma_{Attr(x)}$ and σ_{z_i} denote the standard deviation of $Attr(x)$ and z_i, respectively. $\mu_{Attr(x)}$ and μ_{z_i} are the mean of $Attr(x)$ and z_i, respectively. The absolute PCC ranges from 0 to 1.

Experimental setup and results on synthetic data

In order to find out whether our PCA-AE is able to capture meaningful components which correspond to the parameters of visual objects, we have first tested our algorithm on synthetic data of grayscale images of geometric shapes which are centred in the image, with a single shape per image. We have created images of ellipses in the case of three parameters: two axes, and rotation.

The two ellipse axes a and b are sampled from a uniform distribution on the interval $(0, \frac{m}{2})$ (where $m \times m$ denotes the size of image), and the rotation angle Θ from a uniform distribution on the interval $(0, \frac{\pi}{2})$. In these experiments, we set d to 3, the number of parameters used to create the dataset. A drawback of using data with binary images of shapes is that we have a limited number of centred parametric shapes that we can create, even though we sample the parameters from a continuous space. To solve this problem, we blur the binary shapes slightly with a Gaussian filter with $\sigma = 0.8$ pixels, again using the technique described in Appendix C.1.

Figure 4.26 shows decoded images of interpolated points in the latent space, in the case of ellipses. Table 4.4 shows the numeric evaluation based on the absolute PCC between the attributes of ellipses with respect to three components of the trained latent space. We observe that the latent space of our PCA-AE corresponds to three principal attributes of ellipses : area (A), the ratio of two diameters towards vertical and horizontal directions (R1), the ratio of two diameters towards diagonal directions (R2). The compared methods also create a meaningful latent space whereas AE and VAE learn a latent space where the intrinsic parameters of the ellipses are mixed up. While these are not the parameters with which we created the images (indeed, the autoencoder has absolutely no way of knowing what representation to choose, and we cannot impose one in an unsupervised setting), they are indeed independent; for a given area, the ratio between the axes is

Co.	AE			β-TCVAE			FactorVAE			VAE			PCA-AE		
	HC	HP	GE	HC	HP	GE	HC	HP	GE	HC	HP	GE	HC	HP	GE
z_1	0.20	**0.53**	0.02	0.35	**0.80**	0.04	0.07	0.46	**0.53**	0.35	**0.81**	0.07	**0.70**	0.03	0.14
z_2	**0.36**	0.21	0.04	0.05	0.26	0.25	0.04	0.17	0.03	0.05	0.24	0.24	0.07	**0.80**	0.13
z_3	0.28	0.36	0.23	0.13	0.04	0.04	0.09	**0.66**	0.37	0.13	0.02	0.02	0.16	0.15	**0.56**
z_4	0.20	0.19	**0.58**	**0.53**	0.19	**0.53**	**0.70**	0.11	0.14	**0.59**	0.23	**0.57**	0.03	0.24	0.05
z_5	0.34	0.21	0.49	0.07	0.15	0.33	0.01	0.28	0.12	0.07	0.11	0.35	0.05	0.22	0.09

Table 4.5: Quantitative evaluation of the correlation of latent components with high-level attributes. We have calculated the PCC between the latent components of AE, VAE-based methods and PCA-AE, and three attributes: head pose (HP), hair colour (HC) and gender (GE). We can see that the components of PCA-AE are correlated with one dominant attribute of the semantic feature.

an independent parameter, and vice versa. This gives us a way to interpolate in the latent space in a meaningful manner. These independent parameters are sufficient to describe the ellipse, and each axis is hierarchically more interpretable and navigable than in the case of other methods.

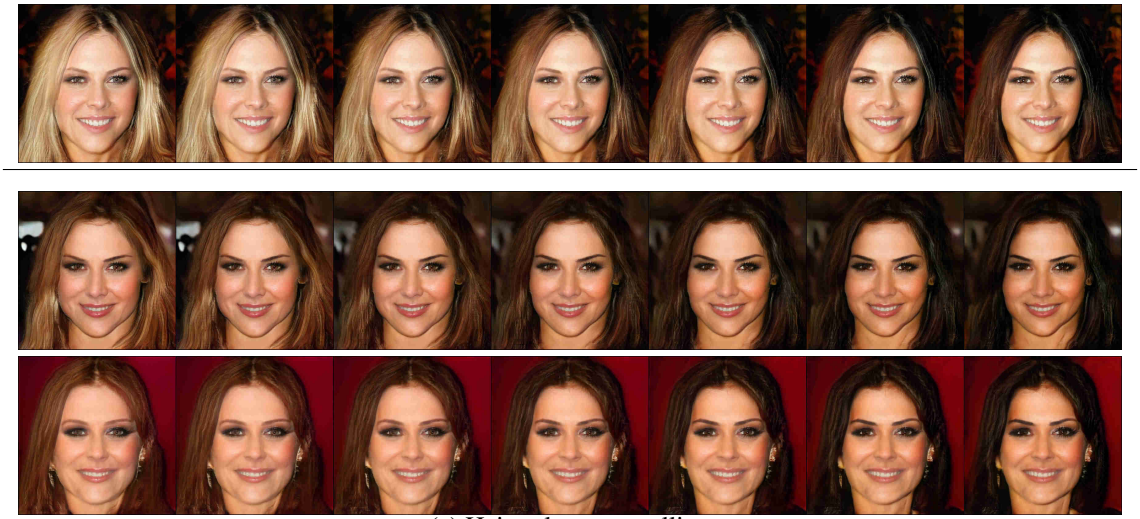
Table 4.4 also shows an ablation study which compares the PCA-AE with the baselines such as a standard AE and our PCA-AE with no covariance loss (*i.e.* $\lambda_{cov} = 0$). We can see that more than one component of the latent space of the PCA-AE with no covariance loss controls the area of the ellipses. In the case of the PCA-AE with no covariance loss, the first and the third component of its latent space correspond to the area attribute simultaneously. This confirms the need of the proposed covariance loss.

Several other methods correlate two latent variables with the area of the ellipses because of the fact that a two-fold correlation may appear. However, in accordance with the underlying idea of transposing the PCA to AEs, we can check for the order of importance of the dimensions produced by our PCA-AE. For example, for the plain AE, increasing the number of dimensions would not help much before the last one. Whereas our PCA-AE's components, within the language of PCA, explain the decreasing of the variability from the first to the last component of the latent space. Due to the non-linear nature of the reconstruction, we do not have exactly decreasing importance but clearly the first dimension is much more powerful than the following (*e.g.* the area of ellipses). In any case our PCA-AE finds that the dimensionality of the data is around $d = 3$. Indeed, our approach ordered the latent space and recovered the approximate dimensionality of the ellipse dataset.

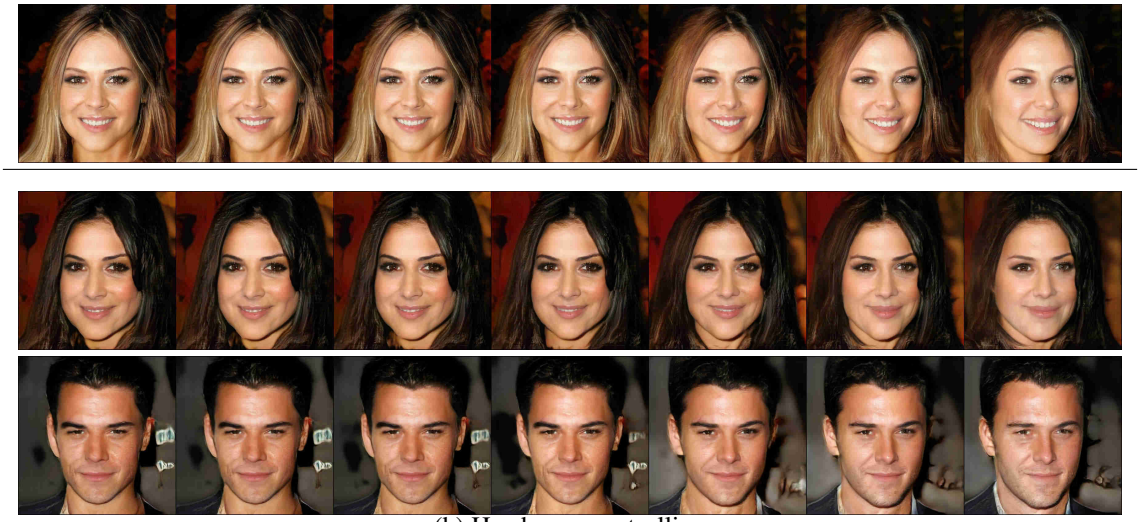
4.6.5 Experimental setup and results of the PCA-AE applied to the latent space of PGAN

In Appendix Figure C.11, we also display the results of our PCA-AE directly to the CelebA data. This gives very blurry results (since the task is very difficult), similar to the results of β-VAE [85] (Figure 4 of their paper), which lead us to our approach to using the PCA-AE applied to pretrained GANs. Therefore, to show the use of our PCA-AE on more high-level data, we take a pre-trained model of PGAN [96] [3] trained with the CelebA dataset [119]. Note that the pre-trained generator is fixed during the training of our PCA-AE. The latent space of PGAN is entangled (we show experiments to support this in Figure C.10), so that a variation along one parameter of this initial code in the latent space can modify several characteristics of the generated images. The latent space size of this pre-trained network is 512. An initial code $\overline{\eta}$, from which the network generates

[3] Pytorch GAN zoo: https://github.com/facebookresearch/pytorch_GAN_zoo



(a) Hair colour controlling



(b) Head pose controlling

Figure 4.28: **Application of PCA-AE to PGAN**. We transfer the learning attributes from the training code (the first row) to other testing codes (the last rows) for (a) hair colour and (b) head pose. The first row shows that we change the hair colour of the generated image from PGAN with respect to a training initial code $\overline{\eta}$ by adjusting the first component of the latent space of PCA-AE. We can see that the hair colours and the head pose of generated images from other testing initial codes are also changed as those of the training code.

a photo-realistic image, is chosen. In order to create the set of random perturbations, we sample from a multivariate Gaussian distribution:

$$\eta \sim \mathcal{N}(0, \sigma^2 \boldsymbol{I}) \tag{4.47}$$

where $\boldsymbol{I}$ is the identity matrix.

We now show our results of PCA-AE for organising the latent space of the pre-trained PGAN [96]. We show an example of the navigation of the latent space of the PGAN in Figure C.12. This is achieved by training our PCA-AE around the code generating the image at the middle of the three grids. We can see that for this example, the first component (z_1) corresponds to the hair

(a) AE

(b) β-VAE$_B$

(c) FactorVAE

(d) PCA-AE

Figure 4.29: **Interpolation in latent space of five components of AE, β-VAE$_B$, FactorVAE and the PCA-AE for the pre-trained PGAN [96].** Two components are adjusted along two axes, the others are set to zeros. We can see that VAE and β-VAE$_B$ mixe hair colour along two components z_1 and z_4, z_2 and z_5 respectively. Head pose corresponds to the components z_1 and z_4 of FactorVAE. Our method, on the contrary, shows that each component of our proposed latent space represents one attribute of the generated images. For example, z_1, z_2, z_3 correspond to hair colours, head poses and gender.

colour from black to blond, the second one (z_2) controls the head pose and the third parameter (z_3) changes the gender.

In order to better visualise the results of the proposed method, we adjust two components which correspond specifically to hair colour and head poses of generated images from the training initial code as shown in the first row of each sub-figure of Figure 4.28. Then, we apply the trained model to other initial codes of the latent space of PGAN. We can see that the image attributes, starting from initial codes $\overline{\eta}$ that have not been seen during training, are successfully edited. This is shown in rows 2,3 and 5,6.

In order to evaluate the disentanglement of the latent space of other methods and ours, we use pre-trained classifiers to determine an attribute of generated images. We choose three main attributes which the classifiers [126] can recognise well, corresponding to the head pose (*i.e.* turning left to right), hair colour (*i.e.* black, brunette and blond) and gender. To demonstrate the performance of our algorithm, we have trained the standard AE, the aforementioned VAE-based methods and our proposed PCA-AE, using the procedure described in Section 4.6.3. Table 4.5 shows the numeric evaluation of the methods and Figure 4.29 shows the generated images of the generator of PGAN from the latent spaces of the other approaches and of our proposed PCA-AE. The other methods construct a latent space where the attributes of the generated images are correlated with more than one component. For example, we can see that the latent space of AE mixes up the attributes. In addition, it can be seen that the fourth parameter of β-TCVAE controls the hair color and the gender of generated images simultaneously. The first parameter of FactorVAE changes the head pose. Then, the third one of this model still corresponds to the head pose. Indeed, the absolute PCC of this model for the head pose is correlated to the first and third components of the latent space. The PCA-AE yields a disentangled latent space which is organised in a hierarchical fashion: the first component corresponds to the colour hair of the generated images, the second one represents head poses (*e.g.* turning left and right), the third parameter corresponds to hair thickness and the last one is mildly correlated to skin tone. Our PCA-AE is able to efficiently separate the different facial attributes and rank them according to their importance in the reconstruction. Thus, the latent space created by our method is easier to interpret and navigate than the original GAN latent space.

We highlight that this procedure can be applied to any pre-trained model, so that the disentangling and organisation of the latent space can be carried out after the initial, computationally expensive, training of a GAN.

4.6.6 Conclusion

This work presented an autoencoder which imitates the properties of the PCA. The first main contribution is a method to progressively increase the size of the autoencoder latent space to organise the latent space by decreasing importance with respect to the ℓ^2 norm. The second contribution is the use of a covariance latent loss which encourages statistical independence of latent codes, thus making these codes correpsond to independent attributes in images. This is also known as disentanglement.

There is one clear deficieny in this approach; in Section 4.6.3, we describe how the application of PCA-AE was only successful *locally* around a certain data point. This is obviously insufficient, since it would require the training of a separate PCA-AE for each data point! Part of my current work is geared towards seeing if it is possible to extend the approach to the whole space.

