Redistributor: Transforming Empirical Data Distributions

Pavol Harar, Dennis Elbrächter, Monika Döfler, Kory D. Johnson

Abstract—We present an algorithm and package, Redistributor, which forces a collection of scalar samples to follow a desired distribution. When given independent and identically distributed samples of some random variable \( S \) and the continuous cumulative distribution function of some desired target \( T \), it provably produces a consistent estimator of the transformation \( R \) which satisfies \( R(S) = T \) in distribution. As the distribution of \( S \) or \( T \) may be unknown, we also include algorithms for efficiently estimating these distributions from samples. This allows for various interesting use cases in image processing, where Redistributor serves as a remarkably simple and easy-to-use tool that is capable of producing visually appealing results. For color correction it outperforms other model-based methods and excels in achieving photorealistic style transfer, surpassing deep learning methods in content preservation. The package is implemented in Python and is optimized to efficiently handle large datasets, making it also suitable as a preprocessing step in machine learning. The source code is available at https://github.com/paloha/redistributor.

Keywords Data preprocessing · Color correction · Density estimation · Histogram matching · Quantile transform

1 INTRODUCTION

Redistributor provably transforms an empirical distribution to a known distribution or to match a second empirical distribution, see Figure 1. Historically, similar methods have been used to transform data to be Gaussian. The idea of transforming data to a known distribution has a long history and is intuitively appealing. The most famous method for achieving a transformation to normality, the Box-Cox transformation [Box and Cox, 1964], has been cited thousands of times in a broad range of fields, cf. [Peltier et al., 1998], [Osborne, 2010]. The transformation is motivated by the fact that normality is a core assumption of many statistical tests. The central idea is a simple translation of raw data to ranks or order statistics and then transforming these into corresponding normal scores, e.g., quantiles from the standard normal distribution. If data is initially provided as ranks, this idea dates back to 1947 [Bartlett, 1947]. Redistributor generalizes the idea to transformation into any distribution for which a percent-point function is available or can be empirically estimated.

In addition to the package and its algorithmic details, we present theoretical results supporting the use of the method as well as a broad array of examples in which transforming distributions is beneficial. In the context of image processing, we can correct color issues in photography, producing far better results than other automated procedures. We also consider further uses of matching the color distribution of an image to some reference image, e.g. translating it into a preferred color scheme for aesthetic purposes, creating photographic mosaics, and performing image data augmentation in color space. Lastly, we discuss the use of Redistributor in signal processing and preprocessing within a machine learning pipeline.

2 RELATED WORK

Redistributor is implemented as a Scikit-learn transformer that is convenient for processing large datasets and integration in machine learning pipelines. While the Scikit-learn [Pedregosa et al., 2011] resp. Scikit-image [Van der Walt et al., 2014] functions quantile_transform and match_histograms have the same mathematical basis, they are limited to specific use cases and do not produce a reusable, estimated data distribution as our LearnedDistribution and KernelDensity classes do. They also do not feature a robust treatment of boundary values, which may cause problems in real-world applications.

A related R [R Core Team, 2022] package, bestNormalize [Peterson, 2021], provides a set of methods for normalizing a distribution and functionality to pick the best transformation from this set. We deviate from testing the parameterization of such transformations or picking the best one and focus instead on the broader applicability and scalability of the procedure. The generality and additional flexibility of Redistributor to estimate and map between two data distributions opens the door for a host of new use cases.

One use case of Redistributor in the context of image processing is color mapping, i.e. transforming the color scheme of a given image to that of some reference image, usually for
Fig. 2: Applying the transformation $R$ from $\hat{F}_S$ to $\hat{F}_T$, where $\hat{F}_S$ is an estimate of a Double Gamma distribution $F_S$ obtained by Algorithm 2 from 1000 iid samples, and $\hat{F}_T$ is an estimate of a Gaussian distribution $F_T$ obtained by Algorithm 2 from 1000 iid samples. Subfigures (a) and (c) display density histograms.

The application of the inverse cumulative distribution function, $X \sim \mathcal{U}[0,1]$, one has

$$F_X(x) \sim \mathcal{U}[0,1]$$

and, if $F_X$ is invertible,

$$X \sim F_X^{-1}(\mathcal{U}[0,1]),$$

where $\mathcal{U}[0,1]$ is the uniform distribution on the interval $[0,1]$. The application of the inverse cumulative distribution function,
or percent-point function (PPF), to samples taken uniformly from $[0, 1]$ in order to obtain samples from a random variable $X$ is known as inverse transform sampling [Devroye, 2006].

Another direct consequence of the above is that, given a source distribution $F_S$ and a target distribution $F_T$ of random variables $S$ and $T$, respectively, the transformation

$$R := F_T^{-1} \circ F_S$$

achieves

$$R(S) \overset{\text{d}}{=} T,$$

where $\overset{\text{d}}{=}$ denotes equality in distribution.

While this is straightforward in principle, in practice one may not have access to the CDFs or PPFs of $S$ and $T$, and needs to estimate them from samples. When the number of samples in small, this can be accomplished nicely via kernel density estimation (see Algorithm 2). In order to produce a continuous and invertible transform in the case of large sample sizes, as may, for example, be encountered when using this algorithm as a preprocessing step for machine learning, we use a continuous piecewise linear empirical estimator (see Algorithm 2) of a cumulative distribution function $F$. Specifically, for samples $X_1, \ldots, X_n$ it maps $X_k$ to $\frac{k}{n+1}$ and is linear in between, whereas the behaviour outside of the range of the observed samples should be chosen depending on the use case. The implementation of these methods is described in more detail in the next Section.

In Section 6 we prove consistency of our empirical estimator, i.e. quantify how the transformation obtained by replacing $F_S$ in $F$ with our empirical estimator, based on $n$ independent and identically distributed (iid) samples, approximates the transformation using the true CDF as $n$ increases. As demonstrated in Section 7 this transformation turns out to be quite useful when applied to images by treating them as a collection of pixels, even though it is certainly not reasonable to assume that the pixels of an image are independent. We will discuss the theoretical underpinning of this in Section 6.1.

5 Algorithms

In order to use our Redistributor Python package to compute the transformation $R$, we have to instantiate the Redistributor class by specifying the source and target distributions. Each of the distributions must be described by an instance of a class that implements at least methods for computing CDF and PPF. Assuming the CDF (and its inverse, the PPF) is continuous, the transformation $R$ and its inverse is then given by composition of those functions as described in Algorithm 1.

5.1 Estimating Distribution from Data

It may happen that we want to compute the transformation $R$ between distributions of which one or both are not known explicitly. In that case, we could estimate the missing distribution(s) from data, by computing the empirical cumulative distribution function (eCDF). As the eCDF is not a continuous function by definition, its inverse is not readily available. Therefore, in the LearnedDistribution class, we use linear interpolation to obtain a continuous and invertible estimate of the CDF. This class also provides the methods for PPF, probability density function (PDF), and a function for random variable sampling (RVS). The implementation is described in Algorithm 2. Note that for large numbers of samples it can become quite inefficient to use all of the samples, so a value for bins can be provided and, if the number of samples provided is larger than that, they will be subsampled to a set of size bins before generating the CDF and PPF. By default, the value of bins is set to the min. of 5000 and the number of samples.

