Compression ensembles quantify aesthetic complexity and the evolution of visual art: Supplementary materials

Andres Karjus, Mar Canet Solà, Tillmann Ohm, Sebastian E. Ahnert and Maximilian Schich

This supplementary appendix provides additional technical details on the method and its implementation, as well as some additional results.

List of transformations in the compression ensemble

Table S1 lays out all the transformations used in the version of the compression ensemble used in this paper. As discussed in the Introduction, the exact number and nature of the transformations is somewhat arbitrary, and subject to optimization. The larger the number of (non-collinear) transformations, the better the approximation of the underlying uncomputable complexity.

The Type values stand for the following. C: compression without transformation, CT: compression of transformed image, S: statistical transformation that does not involve a compression algorithm in the narrow sense. The JPEG algorithm is used in two modes, with the quality parameter at 100 and at 0. Sizes refer to image dimensions (after initial normalization) — values smaller than one indicate fractional size, e.g. 0.4 stands for width and height reduce to 40% of the normalized size, before applying transformations. The flood fill: its color is determined by finding a primary color that is not present or least frequent in the distribution of pixel colors of the input image.

Type	Feature	Compressions	Sizes	Description
C	compress	gif, png, jpeg100, jpeg0	1, .4, .2, .1	Compression of the original, of various resizes and algorithms
CT	blur10	gif, png	1, .4	Blur filter (radius, sigma=10)
CT	blur30	gif	1	Blur with (radius, sigma=30)
CT	colors grayscale	gif	1, .4	Grayscale filter
CT	colors quantize bw	gif	1, .4	Black and white filter
CT	colors quantize bw dither	gif	1, .4	Black and white filter with dithering
CT	colors quantize3	gif	1, .4	Color quantization (n=3)
CT	colors quantize5	gif	1	Color quantization (n=5)
CT	colors acos	gif	1, .4	Color values transformation, arc cosine
CT	colors p10	gif	1, .4	Color values transformation, power of 10
CT	colors sqrt	gif	1	Color values transformation, square root
CT	colors round	gif	1	Color values transformation, rounding
CT	colors brightness	gif	1	Color brightness increased by 300\%
CT	colors saturate	gif	1	Color saturation increased by 5000\%
CT	color chroma divide	gif	1	Color values divided by chroma channel values
CT	colors roundchroma	gif	1	Color transformation, rounding chroma channel values
CT	color luminance divide	gif	1	Color transformation, dividing color by the luminance channel

CT	color luminance lighten	gif	1	Color transformation, adding the luminance channel as linear light effect
CT	color darken intensity	gif	1	Color values transformation, pairwise comparison to negative + intensity filter
CT	colors add2	gif	1	Color values transformation, add-composite of the image with itself
CT	lines bw canny	gif	1, .4	Line (edge) detection, Canny filter
CT	lines cartoon	gif	1, .4	Line detection, multiply-composite of Canny and blurred image
CT	lines division gray	gif, png	1, .4	Line detection via division with blurred grayscale of itself
CT	lines edge5 gray	gif, png	1, .4	Line detection based on grayscaled image (radius=5)
CT	lines edge10 gray	gif	1, .4	Line detection based on grayscaled image (radius=10)
CT	lines edge1 color	gif	1, .4	Line detection based on full color image (radius=1)
CT	lines edge2 color	gif	1	Line detection based on full color image (radius=2)
CT	lines hough40	gif	1	Line detection, Hough filter (width=40, height=40, threshold=20)
CT	lines hough50	gif	1, .4	Line detection, Hough filter (width=50, height=50, threshold=70)
CT	lines color comp	gif	1	Line detection via comparison to despecled original
CT	lines color conv	gif	1	Line detection, Sobel convolution-based

