NA_DAtabase: Generator of Probabilistic Synthetic Geometrical Shape Dataset

Federico Volpini[1,*], Claudia Caudai[1,*], Giulio Del Corso[1,*], and Sara Colantonio[1]

[1]Institute of Information Science and Technologies (ISTI), National Research Council of Italy (CNR) - Pisa, Italy
[*]Share first co-authorship.

Abstract

In this technical report, we detail NA_DA (Not-A-DAtabase), an open-source software written in Python that generates datasets of regular two-dimensional geometric shapes based on probabilistic distributions (`https://github.com/GDelCorso/NA_DAtabase.git`). NA_DA comes with an intuitive GUI (Graphical User Interface) that allows users to define shapes, colors, and distributions of features of datasets consisting of image sets and CSV files containing metadata for each element. These databases can be saved to provide a unique identifier of the dataset, allowing perfect reproducibility or easy modification of the dataset using the GUI or directly by calling the generator class. Therefore, NA_DA is a tool to help and support the investigation of trustworthiness, overconfidence, uncertainty, and computation time of machine learning and deep learning models.

Contents

1 Introduction

With the increasingly rapid and growing development of Deep Learning architectures, the need to find methods to verify the level of Trustworthiness has also grown, especially about the need to estimate the reliability of predictions in several areas, such as medicine, biology, automation, and bioengineering [1, 2, 3, 4, 5, 6, 7]. In several application areas, artificial neural networks suffer from major drawbacks due to problems with the training sets they are trained on, which may be small, noisy, unbalanced, or contain biases and outliers.

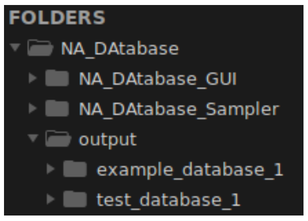
For this reason, it has become increasingly important to focus on the search for models that are more robust and reliable, and that allow us to better evaluate certain aspects such as reproducibility, interpretability, and uncertainty quan-

tification [8, 9].
Given that the behavior of an artificial neural network is strongly dependent on the characteristics of the training set, it is important to be able to test its overconfidence and understand how much the behavior of the network depends on the characteristics of the training set or on the properties of the network itself.
To study the behavior of artificial neural networks, there are several well-known **synthetic databases** in the literature, such as: DSprite [10], 2D-geometric shapes [11], or XYsquares [12]. However, these synthetic databases are suboptimal for the study of probabilistic relationships between input (i.e., images) and model output, particularly for the development of methods and software for quantifying uncertainties. Indeed, each database implicitly assumes a multivariate statistical relationship between the parameters characterising the image (e.g., rotation, magnification, greyscale, presence of shapes, etc.). For example, a database containing all the rotations (0 to 360°) of a geometric shape and a set of possible colour variants (e.g., red, blue and yellow) is equivalent to assuming a uniform distribution of rotation and colour and imposing independence between the two variables. However, the presence of correlations between input variables and heavy-tailed or asymmetric marginal distributions play a key role in the analysis of probabilistic networks and, in particular, their ability to detect out-of-distribution or unreliable values in the test data.
The use of **non-synthetic public databases**, on the other hand, provides more realistic distributions, but it is almost impossible to obtain image-level annotation of the properties described (e.g., rotation, colour variations, etc.). Therefore, these databases can only serve as a confirmation of the generalisation capabilities of the models under investigation by providing little insight into the details of the probabilistic method under consideration. Examples of synthetic dataset include MNIST [13], Fashion-MNIST [14], or Natural-Sprites/Kitti-masks [15].

Formally then, this paper provides a novel tool for constructing a synthetic database (Not-A-DAtabase, **NA_DAtabase**), while maintaining complete control over the multivariate distributions of the parameters characterising the images. NA_DAtabase Tool (hereafter NA_DA) is a Python-based software that generates regular geometric shape databases based on probabilistic distributions (`https://github.com/GDelCorso/NA_DAtabase.git`). The databases consist of image records and CSV files containing metadata for each entry.
NA_DA allows the definition of shapes, colors and distributions through its user-friendly GUI (Graphical User Interface). The distributions of the various characteristics can be managed and modified as needed to allow the study of the behavior of architectures by focusing on specific aspects. This generator also provides the possibility of adding different types of noise to the generated databases, and allows differentiable transitions from polygonal figures to circles through a curvature management function. Furthermore, NA_DA allows the management of the accuracy of the attribution label, so that it can be used as a tool to study the behavior of architectures during the classification task.