There exist other ways of estimating distributions which are arguably more suitable if only a small amount of samples is available. One such technique is kernel density estimation (KDE) [Chen, 2017]. A popular implementation of KDE exists as a part of the Scikit-learn Python package [Pedregosa et al., 2011], however, it does not implement CDF and PPF functions. For convenience, we have introduced a KernelDensity class that extends the Scikit-learn KDE (with Gaussian kernel) by implementing the methods for CDF, PDF, and PPF. This class described in Algorithm 3 makes estimating the source and/or target distributions of Redistributor effortless.

5.2 Handling Duplicate Values

Redistributor assumes that both the source and target distributions are continuous. This implies that a.s. (almost surely) no duplicate values will be present in a random sample from these distributions. Observed data, however, may contain duplicate values that cannot be present in the Algorithm 2 input array as they would render the CDF non-invertible. This limitation would disqualify usage of LearnedDistribution and by extension Redistributor, on a particularly interesting type of data – images. For this reason, we provide a function make_unique that adds uniform noise of desired magnitude to duplicate values in an array. This ensures that the array has no repeating values a.s. so the users can “pretend” the data points come from a continuous distribution even though the array is for example quantized. The function guarantees that the minimum and maximum values are never changed, which is of technical importance in treating the boundaries described in Section 5.3.

5.3 Treating Boundary Values

Estimating a CDF outside of the range of the available samples is a well-known problem as it amounts to extrapolating beyond the observed data. In particular it is already quite difficult to estimate the support of the distribution from which the samples are taken. We deal with this in the following way.

Let min and max denote the smallest resp. largest value of the $n$ samples that were used to generate the Learned Distribution with Algorithm 2 and write $\Delta = 1/(\text{bins} + 1)$. Then, the CDF always maps inputs in $[\text{min}, \text{max}]$ to values in $[\Delta, 1 - \Delta]$ and on these intervals the CDF and PPF are inverses of each other. A choice becomes necessary for inputs

2Roughly speaking, extrapolation to inputs outside the range of the observed samples requires some assumption of the probability distribution from which they are sampled. For more details on this see Section 5.3.

3In Section 5 we use terminology from object-oriented programming.
Algorithm 1 Redistributor

Instantiating

Input: source, target
Output: r (Redistributor instance)

▷ Objects with ppf and cdf methods implemented
▷ Subclass of sklearn.base.TransformerMixin

r.transform

Input: x (Float array)
Output: x’ (Float array)
1: x’ ← target.ppf(source.cdf(x))

▷ Elements from source distribution
▷ Elements of x transformed into target distribution

r.inverse_transform

Input: x’ (Float array)
Output: x (Float array)
1: x ← source.ppf(target.cdf(x’))

▷ Elements from target distribution
▷ Elements of x’ transformed into source distribution

Algorithm 2 Learned Distribution

Instantiating the class

Input:

x (Float array) ▷ Samples from unknown distribution
a (Float, optional) ▷ Assumed left boundary of the data distribution’s support
b (Float, optional) ▷ Assumed right boundary of the data distribution’s support
bins (Int, optional) ▷ Scalar influencing the precision of CDF approximation
... ▷ For a full signature consult the source code

Outline of the code:

1: lp ← Choose the lattice points based on bins ▷ Support of l.ppf
2: lv ← Get the lattice values from x using a partial-sort ▷ Support of l.cdf
3: Make lv unique if desired & necessary ▷ So inversion is possible
5: l.cdf ← Compute a linear interpolant of lp on lv
4: l.ppf ← Compute a linear interpolant of lv on lp
6: l.pdf ← Compute a linear interpolant of the derivative of lv on lp
7: l.rvs ← l.ppf(random uniform sample from the ppf support)

Output:

l (LearnedDistribution instance) ▷ Object suitable as Algorithm 1 input

Available methods

l.cdf ▷ Continuous piecewise linear approximation of CDF
Input: q (Float array)
Output: p (Float array)

l.ppf ▷ Inverse of l.cdf
Input: p (Float array)
Output: q (Float array)

l.pdf ▷ Numerical derivative of l.cdf
Input: q (Float array)
Output: d (Float array)

l.rvs ▷ Random value generator
Input: size (Int)
Output: s (Float array)

▷ Desired number of elements to generate
▷ Random sample from the estimated distribution
Algorithm 3 Kernel Density

Instantiating the class

Input:

\( x \) (Float array)
\( \text{bandwidth} \) (strictly positive Float, optional)
\( \text{cdf\_method} \) (Str, 'precise' or 'fast', optional)
\( \text{grid\_density} \) (Int, optional)
...

▷ Elements from unknown distribution
▷ Standard deviation of the Gaussian kernel
▷ Default behaviour of \( k\text{-cdf} \) function
▷ Scalar influencing the precision of \( k\text{-cdf}/\text{fast}/ \) and \( k\text{-ppf} \)
▷ For a full signature consult the source code

Outline of the code:

1: \( kde \leftarrow \) Define a Gaussian mixture model using elements of \( x \) and \( \text{bandwidth} \)
2: \( k\text{-cdf}/\text{precise}/ \leftarrow \) An explicit evaluation of the \( \text{cdf} \) function under given \( kde \)
3: \( a, b \leftarrow \) Compute the empirical support of \( \text{cdf} \) under given \( kde \)
4: \( lv \leftarrow \) Get the lattice values by evaluating \( k\text{-cdf}/\text{precise}/ \)
5: \( lp \leftarrow \) Compute a linear interpolant of \( lv \) on \( lp \)
6: \( k\text{-pdf} \leftarrow \) Compute a linear interpolant of \( lp \) on \( lv \)

Output:

\( k \) (KernelDensity instance)
▷ Object suitable as Algorithm 1 input

Available methods

\( k\text{-cdf} \)

Input: \( q \) (Float array)
Output: (Float array)
if \( \text{cdf\_method} = \) 'precise' then \( p' \)
if \( \text{cdf\_method} = \) 'fast' then \( p'' \)
▷ cumulative distribution function under given \( kde \)
▷ Quantile
▷ Explicit Gaussian mixture \( \text{cdf} \)
▷ Piecewise linear approximation of \( k\text{-cdf}/\text{precise}/ \)

\( k\text{-ppf} \)

Input: \( p \) (Float array)
Output: \( q \) (Float array)
▷ Inverse of \( k\text{-cdf}/\text{fast}/ \)
▷ Lower tail probability
▷ Quantile

\( k\text{-pdf} \)

Input: \( q \) (Float array)
Output: \( d \) (Float array)
▷ Probability density function under the given \( kde \)
▷ Quantile
▷ Probability density

\( k\text{-rvs} \)

Input: \( size \) (Int)
Output: \( s \) (Float array)
▷ Random value generator
▷ Desired number of elements to generate
▷ Random sample from the estimated distribution

outside of these intervals. In case the upper or lower boundary of the support of the distribution is known it can be specified as \( a \) resp. \( b \), in which case the CDF will take the value 0 at \( a \) and 1 at \( b \) and be linear on the intervals \([a, \text{min}] \) and \([\text{max}, b] \).
In this case the CDF and PPF will be inverses of each other on the interval \([a, b] \) resp. \([0, 1] \). In this case the CDF will not accept inputs smaller than \( a \) or larger than \( b \).