Chapter 5

Conclusion and future work

In this document, we have presented a variety of mathematical models for the purpose of image analysis, synthesis and editing. The first model used was the low-rank model, which is useful for analysing data that can be represented as a few well-chosen vectors, multiplied by a global constant. This is the case for backgrounds in videos with globally changing lighting conditions. Two algorithms were proposed, the first for detecting foreground objects with multilple timescales, and the second for identiying sub-regions in a video where the low-rank model can be well-applied. The second model we discussed was the Boolean model, from stochastic geometry, which we used to imitate physical silver-halide film grain in digital images. In the last Chapter, we investigated deep generative models (autoencoders and GANs), first understanding how they encoded and decoded simple images, then how autoencoders and GANs could be used to edit face images. Finally, we proposed an autoencoder architecture which imitates the Principal Component Analysis method, to create an organised latent space of this autoencoder and discover variations in images in a similar manner to the PCA, and also to carry out image editing.

There are several points which I am very happy with in the previous subjects. My work on film grain synthesis was particularly satisfying, firstly because I discovered a new research domain (stochastic geometry), but also because we were able to propose a completely new, physically realistic model for this synthesis, as well as an algorithm to render the grain. In this sense, it really was an "A-to-Z" treatment of the subject, with the model being mathematically interesting and producing very impressive results. I do not think I have been able to have such a complete and novel approach on any other subject, also because film grain synthesis is quite a niche problem. Actually, the interest from the non-research world has been quite large (I receive emails concerning the Github code every so often), and I believe that a faster algorithm could be of significant use, whereas the research community's interest has been quite low. I am also very satisfied by the work I carried out as a postdoc with Télécom Paris, where we were able to analyse in detail what happens when autoencoding shapes. The conclusions and analyses we obtained are very precise and mathematically clear, even if generalisation to more complex situations remains challenging. Such conclusions are quite rare in deep learning, and I am very happy with them. Finally, I feel that the PCA-Autoencoder approach is extremely interesting, simple and very relevant in a moment where many of the different research directions in interpretable deep generative models can be very difficult to evaluate. Credit is shared here with Saïd Ladjal and Chi-Hieu Pham who were at the centre of this project.

5.1 Future work

As mentioned at the end of each Section, there are obviously limitations with each method and approach. These will lead to future work in my career, which I discuss now with respect to each subject.

Low-rank models I do not, at this moment in time, plan to continue work on low-rank models in image processing. I have not been following the literature on this subject, and I do not know if, currently, the models still present competitive performances. Furthermore, because most of my work revolves around image editing, these tools are not very relevant for my immediate future research directions. Nevertheless, I am very happy to have them in my toolbox, and a situation may arise where such models can be combined with deep learning approaches, in which case I will be happy to come back to them.

Stochastic geometry models I found (and still find) the work carried out on film grain synthesis to be extremely interesting, and I would like to carry on with it. Obviously the main problem to solve is that of execution time. Indeed, the first algorithm requires 4.5 seconds for a 2048×2048 pixel image (see Table 3.1) with a GPU, which is still too slow for film restoration purposes, and is prohibitively slow on a CPU (three hours). As mentioned in Section 3.5, one encouraging possibility, which I intend to pursue, is to train a neural network to add grain. Compared to many of the other types of textures which are possible to synthesise with methods such as that of Gatys *et al.* [67], film grain seems quite simple. However, the difficult task lies in introducing the physical parameters which made our film grain algorithm so powerful. This is not currently commonly done with the most recent deep texture synthesis methods [168], which simply learn to imitate the example at hand (and one network is trained for each texture type). A very tempting option is to use the variational autoencoder [102], which have been briefly mentioned in this manuscript. Indeed, these seem tailor-made for the case of textures, since they learn to represent data in an autoencoder's latent space, and the data in the latent space are encouraged to be distributed with a chosen probability distribution during training. Textures seem to fall into this category exactly. However, I have myself, in the context of my postdoc with Télécom Paris, found this not to be so straightforward. Furthermore, I also worked on a similar problem with Lara Raad (assistant professor with ESIEE) during her postdoc at Télécom Paris, and similar problems were observed. The main solution to including truly random elements in an image via a deep generative model seems to be to inject some noise into the latent space, and then generate the image. StyleGAN [95] takes this approach, with very good results. It may be possible to train an autoencoder-type network whose latent space contains both a stochastic element and a few components containing the physical parameters. This would obviously require training examples with many different variations of the parameters, but this would be possible since we can create as many example as we like with our model, with the parameter annotations if they prove to be necessary.

Another question is that of the model itself. Indeed, while we were able to tune the parameters to imitate real film grain (see Figures 3.12 and 3.13), the imitation is not perfect. As mentioned in Section 3.3.7, certain grain patterns are difficult to imitate, in particular the curious connectedness of white grain in the Ilford Delta 3200 grain of Figure 3.13. This structured behaviour is quite contrary to the Boolean model, since it seems to not be the case for the dark grains, and the Boolean model is supposed to be symmetric. I would very much like to speak to grain experts about such phenomena, in order to figure out whether it is a fundamental limitation of the Boolean model, or simply a quirk of the DxO FilmPack software. If it is indeed a limitation of the model,

there are several things that can be tried. Firstly, we can use different grain shapes. This is not actually a limitation of the Boolean model, since it is defined (Subsection 3.3.1) using any compact random set. In our implementation, we only used disks. I do not believe that this will greatly change the results, since I have carried out some experiments with triangles (see Appendix B.2), with little change. Another question is that of the simplification made when considering that the grains can overlap. Indeed, in reality, the physical grains obviously do not overlap in three dimensions, however since we are looking at a 2D projection of the physical configuration of the grains, the overlapping Boolean model seemed reasonable. There are more complicated models in the stochastic geometry literature, called determinantal point processes [87], where the random objects cannot overlap. This model would be more involved, but more accurate. Its usefulness obviously depends on the magnitude of grains versus that of the film emulsion. A quick search finds that film grains are about 0.2 to 2 micrometres (Wikipedia), whereas the film grain emulsion's thickness is about 10 micrometres [160]. In extreme cases, therefore, it may be the case that the standard Boolean model is too approximate and a determinantal process may be needed. Finally, there is the question of the grain sensitivity. Indeed, in all our work, we have chosen the density of the Poisson point process in the Boolean model to respect the grey-level of the input digital image on which we want to put film grain. However, this may not be the most realistic hypothesis to get the right feel for a certain film type. Indeed, the probability of sensitisation of film grains is related to their size. Thus, in the real world, different film types require different exposure times (this is known as film speed). The process of sensitisation, and in particular the relationship between light recieved and sensitisation probability, is well explained by Anderson [30]. It may be necessary to review and simulate this whole process to get a good feel of different film types. Geigel and Musgrave [68] proposed a complete simulation of the film grain process, but apart from this work I do not know of any other which does this. In all of these potential avenues of research, my first goal would be to reach out to photography specialists (both on the artistic and chemical/physical sides) to find their opinion on our current algorithms and future suggestions. I would have liked to have done this in my postdoc with Université Paris Descartes, however lack of time made this difficult.

Deep generative models At its core, most of my work concerning deep generative models has been geared towards creating latent spaces whose elements are understandable and interpretable. My initial work on autoencoding simple shapes showed that in that case a clearly understandable latent space was indeed achieved. The subsequent work took several approaches, but the one which currently stands out is that of the work of my former PhD student, Xu Yao, on face image attribute editing [11]. In this approach, the deep generative model is trained first, and then we try to use it a posteriori for editing. The main reason why this is such a popular approach is that it basically "outsources" the learning of the deep generative model to other authors who have special experience. Indeed, it is notoriously difficult to train high-quality GANs, and it is not a coincidence that the StyleGAN models were created by authors working at Nvidia, who have access to many GPUs. Thus, they have accumulated great expertise over the years on training GANs and it is difficult to compete with them. The question is therefore, what are the natural properties of the StyleGAN type models, and how can we navigate their latent spaces for efficient editing.

In the work of Xu Yao, this was done by training a network to carry out this navigation. The main problem is that a different network has to be trained for each attribute. This is clearly inefficient. However other simple approaches such as the popular work "InterfaceGAN" [156], which learn a single constant direction for each attribute, do not achieve good disentanglement. Recently (October 2021), I recieved a ANR Jeunes Chercheurs Jeunes Chercheuses grant, called

"IDeGeN" to carry out work on how to edit in the latent space of powerful GANs. In this project, which has already started with a PhD student, Gwilherm Lesné, I wish to see if it is possible to project the latent space to another intermediate space where the attributes are disentangled. This is in fact a similar approach to the one taken in Section 4.6 for GANs, except we are carrying it out on the more complex space $\mathcal{W}$ of StyleGAN. This will be the first task. The second will be to see if we can, by the previous method if possible, create a space where the editing is carried out in a smooth manner, that is to say where a displacement in the latent space in the right direction leads to the same quantitative modification of an attribute. This may not always be possible (as in the case of truly binary attributes - eg. glasses) but is a desirable property for editors. Another question which I will try to tackle is to see whether the editing directions and magnitudes required for a certain edit vary or not depending upon the position in the latent space. The work of Xu Yao supposes that it does indeed, but I feel that this needs more investigation, since preliminary results suggested that the direction did not change that much, whereas the magnitude did seem to. This is linked with a limitation of the work of my postdoc, Chi-Hieu Pham, explained in Section 4.6.3, which is that the PCA-GAN (ie the PCA-AE applied to a pre-trained GAN) had to be trained locally. The algorithm did not seem to work satisfactorily globally, which is clearly a major weakness, because a new PCA-AE would have to be trained for each data point. In this sense, the algorithm is not yet ready for real-world use. A major goal, therefore, will be to try and create such a network which does indeed work globally. This requires investigating the latent space properties of StyleGAN, and my PhD student Gwilherm Lesné is indeed working on this at the moment. Another question which interests me is the following. In the case of StyleGAN-type models, how do we ensure that we remain on the subspace of $\mathcal{W}$ that corresponds to real, photo-realistic images. Indeed, in traditional GANs, this is not a problem, since we know that the data live on a hyper-sphere, given that they are distributed with a Gaussian distribution in a high-dimensional space. However, when $\mathcal{Z}$ is transformed into $\mathcal{W}$ via a series of fully-connected layers, it is not clear how to remain on that space when either navigating or projecting onto it. I would like to investigate how to do this, and in general the sub-space of photo-realistic images. One option is to retrace the path to the latent space $\mathcal{Z}$, and then reproject to $\mathcal{W}$. This requires inverting both the linearities and non-linearities of the fully-connected layers of StyleGAN. Luckily, the non-linearities are Leaky ReLUs, which are invertible. However, the matrices of the linear transformations may have poor condition numbers then this might be difficult. This would imply that the unit sphere gets excessively stretched/distorted by the transformations. This, in turn, seems likely because the authors of StyleGAN suppose that $\mathcal{W}$ is indeed flattened (see Figure 6 of [98]). Thus, this may be challenging or infeasible. Another approach would be to try to determine the sub-space empirically using a PCA on $\mathcal{W}$. Thus, we could resrict editing to be carried out in this sub-space, or project to it, which would result in more robust and reliable editing. A potential application of this is the recent trend in using deep generative models as regularisers in inverse problems, also known as Plug-and-Play priors [32, 73]. In these methods, it is required to project onto such spaces, so these questions are equally of interest. I intend to investigate most of the previous questions in the IDeGeN project, which is to last 48 months, so until October 2026.

Combining classical models into deep learning I mentioned at the very beginning of this manuscript that I had previously worked on patch-based methods for video inpainting. I am continuing my work on inpainting with a PhD student, Nicolas Cherel, financed with an internal grant of Télécom Paris. When looking at the literature on this subject in the deep learning world, it was apparent that most modern methods used what is called *self-attention*. This originates in the world of natural language processing (NLP) [170], and is known as a "Transformer". The initial

goal was to avoid problems in gradient passing through neural networks during training. However, from our point of view, attention, and in particular self-attention, works in a very similar manner to patches. Indeed, it is a module which reconstructs the data of a layer of a neural network using a weighted sum of the information in that layer[1]. More precisely, if Q is a list of query patches of data, K are *key* patches and V is a list of *value* patches, the general formulation of the attention module is defined as:

$$\text{Attention}(Q, K, V) = \text{softmax}(QK^T)V. \tag{5.1}$$

In intuitive terms, the weights are given by comparisons of the queries with the keys via a dot product. The softmax ensures that the weights sum to 1, and the multiplication with V is the final reconstruction. For self-attention, Q, K and V are the same set of patches. This bears a striking resemblance to Non-Local Means [39]. Indeed, one way of seeing self-attention is that is allows for patch-based methods to be inserted into deep neural networks. In the case of inpainting, this is particularly important, since convolutional layers only cannot achieve what patches can achieve. Thus, in most approaches, the two are combined. In the work of Yu *et al*. [184], for example, two paths in the network are proposed: one standard convolution/upsampling path, and one with self-attention. Thus, the network can use both according to the requirements of the situation. Unfortunately, this approach does not scale to large images, let alone to videos. Thus, part of the work I am doing with Nicolas Cherel is to find a solution to this problem. If this can be done, it will be possible to propose hybrid CNN/patch-based methods in high-dimensional problems.

Concerning databases In Section 4.2.5, we considered what happens to autoencoders in the case of incomplete databases. This is indeed the problem of generalisation and/or overfitting. This study was purely emprical, and I would like to obtain more theoretical results if we have either a region of unkown data (in the sense of a connected set of the parameters is missing), or data is randomly missing. While literature exists on this subject in the case of general data, I do not believe that there is any on the case of images. Necessarily, these would be parametric images in a first step. I have started to work towards these questions with Yann Traonmilin (chargé de recherche, CNRS, IMB Bordeaux). More generally, the question of the quality of databases interests me. For example, in my some of my lab works on deep learning, I use the CIFAR10 dataset. My students often remark, quite rightly, that the dimensions of the images in this dataset are so small that it is difficult to distinguish visually between the classes or even identify objects. This leads to the question, what are the conditions which a database needs to verify for a classification to be meaningful? This is a more ambitious question that I have not started to address, but an example of a first approach would be to see how well separated classes are by a support vector machine, depending on the image resolution.