2 The GUI and its elements

The software comes with a graphical user interface (GUI) that helps the user to easily define the dataset properties and wraps the entire dataset definition and image generation.



```
FOLDERS
▾ NA_DAtabase
  ▸ NA_DAtabase_GUI
  ▸ NA_DAtabase_Sampler
  ▾ output
    ▸ example_database_1
    ▸ test_database_1
```

Figure 1: Folder structure.

2.1 On startup

Upon startup, the application will prompt you to either create a new database or load an existing one (Fig. 2). Saved databases are stored in the folder named `output`, located within the main directory (Fig. 1).

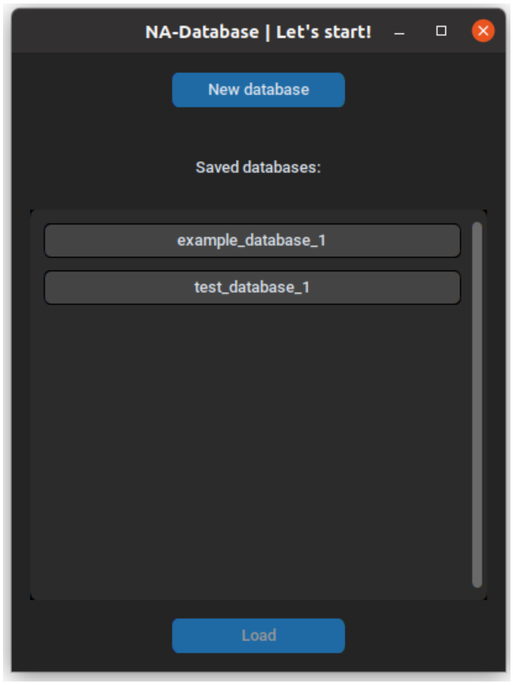


Figure 2: Startup: create a new database or load an existing one.

If the user chooses to create a new database, he/she must specify the desired name (Fig. 3). If a database with the same name already exists, a warning message will be displayed and you will be asked to provide a different name.

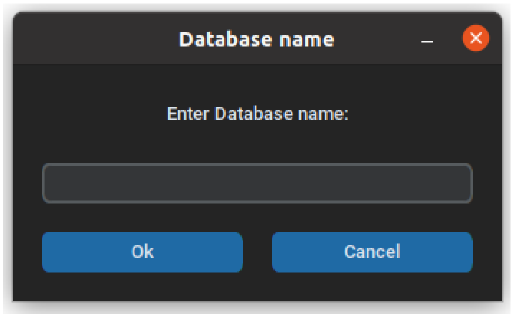


Figure 3: Database name prompt.

Once a unique name has been defined (or an existing database has been loaded), the main interface window will open (Fig. 4). The window is divided into two parts. The upper part contains three buttons and a messaging box. The lower part features a tabbed interface (5 tabs with their respective content).
The buttons in the upper part are:

- `Tab info`: displays a concise informational overview of the currently active tab.
- `Save Database`: saves partial CSV files, one per tab.
- `Generate Images`: merges the partial CSV files into a combined one and generates the image dataset. *The button will remain disabled until the data has been saved for the first time.*

The Message box will display last success/error message that occurred.

2.2 The tabbed interface

The tabbed interface (Fig. 4) contains 5 tabs with their relative content:

- **Shapes and Colors**: in this tab you can create shapes/colors combinations, providing the probability of each of them.
- **Sampler Properties**: This tab contains the properties of the dataset and optionally the addresses for classification ground truth.
- **Uncertainties**: This tab contains the marginal distribution of each continuous random variable and classification noise for every couple of shapes and colors.
- **Continuous distribution**: Provides the marginal distribution of each continuous random variable.
- **Multivariate distribution**: In this tab it is possible to fill the correlation matrix in terms of probability between continuous variables for every couple of shapes and colors.