CT	lines edge lat	gif	1	Line detection, Local Adaptive Thresholding
CT	emboss gray4	gif	1, .4	Emboss filter on grayscaled image (radius=4, sigma=1)
CT	emboss col1	gif	1	Emboss filter on full color image (radius=1, sigma=0.1)
CT	emboss conv grayd	gif	1	Emboss-like effect on grayscale image via convolution with DoG kernel
CT	emboss conv grayp	gif	1	Emboss-like effect on grayscale image via convolution with Prewitt kernel
CT	emboss modulate	gif	1	Emboss-like effect via modulate-compositing the image with itself and despecling
CT	morph pixelate10	gif	1, .4	Pixelation via rescaling down 10x and back to original size
CT	morph pixelate20	gif	1, .4	Pixelation via rescaling down 20x and back to original size
CT	morph add3 pixelate	gif	1	Pixelation via add-compositing the image with itself 3x
CT	morph despecle10	gif	1	Despecle filter, 10x
CT	morph oilpaint	gif	1	Oil painting effect
CT	morph squares	gif	1	Square-shaped dilation effect
CT	flood centre	gif	1, .4	Flood fill image with single color, from centre
CT	flood hole	gif	1, .4	Flood fill image with single color, filled circle
CT	flood corners	gif	1	Flood fill image with single color, from corners
CT	flood thirds	gif	1	Flood fill image with single color, from intersections of thirds
CT	fx deskew zoom	gif	1, .4	Auto-deskew and crop to center

CT	fx implode	gif	1	Gravity well visual effect at center
CT	fx noise	gif	1	Multiplicative noise effect
CT	fx scramble	gif	1	Randomly scramble positions of all pixels
CT	fx stripes	gif	1	Quantize to 20 colors and order pixels by color frequency
CT	fft1	gif	1	Fast Fourier Transform of image, magnitude output
CT	fft2	gif	1	Fast Fourier Transform of image, phase output
CT	fft1 blur10	gif	1	Fast Fourier Transform of blurred image, magnitude output
CT	fft2 blur10	gif	1	Fast Fourier Transform of blurred image, phase output
CT	fft1 colors quantize3	gif	1	Fast Fourier Transform of colors quantize3, magnitude output
CT	fft2 colors quantize3	gif	1	Fast Fourier Transform of colors quantize3, phase output
CT	fft1 lines division gray	gif	1	Fast Fourier Transform of lines division gray, magnitude output
CT	fft2 lines division gray	gif	1	Fast Fourier Transform of lines_division_gray, phase output
S	fractaldim1	NA	1	Fractal dimension estimate (window=m/10, step=m/40, where m is the mean of x and y dimensions)
S	fractaldim2	NA	1	Fractal dimension estimate (window=m/3, step=m/4)
S	fractaldim3	NA	1	Fractal dimension estimate of lines_bw_canny (window=m/5, step=m/6)

S	stats angleentropy	NA	1	Entropy of angles in lines_hough40
S	stats colfreq entropy	NA	1	Color frequency distribution: entropy
S	stats colfreq max	NA	1	Color frequency distribution: maximum
S	stats colfreq mean	NA	1	Color frequency distribution: mean
S	stats colfreq median	NA	1	Color frequency distribution: median
S	stats colfreq sd	NA	1	Color frequency distribution: standard deviation
S	stats colorfulness lab	NA	1	Color complexity estimate M2, LAB space
S	stats colorfulness rgb	NA	1	Color complexity M3, RGB space
S	stats contrastrange	NA	1	Contrastiveness estimate (luminosity values absolute range)
S	stats contrastsd	NA	1	Contrastiveness estimate (luminosity values standard deviation

Table S1: List of transformations in the compression ensemble used in our analyses.

Image transformations illustrated

Figure S1 illustrates a more comprehensive set of image transformations in the compression ensemble, as exemplified in Figure 1A in the main text. Resizes and rotations are not shown.