5.2 Summary of students supervised, collaborations and other responsabilities and contributions to the research community

I have had the pleasure to supervised, or am supervising the following students:

- Arthur Ouaknine, CIFRE PhD (Valéo). Co-supervised with Florence Tupin (Professeure, Télécom Paris) and Patrick Pérez (Valéo). He defended his thesis on the 4th March 2022. This work (detection in radar images) was not included since it is somewhat removed from the themes of the manuscript. Now postdoc at the MILA lab (Montréal, Canada);

[1] Note, that this is a description of self-attention

- Xu Yao, CIFRE PhD (Interdigital). Co-supervised with Pierre Helier (Interdigital) and Yann Gousseau (Télécom Paris), working on facial image editing with deep learning. Defended 8th April 2022. Now reserch engineer with.
- Chi-Hieu Pham (postdoc until 2020). Working on the PCA-Autoencoder. Now Assistant Lecturer with ISEN Yncréa Ouest (Brest).
- Nicolas Cherel (ongoing, started September 2020). Co-supervised with Yann Gousseau (Télécom Paris) and Andrés Almansa (Université Paris Descartes). Working on video inpainting.
- Gwilherm Lesné (ongoing, started October 2021). Co-supervised with Saïd Ladjal (Télécom Paris). Working on image editing with deep generative networks. Financed by the ANR Jeunes Chercheurs, Jeunes Chercheuses grant I obtained in 2021.
- Raphaël Remé (ongoing, to start soon). Co-supervised with Elsa Angelini (Télécom Paris), Thibault Lagache (Institut Pasteur), Jean-Christophe Olivo-Marin (Institut Pasteur). Working on deep learning for the tracking of micro-organisms.

As mentioned above, in 2021 I obtained an ANR Jeunes Chercheurs, Jeunes Chercheuses grant (project name IDeGeN), to finance one PhD and one postdoc over four years, on the subject of image editing with deep generative networks, a central theme of my research. The PhD student started in October 2021, and the postdoc will start later in the project.

I have reviewed papers for the following conferences and journals: AAAI, CVIU, ICIP, IPOL, JMIV, PAMI, SIIMS, TIP, TSP. I currently organise the monthly IMAGES team seminar with Christophe Kervazo (Assistant Professor, Télécom Paris). Also, I have recently taken the initiative, with Vincent Duval (researcher with INRIA), to start up the "Imaging in Paris" series of seminars which take place at the Institut Henri Poincaré (Paris). These very popular seminars existed before the COVID pandemic and were shut down during. The first seminar will take place on the 13th October 2022.

I have, of course, given lessons during my research career. At the moment at Télécom Paris, these take up about 40-50% of my time. I enjoy teaching immensely, and am happy to be able to continue to do so. I teach in about 8 different courses with Télécom Paris, the Data Sciences M2 of the Institut Polytechnique de Paris and the Mathématiques, Vision, Apprentissage (MVA) M2 of Paris Saclay. Furthermore, I teach in two "formation continue" courses at Télécom Paris. I enjoy both the fundamental courses such as digital signal processing and the more advanced courses such as generative models. The latter are a great opportunity to discuss new ideas with students, and potentially recruit new PhD students. Student projects at such a level are in particular a great way to test tentative ideas which a PhD student may not have time to address. Thus, I wish to keep teaching at the centre of my research career.

Chapter 6

Bibliography

Here is the full list of my publications, grouped by journal articles, conference articles, invited colloquium articles, and other. This is followed by the publications of others.

Journal Articles

[J1] Alasdair Newson, Julie Delon, and Bruno Galerne. A Stochastic Film Grain Model for Resolution-Independent Rendering. In *CGF*, volume 36, pages 684–699, 2017.

[J2] Alasdair Newson, Noura Faraj, Bruno Galerne, and Julie Delon. Realistic film grain rendering. *Image Processing On Line*, 7:165–183, 2017.

[J3] Alasdair Newson, Andrés Almansa, Yann Gousseau, and Saïd Ladjal. Processing Simple Geometric Attributes with Autoencoders. *JMIV*, 2019.

[J4] Chi-Hieu Pham, Saïd Ladjal, and Alasdair Newson. PCA-AE: Principal Component Analysis Autoencoder for Organising the Latent Space of Generative Networks. *JMIV*, 64(5):569–585, June 2022.

[J5] Alasdair Newson, Andrés Almansa, Yann Gousseau, and Patrick Pérez. Robust automatic line scratch detection in films. *IEEE TIP*, 2014.

[J6] Alasdair Newson, Andrés Almansa, Matthieu Fradet, Yann Gousseau, and Patrick Pérez. Video Inpainting of Complex Scenes. *SIAM Journal on Imaging Sciences*, 7(4):1993–2019, January 2014.

Conference Articles

[C7] Alasdair Newson, Mariano Tepper, and Guillermo Sapiro. Low-Rank Spatio-Temporal Video Segmentation. In *BMVC*, pages 103–1, 2015.

[C8] Mariano Tepper, Alasdair Newson, Pablo Sprechmann, and Guillermo Sapiro. Multi-temporal foreground detection in videos. In *2015 IEEE International Conference on Image Processing (ICIP)*, pages 4599–4603. IEEE, 2015.

[C9] Alasdair Newson, Noura Faraj, Julie Delon, and Bruno Galerne. Analysis of a physically realistic film grain model, and a Gaussian film grain synthesis algorithm. In *SSVM*, pages 196–207. Springer, 2017.

[C10] X. Yao, G. Puy, A. Newson, Y. Gousseau, and P. Hellier. High Resolution Face Age Editing. In *ICPR*, 2020.

[C11] Xu Yao, Alasdair Newson, Yann Gousseau, and Pierre Hellier. A Latent Transformer for Disentangled Face Editing in Images and Videos. pages 13789–13798, 2021.

[C12] Alasdair Newson, Patrick Pérez, Andrés Almansa, and Yann Gousseau. Adaptive line scratch detection in degraded films. In *CVMP*, pages 66–74, 2012.

[C13] Alasdair Newson, Andrés Almansa, Yann Gousseau, and Patrick Pérez. Temporal filtering of line scratch detections in degraded films. In *ICIP*, pages 4088–4092. IEEE, 2013.

[C14] Alasdair Newson, Andrés Almansa, Matthieu Fradet, Yann Gousseau, and Patrick Pérez. Towards fast, generic video inpainting. In *CVMP*, pages 1–8, 2013.

[C15] A. Ouaknine, A. Newson, J. Rebut, F. Tupin, and Patrick Pérez. CARRADA Dataset: Camera and Automotive Radar with Range-Angle-Doppler Annotations. In *ICPR*, 2020.

[C16] Xu Yao, Gilles Puy, Alasdair Newson, Yann Gousseau, and Pierre Hellier. High Resolution Face Age Editing. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 8624–8631, January 2021. ISSN: 1051-4651.

[C17] Arthur Ouaknine, Alasdair Newson, Patrick Pérez, Florence Tupin, and Julien Rebut. Multi-View Radar Semantic Segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15671–15680, 2021.

[C18] Nicolas Cherel, Andrés Almansa, Yann Gousseau, and Alasdair Newson. Patch-Based Stochastic Attention for Image Editing, February 2022. arXiv:2202.03163 [cs] (accepted to ICIP 2022).

[C19] Xu Yao, Alasdair Newson, Yann Gousseau, and Pierre Hellier. Feature-Style Encoder for Style-Based GAN Inversion, February 2022. arXiv:2202.02183 [cs], (accepted to ECCV 2022).

Invited Colloquiums

[I20] Alasdair Newson, Andres Almansa, Matthieu Fradet, Yann Gousseau, and Patrick Perez. Variational Patch-Based Video Inpainting. In *Rank Prize Funds Symposium on Computer Vision and Video Effects Generation*, 2016.

PhD Thesis

[T21] Alasdair Newson. *On Video Completion: Line Scratch Detection in Films and Inpainting of Complex Scenes*. PhD thesis, Telecom Paris, 2014.

Other

[O22] Alasdair Newson and Julie Delon. Un Modele Aleatoire pour le Grain Photographique. *Image des Mathématiques*, August 2020.

Bibliography

[23] FILMPAC Cinematic Stock Footage and Premium Stock Music, 2017.

[24] Rameen Abdal, Yipeng Qin, and Peter Wonka. Image2stylegan: How to embed images into the stylegan latent space? In *ICCV*, pages 4432–4441, 2019.

[25] Rameen Abdal, Yipeng Qin, and Peter Wonka. Image2StyleGAN++: How to Edit the Embedded Images? pages 8296–8305, 2020.

[26] Rameen Abdal, Peihao Zhu, Niloy J. Mitra, and Peter Wonka. StyleFlow: Attribute-conditioned Exploration of StyleGAN-Generated Images using Conditional Continuous Normalizing Flows. *ACM Transactions on Graphics*, 40(3):21:1–21:21, 2021.

[27] David H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski. A learning algorithm for boltzmann machines. *Cognitive Science*, 9(1):147–169, January 1985.

[28] Guillaume Alain and Yoshua Bengio. What regularized auto-encoders learn from the data-generating distribution. *The Journal of Machine Learning Research*, 15(1):3563–3593, January 2014.

[29] Luigi Ambrosio and Vincenzo Maria Tortorelli. Approximation of functional depending on jumps by elliptic functional via t-convergence. *Communications on Pure and Applied Mathematics*, 43(8):999–1036, 1990.

[30] William J. Anderson. Probabilistic Models of the Photographic Process. In *Advances in the Statistical Sciences: Applied Probability, Stochastic Processes, and Sampling Theory*, volume 34, pages 9–40. Springer Netherlands, 1987.

[31] Grigory Antipov, Moez Baccouche, and Jean-Luc Dugelay. Face aging with conditional generative adversarial networks. In *2017 IEEE international conference on image processing (ICIP)*, pages 2089–2093, 2017.

[32] Simon Arridge, Peter Maass, Ozan Öktem, and Carola-Bibiane Schönlieb. Solving inverse problems using data-driven models. *Acta Numerica*, 28:1–174, May 2019. Publisher: Cambridge University Press.

[33] Soonmin Bae, Sylvain Paris, and Frédo Durand. Two-scale tone management for photographic look. *ACM Transactions on Graphics*, 25(3):637–645, 2006.

[34] B. E. Bayer. Relation Between Granularity and Density for a Random-Dot Model. *Journal of the Optical Society of America*, 54(12):1485+, December 1964.

[35] Yoshua Bengio and Martin Monperrus. Non-local manifold tangent learning. *Advances in Neural Information Processing Systems*, 2005.

[36] Hervé Bourlard and Yves Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*, 59(4):291–294, September 1988.

[37] Thibault Briand, Jonathan Vacher, Bruno Galerne, and Julien Rabin. The Heeger & Bergen pyramid based texture synthesis algorithm. *Image Processing On Line*, 4:276–299, 2014.

[38] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large Scale GAN Training for High Fidelity Natural Image Synthesis. Technical Report arXiv:1809.11096, arXiv, February 2019.

[39] Antoni Buades, Bartomeu Coll, and Jean-Michel Morel. A non-local algorithm for image denoising. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 60–65 vol. 2, June 2005.

[40] Adrian Bulat and Georgios Tzimiropoulos. How Far Are We From Solving the 2D & 3D Face Alignment Problem? (And a Dataset of 230,000 3D Facial Landmarks). In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1021–1030, 2017.

[41] Christopher P. Burgess, Irina Higgins, Arka Pal, Loic Matthey, Nick Watters, Guillaume Desjardins, and Alexander Lerchner. Understanding disentangling in β-VAE. *arXiv:1804.03599 [cs, stat]*, April 2018.

[42] Emmanuel J Candès, Xiaodong Li, Yi Ma, and John Wright. Robust principal component analysis? *Journal of the ACM (JACM)*, 58(3):1–37, 2011.

[43] Vincent Caselles, Ron Kimmel, and Guillermo Sapiro. Geodesic active contours. In *Proceedings of IEEE International Conference on Computer Vision*, pages 694–699, June 1995.

[44] P. E. Castro, J. H. B. Kemperman, and E. A. Trabka. Alternating renewal model of photographic granularity. *Journal of the Optical Society of America*, 63(7):820+, July 1973.

[45] Imen Charfi, Johel Miteran, Julien Dubois, Mohamed Atri, and Rached Tourki. Definition and Performance Evaluation of a Robust SVM Based Fall Detection Solution. In *2012 Eighth International Conference on Signal Image Technology and Internet Based Systems*, pages 218–224, November 2012.

[46] Bor-Chun Chen, Chu-Song Chen, and Winston H. Hsu. Cross-age reference coding for age-invariant face recognition and retrieval. In *ECCV*, pages 768–783. Springer, 2014.

[47] Ricky T. Q. Chen, Xuechen Li, Roger B Grosse, and David K Duvenaud. Isolating Sources of Disentanglement in Variational Autoencoders. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[48] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. InfoGAN: interpretable representation learning by information maximizing generative adversarial nets. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pages 2180–2188, Red Hook, NY, USA, December 2016.

[49] Xuanhong Chen, Bingbing Ni, Naiyuan Liu, Ziang Liu, Yiliu Jiang, Loc Truong, and Qi Tian. CooGAN: A Memory-Efficient Framework for High-Resolution Facial Attribute Editing. In *Computer Vision – ECCV 2020*, Lecture Notes in Computer Science, pages 670–686, Cham, 2020.

[50] Ying-Cong Chen, Xiaohui Shen, Zhe Lin, Xin Lu, I.-Ming Pao, and Jiaya Jia. Semantic Component Decomposition for Face Attribute Manipulation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9859–9867, 2019.

[51] Brian Cheung, Jesse A. Livezey, Arjun K. Bansal, and Bruno A. Olshausen. Discovering Hidden Factors of Variation in Deep Networks. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*, 2015.

[52] Sung Nok Chiu, Dietrich Stoyan, Wilfried S. Kendall, and Joseph Mecke. *Stochastic geometry and its applications*. John Wiley & Sons, third edition, 2013.

[53] 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.

[54] Edo Collins, Raja Bala, Bob Price, and Sabine Susstrunk. Editing in style: Uncovering the local semantics of gans. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5771–5780, 2020.

[55] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, December 1989.

[56] Andrew Delong, Anton Osokin, Hossam N. Isack, and Yuri Boykov. Fast Approximate Energy Minimization with Label Costs. *International Journal of Computer Vision*, 96(1):1–27, January 2012.

[57] Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur. A Learned Representation For Artistic Style. Technical Report arXiv:1610.07629, arXiv, February 2017.

[58] DxO. DxO Film Pack 5, 2016. http://www.dxo.com/us/photography/photo-software/dxo-filmpack.