If one does not want to make a choice about the assumed support of the distribution, the boundaries can be left unspecified in which case the CDF will simply map all inputs smaller than \text{min} to \Delta resp. all inputs larger than \text{max} to \( 1 - \Delta \). Similarly, the PPF will map all inputs in \([0, \Delta] \) to \text{min} resp. all inputs in \([1 - \Delta, 1] \) to \text{max}. This option is essentially saying that we do not expect to receive inputs to the CDF, which are outside of those we have already seen, but in case we do happen to receive such inputs we do not want to extrapolate since this might lead the Redistributor transformation to produce unreasonably large outputs, depending on the target PPF. Doing so could cause issues, for example, in the case of the target being a Gaussian distribution, where the PPF tends to \( \pm \infty \) very quickly as the input gets close to 1 resp. 0. It is possible to only specify either \( a \) or \( b \) and leave the other unspecified. In case of Algorithm 3 the situation is simpler due to the fact that it uses the Gaussian kernel to estimate the density, which has a natural tail behaviour.
Fig. 3: Treating boundary values in Algorithm 2 – The simplest possible example using only 3 data points. Each subfigure shows where the supported values of respective functions map based on whether the boundaries are explicitly set or not. Endpoints denote the “valid” support, i.e. the interval where the function is strictly increasing. E.g., if the boundary \( a \) is not set, all CDF values from the interval \([-\infty, \min]\) map to the constant \( \Delta = 1/(\text{bins}+1) \). The PPF maps values from the interval \([0, \Delta]\) to \(\min\), i.e. the minimum value of the provided data. Analogously, the same applies for the boundary value \( b \), which can be specified or not independently of \( a \).

Fig. 4: Timing Algorithms 2 and 3 on a consumer grade CPU – Intel® Core™ i7-8565U 1.80GHz. In both subfigures, \( N \) denotes the number of input data points. In (a), \( K \) denotes the number of bins and in (b), \( K \) denotes the grid density. Note that in comparison to Algorithm 3, Algorithm 2 can handle approx. 4 orders of magnitude more data points in the same amount of time making it applicable in larger data-processing pipelines.

5.4 Time & Space Complexity

The time complexity of Algorithm 2 in the worst case is \( O(N \log K) \), where \( N \) is the number of data points and \( K \) is the number of bins. The bottleneck is partial sorting the data with introspective sort [Musser, 1997] to get the lattice values. In the best case, the input array is already sorted and the time complexity is \( O(N) \). In the worst case \( K = N \) and we need to do a full sort instead of a partial sort. In case the lattice values contain repeated values, a call to `make_unique` will add \( O(K \log(K)) \). Computation time as a function of \( N \) on a specific hardware is shown in Figure 4(a).

The space complexity of Algorithm 2 is \( O(N) \). Auxiliary space complexity is \( O(N) \) if \( \text{keep}_x\text{ unchanged} = \text{False}, \) and \( O(K) \) otherwise. In practical terms, when we use a 1 GB input array, the peak memory will be approx. 2 GB if we want to keep the order of elements in the input array unchanged. If changing the order is not a problem, the peak memory will be approx. 1 GB (not counting the negligible overhead costs of the interpreter, loaded modules, and K-sized arrays stored in the interpolants), i.e. almost no additional memory is used. Being aware of space complexity of this algorithm is important, as its speed allows us to process reasonably large amounts of data points for it to be relevant.

The time complexity of Algorithm 3 is \( O(NK) \), where \( N \) is the number of data points and \( K \) is the grid density. Most of the time is spent on \( N \) evaluations of the Gaussian
If $\varphi'[F]$ is defined and continuous on the entirety of $X$, we also have

$$r_n(\varphi(F_n) - \varphi(F)) = [\varphi'[G]](r_n(F_n - F)) + o_P(1).$$

As we are primarily interested in cases where the cumulative distribution function $F$ is continuous and we are generating a continuous empirical cumulative distribution function, a natural Banach space to consider would be $(C_b(\mathbb{R}, \mathbb{R}), \| \cdot \|)$, i.e. the space of bounded continuous functions from $\mathbb{R}$ to $\mathbb{R}$ equipped with the uniform norm

$$\| f \|_\infty := \sup_{t \in \mathbb{R}} |f(t)|.$$

However in order to derive the desired convergence results, we will need to deal with usual empirical cumulative distribution functions which are discontinuous. Therefore we also introduce the larger space $S = (S, \| \cdot \|_\infty)$, where

$$S := \{ f : \mathbb{R} \to \mathbb{R} : \forall x \in \mathbb{R} : (\exists \lim_{t \searrow x} f(t) \land \lim_{t \nearrow x} f(t) = x) \}$$

is the set of functions which are right continuous and have left limits. We can now consider, for $g \in C((0,1), \mathbb{R})$ and $x \in \mathbb{R}$, the real-valued statistical functional

$$R_{g,x} : S_x \to \mathbb{R}$$

$$F \to g(F(x)),$$

where

$$S_x := \{ F \in S : F(x) \in (0,1) \}.$$

The functional $R_{g,x}$ maps a cumulative distribution function $F \in S_x$ to the value at $x$ of the Redistributor transformation generated from $g$ and $F$. We consider the inverse cumulative distribution function $g$ as a function on $(0,1)$ and restrict the domain of $R_{g,x}$ to $S_x$ in order to avoid the possibility of $g(F(x)) \in (-\infty, \infty)$. As we have $P_{F \to g}[F(x) \in (0,1)] = 1$ anyway, this is not a practically relevant restriction. Note that $F \in S_x$ if $x$ is in the interior of the support of the density function corresponding to $F$. We can now show the following

**Lemma 6.3.** Let $g \in C^1((0,1), \mathbb{R})$, $x \in \mathbb{R}$, and $F \in S_x$. Then the Hadamard derivative of $R_{g,x}$ at $F$ is continuous and given by $[R'_{g,x}(F)](h) = g'(F)h(x)$ for $h \in S$.

**Proof.** Let $h \in S$, $(h_n)_{n \in \mathbb{N}} \subseteq S$, $(t_n)_{n \in \mathbb{N}} \subseteq \mathbb{R}$ with $\lim_{n \to \infty} \| h_n - h \| = 0$, $\lim_{n \to \infty} |t_n| = 0$, and $x + t_n h_n \in D$, it holds that

$$\lim_{n \to \infty} \| [\varphi(F(t_n h_n + F)) - \varphi(F)](h) \|_\infty = 0.$$

With this notion of differentiability we can make use of the following theorem.