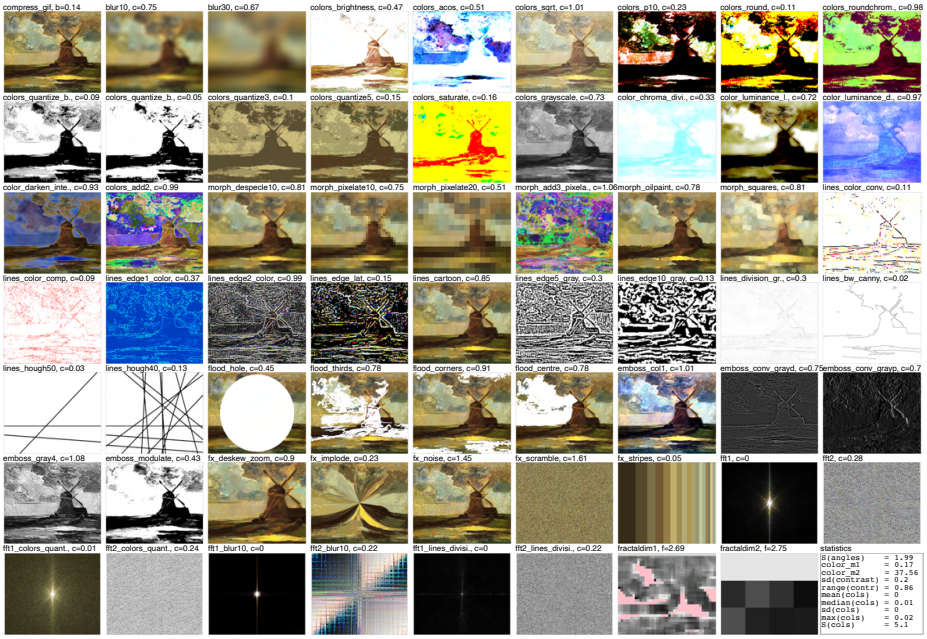

Figure S1 Image transformations in the compression ensemble for the example of "Windmill in the Gein" (1906-1907) by Piet Mondrian. The labels above the transformed image correspond to the feature in table S1 above. The b and c values indicate the resulting raw compression and compression ratio respectively.

Compression ensemble pipeline technical details

This section gives a step-by-step overview of the pipeline used in this work implementing the compression ensemble approach. The pipeline was implemented in R (version 4.1.2). The visual transformations and export into various compression formats is handled by the magick R package (2.7.3), which is a wrapper for the Imagemagick cross-platform software suite (6.9.12.3).

We normalized all images to be the same size in terms of the number of pixels n, while retaining original aspect ratio. We set $n = 160000$, i.e. 400x400 in case of a perfectly square image. Since not all aspect ratios fit into this number, some variation remains in the bitmap file sizes, but we account for that using individual file sizes for the respective compression ratio calculations.

The compressed file sizes are recorded as the mean of two sizes, that of compression of the image in the original orientation, and after 90 degree rotation (as these values usually differ slightly due to the way image compression works). For the Fast Fourier Transform, and additional rotation step is is applied both before doing FFT (as FFT in Imagemagick uses the width of the original image for the dimensionality of the square-shaped outputs).

The statistical transformations include the following. We use both the LAB and RGB space based measures of colorfulness [**?**]; standardize images by quantizing down to 200 colors and record statistics of contrast (range and standard deviation of the lightness channel values in LAB space), color distribution mean, median, max, standard deviation and entropy (which all provide insight into color complexity). We also attempt to estimate composition regularity, first as entropy of the angles of composition lines (based on the Hough transform applied on Canny filtered, i.e. edge-detected images). We also estimate fractal or Hausdorff dimension on bilevel-quantized versions of the images, using both a small and large window size, and on a Canny-filtered image [**?**].

We recommend carrying out the image compression steps in memory, as saving all the files to disk would be computationally inefficient. Parallelization of the pipeline is recommended if the dataset is large. Note that due to the architecture of Imagemagick, the file sizes of image outputs in different formats, if implemented this way, may vary slightly between operating systems. As long as all images in a given dataset go through the pipeline on the same system, this should not be a problem

though. We carried out all processing on an Nvidia DGX Station A100 running Red Hat Enterprise Linux (AS release 4) hosting a Docker running Ubuntu 20.04.3.

The vector of compression ratios and statistical transformations is calculated as follows.

- Import image file, convert to RGB colorspace, downsize so the total number of pixels is (as close as possible to) n. Smaller images are not upsized. If the original image is smaller than 50% of that, stop processing.
- Record the RGB bitmap file size of this normalized image as f
- Compress the image using GIF compression, record the ratio of the size of this file to f as b_{gif}. This is the baseline value that all transformations compressed using GIF will be compared to.
- Also compress the image using other compression algorithms (PNG, JPEG — we do the latter both with quality parameter 0 and 100), record the ratio, these file sizes divided by f. Record the PNG size as b_{png}.
- Downscale the image to 40, 20 and 10% of n, record compression ratio to f with the different algorithms.
- Transform the image using all the visual transformations CT (see list above), compress using GIF, record the ratio c by dividing the compressed file size with b_{gif}
- Also transform the image using a subset of the visual transformations (see Table S1), compress using PNG, record the ratio c by dividing the compressed file size with b_{png}
- Carry out the statistical transformations S (see list above).
- Concatenate the compression ratios and statistical transformation values into a vector.