[59] Jose I. Echevarria, Gregg Wilensky, Aravind Krishnaswamy, Byungmoon Kim, and Diego Gutierrez. Computational Simulation of Alternative Photographic Processes. *Computer Graphics Forum*, 32(4):7–16, 2013. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.12146.

[60] Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, September 1936.

[61] Jeffrey L. Elman and David Zipser. Learning the hidden structure of speech. *The Journal of the Acoustical Society of America*, 83(4):1615–1626, April 1988.

[62] Yun Fu, Guodong Guo, and Thomas S. Huang. Age Synthesis and Estimation via Faces: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(11):1955–1976, November 2010.

[63] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, April 1980.

[64] Bruno Galerne, Yann Gousseau, and Jean-Michel Morel. Random Phase Textures: Theory and Synthesis. *IEEE Trans. Image Process.*, 20(1):257 – 267, 2011.

[65] Bruno Galerne, Yann Gousseau, and Jean-Michel Morel. Random Phase Textures: Theory and Synthesis. *IEEE Transactions on Image Processing*, 20(1):257–267, January 2011.

[66] Bruno Galerne, Arthur Leclaire, and Lionel Moisan. Texton Noise. Technical Report 2016-09, MAP5, 2016.

[67] Leon Gatys, Alexander S Ecker, and Matthias Bethge. Texture Synthesis Using Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[68] Joe Geigel and F. Kenton Musgrave. A model for simulating the photographic development process on digital images. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '97, pages 135–142, USA, August 1997.

[69] Ross Girshick. Fast R-CNN. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1440–1448, 2015.

[70] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014.

[71] G'MIC. GREYC's Magic for Image Computing, 2016.

[72] Lore Goetschalckx, Alex Andonian, Aude Oliva, and Phillip Isola. Ganalyze: Toward visual definitions of cognitive image properties. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5744–5753, 2019.

[73] Mario González, Andrés Almansa, and Pauline Tan. Solving Inverse Problems by Joint Posterior Maximization with Autoencoding Prior. *SIAM Journal on Imaging Sciences*, 15(2):822–859, June 2022.

[74] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[75] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[76] Grubbasoftware. TrueGrain, 2015.

[77] Gurney. The theory of the photolysis of silver bromide and the photographic latent image. *Proceedings of the Royal Society of London*, 164:151–167, 1938.

[78] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality Reduction by Learning an Invariant Mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1735–1742, June 2006.

[79] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. Mask R-CNN. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2961–2969, 2017.

[80] 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.

[81] Keke He, Yanwei Fu, Wuhao Zhang, Chengjie Wang, Yu-Gang Jiang, Feiyue Huang, and Xiangyang Xue. Harnessing Synthesized Abstraction Images to Improve Facial Attribute Recognition. In *IJCAI*, pages 733–740, 2018.

[82] Zhenliang He, Meina Kan, Shiguang Shan, and Xilin Chen. S2GAN: Share Aging Factors Across Ages and Share Aging Trends Among Individuals. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9439–9448, October 2019. ISSN: 2380-7504.

[83] Zhenliang He, Wangmeng Zuo, Meina Kan, Shiguang Shan, and Xilin Chen. AttGAN: Facial Attribute Editing by Only Changing What You Want. *IEEE Transactions on Image Processing*, 28(11):5464–5478, November 2019. Conference Name: IEEE Transactions on Image Processing.

[84] David J. Heeger and James R. Bergen. Pyramid-based texture analysis/synthesis. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '95, pages 229–238, New York, NY, USA, September 1995. Association for Computing Machinery.

[85] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. 2016.

[86] Wassily Hoeffding. The strong law of large numbers for U-statistics. *Institute of Statistics mimeo series*, 302, 1961. Publisher: Chapel Hill UNC.

[87] J. Ben Hough, Manjunath Krishnapur, Yuval Peres, and Bálint Virág. Determinantal Processes and Independence. *Probability Surveys*, 3(none), January 2006.

[88] Xun Huang and Serge Belongie. Arbitrary Style Transfer in Real-Time With Adaptive Instance Normalization. pages 1501–1510, 2017.

[89] Xun Huang, Ming-Yu Liu, Serge Belongie, and Jan Kautz. Multimodal Unsupervised Image-to-image Translation. pages 172–189, 2018.

[90] D. H. Hubel and T. N. Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of Physiology*, 195(1):215–243, March 1968.

[91] Erik Härkönen, Aaron Hertzmann, Jaakko Lehtinen, and Sylvain Paris. Ganspace: Discovering interpretable gan controls. *arXiv preprint arXiv:2004.02546*, 2020.

[92] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 448–456. PMLR, June 2015. ISSN: 1938-7228.

[93] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-To-Image Translation With Conditional Adversarial Networks. pages 1125–1134, 2017.

[94] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual Losses for Real-Time Style Transfer and Super-Resolution. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, Lecture Notes in Computer Science, pages 694–711, 2016.

[95] Teo Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *CVPR*, pages 8110–8119, 2020.

[96] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

[97] Tero Karras, Miika Aittala, Samuli Laine, Erik Härkönen, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Alias-Free Generative Adversarial Networks. Technical Report arXiv:2106.12423, arXiv, October 2021. arXiv:2106.12423 [cs, stat] type: article.

[98] Tero Karras, Samuli Laine, and Timo Aila. A Style-Based Generator Architecture for Generative Adversarial Networks. pages 4401–4410, 2019.

[99] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, January 1988.

[100] Hyunjik Kim and Andriy Mnih. Disentangling by Factorising. In *Proceedings of the 35th International Conference on Machine Learning*, pages 2649–2658. PMLR, July 2018.

[101] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[102] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *ICLR*, 2014.

[103] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[104] Abhishek Kumar, Prasanna Sattigeri, and Avinash Balakrishnan. Variational Inference of Disentangled Latent Concepts from Unlabeled Observations, December 2018.

[105] A. Lagae, S. Lefebvre, R. Cook, T. DeRose, G. Drettakis, D. S. Ebert, J. P. Lewis, K. Perlin, and M. Zwicker. A Survey of Procedural Noise Functions. *Computer Graphics Forum*, 29(8):2579–2600, December 2010.

[106] A. Lagae, S. Lefebvre, G. Drettakis, and P. Dutré. Procedural Noise using Sparse Gabor Convolution. *SIGGRAPH '09*, 28(3), August 2009.

[107] Guillaume Lample, Neil Zeghidour, Nicolas Usunier, Antoine Bordes, Ludovic DENOYER, and Marc' Aurelio Ranzato. Fader Networks:Manipulating Images by Sliding Attributes. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[108] William H. Lawton, Eugene A. Trabka, and Donald R. Wilder. Crowded Emulsions: Granularity Theory for Multilayers. *JOSA*, 62(5):659–667, May 1972. Publisher: Optica Publishing Group.

[109] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551, December 1989. Conference Name: Neural Computation.

[110] José Lezama. Overcoming the Disentanglement vs Reconstruction Trade-off via Jacobian Supervision. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[111] Liyuan Li, Weimin Huang, Irene Y. H. Gu, and Qi Tian. Foreground object detection from videos containing complex background. In *Proceedings of the eleventh ACM international conference on Multimedia*, pages 2–10, New York, NY, USA, November 2003.

[112] Liyuan Li, Weimin Huang, Irene Yu-Hua Gu, and Qi Tian. Statistical modeling of complex backgrounds for foreground object detection. *IEEE Transactions on Image Processing*, 13(11):1459–1472, November 2004.

[113] Peipei Li, Yibo Hu, Ran He, and Zhenan Sun. Global and Local Consistent Wavelet-Domain Age Synthesis. *IEEE Transactions on Information Forensics and Security*, 14(11):2943–2957, November 2019.

[114] Yiyi Liao, Yue Wang, and Yong Liu. Graph Regularized Auto-Encoders for Image Representation. *IEEE Transactions on Image Processing*, 26(6):2839–2852, June 2017.

[115] Zhouchen Lin, Minming Chen, and Yi Ma. The Augmented Lagrange Multiplier Method for Exact Recovery of Corrupted Low-Rank Matrices. *CoRR*, abs/1009.5055, 2010.

[116] Ming Liu, Yukang Ding, Min Xia, Xiao Liu, Errui Ding, Wangmeng Zuo, and Shilei Wen. STGAN: A Unified Selective Transfer Network for Arbitrary Image Attribute Editing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3673–3682, 2019.

[117] Rosanne Liu, Joel Lehman, Piero Molino, Felipe Petroski Such, Eric Frank, Alex Sergeev, and Jason Yosinski. An Intriguing Failing of Convolutional Neural Networks and the CoordConv Solution. *arXiv preprint arXiv:1807.03247*, 2018.

[118] Yunfan Liu, Qi Li, and Zhenan Sun. Attribute-Aware Face Aging With Wavelet-Based Generative Adversarial Networks. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11869–11878, June 2019. ISSN: 2575-7075.

[119] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep Learning Face Attributes in the Wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[120] Robert Livingston. The Theory of the Photographic Process. By C. E. Kenneth Mees. *J. Phys. Chem.*, 49(5):509, May 1945.

[121] Chen Change Loy, Timothy M. Hospedales, Tao Xiang, and Shaogang Gong. Stream-based joint exploration-exploitation active learning. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1560–1567, June 2012.

[122] Lucia Maddalena and Alfredo Petrosino. A fuzzy spatial coherence-based approach to background/foreground separation for moving object detection. *Neural Computing and Applications*, 19(2):179–186, March 2010.

[123] Alireza Makhzani and Brendan Frey. K-sparse autoencoders. *arXiv preprint arXiv:1312.5663*, 2013.

[124] Xudong Mao, Qing Li, Haoran Xie, Raymond Y. K. Lau, Zhen Wang, and Stephen Paul Smolley. Least Squares Generative Adversarial Networks. pages 2794–2802, 2017.

[125] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, December 1943.

[126] megvii. Face++. https://www.faceplusplus.com/.

[127] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which Training Methods for GANs do actually Converge? In *Proceedings of the 35th International Conference on Machine Learning*, pages 3481–3490. PMLR, July 2018.

[128] Mehdi Mirza and Simon Osindero. Conditional Generative Adversarial Nets. *arXiv:1411.1784 [cs, stat]*, November 2014. arXiv: 1411.1784.

[129] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral Normalization for Generative Adversarial Networks. February 2018.

[130] David Mumford and Jayant Shah. Optimal approximations by piecewise smooth functions and associated variational problems. *Communications on Pure and Applied Mathematics*, 42(5):577–685, July 1989.

[131] Richard Nock and Frank Nielsen. Statistical region merging. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(11):1452–1458, November 2004.

[132] P. G. Nutting. On the absorption of light in heterogeneous media. *Philosophical Magazine*, 26(153):423–426, September 1913.

[133] Byung Tae Oh, Shaw-min Lei, and C.-C. Jay Kuo. Advanced Film Grain Noise Extraction and Synthesis for High-Definition Video Coding. *IEEE Transactions on Circuits and Systems for Video Technology*, 19(12):1717–1729, December 2009.

[134] Ron Ohlander, Keith Price, and D. Raj Reddy. Picture segmentation using a recursive region splitting method. *Computer Graphics and Image Processing*, 8(3):313–333, December 1978.

[135] Sveinn Palsson, Eirikur Agustsson, Radu Timofte, and Luc Van Gool. Generative Adversarial Style Transfer Networks for Face Aging. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2165–21658, June 2018. ISSN: 2160-7516.

[136] Taesung Park, Jun-Yan Zhu, Oliver Wang, Jingwan Lu, Eli Shechtman, Alexei Efros, and Richard Zhang. Swapping Autoencoder for Deep Image Manipulation. In *Advances in Neural Information Processing Systems*, volume 33, pages 7198–7211. Curran Associates, Inc., 2020.

[137] Omkar M. Parkhi, Andrea Vedaldi, and Andrew Zisserman. Deep Face Recognition. In *Procedings of the British Machine Vision Conference 2015*, pages 41.1–41.12, Swansea, 2015.

[138] Stanislav Pidhorskyi, Donald A. Adjeroh, and Gianfranco Doretto. Adversarial Latent Autoencoders. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14104–14113, 2020.

[139] Yael Pritch, Eitam Kav-Venaki, and Shmuel Peleg. Shift-map image editing. In *2009 IEEE 12th International Conference on Computer Vision*, pages 151–158, September 2009.

[140] Albert Pumarola, Antonio Agudo, Aleix M. Martinez, Alberto Sanfeliu, and Francesc Moreno-Noguer. GANimation: Anatomically-aware Facial Animation from a Single Image. pages 818–833, 2018.

[141] Patrick Pérez, Michel Gangnet, and Andrew Blake. Poisson image editing. In *ACM SIGGRAPH 2003 Papers*, pages 313–318. 2003.

[142] Shengju Qian, Kwan-Yee Lin, Wayne Wu, Yangxiaokang Liu, Quan Wang, Fumin Shen, Chen Qian, and Ran He. Make a Face: Towards Arbitrary High Fidelity Face Manipulation. pages 10033–10042, 2019.

[143] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[144] M. Ranzato, Y. Boureau, and Y LeCun. Sparse feature learning for deep belief networks. In *Conference on Neural Information Processing Systems*, 2007.

[145] Benjamin Recht, Maryam Fazel, and Pablo A. Parrilo. Guaranteed Minimum-Rank Solutions of Linear Matrix Equations via Nuclear Norm Minimization. *SIAM Review*, 52(3):471–501, January 2010. Publisher: Society for Industrial and Applied Mathematics.

[146] Scott Reed, Kihyuk Sohn, Yuting Zhang, and Honglak Lee. Learning to disentangle factors of variation with manifold interaction. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pages II–1431–II–1439, Beijing, China, June 2014.

[147] 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*, volume 28. Curran Associates, Inc., 2015.

[148] Elad Richardson, Yuval Alaluf, Or Patashnik, Yotam Nitzan, Yaniv Azar, Stav Shapiro, and Daniel Cohen-Or. Encoding in style: a stylegan encoder for image-to-image translation. *arXiv preprint arXiv:2008.00951*, 2020.

[149] Christof Ridder, Olaf Munkelt, and Harald Kirchner. Adaptive Background Estimation and Foreground Detection using Kalman-Filtering. In *Proceedings of International Conference on recent Advances in Mechatronics*, pages 193–199, 1995.