**Theorem 6.2.** (Vaart, 1998, Thm.20.8). Let $(X, \| \cdot \|_X), (Y, \| \cdot \|_Y)$ be Banach spaces, $D \subseteq X$ and let $\varphi : D \to Y$ be Hadamard differentiable at $F \in X$ if there exists a continuous linear map $\varphi'(F) : Y \to X$ such that for every $h \in X$, $(h_n)_{n \in \mathbb{N}} \subseteq X$, $(t_n)_{n \in \mathbb{N}} \subseteq \mathbb{R}$ with $\lim_{n \to \infty} \| h_n - h \| = 0$, $\lim_{n \to \infty} |t_n| = 0$, and $x + t_n h_n \in D$, it holds that

$$r_n(\varphi(F_n) - \varphi(F)) \Rightarrow [\varphi'(F)](G).$$

We do not explicitly discuss the underlying probability spaces, but refer the interested reader to Vaart, 1998.
By considering the Taylor expansion of \( g \) around \( F(x) \) evaluated at \( F(x) + t_n h_n(x) \) we get that
\[
g(F(x) + t_n h_n(x)) = g(F(x)) + g'(F(x)) t_n h_n(x)
+ r(F(x) + t_n h_n(x)) t_n h_n(x),
\]
where
\[
\lim_{n \to \infty} r(F(x) + t_n h_n(x)) = 0.
\] (2)

Consequently we get
\[
\left| \frac{g(F(x) + t_n h_n(x)) - g(F(x))}{t_n} - g'(F(x)) h(x) \right| = \left| g'(F(x)) h_n(x) + r(F(x) + t_n h_n(x)) h_n(x) - g'(F(x)) h(x) \right|
\leq |g'(F(x))| |h_n(x) - h(x)| + |r(F(x) + t_n h_n(x)) h_n(x)|.
\]
Due to (2) and \( \lim_{n \to \infty} ||h_n - h||_\infty = 0 \) this implies
\[
\lim_{n \to \infty} \left| R_{g, x}(F + t_n h_n) - R_{g, x}(F) - g'(F(x)) h(x) \right| = 0,
\]
which proves that \( |R_{g, x}(F)|(h) = g'(F(x)) h(x) \).
Moreover, \( R_{g, x}(F) \) is continuous as, for all \( h_1, h_2 \in S \), it holds that
\[
|R(h_1) - R(h_2)| = |g'(F(x)) h_1(x) - g'(F(x)) h_2(x)|
\leq |g'(F(x))| ||h_1 - h_2||_\infty.
\]

Note that each empirical cumulative distribution function \( F_n \in S \) is derived from a real valued random variable with cumulative distribution function \( F \in C_b([0, 1]) \) by application of a deterministic function \( h_n^* : T^n \to S \) to \( (X_1, \ldots, X_n) \), where \( X_i \sim F, i \in \{1, \ldots, n\} \), iid and
\[
T^n := \{x \in \mathbb{R}^n : x_1 < \cdots < x_n \}.
\]

Specifically we have
\[
F_n(t) = (h_n^*(X_{\pi(1)}, \ldots, X_{\pi(n)}))(t) := \frac{1}{n} \sum_{k=1}^{n} 1_{[X_k \leq t]},
\]
where \( (X_{\pi(1)}, \ldots, X_{\pi(n)}) \) is an increasing rearrangement of \( (X_1, \ldots, X_n) \).

Colloquially speaking, the sequence \( (h_n^*)_{n \in \mathbb{N}} \) describes the deterministic method to generate the usual empirical cumulative distribution function out of real-valued samples from \( F \). In this functional setting the convergence of the empirical cumulative distributions functions can be described by the following Theorem from [Vaart, 1998].

**Theorem 6.4.** Let \( F \in S \) be the cumulative distribution function of a real-valued random variable and \( (F_n)_{n \in \mathbb{N}} \) the corresponding sequence of empirical cumulative distribution functions. Then we have
\[
\sqrt{n} (F_n - F) \rightsquigarrow \mathbb{G}_F,
\] (3)

I.e., convergence in distribution in the space \( (S, ||\cdot||_\infty) \), where \( \mathbb{G}_F \) is a zero-mean Gaussian process with covariance
\[
\text{cov}[\mathbb{G}_F(s), \mathbb{G}_F(t)] = \min\{F(s), F(t)\} - F(s) F(t).
\]

Our algorithm, however, generates a continuous empirical distribution function out of a vector of samples \( (X_1, \ldots, X_n) \). We therefore need to consider what conditions on a sequence \( h = (h_n)_{n \in \mathbb{N}} \) of functions \( h_n : T^n \to S \) are required to ensure that (3) remains valid. More precisely, we write
\[
F_n^h := h_n(X_{\pi(1)}, \ldots, X_{\pi(n)}) \in S
\]
for the function-valued random variable derived by the deterministic function \( h_n \) from the real-valued cumulative distribution function \( F \in C_b([0, 1]) \). This is done via taking \( X_i \sim F, i \in \{1, \ldots, n\} \), iid, where, again, \( (X_{\pi(1)}, \ldots, X_{\pi(n)}) \) is an increasing rearrangement of \( (X_1, \ldots, X_n) \). We then need to show that with this construction, we still obtain
\[
\sqrt{n} (F_n^h - F) \rightsquigarrow \mathbb{G}_F.
\]

To accomplish this we will make use of a continuous mapping theorem for Banach-space-valued random variables.

**Theorem 6.5 ([Vaart, 1998], Thm.18.11).** Let \( (D, ||\cdot||_D) \), \( (E, ||\cdot||_E) \) be Banach spaces. Assume that \( D_n \subseteq D \) and \( g_n : D_n \to E, n \in \mathbb{N} \cup \{\infty\} \), satisfy for every sequence \( (x_n)_{n \in \mathbb{N}} \) with \( x_n \in D_n \), that for every subsequence \( (x_{n_k})_{k \in \mathbb{N}} \subseteq \mathbb{N} \) we have
\[
\exists s \in D_\infty : \lim_{k \to \infty} ||x_{n_k} - s||_D = 0
\implies \lim_{k \to \infty} ||g_{n_k}(x_{n_k}) - g(s)||_E = 0.
\]

Then, for any sequence of random variables \( X_n \in D_n, n \in \mathbb{N} \), and random variable \( X \in D_\infty \) with \( g_\infty(X) \in E \), it holds that
\[
X_n \rightsquigarrow X \implies g_n(X_n) \rightsquigarrow g_\infty(X).
\]

We obtain the following Lemma, which essentially states that, if a deterministic construction method \( (h_n)_{n \in \mathbb{N}} \) is sufficiently close to the one of the usual empirical cumulative distribution function, i.e., \( (h_n^*)_{n \in \mathbb{N}} \) then the corresponding sequences of function-valued random variables \( (F_n^h)_{n \in \mathbb{N}} \) and \( (F_n)_{n \in \mathbb{N}} \) enjoy the same kind of convergence.

