Advanced Machine Learning, Fall 2023

Thomas Kientz
ENSAE Paris
thomas.kientz@ensae.fr

Elena Loumagne
ENSAE Paris
elena.loumagne@ensae.fr

Adèle Moreau
ENSAE Paris
adele.moreau@ensae.fr

Abstract

This study focuses on removing the background from datasets of faces to gauge the effect on the training and performance of facial generative models. We are also interested on the effect on the interpretability of the latent spaces of the models.

1 Introduction

The realm of image generation has seen some major breakthroughs in recent years. An area of specific interest to us was that of the latent representations of complex images in generative models. Observing these gives us an insight into what best characterizes an image, in the eyes of the trained algorithm. When applied to a dataset of human faces, they can give us objective* (*depending on dataset bias…) insight into our most defining traits.

Most image generation techniques will create an image from a scalar vector as input. During training, the algorithm will learn to map the dimensions of input vectors to different aspects of the image. This vector is what we are calling the latent vector, and its shape is the latent dimension. It can very well be interpreted as a compressed version of the image - indeed, a 128x128x3 RGB image is 49k+ scalars, and latent dimension is typically chosen to have 64-512 scalars.

The key observation that we made was that in many trained face generation algorithms, the background of the image occupied a large portion* (* metrics discussed later - PCA eigenvalues) of the captured variance - i.e. of the latent space. This means that some precious space from the latent vector is wasted on encoding what is behind the face, instead of the face itself.

There are a few neutral-background datasets, but none of them have nearly enough samples to train generative algorithms, and are mostly used to train or fine tune classifiers (https://libguides.princeton.edu/facedatabases). We should aim for 10k-100k+ unique images, but none here exceed 1k with a neutral background.

2 Related work

For the face generation task, some authors decided to circumvent the effect of the background by using a cropped version of their dataset to prevent various noisy backgrounds (Gauthier, 2014) from interacting with the model. Jon Gauthier. 2014. Conditional Generative Adversarial Nets for Convolutional Face Generation. Technical Report. Stanford University (https://www.foldl.me/uploads/2015/conditional-gans-face-generation/paper.pdf)

In a context of face generation with emotions with Sinha et al, 2022, the background issue is also circumvented by first replacing the background of the original image with a specific color (solid

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

dark green) and, as post-processing, the background of the original image is then added back to the generated image. The goal of the authors was different though - they only did this because the "emotions" dataset they used to train their model (MEAD) had a solid dark green background.

Sanjana Sinha, Sandika Biswas, Ravindra Yadav, and Brojeshwar Bhowmick. Emotion-controllable generalized talking face generation. International Joint Conferences on Artificial Intelligence Organization, 2022. (https://arxiv.org/pdf/2205.01155.pdf)

The impact of the background removal on Convolutional Neural Networks has been proved for classification tasks (KC and al., 2021). Achieving a 12% higher accuracy on the task of Plant Disease Classification In-Situ for a dataset with images taken with a homogeneous background compared to that of a dataset with a cluttered background. KC, K.; Yin, Z.; Li, D.; Wu, Z. Impacts of Background Removal on Convolutional Neural Networks for Plant Disease Classification In-Situ. Agriculture 2021, 11, 827. (https://www.mdpi.com/2077-0472/11/9/827)

All these tricks suggest an important effect of the background on image generation. However, we have not found any studies on the importance of this effect of the background on image generation, and particularly face generation.

3 Methodology

Our idea was to create our own neutral-background dataset, from a normal face dataset, the Flicker-Faces-HQ dataset, often used for facial generation tasks.

To do this, we first decomposed the image by isolating the face and background using the *removebg* library. Then we created two datasets, one with a large Gaussian blur background and the other with a grey background. With over 70,000 images, our neutral background datasets are effective for training generative algorithms.

Our main hypothesis is that the removal of the background will have a positive impact on the performance and the training of the generative models. Furthermore, we expect the latent space to be more interpretable.

To test our hypothesis, we trained models on both the original and the greyed-out versions of the same dataset. We picked versions of the two main categories of generative models: Autoencoders (AE) and Generative Adversarial Networks (GAN).

3.1 Auto-encoders

Autoencoders are unsupervised artificial neural networks whose main objective is to learn efficient representations of input data. An autoencoder consists of two sub-networks: an encoder and a decoder. Firstly, the encoder compresses the input data (image) into a lower-dimensional representation. It reduces the dimensionality of the input data by mapping it to a set of hidden variables, often referred to as 'latent space'. The decoder then takes the encoded representation and reconstructs the original input data from it. The aim is to minimize the difference between the input and the reconstructed output, encouraging the model to learn a meaningful and compact representation.