[150] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th international conference on machine learning*, 2011.

[151] Rasmus Rothe, Radu Timofte, and Luc Van Gool. DEX: Deep EXpectation of Apparent Age From a Single Image. pages 10–15, 2015.

[152] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252, December 2015.

[153] 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*, pages 2234–2242, 2016.

[154] Peter Schallauer and Roland Mörzinger. Film grain synthesis and its application to re-graining. volume 6059, pages 60590Z–60590Z–7, 2006.

[155] Rolf Schneider and Wolfgang Weil. *Stochastic and Integral Geometry*. Springer, 2008.

[156] Yujun Shen, Jinjin Gu, Xiaoou Tang, and Bolei Zhou. Interpreting the Latent Space of GANs for Semantic Face Editing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9243–9252, 2020.

[157] Jianbo Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, August 2000.

[158] Andrews Sobral. BGSLibrary: An opencv c++ background subtraction library. *IX Workshop de Visao Computacional.*, 27, 2013.

[159] Jingkuan Song, Jingqiu Zhang, Lianli Gao, Xianglong Liu, and Heng Tao Shen. Dual Conditional GANs for Face Aging and Rejuvenation. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, pages 899–905, Stockholm, Sweden, July 2018.

[160] Perry Sprawls. The Photographic Process and Film Sensitivity.

[161] Pablo Sprechmann, Alexander M. Bronstein, and Guillermo Sapiro. Learning Efficient Sparse and Low Rank Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(9):1821–1833, September 2015.

[162] Chris Stauffer and W. Eric L. Grimson. Adaptive background mixture models for real-time tracking. In *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149)*, volume 2, pages 246–252 Vol. 2, June 1999. ISSN: 1063-6919.

[163] Ian Stephenson and A. Saunders. Simulating Film Grain using the Noise-Power Spectrum. In *Eurographics UK Theory and Practice of Computer Graphics*, 2007.

[164] Stoyan. *Stochastic geometry and its applications*, volume 2. Wiley New York, 1987.

[165] Kazuo Tanaka and Suguru Uchida. Extended random-dot model. *Journal of the Optical Society of America*, 73(10):1312+, October 1983.

[166] Ayush Tewari, Mohamed Elgharib, Gaurav Bharaj, Florian Bernard, Hans-Peter Seidel, Patrick Perez, Michael Zollhofer, and Christian Theobalt. StyleRig: Rigging StyleGAN for 3D Control Over Portrait Images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6142–6151, 2020.

[167] The CGAL Project. *CGAL User and Reference Manual*. 4.9 edition, 2016.

[168] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Improved Texture Networks: Maximizing Quality and Diversity in Feed-Forward Stylization and Texture Synthesis. pages 6924–6932, 2017.

[169] P. Upchurch, Jacob V. Gardner, Geoff Pleiss, K. Bala, Robert Pless, Noah Snavely, and Kilian Q. Weinberger. Deep Feature Interpolation for Image Content Changes. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[170] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[171] Yuri Viazovetskyi, Vladimir Ivashkin, and Evgeny Kashin. StyleGAN2 Distillation for Feed-Forward Image Manipulation. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV 2020*, Lecture Notes in Computer Science, pages 170–186, 2020.

[172] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-Resolution Image Synthesis and Semantic Manipulation With Conditional GANs. pages 8798–8807, 2018.

[173] Zongwei Wang, Xu Tang, Weixin Luo, and Shenghua Gao. Face Aging With Identity-Preserved Conditional Generative Adversarial Networks. pages 7939–7947, 2018.

[174] G. Wernicke. Silver-Halide Recording Materials for Holography and Their Processing. *Zeitschrift für Physikalische Chemie*, 187(2):322–323, January 1994.

[175] Steven Worley. A cellular texture basis function. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '96, pages 291–294, New York, NY, USA, August 1996.

[176] Cristopher Richard Wren, Ali Azarbayejani, Trevor Darrell, and Alex Paul Pentland. Pfinder: real-time tracking of the human body. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):780–785, July 1997.

[177] Rongliang Wu and Shijian Lu. LEED: Label-Free Expression Editing via Disentanglement. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *ECCV*, Lecture Notes in Computer Science, pages 781–798, 2020.

[178] Zongze Wu, Dani Lischinski, and Eli Shechtman. StyleSpace Analysis: Disentangled Controls for StyleGAN Image Generation. pages 12863–12872, 2021.

[179] Taihong Xiao, Jiapeng Hong, and Jinwen Ma. ELEGANT: Exchanging Latent Encodings with GAN for Transferring Multiple Face Attributes. pages 168–184, 2018.

[180] J. C. K. Yan. Statistical methods for film grain noise removal and generation. Master's thesis, University of Toronto, 1997.

[181] Xinchen Yan, Jimei Yang, Kihyuk Sohn, and Honglak Lee. Attribute2Image: Conditional Image Generation from Visual Attributes. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *ECCV*, Lecture Notes in Computer Science, pages 776–791, 2016.

[182] Hongyu Yang, Di Huang, Yunhong Wang, and Anil K. Jain. Learning Face Age Progression: A Pyramid Architecture of GANs. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 31–39, June 2018. ISSN: 2575-7075.

[183] Jian Yao and Jean-Marc Odobez. Multi-Layer Background Subtraction Based on Color and Texture. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, June 2007. ISSN: 1063-6919.

[184] Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S. Huang. Generative Image Inpainting With Contextual Attention. pages 5505–5514, 2018.

[185] Lihi Zelnik-manor and Pietro Perona. Self-Tuning Spectral Clustering. In *Advances in Neural Information Processing Systems*, volume 17. MIT Press, 2005.

[186] Jiangfeng Zeng, Xiao Ma, and Ke Zhou. Photo-realistic face age progression/regression using a single generative adversarial network. *Neurocomputing*, 366:295–304, November 2019.

[187] Zhifei Zhang, Yang Song, and Hairong Qi. Age Progression/Regression by Conditional Adversarial Autoencoder. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 4352–4360. IEEE Computer Society, 2017.

[188] Zihan Zhou, Xiaodong Li, John Wright, Emmanuel Candès, and Yi Ma. Stable Principal Component Pursuit. In *2010 IEEE International Symposium on Information Theory*, pages 1518–1522, June 2010.

[189] Jiapeng Zhu, Yujun Shen, Deli Zhao, and Bolei Zhou. In-Domain GAN Inversion for Real Image Editing. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *ECCV*, Lecture Notes in Computer Science, pages 592–608, 2020.

[190] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2223–2232, 2017.

Appendix A

Low Rank models for Background Estimation

A.1 Comparing spatial and temporal merging

In this Section, we justify the weighting value of merging spatio-temporal regions in our video segmentation algorithm in Section 2.6.2. This weight is required because merging either spatial or temporal regions is not the same from the point of view of the RPCA error. In the spatial case, we are adding new variables (pixels) to the optimisation problem, whereas in the temporal case, we are adding new observations (in time) of the same pixels. In a simple toy case of a static background with noise, the estimated background $\mathbf{L}$ will converge to the true background as time increases, whereas increasing the number of pixels will not induce this convergence.

Let us first recall the merging cost function of two regions Ω_i and Ω_j:

$$d(\Omega_i, \Omega_j) = \frac{|e_i + e_j - e_{i\cup j}|}{\phi_{i\cup j}}, \tag{A.1}$$

where $e_i = \|\mathbf{X}^i - \mathbf{L}^i - \mathbf{S}^i\|_F^2$ in the region Ω_i, and likewise for Ω_j and $\Omega_{i\cup j}$. $\mathbf{X}^i$, $\mathbf{L}^i$ and $\mathbf{S}^i$ are, respectively, the input video, the estimated low-rank background and the sparse foreground, in the region Ω_i (these are all matrices). When we decide whether to merge two spatio-temporal regions in our low-rank video segmentation algorithm, we need to make sure that the cost of merging the regions are calculated in a "fair" manner. To explain this in more detail, let us consider the following simple case.

Let us suppose that the video consists of a background which does not change, plus some Gaussian noise:

$$\mathbf{X}_{k,\ell} = \mu_k + \varepsilon_{k,\ell}, \tag{A.2}$$

where μ_k is a scalar (the background) and $\varepsilon_{k,\ell} \sim \mathcal{N}(0, \sigma^2)$, with some variance σ^2. Furthermore, let us choose $r = 1$, in other words we are considering rank-1 decompositions of each region. This is reasonable for such a simple case, and it is in fact the value we use in the algorithm in practice.

We will show that in this simple situation, the expected value of the cost function is very different in the case of merging two regions which are spatially adjacent or temporally adjacent. This means that our segmentation algorithm will be highly biased towards merging in the *spatial* direction, which is problematic. Therefore, we shall scale the cost function in order to give the two an equal footing.

To find the expected value of the cost of merging two adjacent regions in the simple case, we need to know what the rank-1 decomposition will be. We consider here that the sparse component will be 0, so that the error matrix $\mathbf{E}$ (defined in Equation (2.5)) is given by $\mathbf{E} = \mathbf{X} - \mathbf{L}$.

In such a case, the decomposition should produce a rank-1 component $\mathbf{L}$ which contains the mean value along the columns, repeated:

$$\mathbf{L} = \begin{pmatrix} \frac{1}{t}\sum_{q=1}^{t}\mathbf{X}_{1,q} & \cdots & \frac{1}{t}\sum_{q=1}^{t}\mathbf{X}_{1,q} & \cdots & \frac{1}{t}\sum_{q=1}^{t}\mathbf{X}_{1,q} \\ \cdots & & \cdots & & \\ \frac{1}{t}\sum_{q=1}^{t}\mathbf{X}_{k,q} & \cdots & \frac{1}{t}\sum_{q=1}^{t}\mathbf{X}_{k,q} & \cdots & \frac{1}{t}\sum_{q=1}^{t}\mathbf{X}_{k,q} \\ \cdots & & \cdots & & \\ \frac{1}{t}\sum_{q=1}^{t}\mathbf{X}_{mn,q} & \cdots & \frac{1}{t}\sum_{q=1}^{t}\mathbf{X}_{mn,q} & \cdots & \frac{1}{t}\sum_{q=1}^{t}\mathbf{X}_{mn,q} \end{pmatrix}. \tag{A.3}$$

Indeed, since there is no sparse compenent, the RCPA problem reduces to the PCA, which finds the average as its first component.

Thus, for a given component of the error matrix $\mathbf{E}_{k,\ell}$ we have:

$$\begin{aligned} \mathbf{E}_{k,\ell} &= \mathbf{X}_{k,\ell} - \frac{1}{t}\sum_{q=1}^{t}\mathbf{X}_{k,q} && \text{(A.4)} \\ &= \frac{t\mathbf{X}_{k,\ell} - \sum_{q=1}^{t}\mathbf{X}_{k,q}}{t} \\ &= \frac{t\left(\mu_k + \varepsilon_{k,\ell}\right) - \sum_{q=1}^{t}\left(\mu_k + \varepsilon_{k,q}\right)}{t} && \text{(A.5)} \\ &= \varepsilon_{k,\ell} - \frac{1}{t}\sum_{q=1}^{t}\varepsilon_{k,q}. && \text{(A.6)} \end{aligned}$$

We wish to know the Frobenius norm of $\mathbf{E}$. However, it is clear that each component of one row of $\mathbf{E}$ contains repeated values, so we cannot consider them to be independant random samples.

For one row of $\mathbf{E}$, we have:

$$\sum_{\ell=1}^{t}(\mathbf{E}_{k,\ell})^2 = \sum_{\ell=1}^{t}\left(\varepsilon_{k,\ell} - \frac{1}{t}\sum_{q=1}^{t}\varepsilon_{k,q}\right)^2 \tag{A.7}$$

$$= \sum_{\ell=1}^{t}\varepsilon_{k,\ell}^2 - \frac{2}{t}\sum_{\ell=1}^{t}\varepsilon_{k,\ell}\sum_{q=1}^{t}\varepsilon_{k,q} + \sum_{\ell=1}^{t}\left(\frac{1}{t^2}\sum_{q=1}^{t}\varepsilon_{k,q}\sum_{q=1}^{t}\varepsilon_{k,q}\right) \tag{A.8}$$

$$= \sum_{\ell=1}^{t}\varepsilon_{k,\ell}^2 - \frac{2}{t}\sum_{\ell=1}^{t}\varepsilon_{k,\ell}\sum_{q=1}^{t}\varepsilon_{k,q} + \frac{1}{t}\sum_{q=1}^{t}\varepsilon_{k,q}\sum_{q=1}^{t}\varepsilon_{k,q} \tag{A.9}$$

$$= \sum_{\ell=1}^{t}\varepsilon_{k,\ell}^2 - \frac{1}{t}\sum_{\ell=1}^{t}\varepsilon_{k,\ell}\sum_{q=1}^{t}\varepsilon_{k,q} \tag{A.10}$$

$$= \sum_{\ell=1}^{t}\varepsilon_{k,\ell}^2 - \frac{1}{t}\sum_{\ell=1}^{t}\varepsilon_{k,\ell}^2 - \frac{2}{t}\sum_{\substack{\ell,q\\ q>\ell}}^{t}\varepsilon_{k,\ell}\varepsilon_{k,q} \tag{A.11}$$

$$= \frac{t-1}{t}\sum_{\ell=1}^{t}\varepsilon_{k,\ell}^2 - \frac{2}{t}\sum_{\substack{\ell,q\\ q>\ell}}^{t}\varepsilon_{k,\ell}\varepsilon_{k,q}. \tag{A.12}$$

If we sum this over all the rows (recall that these rows correspond to the number of pixels in a single frame), we find:

$$e = \|\mathbf{E}\|_F^2 = \frac{t-1}{t}\sum_{k=1}^{q_w q_h}\sum_{\ell=1}^{t}\varepsilon_{k,\ell}^2 - \frac{2}{t}\sum_{k=1}^{q_w q_h}\sum_{\substack{\ell,q\\ q>\ell}}^{t}\varepsilon_{k,\ell}\varepsilon_{k,q}, \tag{A.13}$$

where q_w and q_h are the width and height of an image in the video.