**Lemma 6.6.** Let \( F \in C_b([0, 1]) \) be the cumulative distribution function of a real-valued random variable and \( h_n : T^n \to S, n \in \mathbb{N} \), a sequence of functions. Assume that there exist constants \( C > 0 \) and \( c > \frac{1}{2} \) such that for all \( n \in \mathbb{N}, x \in \mathbb{R}^n \) it holds that
\[
||h_n(x) - h_n^*(x)||_\infty \leq C n^{-c}.
\] (4)

Then we have
\[
\sqrt{n} (F_n^h - F) \rightsquigarrow \mathbb{G}_F,
\]

A sequence \( (X_n)_{n \in \mathbb{N}} \) of random variables in some metric space \( (M, \rho) \) converges in distribution to \( X \in M \), written as \( X_n \rightsquigarrow X \), if it holds for every bounded, continuous functional \( f : M \to \mathbb{R} \) that \( \lim_{n \to \infty} E[f(X_n)] - E[f(X)] = 0 \). Note that some rather subtle issues of measurability arise here, when considering the underlying probability spaces, for a treatment of which we refer to [Vaart, 1998].

With some variations at the boundaries depending on the use case.
where $G_F$ is a zero-mean Gaussian process with covariance
\[ \text{cov}[G_F(s), G_F(t)] = \min\{F(s), F(t)\} - F(s)F(t). \]

**Proof.** This proof is effected by a fairly straightforward application of Theorem 6.5. To this end, let $D = D_\infty = E = (S, \| \cdot \|_\infty)$ and, for $n \in \mathbb{N},$
\[
D_n := \{ \bar{x} \in S : x \in T^n \},
\]
where
\[
\bar{x}(t) := \sqrt{n}(F(t) - \frac{1}{n} \sum_{k=1}^{n} \mathbf{1}_{[x_k \leq t]} \} = \sqrt{n}(F(t) - (h_n^*)(x(t))),
\]
x $\in T^n$, $t \in \mathbb{R}$. In particular we have that $\sqrt{n}(F_n - F)$ is a random variable in the function space $D_n \subseteq S$. Moreover, the map $T^n \ni x \mapsto \bar{x} \in D_n$ is bijective since $F$ is continuous, which means that the points of discontinuity of $\bar{x}$ uniquely determine $x$. Thus there are well-defined maps $g_n : D_n \to S$, $n \in \mathbb{N}$, which satisfy for all $n \in \mathbb{N}$, $\bar{x} \in D_n$, that
\[
g_n(\bar{x}) = \sqrt{n}(F - h_n(x)).
\]
We will show that the requirement in Theorem 6.5 is satisfied
with $g_\infty : S \to S$ being the identity. To this end consider $(\bar{x}_n)_{n \in \mathbb{N}}$ with $\bar{x}_n \in D_n$ and assume there exists a subsequence $(\bar{x}_{n_k})_{k \in \mathbb{N}}$ and $s \in S$ such that
\[
\lim_{k \to \infty} \| \bar{x}_{n_k} - s \|_\infty = 0.
\]
We have
\[
\| g_{n_k}(\bar{x}_{n_k}) - g_\infty(s) \|_\infty \leq \sqrt{n_k} \| h_{n_k}(x_{n_k}) - h_{n_k}(x_{n_k}) \|_\infty \leq \frac{Cn_k^{\frac{1}{2} - \epsilon}}{n_{k}},
\]
whereas the second term vanishes for $k \to \infty$ due to assumption (5). Consequently we have
\[
\lim_{k \to \infty} \| g_{n_k}(\bar{x}) - g_\infty(s) \|_\infty = 0
\]
as claimed. Note that
\[
g_n(\sqrt{n}(F_n - F)) = g_n(\sqrt{n}(F - h_n^* (X_{\pi(1)}, \ldots, X_{\pi(n)})))
\]
\[
= \sqrt{n}(F - F_n^* (X_{\pi(1)}, \ldots, X_{\pi(n)}))
\]
\[
= \sqrt{n}(F_n^* - F).
\]
Since $\sqrt{n}(F_n^* - F) \Rightarrow G_F$ by Theorem 6.4 application of Theorem 6.5 yields $\sqrt{n}(F_n^* - F) \Rightarrow G_F$, which completes the proof.

We next argue that the constructions $(h_n)_{n \in \mathbb{N}}$ we use to obtain the continuous empirical distribution function fulfill condition (4). We first observe that for $n \in \mathbb{N}$ and $x \in T^n$, the functions $(h_n)_{n \in \mathbb{N}}$ satisfy
\[
(h_n(x))(t) \in [0, \frac{1}{n+1}] \quad \text{for } t \leq x_1,
\]
\[
(h_n(x))(t) \in [\frac{n}{n+1}, 1] \quad \text{for } t \geq x_n,
\]
\[
(h_n(x))(x_k) = \frac{k}{n+1} \quad \text{for } k \in [n],
\]
and are linear on each interval $[x_k, x_{k+1}], k \in \{1, \ldots, n-1\}$. We therefore have for $n \in \mathbb{N}$, $x \in T^n$, $k \in \{1, \ldots, n-1\}$, $t \in [x_k, x_{k+1}]$
\[
|(h_n(x))(t) - (h_n^*(x))(t)| \leq \frac{1}{n+1} \leq [h_n(x))(x_k) - (h_n^*(x))(x_k)] + [h_n(x))(x_{k+1}) - (h_n^*(x))(x_k)] + [h_n(x))(x_k) - (h_n^*(x))(x_k)] 
\]
\[
\leq \frac{1}{n+1} + \frac{k}{n+1} \leq 2n^{-1}.
\]
Moreover, we have for $t < x_1$
\[
|(h_n(x))(t) - (h_n^*(x))(t)| = |1 - (h_n(x))(t)| 
\]
\[
\leq |1 - (h_n(x))(x_1)| = \frac{1}{n+1}.
\]
As such, our construction satisfies (4) and is covered by the following proposition.

**Proposition 6.7.** Let $g \in C^1([0, 1], \mathbb{R})$, $x \in \mathbb{R}$, $F \in S_x$, and $h = (h_n)_{n \in \mathbb{N}}$ such that the assumptions of Lemma 6.6 are satisfied. Then
\[
\sqrt{n}(R_{g,x}(F_n^h) - R_{g,x}(F)) \Rightarrow g'(F(x))G_F(x)
\]
and
\[
g'(F(x))G_F(x) = N(0, (g'(F(x)))^2(F(x) - F(x)^2)).
\]

**Proof.** Lemma 6.6 ensures $\sqrt{n}(F_n^h - F) \Rightarrow G_F$ and thus Theorem 6.2 yields
\[
\sqrt{n}(R_{g,x}(F_n^h) - R_{g,x}(F)) \Rightarrow [R'_{g,x}(F)](G_F(x)).
\]
Due to Lemma 6.3 we further have
\[
[R'_{g,x}(F)](G_F(x)) = g'(F(x))G_F(x)
\]
\[
= \frac{g'(F(x))}{N(0, (g'(F(x)))^2(F(x) - F(x)^2))}.
\]