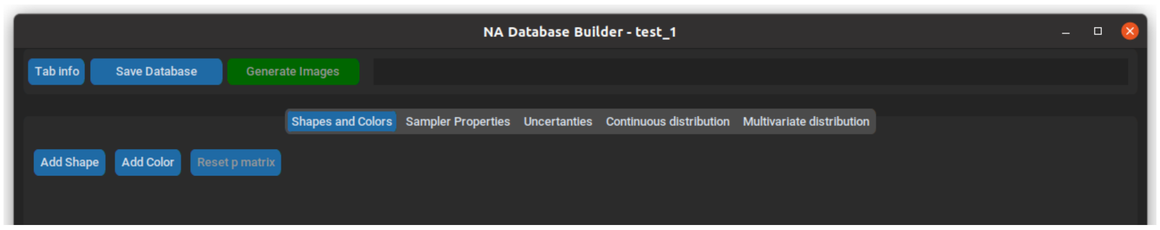


Figure 4: Main GUI panel: the tabbed interface allows to switch between different tabs to define the database properties.

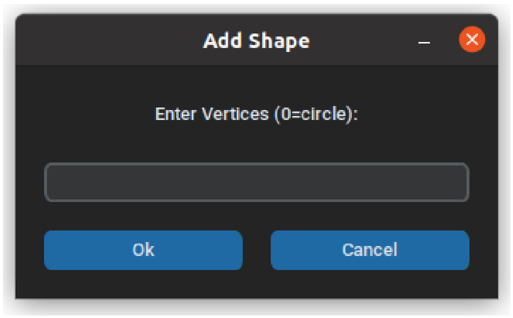


Figure 5: Add Shape Color Prompt. Valid values include 0 (circle) or a number of vertices greater than or equal to 2.

2.2.1 Shapes and Colors

In this section it's possible to create several combinations of shapes and colors, organized in an intuitive matrix. The colors will populate the rows and the shapes will be inserted into the columns. Shapes are determined by the *number of vertices*, and colors by their *hexadecimal value*.
By clicking on the `Add Shape/Add Color` button, you can set these values and add a shape/color into the matrix.
For shapes, the minimum value is 3 (a triangle), except for zero which defines a circle (Fig. 5).
You can pick a color using the color wheel (adjusting the brightness with the slider) or by typing the hex code into the designated field (Fig. 6).
The probability matrix (Fig. 8) is highly customizable at different level:

- **Detail level**: Modify the probability of each individual shape-color pair;
- **Shape level**: Modify the probability of a single shape, regardless of its associated color. The probabilities of each shape-color pair are updated accordingly to ensure that they sum to 1.
- **Color level**: Modify the probability of a single color, regardless of its associated shape. The probabilities of each shape-color pair are updated accordingly to ensure that they sum to 1.

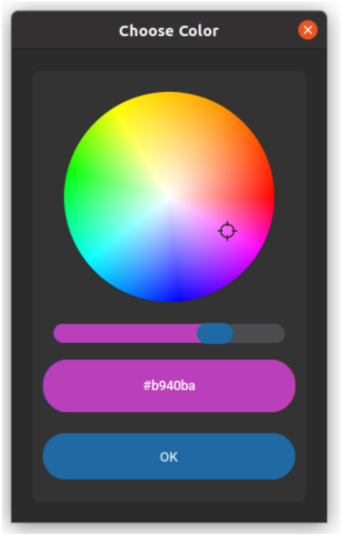


Figure 6: Add Color Prompt, values can be selected using the color wheel or by entering the hex code directly.

- **Automatic locking**: Changes lock dependent probabilities to ensure matrix consistency.
- **Custom locking**: Use the switches to lock/unlock cells/rows/columns. Locked