We wish to find the expected value of Equation (A.1), which will itself contain repeated values (since we are analysing the concatenation of two adjacent regions). The exact expression for Equation (A.1) will depend on whether we are considering temporal concatenation or spatial concatenation. Let us look at the temporal case first:

A.1.1 Temporal merging

In the temporal merging case, we have:

$$\begin{aligned}\|\mathbf{E}^i\|_F^2 + \|\mathbf{E}^j\|_F^2 - \|\mathbf{E}^{i\cup j}\|_F^2 &= \frac{t-1}{t}\sum_{k=1}^{q_w q_h}\sum_{\ell=1}^{2t}\varepsilon_{k,\ell}^2 - \frac{2}{t}\sum_{k=1}^{q_w q_h}\sum_{\substack{\ell,q\\ q>\ell}}^{2t}\varepsilon_{k,\ell}\varepsilon_{k,\ell}\\ &\quad - \left(\frac{2t-1}{2t}\sum_{k=1}^{q_w q_h}\sum_{\ell=1}^{2t}\varepsilon_{k,\ell}^2 - \frac{2}{2t}\sum_{k=1}^{q_w q_h}\sum_{\substack{\ell,q\\ q>\ell}}^{2t}\varepsilon_{k,\ell}\varepsilon_{k,q}\right)\\ &= -\frac{1}{2t}\sum_{k=1}^{q_w q_h}\sum_{\ell=1}^{2t}\varepsilon_{k,\ell}^2 - \frac{1}{t}\sum_{k=1}^{q_w q_h}\sum_{\substack{\ell,q\\ q>\ell}}^{2t}\varepsilon_{k,\ell}\varepsilon_{k,q}.\end{aligned} \tag{A.14}$$

Finally, we can calculate the expected value of Equation (A.14). The expected value of the second term on the right hand side of Equation (A.14) is equal to 0. Therefore, in the temporal case, we have:

$$\mathbb{E}(\|\mathbf{E}^i\|_F^2 + \|\mathbf{E}^j\|_F^2 - \|\mathbf{E}^{i\cup j}\|_F^2) = -\frac{1}{2t}(q_w q_h)2t\sigma^2 = -q_w q_h \sigma^2. \tag{A.15}$$

We note that in reality, the situation is slightly more complicated since we should be taking the expected value of the *absolute value* of $\|\mathbf{E}^i\|_F^2 + \|\mathbf{E}^j\|_F^2 - \|\mathbf{E}^{i\cup j}\|_F^2$. However, this is a more involved calculation, so we approximate the expected value of the cost function Equation (A.1) with $q_w q_h \sigma^2$. Experiments show that this is reasonably accurate, and normalises the temporal merging cost to a value close to 1 in the case of static background with noise.

An interesting secondary result of this analysis is that the merging of two temporally adjacent regions is always worse than considering the two regions separately, in the case of static background with noise, using a rank-1 background approximation and our cost function Equation (A.1).

A.1.2 Spatial merging

In the case of merging two spatially adjacent regions we have:

$$\begin{aligned}\|\mathbf{E}^i\|_F^2 + \|\mathbf{E}^j\|_F^2 - \|\mathbf{E}^{i\cup j}\|_F^2 = \frac{t-1}{t}\sum_{k=1}^{2q_w q_h}\sum_{\ell=1}^{t}\varepsilon_{k,\ell}^2 - \frac{2}{t}\sum_{k=1}^{2q_w q_h}\sum_{\substack{\ell,q\\ \ell\neq q}}^{t}\varepsilon_{k,\ell}\varepsilon_{k,q} \\ - \left(\frac{t-1}{t}\sum_{k=1}^{2q_w q_h}\sum_{\ell=1}^{t}\varepsilon_{k,\ell}^2 - \frac{2}{t}\sum_{k=1}^{2q_w q_h}\sum_{\substack{\ell,q\\ \ell\neq q}}^{t}\varepsilon_{k,\ell}\varepsilon_{k,q}\right) \\ = 0.\end{aligned}$$

Therefore in the spatial case, the merging cost should be equal to zero. This is coherent, since in our simple setting, the rows are treated independently by the decomposition algorithm, meaning that it will not change anything to consider regions separately or merged. In practice, this may not be the case due to approximation errors. However, for lack of a scaling parameter, we do not scale the cost function in the spatial merging case.

Appendix B

Film grain

B.1 Illustration of the dithering effect on film grain

In Figure B.1, we show the subjective effect of increased resolution that adding film grain gives, an effect which is known as dithering. This shows that film grain rendering can both increase the artistic value of an image and improve the perceived resolution.

B.2 Comparison of variable grain shapes

In Figure B.2, we compare the results of our film grain rendering with disks or triangles, as it is possible to chose a variety of shapes in our algorithm. We parameterize the triangles by their circumscribed circle, and have chosen the radius of this circumscribed circle so that the area of the disks and the triangles are equal. We have voluntarily used large grain parameters in order to test the algorithm. At these resolutions, the graininess impression is quite similar for both disks and triangles; nevetheless it is a significant advantage to have such flexibility in our algorithm.

Input

Film grain rendering, zoom : $\times 1$

Film grain rendering, zoom : $\times 8$

Figure B.1: **Example of the "dithering" effect of film grain**. We show the subjective impression of increased "resolution" with a low-resolution example. This example shows that the only resolution limitation of our algorithm comes from the pixel grid of the input.

Rendering with disks ($r = 0.1$) Rendering with triangles

Rendering with disks ($r = 0.2$) Rendering with triangles

Figure B.2: **Rendering results with varying shapes of film grain**. We show film grain with disks of constant radius and triangles. The triangles are parameterized by their circumscribed circle and the radius of the latter is chosen to produce an equal area to the disks, in the interest of fair comparisons. At this resolution, the visual difference is not large, however it is very useful to have this flexibility in our algorithm.

Appendix C

Deep learning

C.1 Creating a disk dataset

The goal of this section of the appendix is to create a dataset which contains images of centred disks. This is done with the same approach as in Section 3.3.4, by using a Monte Carlo simulation.

Since the autoencoder must project each image to a continuous scalar, it makes sense to generate the disks with a continous parameter r, and that the disks also be "continuous" in some sense (each different value of r should produce a different disk. For this, as we mentioned in Section 4.2.2, we create the training images x_r as

$$x_r = g_\sigma * \mathbb{1}_{\mathbb{B}_r}, \tag{C.1}$$

where $\mathbb{1}_{\mathbb{B}_r}$ is the indicator function of the ball of radius r, and g_σ is a Gaussian kernel with variance σ. In practical terms, we carry this out using a Monte Carlo simulation to approximate the result of the convolution of an indicator function with a disk. Indeed, let $\xi_{i,i=1\dots N}$ be a sequence of independently and identically distributed (iid) random variables, with $\xi_i \sim \mathcal{N}(0, \sigma)$. Each pixel at position t is evaluated as

$$x_r(t) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}_{B_r}(\xi_i). \tag{C.2}$$

According to the law of large numbers, this tends to the exact value of $g_\sigma * \mathbb{1}_{\mathbb{B}_r}$, and gives a method of producing a continuous dataset.

While other approaches are available (evaluating the convolution in the Fourier domain, for example), this is simple to implement and generalises to any shape which we can parametrise. We also note that the large majority of deep learning synthesis papers suppose that the data lie on some manifold, but this hypothesis is never checked. In our case, we explicitly sample the data in a smooth space.

C.2 Contractive encoders learn the area of disks

We study the encoder part of an autoencoder that takes an image and outputs a one-dimensional feature. We show that, with a simple constraint that the output of the encoder is not constant and in the absence of any other loss than the contractive loss, the feature is merely the area of the disk presented to the encoder.

We refer to the input image as x_r, where r is the radius of the disk present in the image (one disk for each image, and each disk centred). In this simple setting we will seek to find a function

$z : L^2(\Omega) \to \mathbb{R}$ that stands for the encoder E, where Ω is the support of the images. The loss associated with a contractive auto-encoder [150] is

$$\mathcal{L}(z) = \sum_{r=0}^{R_{max}} \|\nabla z(x_r)\|^2, \tag{C.3}$$

where $\nabla z \in L^2(\Omega)$ stands for the gradient of the latent z with respect to the input image, when the input image is x_r (it is an image). R_{max} is the maximum radius observed in the dataset, which we normalise to 1. Although the parameter of a loss is typically the set of parameters θ of a network and is usually written $\mathcal{L}(\theta)$, here we minimise among all possible encoders z simulating an infinite capacity of the the encoder hence the notation $\mathcal{L}(z)$.

We can take a continuous proxy for this loss and write

$$L(z) = \int_0^1 \|\nabla z(x_r)\|_2^2 dr, \tag{C.4}$$

Note the integration against the simple measure dr reflects the fact that the distribution of the radii is uniform. In anticipation of the derivations ahead we suppose that the encoder function is smooth and that the edges of the shapes are also smooth. We will investigate what happens when the shapes become infinitely sharp after. We can express this by

$$x_r(p_x, p_y) = \varphi\left(\frac{\sqrt{p_x^2 + p_y^2} - r}{\sigma}\right) = \varphi_\sigma\left(\sqrt{p_x^2 + p_y^2} - r\right), \tag{C.5}$$

where $p = (p_x, p_y)$ is a position, φ is some smooth real function that is equal to 1 before -1, 0 after 1 (think of a simplified tanh function) and σ is a scaling factor. When σ goes to zero we will be in the case of sharp edges. Other smooth representations of a disk are possible, for example $x_r(p_x, p_y) = (\mathbb{1}_{B_r} * g_\sigma)(p_x, p_y)$, where $\mathbb{1}_{B_r}$ is the indicator function of the ball of radius r, as used in our experiments, and when σ goes to zero we are back to sharp edges again. We will stick to the representation in Equation (C.5) since it simplifies our calculations further on, in particular in Section C.2.1.

To avoid trivial cases, we also require our encoder not to be constant.[1] Once scaled, this constraint can be written

$$1 = z(x_1) - z(x_0) = \int_0^1 \frac{\partial z}{\partial r} dr = \int_0^1 < \nabla z | \frac{\partial x_r}{\partial r} > dr, \tag{C.6}$$

the last equality being the chain rule. Let us denote

$$h_r(p) := \frac{\partial x_r}{\partial r}(p). \tag{C.7}$$

Now our problem boils down to

$$\begin{array}{ll} \text{Minimise :} & \int_0^1 \|\nabla z(x_r)\|^2 dr \\ \text{Under the constraint :} & \int_0^1 \langle \nabla z(x_r) | h_r \rangle \, dr = 1 \end{array} \tag{C.8}$$

[1] This obviously cannot happen in the case of a full autoencoder, but we must impose it when studying the encoder only.

This minimisation is carried out among all possible z functions that are smooth enough to have a gradient with respect to its input x.

For a fixed r, among all $\nabla z(x_r)$ satisfying

$$\langle \nabla z(x_r) | h_r \rangle = C(r)$$

for some constant $C(r)$, the one with minimal $\|\nabla z(x_r)\|$ is of the form $c(r)h_r$. To see this, write $\nabla z(x_r) = \beta h_r + h_r^{\perp}$, a decomposition of $\nabla z(x_r)$ on $\mathrm{Vect}(h_r)$ and its orthogonal space (in $L^2(\Omega)$). Hence, we can decrease the quantity to minimise in Equation (C.8) without changing the constraint by projecting $\nabla z(x_r)$ on $\mathrm{Vect}(h_r)$. Thus, we can make the assumption that our solution z is such that

$$\nabla z(x_r) = c(r)h_r, \tag{C.9}$$

and we are reduced to finding a single function c that satisfies:

$$\begin{array}{ll} \text{Minimise :} & \int_0^1 c(r)^2 H_2(r)dr \\ \text{Under the constraint :} & \int_0^1 c(r)H_2(r)dr = 1, \end{array} \tag{C.10}$$

where

$$H_2(r) = \iint h_r(p_x, p_y)^2 dp_x dp_y \tag{C.11}$$

Let us consider a small perturbation of the solution, $c(r) + \epsilon\delta$, for some smooth function δ which satisfies $\int_0^1 \delta(r)H_2(r)dr = 0$ (to ensure that $c(r)+\epsilon\delta$ indeed verifies the constraint). Then, we have

$$\frac{d}{d\epsilon}\left(\int_0^1 \left(c(r) + \epsilon\delta(r)\right)^2 H_2(r)dr\right) = \int_0^1 \left(2c(r)\delta(r) + 2\epsilon\delta(r)^2\right) H_2(r)dr. \tag{C.12}$$

If we take the limit when $\epsilon \to 0$, we have the condition

$$\int_0^1 c(r)\delta(r)H_2(r) = 0 \tag{C.13}$$

The solution of the system (C.10) is $c(r) = C$ for C some constant, since the only function $c(r)$ that satisfies Equation (C.13) for any valid increment δ is a constant one. Indeed, we have the two conditions $\delta \in \mathrm{Vect}(H_2)^{\perp}$ (the constraint in Equation (C.10)) and $\langle c(r)H_2, \delta\rangle = 0$. This means that $c(r)H_2 \in \left(\mathrm{Vect}(H_2)^{\perp}\right)^{\perp} = \mathrm{Vect}(H_2)$.

Finally, when σ, the edge width goes to zero the function h_r tends to be concentrated on a circle of radius r (see next section C.2.1) and a value that is almost constant over the range of r. Roughly speaking this gives

$$H_2(r) = 2\pi r\alpha. \tag{C.14}$$

For the sake of completeness, we have verified experimentally that the function h_r is indeed concentrated on a circle of radius r. These results can be seen in Figure C.1.

Finally, by integrating, we have

$$z(r) = \int_0^r \frac{dz}{d\rho}d\rho = \int_0^r c(\rho)H_2(\rho)d\rho = \gamma r^2, \tag{C.15}$$

where γ is some constant.