### 6.1 Componentwise Application to Multi-dimensional Random Variables

So far we have analyzed the behaviour of the Redistributor transformation using an empirical estimator generated with iid samples from some real-valued random variable. We could however also consider a vector-valued random
variable \( X \in \mathbb{R}^d \). Given samples \( X^i \sim X, \, i \in [n] \), we can consider the collection of scalar values \( (X^i_j)_{i \in [n], j \in [d]} \subseteq \mathbb{R} \) and take[10] the empirical cumulative distribution function \( h^*_{nd}((X^i_j)_{i \in [n], j \in [d]}) \) as if it was a vector of samples from a real-valued random variable. We note that
\[
(h^*_{nd}((X^i_j)_{i \in [n], j \in [d]}))(t) = \frac{1}{nd} \sum_{j \in [d]} \sum_{i \in [n]} 1_{X^i_j \leq t} = \frac{1}{d} \sum_{j \in [d]} h^*_{n}((X^j_j)_{i \in [n]}).
\]

Which means this would be the empirical cumulative distribution function of a scalar-valued random \( \bar{X} \) variable with cumulative distribution function \( \bar{F} = \frac{1}{d} \sum_{j \in [d]} F_j \) with \( F_j \) being the cumulative distribution function of \( X^j \in \mathbb{R} \). Similarly taking our empirical estimator to get \( \bar{F}^h_n = h^*_{n}((X^j_j)_{i \in [n], j \in [d]}) \) would yield an approximation, in the sense of Proposition 6.7 of the transformation \( \bar{R} \) which satisfies
\[
\bar{R}(\bar{X}) \overset{d}{=} Y,
\]

where \( Y \) is a random variable with cumulative distribution function \( g \). While the statistical implications of applying a thusly created transformation component-wise to a vector \( x \in \mathbb{R}^d \), is quite unclear, this type of use of the Redistributor transformation to, e.g. images, yields some appealing results as can be seen in the next section.

7 Use Cases

We demonstrate a range of use cases in the context of image processing and discuss potential further applications.

7.1 Color Correction

Poorly lit or over/under-exposed photos are quite common, leading photo editing software to incorporate automated tools for their correction. Typically, these reference-less tools adjust the entire image’s exposure or handle light and dark regions separately. However, Redistributor takes a different approach. It modifies the pixel values to match their distribution with that of a reference image. This reference image represents what the photo should look like had it been properly exposed. Notably, the process extends beyond simple exposure correction; it also addresses other color issues. In the Figure 5 example, each RGB channel of the source image is redistributed individually to match the distribution of the target image outperforming a standard auto-color tool available in Photopica.com software.

In Section 7.4 we describe more advanced approaches to redistribution that are also capable of correcting illumination, color temperature, and other visual aspects of the images.

In Section 8.1 we evaluate Redistributor on a dataset with available ground truth in order to objectively compare it’s performance to other color correction methods that use a reference image.

[10]To simplify notation we write \( [m] := \{1, \ldots, m\} \) for \( m \in \mathbb{N} \).
(a) Mosaic 1 – 50 × 50 mosaic made of 1000 px² tiles redistributed to a Gaussian with \( \mu = \) desired pixel value and \( \sigma = 0.05 \). The original image is overlaid over the mosaic with \( \alpha = 0.25 \) to regain high-frequency content.

(b) \( \alpha = 0.00, \sigma = 0.05 \)

(c) \( \alpha = 0.25, \sigma = 0.20 \)

(d) \( \alpha = 0.25, \sigma = 0.40 \)

Fig. 7: Mosaic effect achieved by tiling redistributed images from the LFW dataset [Huang et al., 2007].
of the mosaic can be easily modified using a parameter \( \sigma \) to resemble using only a single image. Furthermore, the granularity of each region, each of which is replaced by an image with similar mean color value as the original tile. Usually, mosaics are created by selecting the smaller images with fitting mean from a large collection. The resulting granularity is a function of both the size of the collection as well as the size of the tiles. Redistributor can instead transform any provided image to a Gaussian distribution with mean \( \mu \) matching the tile to replace. By doing so, one could even make a mosaic by repeatedly using only a single image. Furthermore, the granularity of the mosaic can be easily modified using a parameter \( \sigma \) of the target Gaussian distribution, either keeping or compressing the variability in an individual sub-image. Moreover, the original full-resolution image can be overlaid over the mosaic to regain high-frequency content, adding another parameter \( \alpha \) influencing the ratio between the mosaic and the original image it depicts. Examples of mosaics created with varying parameters are shown in Figure 7.

### 7.3 Making Photomosaics

As a further use case in image modification, Redistributor can easily be used to make photographic mosaics (photomosaics). These are images that have been divided into smaller, tiled regions, each of which is replaced by an image with similar mean color value as the original tile. Usually, mosaics are created by selecting the smaller images with fitting mean from a large collection. The resulting granularity is a function of both the size of the collection as well as the size of the tiles. Redistributor can instead transform any provided image to a Gaussian distribution with mean \( \mu \) matching the tile to replace. By doing so, one could even make a mosaic by repeatedly using only a single image. Furthermore, the granularity of the mosaic can be easily modified using a parameter \( \sigma \) of the target Gaussian distribution, either keeping or compressing the variability in an individual sub-image. Moreover, the original full-resolution image can be overlaid over the mosaic to regain high-frequency content, adding another parameter \( \alpha \) influencing the ratio between the mosaic and the original image it depicts. Examples of mosaics created with varying parameters are shown in Figure 7.

### 7.4 Applying the Transformation in Alternative Spaces

As opposed to the previous use cases, this subsection describes an approach where processing happens in spaces other than the RGB color space. The idea of using different color representation spaces for color mapping is well established, cf., e.g., Section 2.2.2 in [Faridul et al., 2016], where transformations in LMS cones space or (CIE)LAB space, among others, are reviewed. We present two useful extensions.

On the one hand, we consider application of Redistributor in Fourier space. More precisely, the amplitudes of an image’s channels’ Fourier coefficients are redistributed, while original phases are kept. Results are shown in Figure 8, where both combined or separate channel processing is demonstrated. Both cases adjust image sharpness.

On the other hand, we devise and examine advanced sequential transformations that let us consecutively match histograms of chosen channels within different color spaces. This approach produces promising results, especially when combined with masking and weighting of each distinct transformation for better and more intuitive control of the final result. To be more precise, by masking we mean applying Redistributor between pre-specified regions of source and target images. Masking can be used to enforce mapping of the colors between semantically corresponding regions while weighting certain transformations helps to reduce undesired artifacts caused by heavily-skewed distributions of certain channels. See Figure 9 for a visual aid explaining what we mean by weighting in this context. To produce the results presented in Section 8, we show in Figures 11 to 13 we iteratively applied a sequence of changes over the Saturation (in HSV color space), Lightness (in LAB), Red, Green, Blue (in RGB), and again Lightness (in LAB) channels. This approach is explainable, as each of the channels bears a specific meaning, and also traceable, as one can visualize the intermediate result after each transformation.