Note that the compression ensemble approach is general and the specific number of transformations is unimportant. As demonstrated in the Evaluation section, and handful transformations may be enough for some tasks. Here, we opted to implement a fairly large array of transformations and additional compressions with different sizes and algorithms, in order to explore the transformation space in a relatively comprehensive manner. If, unlike here, computation time is important, a smaller set of transformations is likely a better choice.

Transformations mapped onto UMAP

Figure S2 illustrates individual transformations and how they constitute the UMAP shown in Figure 1.C in the main text.

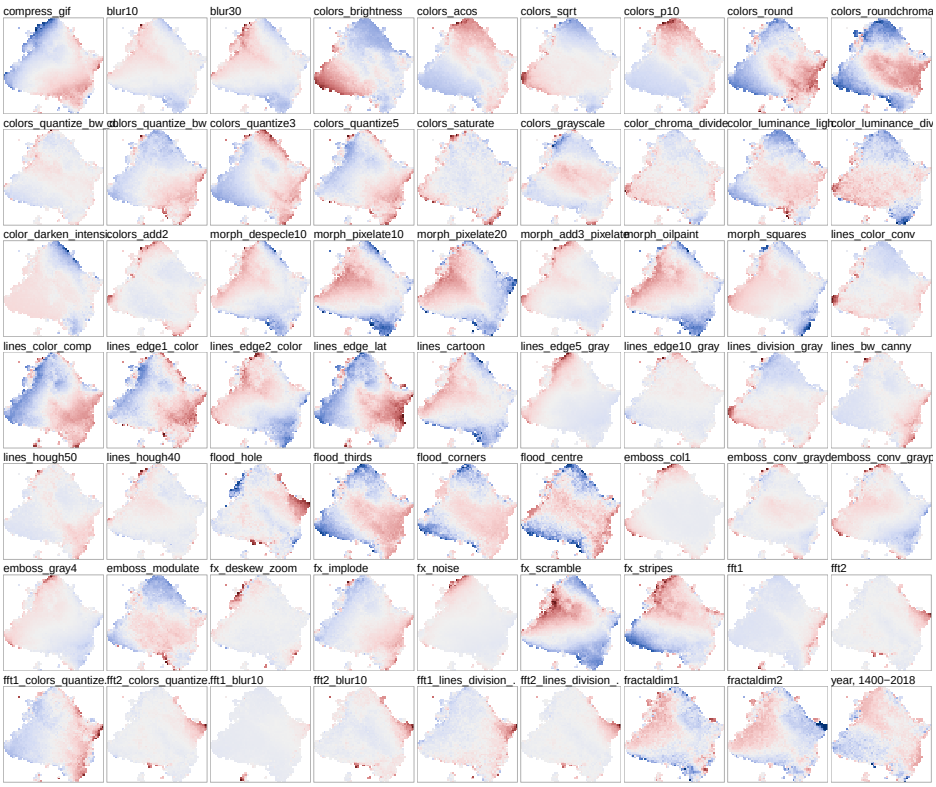

Figure S2 The transformations illustrated in Figure 1.A and S1 mapped onto the UMAP projection of Figure 1.B, here as a heatmap (for better visibility), where each cell represents the mean value of the points in a given area of the UMAP. Blue stands for low, gray for mean and red for high values in a given compression ratio or variable. Date of creation is added as an additional map to the bottom right.