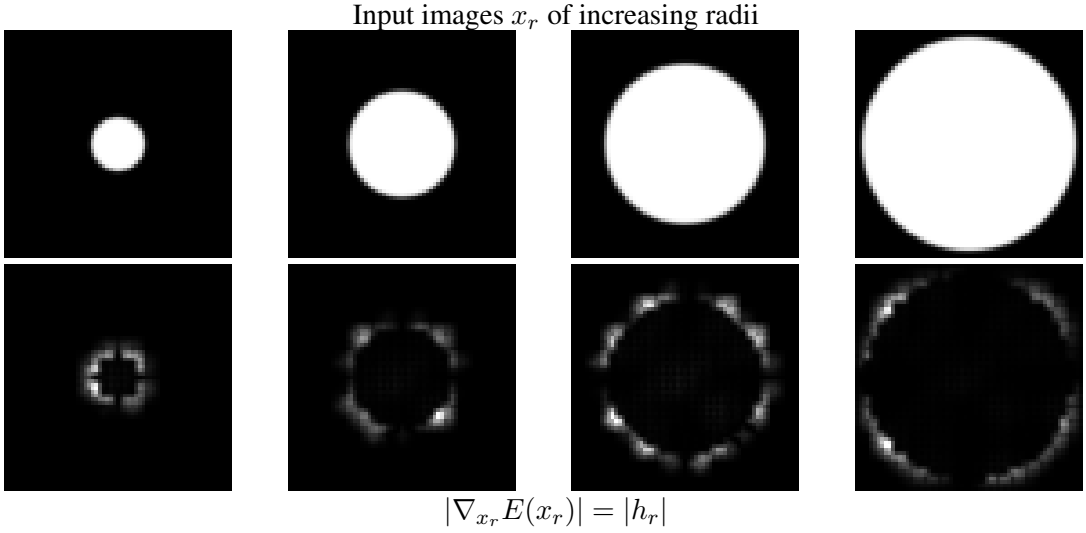


Figure C.1: **Absolute value of the gradient of the code z with respect to x_r.** We verify that in the case of a contractive encoder, the gradient of the code z of the disk image x_r, with respect to the image itself, is indeed concentrated on a circle of radius r. This behaviour is important to show that the contractive encoder indeed extracts the area of the disk.

C.2.1 Infinitely thin edges

Here we show our claim when the edge width goes to 0, $z(r)$ is indeed proportional to the disk area (Equation (C.14)). We do this with the model described in Equation (C.5).

$$x_r(p_x, p_y) = \varphi\left(\frac{\sqrt{p_x^2 + p_y^2} - r}{\sigma}\right) = \varphi_\sigma\left(\sqrt{p_x^2 + p_y^2} - r\right), \tag{C.16}$$

where φ is some smooth function that is equal to 1 before -1, 0 after 1.

In this case we have

$$\frac{\partial x_r}{\partial r}(p_x, p_y) = -\varphi'_\sigma(\sqrt{p_x^2 + p_y^2} - r). \tag{C.17}$$

The support of φ'_σ is $[-\sigma, \sigma]$. This function is radial and we are interested in computing (C.11), which gives

$$H_2(r) = \int_{r-\sigma}^{r+\sigma} 2\pi u \left(\varphi'_\sigma(u - r)\right)^2 du \tag{C.18}$$

with the variable u being $\sqrt{p_x^2 + p_y^2}$.

For $r \geq \sigma$ we have the following simple inequalities

$$2\pi(r - \sigma)\sigma^2 C \leq H_2(r) \leq 2\pi(r + \sigma)\sigma^2 C \tag{C.19}$$

where $C = \int \varphi'^2(t)dt$. This confirms the behavior of $H_2(r)$ as being merely proportional to r.

More precisely we obtain (by integration as in (C.15))

$$\gamma(r^2 - \sigma r) \leq z(r) \leq \gamma(r^2 - \sigma r), \tag{C.20}$$

which is the announced behavior for z. □

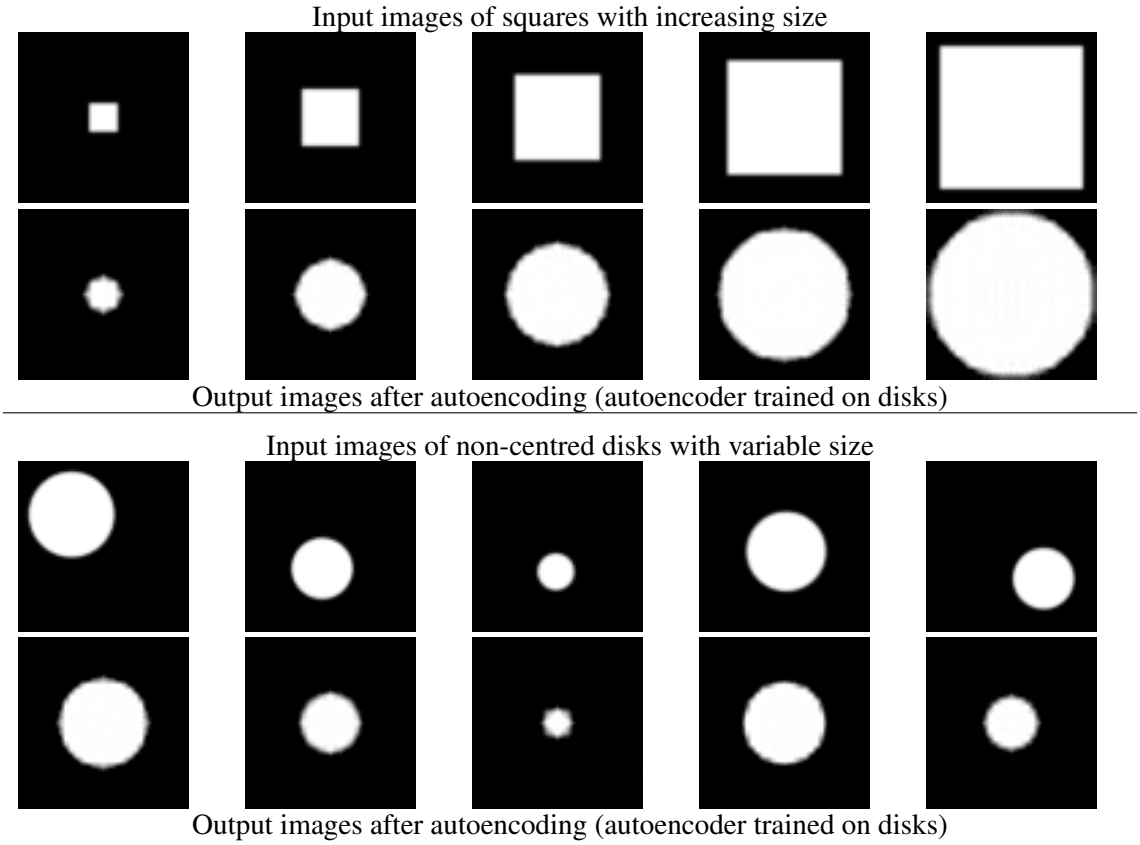


Figure C.2: **Output of an autoencoder trained on disks, applied to squares and non-centred disks during testing**. The autoencoder indeed extracts the area of the object, regardless of its shape or position. Since it was trained on disks, it outputs the disks with a similar area to the objects observed during testing. Note that the autoencoder does not extract position, since it was trained on centred disks.

C.2.2 Experimental results

To further test this behaviour experimentally, we have used our contractive autoencoder trained on disks, and applied it to a test set of images with squares and non-centred disks. In Figure C.2, it can be seen that the encoder indeed extracts the area of these objects, and then outputs the disk with the closest area (since it has been trained on a disk database). This further confirms that the encoder is indeed extracting the area.

C.3 Decoding of a disk (network with no biases)

During the training of the autoencoder for the case of disks (with no bias in the autoencoder), the objective of the decoder is to convert a scalar into the image of a disk with the ℓ_2 distance as a metric. Given the profiles of the output of the autoencoder, we have made the hypothesis that the decoder approximates a disk of radius r with a function $y(t;r) := D(E(\mathbb{1}_{B_r})) = h(r)f(t)$, where f is a continuous function. We show that this is true experimentally in Figure C.3 by determining f experimentally by taking the average of all output profiles, and then comparing our code z against its theoretically optimal value $\langle f, \mathbb{1}_{B_r} \rangle$. We see that they are the same up to a

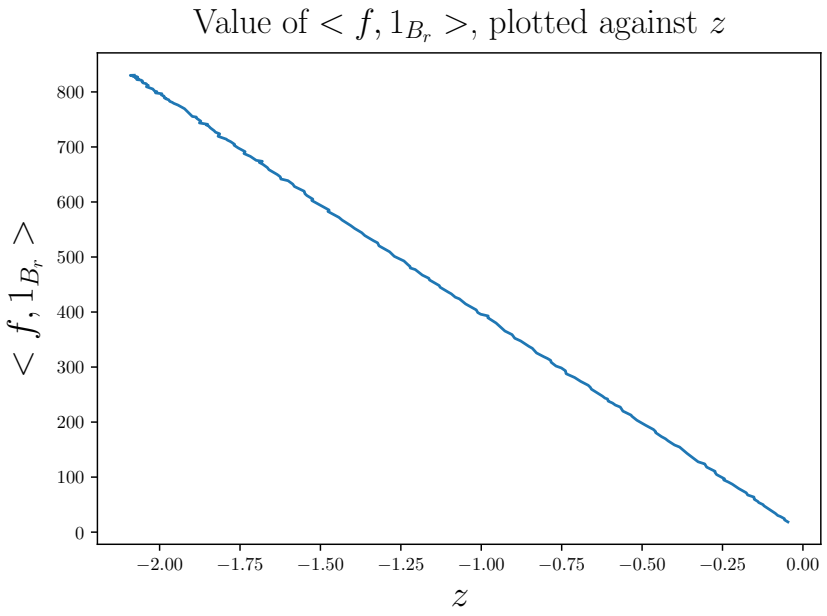


Figure C.3: **Verification of the theoretical derivations that use the hypothesis that $y(t, r) = h(r)f(t)$ for decoding, in the case where the autoencoder contains no bias.**. We have plotted z against the theoretically optimal value of h ($C \langle f, \mathbb{1}_{B_r} \rangle$, where C is some constant accounting for the arbitrary normalization of f). This experimental sanity check confirms our theoretical derivations.

multiplicative constant C.

We now compare the numerical optimisation of the energy in Equation (4.16) using a gradient descent approach with the profile obtained by the autoencoder without biases. The resulting comparison can be seen in Figure C.4. One can also derive a closed form solution of Equation (4.16) by means of the Euler-Lagrange equation and see that the optimal f for Equation (4.16) is the solution of the differential equation $y'' = -kty$ with initial state $(y, y') = (1, 0)$, where k is a free positive constant that accommodates for the position of the first zero of y. This gives a closed form of the f in terms of Airy functions.

C.4 Autoencoding disks with a database with a limited observed radius (network with no biases)

In Figure C.5, we see the grey-levels of the input/output of an autoencoder trained (without biases) on a restricted database, that is to say a database whose disks have a maximum radius R which is smaller than the image width. We have used $R = 18$ for these experiments. We see that the decoder learns a useful function f which only extends to this maximum radius. Beyond this radius, another function is used corresponding to the other sign of codes (see proposition 4) that is not tuned.

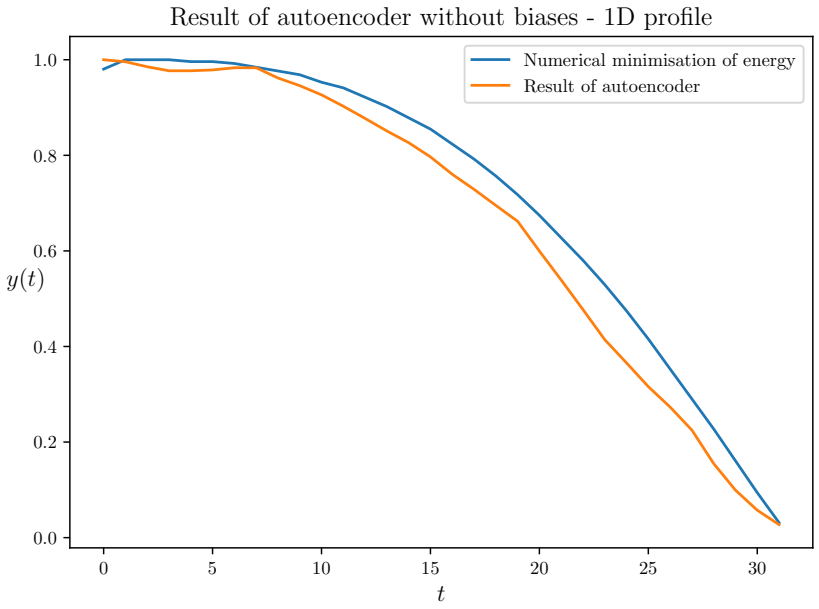


Figure C.4: **Comparison of the empirical function f of the autoencoder without biases with the numerical minimisation of Equation (4.16).** We have determined the empirical function f of the autoencoder and compared it with the minimisation of Equation (4.16). The resulting profiles are similar, showing that the autoencoder indeed succeeds in minimising this energy.

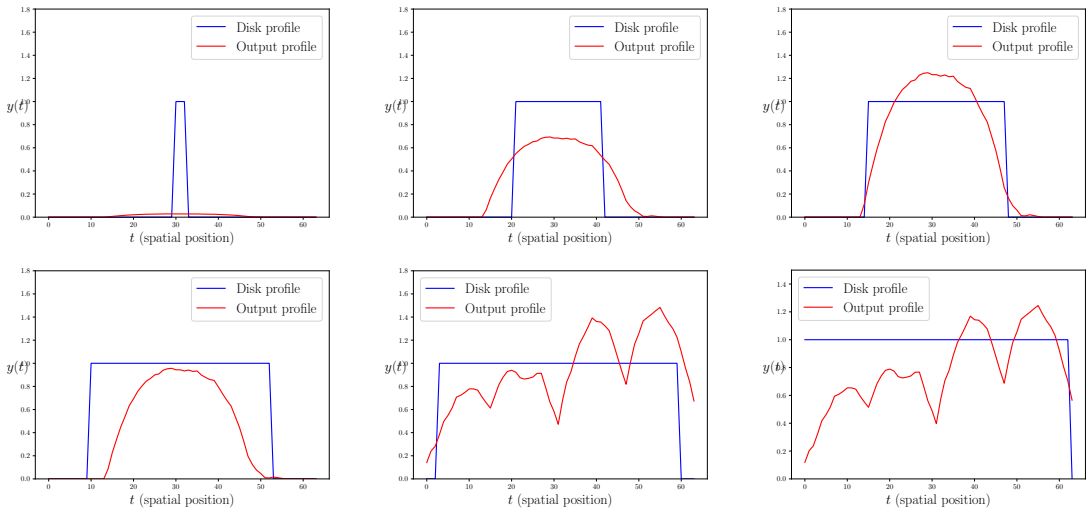


Figure C.5: **Profile of the encoding/decoding of centred disks, with a restricted database**. The decoder learns a profile f which only extends to the largest observed radius $R = 18$. Beyond this radius, another profile is learned that has is obviously not tuned to any data.