### 7.5 Image Data Augmentation

Redistributor may also be used for Image data augmentation in color space, as described in [Shorten and Khoshgoftaar, 2019], i.e. taking an image and producing a similar one by modifying the histograms of the RGB color channels. Figure 10 illustrates the example of a batch of 6 images, where out of each image 5 new ones have been created by channel-wise redistribution to match each of the other images. One can also create new versions of an image by redistributing them to some manually chosen distribution of interest, depending on the specifics of the intended application. Naturally, Redistributor could also be applied to any other type of data as long as it consists of some collection of scalar values. Of course, it is not always clear whether these augmentations are necessarily sensible for the intended task. While the benefits and drawbacks need to
be carefully considered based on the goals of the intended application. Redistributor always provides an efficient and flexible tool for systematic data augmentation.

7.6 Preprocessing for Machine Learning Tasks

Another potential application of this algorithm would be as a preprocessing step for machine learning tasks. It is designed to efficiently handle a large number of samples with a low memory footprint, and as it is implemented as a Scikit-learn transformer, it can be conveniently inserted in common machine learning pipelines. It always produces an invertible transformation, i.e. does not, in principle, reduce the input-output relationships which may be learned. In addition, Section 6 shows that given \( n \) iid samples of a CDF \( F_S \) and some specified target CDF \( F_T \), it is a consistent estimator of \( F_T^{-1} \circ F_S \), i.e. it can be expected to behave predictably on further samples of \( F_S \). Particularly when applied component-wise to high-dimensional data, e.g. images, it seems unlikely that it would improve things from a statistical perspective.

Anecdotally, we have observed various cases of improved performance on a number machine learning tasks using neural networks. The improvement is, however, not very consistent and often rather mild. Although, given the simple and inexpensive nature of this preprocessing step it would be quite unreasonable to expect a consistent significant improvement anyway. We abstain from presenting numerical examples for this use case, since this is not the focus of the paper and the black-box nature of neural networks means that a brief demonstration would require significant cherry picking.

7.7 Connection to Transport Based Signal Processing

Thus far we have been interested in the output we can obtain by applying the Redistributor transformation to some input of interest. To round out this section we will briefly mention a rather different usage of the basic idea behind the Redistributor, which has been put forward in [Park et al., 2015]. We give a somewhat simplified and slightly paraphrased description. Let \( f: \mathbb{R} \to \mathbb{R} \) be a continuous function which satisfies \( \|f\|_1 := \int_{\mathbb{R}} f(x)dx = 1 \), i.e. such that we can interpret it as a density function, and let \( F_T = \int_{-\infty}^x f(t)dt \) denote the corresponding CDF. Moreover, we fix some reference probability measure with continuous density function \( f_S \) and CDF \( F_S \). We can now consider the so-called Cumulative Distribution Transform \( \hat{f}: \mathbb{R} \to \mathbb{R} \) of \( f \), which can be written as

\[
\hat{f}(x) = \left(R(x) - x\right)\sqrt{f_s(x)},
\]

where \( R = F_T^{-1} \circ F_S \) is simply the Redistributor transformation. In this setting, one considers data \( (f_i)_{i \in \mathbb{N}} \) where each \( f_i \) can be represented suitably as a function. In [Park et al., 2015], a number of advantages of applying the CDT to each of the data points and working with the set \( (\hat{f}_i)_{i \in \mathbb{N}} \) are shown theoretically and demonstrated empirically. This has been developed into a number of interesting methods for signals processing, see [Rubaiyat et al., 2021].

8 Comparisons

We present two comparisons of Redistributor to existing methods. First, we consider a color correction task where we compare it to model-based methods using a dataset and evaluation criteria from [Faridul et al., 2016]. Second, we compare it to results obtained by a deep-learning approach to photorealistic style transfer [Luan et al., 2017], where they provide and make use of masks for the input and reference images, in order to match semantically corresponding areas of the respective images (e.g. sky, buildings, water). Due to its flexible implementation, Redistributor can conveniently take advantage of these masks as well.

8.1 Color Correction Comparison

For this comparison we use a dataset\(^{11}\) for ground-truth-based evaluation, first presented in [Faridul et al., 2016]. The idea is to capture an image of a scene with the desired illumination in order to create a ground truth, as well as a series of test images of the scene from the same viewpoint but with changes in imaging modalities, like shutter speed, illuminant, or white balance. The task is to transform these test images into something close to the ground truth by using a reference image of the same scene with the correct illumination but

\(^{11}\)Available at [https://sites.google.com/site/reviewcolormapping/](https://sites.google.com/site/reviewcolormapping/)
Fig. 11: Performance of Redistributor on the Color Mapping Dataset. Each of the three rows contains one example image, namely 4, 39, and 113, from the three image sets. Input images shown here represent the “middle difficulty” from each of the sets. The whole dataset is displayed in Figure 13 of [Faridul et al., 2016].

Fig. 12: Comparing Redistributor on the Color Mapping Dataset to Pitie [Pitié et al., 2007], Hacohen [HaCohen et al., 2011], Faridul [Faridul et al., 2013], Pouli [Pouli and Reinhard, 2011], and Reinhard [Reinhard et al., 2001]. The rows correspond to image sets 1–3. The x-axis in the first two rows represents the shutter speed, in the last row, the color temperature. Each box shows the median CIE $\Delta E_{00}$, and the 25th and 75th percentiles. The whiskers extend from the box by $1.5 \times IQR$. 
from a different viewpoint. The dataset is organized into three sets of images corresponding to different changes in imaging modalities. In the first one the shutter speed is varied from $1/10\,s$ to $1/3200\,s$, in the second one both shutter speed and illuminant is varied, in the third one the color temperature of the camera is varied from $4500\,K$ to $2500\,K$. In Figure 11 each row shows an example from one the three sets. The result of the transformation is then compared to the ground truth by taking the color difference in CIE $\Delta E00$ unit [Luo et al., 2001] for each pixel, which is then summarized via boxplots in Figure 12. When employing simple masks Redistributor outperforms the other methods across the board and still produces competitive results without masking.

8.2 Photorealistic Style Transfer Comparison

In this section we present a comparison to a deep learning approach [Luan et al., 2017], which aims for photorealistic style transfer, by attempting to suppress the distortion of the image content which is often produced by NST methods. By contrast, our method, as well as the other model-based methods considered in the previous subsection, guarantee that the structure of the image is kept, but may struggle to convincingly match the style of the reference image. We demonstrate that Redistributor is capable of yielding visually satisfying results on the images considered in [Luan et al., 2017]. In particular it is notably better at maintaining fine structures of the images, e.g. the tree in the image in row 5 and flower petals in the image in row 16. As demonstrated in rows 8 and 11 our method avoids introducing unnatural green resp. blue spot illuminations. Redistributor does however fall short of fully matching the reference style in row 10, where there is both a large difference in colors and content, i.e. image to be transformed contains mountains and trees, but the reference image does not.

9 Conclusion

We introduced Redistributor, a method and Python package for estimating and transforming empirical data distributions, and provided a range of applications in image and signal processing as well as machine learning.