Results of pairwise vector additions

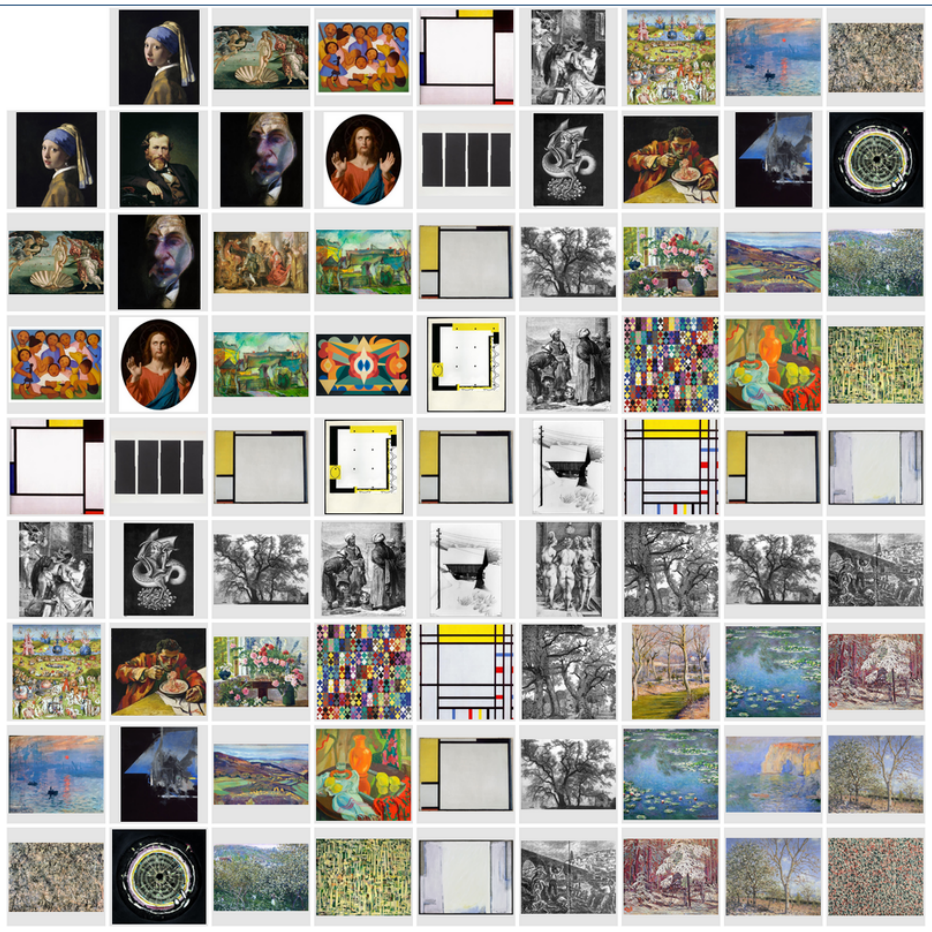

Figure S3 This figure exemplifies the results of vector operations discussed in the main text as a form of latent space navigation, and supplements Figure 1. Each cell is a result of the following operation: take the compression ensemble vectors of the first image in a given row and column, sum them element-wise, find the cosine-nearest neighbor in the ensemble for that new vector. This figure shows how vector addition can be used to navigate the space of aesthetic complexity.

Style periods confusion matrix

Figure S4 supplements the discussion on the art classifier in the Evaluation section in Methods (see also Figure 4 therein).

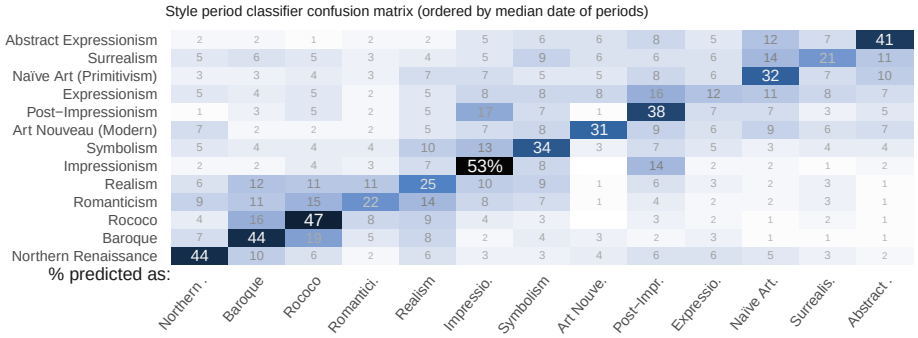

Figure S4 Confusion matrix of the style period classifier. Each row represents the percentage a given style gets predicted as another style; the diagonal stands for correct predictions. The values are from a LDA classifier using 1000 training examples and the full set of transformations; the values are means of 1000 re-runs. This figure illustrates the extent a simple classifier manages to correctly predict style based on the compression ensemble vectors, and which styles are more often confused than others.