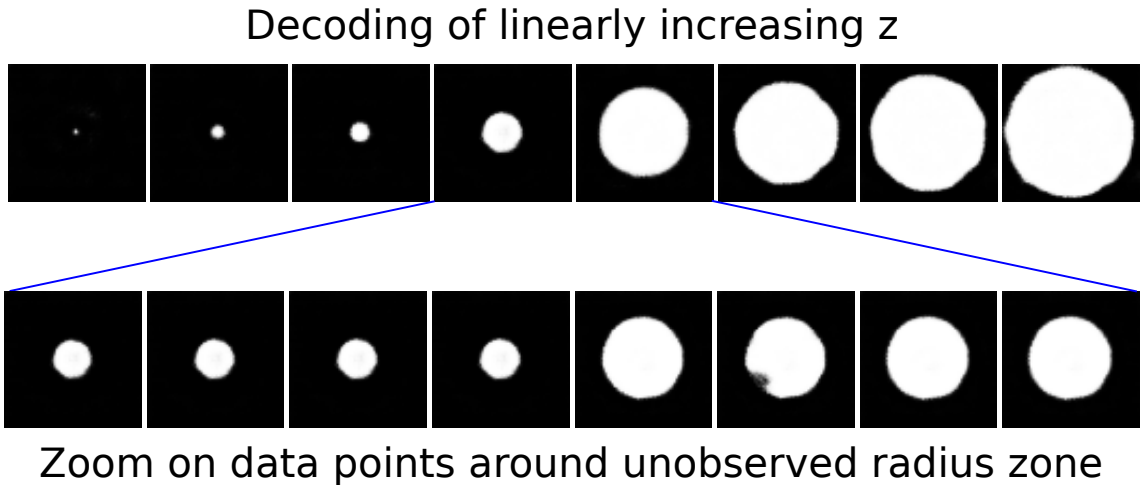


Figure C.6: **Output of the DCGAN of Radford et al.[143] ("IGAN") for disks when the database is missing disks of certain radii (11-18 pixels).** We can see that the DCGAN is not capable of reconstructing the disks which were not obeserved in the training dataset. This is a clear problem for generalisation. In the second line we have zoomed on the datapoints around the radius zone which is unobserved in the training dataset.

C.5 Autoencoding disks with a DCGAN [143]

In Figure C.6, we show the autoencoding results of the DCGAN network of Radford et al. We trained their network with a code size of $d = 1$. As can be seen, the DCGAN learns to force the training data to a predefined distribution, which cannot be modified during training (contrary to the autoencoder). Thus the network fails to correctly autoencode disks in the missing radius region which has not been observed in the training database.

C.6 Editing face image attributes in videos

As described in Section 4.5.4, we present additional facial attribute editing results on videos. These can be seen in Figure C.7. Each sub-figure corresponds to a frame extracted from the corresponding video, in which the indicated attributes are modified. For each video, we edit two attributes sequentially, and generate disentangled manipulation results. For example, in Figure **??**(c) when changing the person to woman, our method does not influence the attribute 'beard', despite the fact that it is correlated with gender. Besides, by varying the scaling factor progressively along the sequence, we achieve progressive attribute editing on videos. As shown by the video in Figure C.8, we can simulate a progressive smiling process by smoothly varying the scaling factor. Overall, our method generates stable and consistent manipulation results on videos, provided that motion is not too strong. When there are quick changes of pose, we observe lighting or geometric artifacts. These artifacts are in fact due to the projection in the latent space, and therefore necessarily extend to the manipulated videos. As can be seen from the video in Figure C.9, the manipulation during the first half of the video is realistic and consistent. But when the face turns to a side pose, the projected face is not well reconstructed and therefore neither is the manipulated face. This may be due to the limited reconstruction capacity of the pre-trained encoder and StyleGAN model when the pose is not frontal.

C.7 PCA Autoencoder

We now present further results concerning the PCA Autoencoder seen in Section 4.6. In Figure C.10, we show interpolation results in the native latent space of PGAN [96]. It is clear that several attributes are modified when we move along one dimension. For example, in the first dimension, we modify both gender and smile, and in the second one smile again and hair colour. This behaviour is problematic for editing in the latent space.

In Figure C.11, we display images of the results of our PCA-AE applied directly to Celeb-A face data. As we can see, the PCA-AE correctly separates different visual attributes in the latent space, because of its architecture and loss function. We recall that the PCA-AE's latent space is trained by increasing the latent space size progressively and then freezing each latent code. This is coupled with a latent code correlation loss which decorrelates attributes in the latent space. In spite of these useful properties, the resulting PCA-AE greatly blurs the images, meaning that, unfortunately, it is not useable for image editing. To tackle this problem, instead of autoencoding the images themselves, we apply PCA-AE to the pre-trained latent space of PGAN.

In Figure C.12, we see our PCA-AE approach applied to the latent space of PGAN [96]. Contrary to the native latent space of PGAN, our method separates and organises the latent space with respect to visual attributes in the images. For example, the first axis corresponds to hair colour, which is very present in the ℓ_2 norm of the reconstruction error. The second axis corresponds to pose. The PCA-AE is trained locally with respect to a central code $\overline{\eta}$.

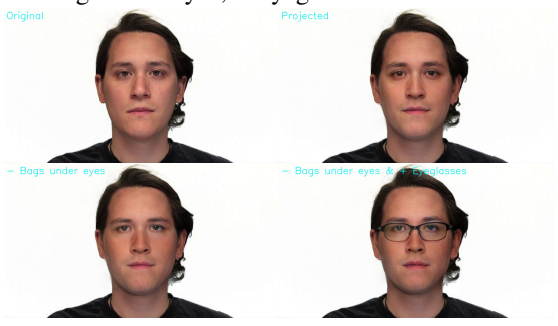
(a) 1_man_with_hat_Arched_Eyebrows_Beard.avi
- Arched Eyebrows, - Beard

(b) 2_woman_with_bricks_Eyeglasses_Age.avi
+ Eyeglasses, + Age

(c) 3_man_in_forest_Gender_Beard.avi
Gender, - Beard

(d) 4_man_with_muscle_Smiling_Young.avi
+ Smile, - Age

(e) 5_man_talking_Bags_Under_Eyes_Eyeglasses.avi
- Bags under eyes, + Eyeglasses



(f) 6_woman_turning_Smiling_Makeup.avi
- Smile, + Makeup

Figure C.7: **Facial attribute editing on videos.** Each sub-figure corresponds to a frame extracted from the specified video, corresponding to the manipulation result of the indicated attributes. In each sub-figure, the upper row shows the original frame and the projected frame reconstructed with the encoded latent code in StyleGAN, the bottom row shows the manipulated frames for the first attribute and then for two attributes. Please open the video files to visualize the manipulation details.

7_woman_with_bricks_progressive_Smiling.avi, + Smile progressively

Figure C.8: **Progressive attribute editing on videos.** By varying the scaling factor progressively along the sequence, the corresponding attribute is gradually varied. This figure show a frame extracted from the edited video, which corresponds to the progressive manipulation of the attribute "smile'. From left to right: the original frame, the projected frame, and the manipulated frame. Please open the video file to fully visualize the manipulation.

failure_case_woman_sitting_Makeup.avi, + Makeup

Figure C.9: Failure case of attribute manipulation on a video. This is a side pose frame extracted from the named video, which is the manipulation result of the attribute 'makeup'. From left to right: the original frame, the projected frame, and the manipulated frame. The face is not well reconstructed in the projected frame, and consequently the manipulated output contains defects. This is due to the limited generation capacity of the pre-trained encoder and the StyleGAN generator.

Figure C.10: **Interpolation in the original latent space of PGAN (with 512 components)**. From the initial code $\overline{\eta} = [\overline{\eta}_1, \overline{\eta}_2, \overline{\eta}_3, ..., \overline{\eta}_{512}]$ as shown in the middle column, we adjust the n^{th} component by adding a constant shown above the image, other codes are not shown that are fixed. We can see that it is difficult to interpret this latent space. Several attributes such as hair colour or head pose are varied within the same component of the latent space of PGAN.

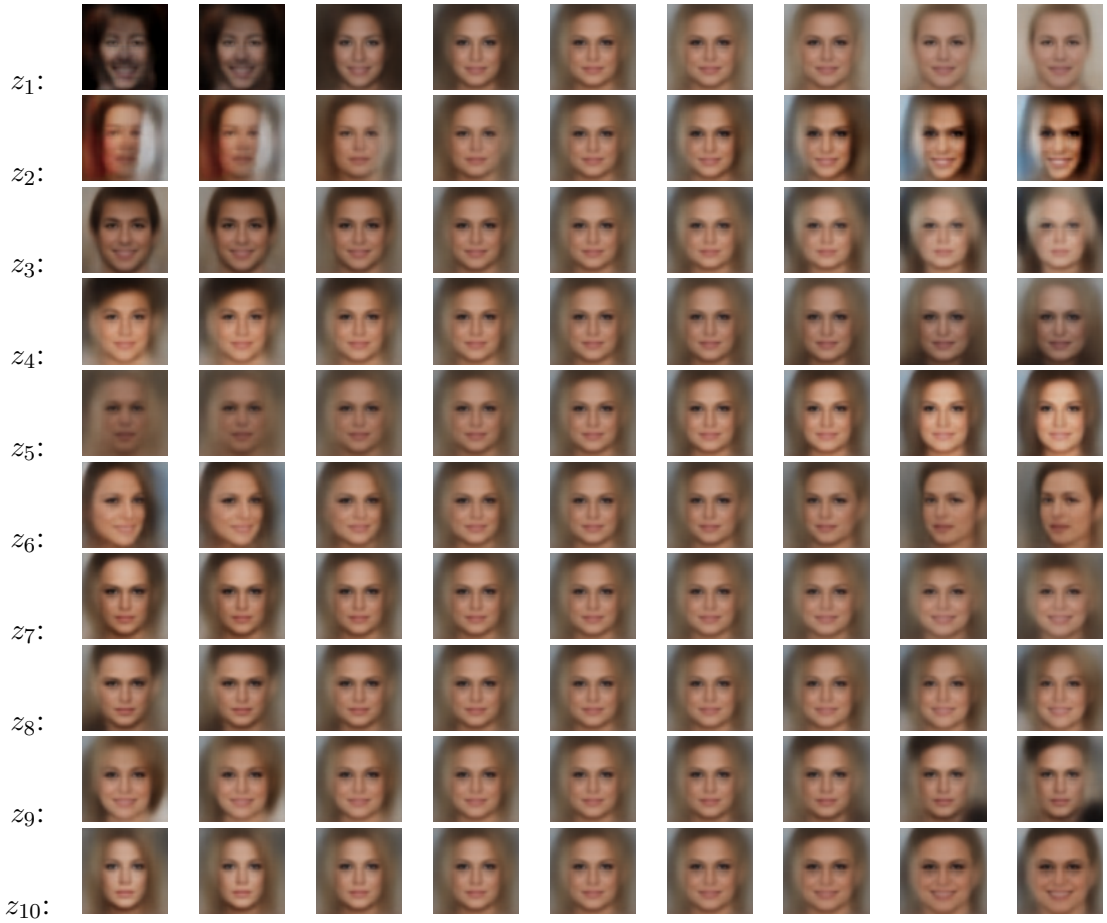


Figure C.11: **Interpolation in latent space of ten components of a PCA-AE, applied directly to the CelebA dataset with the size image of** 64×64. The code shown in the left side is used to adjusted, other codes are set to zeros. The middle column corresponding the images with the codes of all zeros. This leads to blurry results, which is why we chose to apply our PCA-AE a posteriori to a pre-trained GAN.

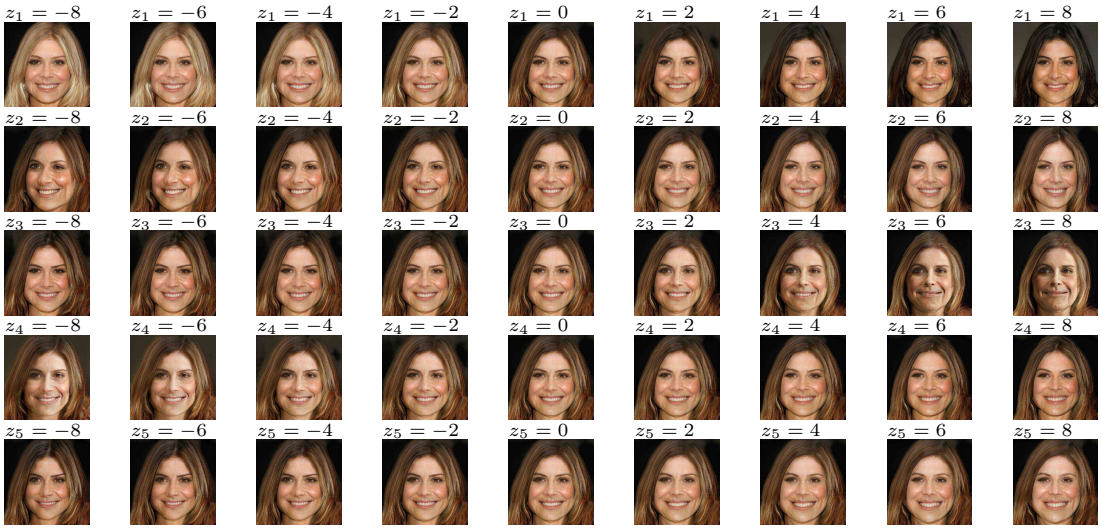


Figure C.12: **Results of navigation in the latent space of the PCA-AE for a pre-trained PGAN**. We trained this PCA-AE around the code $\overline{\eta}$ corresponding to the middle column. On each row, we have modified a single component (the other components are set to 0). We see that the component z_1 of the latent space z of the PCA-AE represents hair colour, while z_2 corresponds head poses, and in this case z_3 seems to correspond to gender and z_5 to the mouth posture.