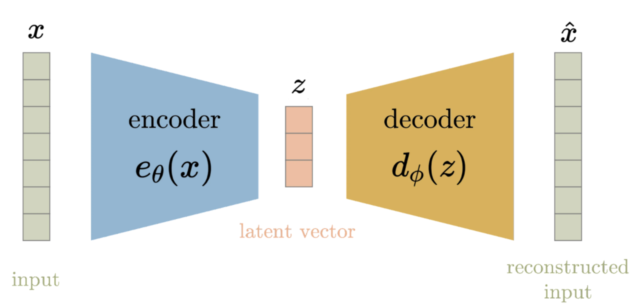

Figure 1: Autoencoder architecture

Let e_θ be the encoder function and d_ϕ the decoder function. During training, input image x is fed to the encoder function and passed through a series of layers that reduce their dimensions to give a compressed latent vector z. The number, type and size of the layers, as well as the size of the latent space, are parameters that can be controlled. Next, the decoder function maps the $mathcalF$ latent space at the bottleneck to the output, attempting to recreate the original image after some generalized non-linear compression.

$$e_\theta : \mathcal{X} \rightarrow \mathcal{F}, d_\phi : \mathcal{F} \rightarrow \mathcal{X}$$

The loss function used to train the neural network through the backpropagation procedure is as follows:

$$\mathcal{L}(x, \hat{x}) = \|x - \hat{x}\|_2 = \|x - d_\phi(e_\theta(x))\|_2$$

Ideally, we would like to have a small latent space to obtain maximum compression, while obtaining a sufficiently small error. But, reducing the size of the latent space beyond a certain value leads to a significant loss of information. However, the latent space of an autoencoder is subject to irregularity due to overfitting. The regularity of the latent space depends on its dimension, the distribution of data in the initial space and the encoder architecture. It is therefore not possible to directly define a generative process by simply sampling a point in the latent space and passing it through the decoder to generate new data.

This is why we introduce the Variational Autoencoder (VAE). This architecture, proposed by King et al. in 2013, addresses the problem of irregular latent space in a conventional autoencoder because, instead of mapping an input to a fixed vector, the encoder outputs parameters of a predefined distribution in the latent space for each input. The VAE then imposes a constraint on this latent distribution, forcing it to be a Gaussian distribution to ensure that the latent space is regularized.

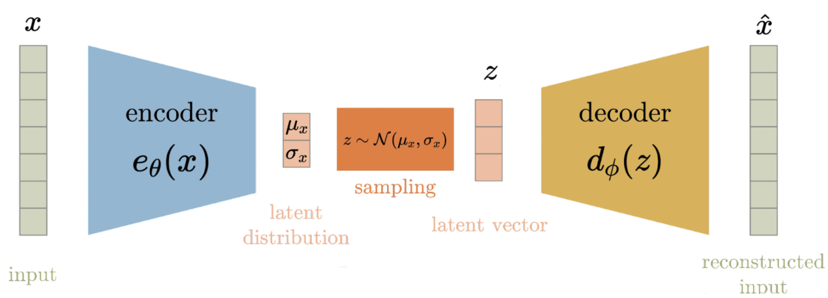

Figure 2: Variational Autoencoder architecture

The main difference between AE and VAE lies in the encoder output. Unlike the autoencoder, where the encoder output was the latent vector, here the encoder produces the mean and standard deviation for each latent variable. The latent vector is then sampled from this Gaussian distribution and passed to the decoder to reconstruct the input as before. The VAE loss function takes into account two terms, the reconstruction loss and the similarity loss (regularizer). As in the AE, the reconstruction loss is the mean squared loss of the input and reconstructed output. The regularizer corresponds to the Kullback-Leibler divergence between the latent space distribution and a standard Gaussian distribution.

We can define the reconstruction loss as follows:

$$\text{reconstruction loss} = \|x - \hat{x}\|_2 = \|x - d_\phi(z)\|_2 = \|x - d_\phi(\mu_x + \sigma_x \epsilon)\|_2$$

where

$$\mu_x, \sigma_x = e_{\theta(x)} \text{ and } \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

And the similarity loss:

$$\text{similarity loss} = \mathrm{D}_{\mathrm{KL}}(\mathcal{N}(\mu_x, \sigma_x), \mathcal{N}(\mathbf{0}, \mathbf{I}))$$

Finally,

$$\text{loss} = \text{reconstruction loss} + \text{similarity loss}$$

As previously noted, the encoder-generated distribution is sampled to obtain the latent vector before it is input into the decoder. This random sampling makes back-propagation difficult for the encoder, since we cannot trace the errors. To address this issue, we employ a reparameterization trick to model the sampling process, enabling the propagation of errors through the network. The latent vector z is represented as a function of the output from the encoder. $z = \mu_x + \sigma_x \epsilon, \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

VAEs generally perform better in image generation tasks. First, by learning a probabilistic latent space representation, it allows for smoother transitions and better interpolation in the latent space, which is crucial for generating diverse and realistic images. Then, VAEs inherently include a regularization term in their objective function, which encourages the learned latent space to be well-structured. This regularization helps prevent overfitting and aids in disentangling meaningful features in the latent space.