Adjustment of temporal resemblance

As an important technical detail, we need to adjust the distances in the Temporal Resemblance model reported in the Results section, due to two biasing factors in the Historical dataset: the boundedness of the dataset (works in the last years have a higher likelihood of having neighbors in the past and vice versa) and its imbalance (much more data in some decades than others). The adjustment works as follows. We calculate temporal distances for all works between 1800-1990. We limit ourselves to this period, where the amount of works per unit of time is more consistent, compared to other parts of the corpus where there is less data but also more variation between years in terms of data points. When finding nearest neighbors, the entire dataset is still taken into account, e.g. a work from 1801 can theoretically have one of its nearest neighbor in 1400 or 2018. We then fit a generalized additive regression model, predicting distance by year. The residuals from that model, still on the scale of years, approximate temporal resemblance given the shape and bounds of the data.

Artworks in temporal resemblance

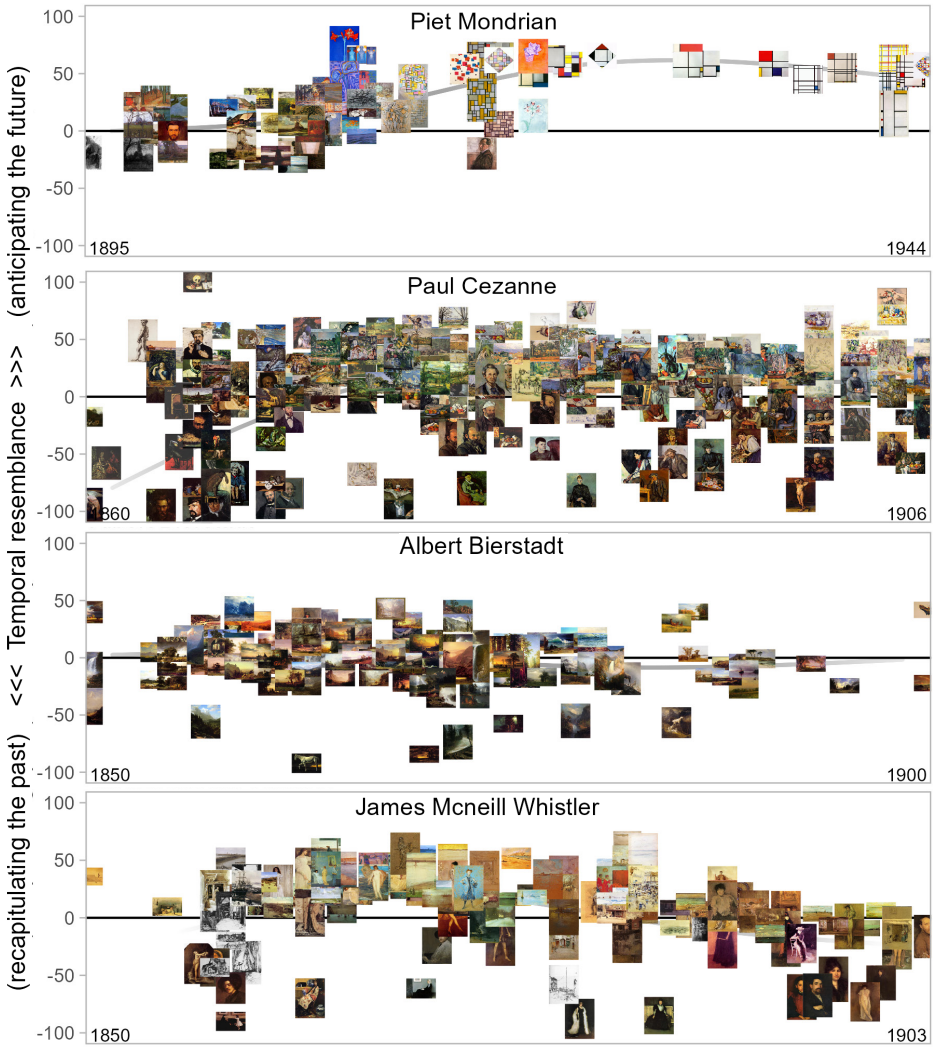

Figure S5 Example artists from Figure 3 with larger thumbnails for qualitative exploration, centered on their coordinates on the vertical and horizontal axes.