Subsequently, we demonstrated its utility in various applications. In the context of image processing, it offers an powerful approach to color correction and photorealistic style transfer. It effectively changes an image’s color scheme using a reference image, yielding visually pleasing results while preserving the original image’s content. Unlike neural style transfer methods, Redistributor guarantees content fidelity and offers greater flexibility, explainability, and computational efficiency.

We also showed an application in image data augmentation, enabling the creation of variations of images for the purpose of expanding datasets for neural network training. Moreover, due to its implementation, Redistributor can be integrated as a preprocessing step into machine learning pipelines.

Redistributor outperformed other model-based methods in color correction tasks, demonstrating its effectiveness in a ground-truth-based evaluation. Additionally, we showcased...
its ability to achieve photorealistic style transfer with superior content preservation compared to deep learning methods.

In conclusion, Redistributor is a versatile and efficient tool for transforming data distributions, offering a wide range of applications in image processing, signal processing, machine learning, and beyond. Its strong theoretical foundation, along with its proven performance in various use cases, makes it a valuable addition to the toolbox of researchers and practitioners in these fields.

ACKNOWLEDGEMENT

Photos in Figure 5 by S. Markovicova. Images in Figure 6 by S. Oulashin, C. Maximilian, J. Plenio, V. Solomon. S. Weeteling, C. Gottardi, J. Street, and L. Pieters. Photo in Figure 7 by S. Mohal. Images in Figure 10 by E.S. Agbonyeme, L. Reis, S. Sokolov, Andrey K, K. Hliznitssova, and Taha. Free image editing software Photopea.com by L. Kutsik was used to preprocess and finalize multiple of the presented figures. We thank J. Schlüter for the initial inspiration, and J. Berner for many valuable discussions.

AUTHOR CONTRIBUTIONS

HP - conceptualization, investigation, methodology, implementation, experimentation, visualization, writing. DE - conceptualization, investigation, methodology, formal analysis, experimentation, writing. MD - formal analysis, writing, supervision. KDJ - investigation, experimentation, writing.

FUNDING

This work has been funded in part by Research Network Data Science @ Uni Vienna and in part by ETH Foundations of Data Science (ETH-FDS).

DATA AVAILABILITY

The datasets utilized in this study are available at https://sites.google.com/site/reviewcolormapping and https://github.com/luanfujun/deep-photo-styletransfer, and http://vis-www.cs.umass.edu/lfw/

CODE AVAILABILITY

Source code for the Redistributor package as well as the code for reproducing the figures used in this study are available at https://github.com/paloha/redistributor

CONFLICT OF INTEREST

The authors have no competing interests to declare that are relevant to the content of this manuscript.

REFERENCES

[Bartlett, 1947] Bartlett, M. S. (1947). The use of transformations. Biometrics, 3(1):39–52.
[Box and Cox, 1964] Box, G. E. P. and Cox, D. R. (1964). An analysis of transformations. Journal of the Royal Statistical Society: Series B (Methodological), 26(2):211–243.
[Chen, 2017] Chen, Y.-C. (2017). A tutorial on kernel density estimation and recent advances. Biostatistics & Epidemiology, 11:161–187.
[Devroye, 2006] Devroye, L. (2006). Nonuniform random variate generation. Handbook of operations research and management science, 13:93–121.
[Faridul et al., 2016] Faridul, H. S., Pouli, T., Chamarat, C., Stauder, J., Reinhard, E., Kuzovkin, D., and Tremeau, A. (2016). Colour mapping: A review of recent methods, extensions and applications. Computer Graphics Forum, 35(1):59–88.
[Faridul et al., 2013] Faridul, H. S., Stauder, J., Kervec, J., and Tremeau, A. (2013). Approximate cross channel color mapping from sparse color correspondences. In 2013 IEEE International Conference on Computer Vision Workshops, pages 860–867.
[HaCohen et al., 2011] HaCohen, Y., Shechtman, E., Goldman, D. B., and Lischinski, D. (2011). Non-rigid dense correspondence with applications for image enhancement. ACM Trans. Graph., 30(4).
[Huang et al., 2007] Huang, G. B., Ramesh, M., Berg, T., and Learned-Miller, E. (2007). Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical Report 07-49, University of Massachusetts, Amherst.
[Luan et al., 2017] Luan, F., Paris, S., Shechtman, E., and Bala, K. (2017). Deep photo style transfer. In 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 6997–7005, Los Alamitos, CA, IEEE Computer Society.
[Luo et al., 2001] Luo, M. R., Cui, G., and Rigg, B. (2001). The development of the cie 2000 colour-difference formula: Cie2000. Color Research & Application, 26(5):340–350.
[Musser, 1997] Musser, D. R. (1997). Introspective sorting and selection algorithms. Software: Practice and Experience, 27(6):983–993.
[Osborne, 2010] Osborne, J. (2010). Improving your data transformations: Applying the Box-Cox transformation. Practical Assessment, Research, and Evaluation, 15(12).
[Park et al., 2015] Park, S., Kolouri, S., Kundu, S., and Rohde, G. (2015). The cumulative distribution transform and linear pattern classification. Applied and Computational Harmonic Analysis, 45.
[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12:2825–2830.
[Peltier et al., 1998] Peltier, M. R., Wilcox, C. J., and Sharp, D. C. (1998). Technical note: Application of the Box-Cox data transformation to animal science experiments. Journal of Animal Science, 76(3):847–849.
[Peterson, 2021] Peterson, R. A. (2021). Finding Optimal Normalizing Transformations via bestNormalize. The R Journal, 13(1):310–329.
[Pitti et al., 2007] Pitié, F., Kokaram, A. C., and Dahyot, R. (2007). Automated colour grading using colour distribution transfer. Computer Vision and Image Understanding, 107(1):123–137. Special issue on color image processing.
[Pouli and Reinhard, 2011] Pouli, T. and Reinhard, E. (2011). Progressive color transfer for images of arbitrary dynamic range. Computers & Graphics, 35(6):67–80. Extended Papers from Non-Photorealistic Animation and Rendering (NPAR) 2010.
[PyTransKit Team, 2022] R Core Team (2022). R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria.
[Reinhard et al., 2001] Reinhard, E., Adhikhmin, M., Gooch, B., and Shirley, P. (2001). Color transfer between images. IEEE Computer Graphics and Applications, 21(5):34–41.
[Rubaiyat et al., 2021] Rubaiyat, A. H. M., Rabbi, M. S. E., Cattell, L., Yin, X., Li, S., Zhuang, Y., Rohde, G., Kolouri, S., and Park, S. (2021). PyTransKit. https://github.com/rostdlibs/PyTransKit.
[Shorten and Khoshgoftaar, 2019] Shorten, C. and Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning. Journal of big data, 6(1):1–48.
[Vaart, 1998] Vaart, A. W. v. d. (1998). Asymptotic Statistics. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge Univ.Press.
[Van der Walt et al., 2014] Van der Walt, S., Schönberger, J. L., Nunez-Iglesias, J., Boulogne, F., Warner, J. D., Yager, N., Gouillart, E., and Yu, T. (2014). scikit-image: image processing in python. PeerJ, 2:e453.