Rationale for choosing VAEs: By structure, VAEs are more likely to learn a latent space that is interpretable. Also, they are relatively stable to train.

3.2 Generative Adversarial Networks

Introduced by Ian Goodfellow and al. in 2014, the Generative Adversarial Network consists of a Generator and a Discriminator, two neural networks that are trained simultaneously by adversarial training.
Firstly, the Generator model attempts to model the distribution of data in order to generate realistic samples. It takes as input a random noise sample of fixed size from a latent space and generates new false images. The main objective is to fool the Discriminator by transforming the random noise into images indistinguishable from the real ones, making it harder to classify images as true or false. Then, acting like a binary classifier, the Discriminator attempts to classify the input data as real or generated one.

The generator and discriminator are trained in a loop, with the generator trying to improve its ability to generate realistic samples and the discriminator improving its ability to distinguish between real and fake samples. A loss is then measured and back propagated to update the weights of the two models. Training stops when the generated images look like real images.
In other words, the Generator and the Discriminator play the following two-player minimax game :

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_{\text{latent}}(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))]$$

where:

- $\mathbf{z}$ is a random noise vector drawn from the latent space distribution
- $D(\mathbf{x})$ is the output of the discriminator for a real data sample
- $D(G(\mathbf{z}))$ is the output of the discriminator for a generated data sample

The discriminator tries to maximize the objective function while the generator tries to minimize the objective function. This is what we call adversarial training where the loss of one is the benefit of the other. By alternating between gradient ascent and descent, the network can be trained.

According to the authors of the article, in practice, the Generator objective function does not provide sufficient gradient to allow good learning. At the start of the training, when the sample is likely to be classified as false by the Discriminator, the function $\log(1 - D(G(z)))$ saturates because the gradients are relatively flat. As a result, the Generator is trained to maximize $log(D(G(z)))$, which provides much stronger gradients early in the learning. Thus, Instead of minimizing the likelihood of the Discriminator being correct, we maximize the likelihood of the Discriminator being wrong.

In practice, “vanilla” GANs are hard to train and do not yield very realistic results, by 2024 standards. The first improvement we implemented over the standard GAN architecture was to use deep convolutions (and deconvolutions) for the Generator (and Discriminator), making it a DC-GAN (Deep

Convolutional GAN). When dealing with image data, convolutions are more spatially aware than regular fully-connected layers because they only “mix” the values of pixels close together.

While more efficient than plain GANs, DCGANs have not been shown to generate realistic images. Their latent space is also very disorganized and hard to interpret. To tackle both of these problems, we decided to implement a custom version of NVidia’s StyleGAN, from scratch.

StyleGAN is a variety of ProGANs (Progressive Growing GANs). The key breakthroughs of those models lies in the progressive refinement of images, commencing from a low resolution and gradually advancing to a higher resolution. In the StyleGAN paper, the authors present an enhanced version of ProGAN’s image generator, specifically focusing on advancements in the generator network. One of the key features of the ProGAN architecture is the progressive nature of the layers, which enables the different visual characteristics of an image to be taken into account effectively. The concept is simple: the lower the layer and resolution, the greater the impact on coarser features. This granularity means that the inputs at each level can be modified independently, allowing precise control of the visual elements, from general characteristics to complex details, without affecting the other levels. The result is the generation of high-resolution images with a higher level of authenticity than previous models. To further enhance visual expression, StyleGAN incorporates background noise into each channel before the Adaptive Normalisation (AdaIN) module. This adjustment subtly influences the visual characteristics of the resolution level on which it operates, contributing to the realism of the images generated. On top of this, they introduce a style vector, called "w", which is used in place of the traditional latent vector "x" to give finer control over the appearance of the images generated.

References

References follow the acknowledgments. Use unnumbered first-level heading for the references. Any choice of citation style is acceptable as long as you are consistent. It is permissible to reduce the font size to `small` (9 point) when listing the references. **Note that the Reference section does not count towards the eight pages of content that are allowed.**

[1] Alexander, J.A. & Mozer, M.C. (1995) Template-based algorithms for connectionist rule extraction. In G. Tesauro, D.S. Touretzky and T.K. Leen (eds.), *Advances in Neural Information Processing Systems 7*, pp. 609–616. Cambridge, MA: MIT Press.

[2] Bower, J.M. & Beeman, D. (1995) *The Book of GENESIS: Exploring Realistic Neural Models with the GEneral NEural SImulation System.* New York: TELOS/Springer–Verlag.

[3] Hasselmo, M.E., Schnell, E. & Barkai, E. (1995) Dynamics of learning and recall at excitatory recurrent synapses and cholinergic modulation in rat hippocampal region CA3. *Journal of Neuroscience* **15**(7):5249-5262.