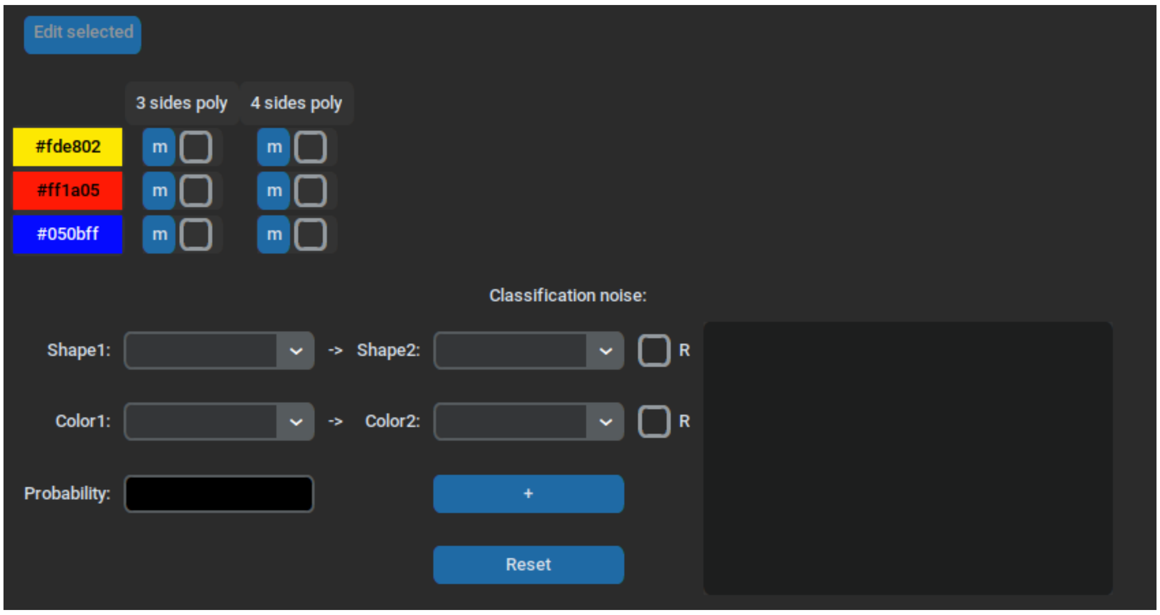
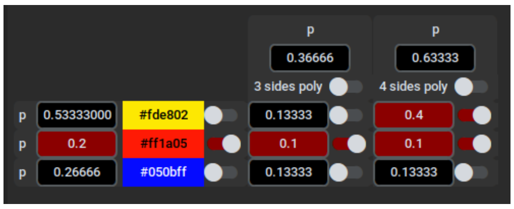


Figure 7: Uncertainties panel.

cells cannot be updated until they are unlocked, allowing the user to fine-tune the desired probabilities.



			p	p
			0.36666	0.63333
			3 sides poly	4 sides poly
p	0.53333000	#fde802	0.13333	0.4
p	0.2	#ff1a05	0.1	0.1
p	0.26666	#050bff	0.13333	0.13333

Figure 8: Shapes and Color probability matrix example. Red cells correspond to locked values. Switches allow to lock or unlock single cells or entire rows/columns.

At any time, by clicking the `Reset p matrix` button, the probability values of the matrix can be reset redistributing uniformly the probabilities such that the sum of each row and column is exactly 1 (i.e., a probability of 100%).

2.2.2 Sampler properties

In this tab it's possible to define the sampler and database properties, including the addresses for classification ground truth. Let's examine these properties:

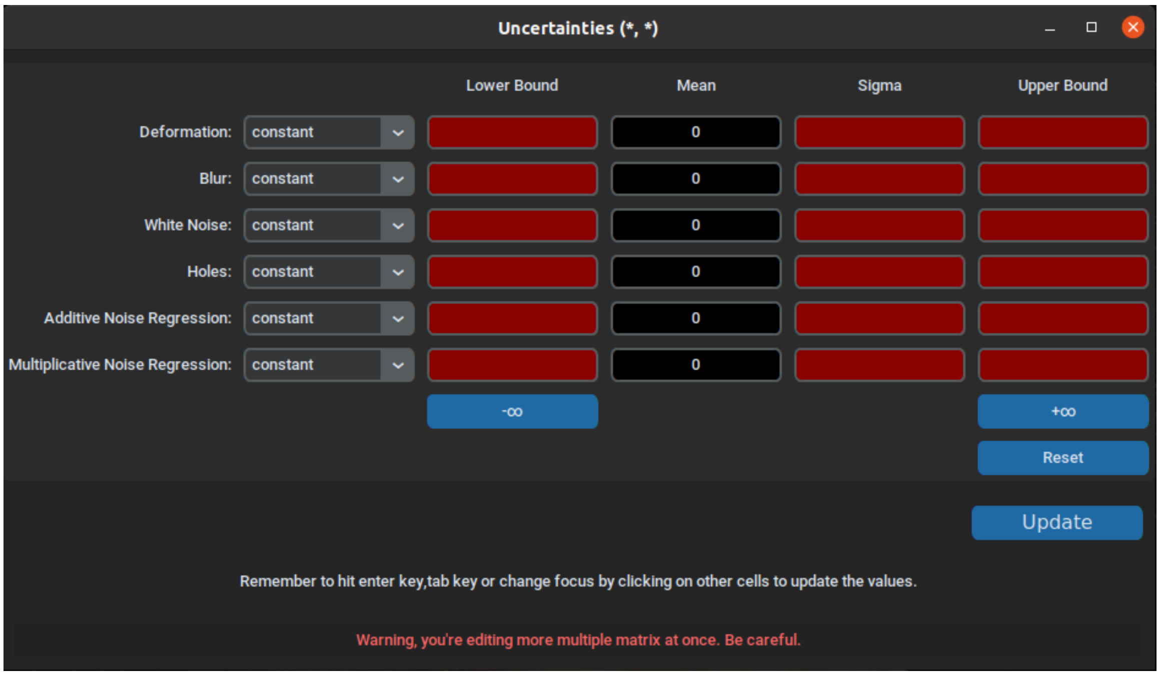
- **Dataset size**: the maximum number of generated images. The number of images to be generated is divided among the classes defined in the shape and color probability matrix. In this way, each shape and color is represented by a number of images exactly equal to the chosen probability.

- **Sampling strategy**: define the technique for sampling a probability distribution. Possible values are:

  – **MC**: Monte Carlo, the standard random sampling strategy characterized by a given random seed for reproducibility.

  – **LHC**: Latin HyperCube Sampling, an efficient sampling strategy that drastically smoothes the empirical sampling distribution.

  – **LDS**: (Sobol') Low Discrepancy Sequence, an efficient pseudo-random


Uncertainties (*, *)

		Lower Bound	Mean	Sigma	Upper Bound
Deformation:	constant		0		
Blur:	constant		0		
White Noise:	constant		0		
Holes:	constant		0		
Additive Noise Regression:	constant		0		
Multiplicative Noise Regression:	constant		0		

Remember to hit enter key,tab key or change focus by clicking on other cells to update the values.

Warning, you're editing more multiple matrix at once. Be careful.

Figure 9: The Uncertainties Popup panel for a given subset of shapes/colors.

sampling strategy that further improves sampling smoothness for low-dimensional parameter spaces.

- **Random seed**: The number used to initialize the random generator. It ensures full reproducibility of data generation.
- **Resolution**: The resolution in pixels of the images sides (the canvas is a square).
- **Background color**: The color in Hexadecimal value of the images background.
- **Allow out of border**: If you select this option, shapes can extend beyond the image borders, but their center will always remain inside. (*Unless this option is selected, Monte Carlo sampling is the sole permitted method.*)
- **Correlation**: The nonparametric correlation measure. It's a read only value and is set on `Spearman` monotonic correlation.

It's also possible to redefine the mappings between shapes and colors within the Correct Classes panel. In fact, even if the database itself contains all the information to allow users to define their own correct class, the GUI allows the correct class to be selected using internal logical operators. As a result, the final database will contain a column with a ready-to-use classification label to speed model testing. More complex labels (or multi-label alternatives) can be defined by users directly in the final database.

2.3 Uncertainties

The Uncertainty panel allows the user to determine the marginal distribution of any continuous random variable associated with any specific shape-color combination (Fig. 7).
Classification noise can be set globally for all shapes and color pairs. It quantifies the uncertainty in predicting a class for a given instance. This is also known as labeling noise, because it represents the risk of incorrectly labeled data. By choosing an original shape/color (Shape 1/Color

1), a target shape/color (Shape 2/Color 2), and a probability p, each instance of the original Shape 1/Color 1 is given a wrong Shape 2/Color 2 label with probability p. If one of the fields is empty, only the other field is considered (i.e. Red/- to Blue/-, $p = 0.3$ maps 30% of red shapes to blue ones, regardless of shape). Conversely, use the randomize (r) switch to randomly reshuffle to any possible shape or color.
This tab also allows the user to manipulate noise uncertainty distributions. These distributions affect the `Deformation` (from original shape to circle), `Blur`, amount of `White Noise`, presence of `Holes`, and `Additive Noise` / `Multiplicative Noise` (which affect the regression values). Users can adjust the noise uncertainties for shape-color pairs, either one at a time (using the `m` button) or for multiple pairs at once (by selecting them and clicking the `Edit selected` button). In both cases, a top panel will pop up (Fig. 9). In the Popup panel , for each variable, it is possible to specify a distribution of different kinds of uncertainties/noises and the corresponding values of the lower bound, upper bound, mean and standard deviation (sigma). Possible distributions are:

- **Constant:** mean;
- **Uniform:** lower, higher bounds;
- **Gaussian:** mean, standard deviation;
- **Truncated Gaussian:** lower, higher bounds, mean, standard deviation.

2.4 Continuous distribution

In this tab, as with continuous random variables, we can characterize each variable through a probability distribution and its associated values (Fig. 10). Following the previous parameter, the possible continuous distributions can be defined as Constant, Uniform, Gaussian or Truncated Gaussian. The selected distributions affect each color/shaped combination. In particular, the center position (described by its x/y relative values), the radius and the rotation (in degrees).

2.5 Multivariate distribution

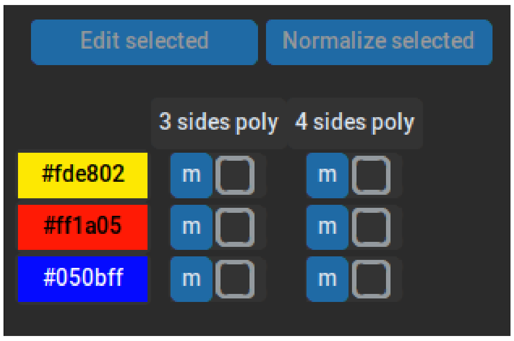
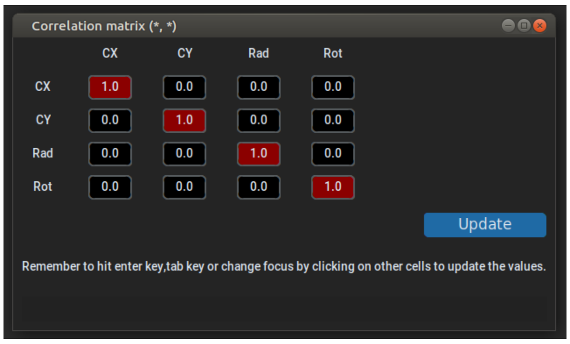


Figure 11: Multivariate distribution matrix.

To modify the relationship between the continuous variables introduced in the Continuous Distribution tab, we can act on the multivariate relationship between the variables (Fig. 11). Following Sklar's theorem and the copula formalism, for each shape/color combination we can define the corresponding Spearman correlation values (used to define the Gaussian copula). For details on copula formalism and Sklar's theorem, see [16]. Users can modify the correlation matrix for individual shape-color pairs using the `m` button. To adjust multiple pairs at once, select them and click the Edit Selected button.

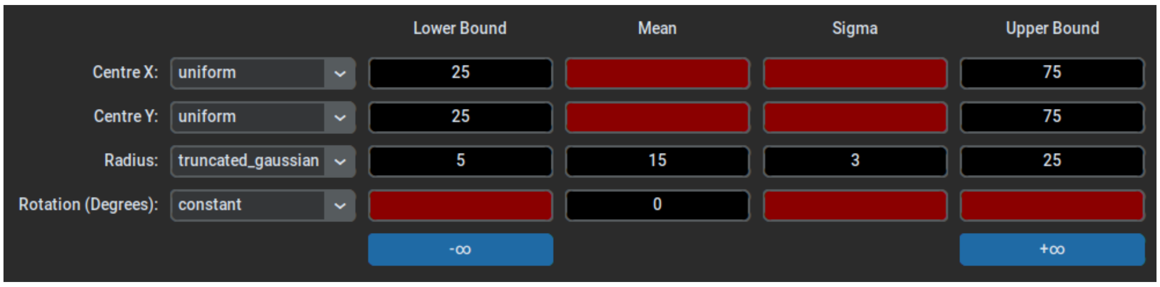

Correlation matrix (*, *)

	CX	CY	Rad	Rot
CX	1.0	0.0	0.0	0.0
CY	0.0	1.0	0.0	0.0
Rad	0.0	0.0	1.0	0.0
Rot	0.0	0.0	0.0	1.0

Remember to hit enter key,tab key or change focus by clicking on other cells to update the values.

Figure 12: Spearman Correlation Matrix.

In both cases the correlation matrix will pop up (Fig. 12). Values showing `*` imply that some color/shape couples have different values; modifying these values in the matrix updates each couple to the provided value. Values must be floating-point number between -1 and 1, including -1 and 1 themselves. *The correlation matrix must be positive definite, a warning is given if*



		Lower Bound	Mean	Sigma	Upper Bound
Centre X:	uniform	25	🟥	🟥	75
Centre Y:	uniform	25	🟥	🟥	75
Radius:	truncated_gaussian	5	15	3	25
Rotation (Degrees):	constant	🟥	0	🟥	🟥
		-∞			+∞

Figure 10: Continuous parameter distribution. This applies to any color/shape combination. By acting on the correlation matrix, the multivariate distribution can be controlled.

this condition is not met. It's also possible to normalize one or more matrices by selecting them and clicking the "Normalize Selected" button to ensure positive definiteness.

3 Code review

NA_DA is a Python software application entirely developed for the purpose of constructing image databases. Its core is built upon a robust infrastructure composed of fundamental numerical libraries such as *NumPy* and *Pandas*, providing a solid foundation for data manipulation and analysis.
In order to meet specific requirements of the application domain, a custom library has been developed, specifically designed for shape drawing.

3.1 The GUI

The Graphical User Interface (GUI) has been developed leveraging the flexibility of *CustomTkinter* [17], a third-party library that enables the creation of modern and highly adaptable graphical interfaces. The GUI was designed using an object-oriented approach to improve code readability and extensibility. Classes were distributed across multiple files to enhance modularity and maintainability.
Each tab within the application is controlled by a dedicated class, promoting better organization. To further improve flexibility, dependency injection was implemented. This ensures that class dependencies are explicitly managed, reducing tight coupling.

Additionally, helper classes were introduced to encapsulate common functionalities, reducing code redundancy.

Finally, the Color Picker component was extended to allow direct input of hexadecimal color values, offering users more precise control over color selection.

3.2 Sampling

The Sampling strategies are included as standalone class (NA_DA package). Initializing the class `NA_DA.random_sample(···)` requires the dataframes that describe the properties of the dataset. The dataframes are those generated by the GUI, but the class can be initialized using existing dataframes. This allows to modify the *.csv* directly with ad hoc software without the need of the graphical interface.

Once the class is initialized, the whole process can be called using the method `autoprocess()`. This method imports the data from the *.csv* files, defines the multivariate distribution according to Sklar's theorem and the Spearman correlation matrix using the OpenTURNS Python library [18], and then generates a `combined_dataframe.csv`. This data frame is organized into rows, each corresponding to an image with all the features (image properties, figure features, uncertainties, noise, etc.) needed to generate the images.

3.3 Shape Generator

The process of synthesizing images, starting from the structured CSV data generated by the sampler, is orchestrated by a pipeline composed of four interdependent classes. Parallel execution, enabled by a dedicated multiprocessing library, leverages available hardware resources to significantly accelerate the generation of visual output, especially on multi-core systems.
Here's the classes involved:

- **MorphShapes_DB_Builder**: This class serves as the entry point for the image generation process. It is invoked by the sampler's `generate_images()` method and takes as input the path to the CSV file `combined_dataframe.csv`, which is also generated by the sampler. The class additionally accepts parameters such as a GUI instance for displaying progress, if applicable, and a flag to enable parallel processing.

- **Main_Surface**: This class defines a two-dimensional canvas on which geometric shapes can be drawn. It provides an interface to configure the visual properties of the surface, such as dimensions, background color, and an optional background image.

  The class additionally provides the following methods:

  – `drawShape`: Accepts an instance of a Shape class, representing the geometric shape to be drawn. An optional boolean parameter specifies whether to enable anti-aliasing, an algorithm that improves the visual quality of shape edges by smoothing them to avoid the 'jagged' effect. Anti-aliasing is enabled by default.

  – `blur`: Applies a Gaussian blur to the entire surface, with a blur radius determined by the power argument (default value: 2). Higher power values result in a stronger blur effect.

  – `addNoise`: Applies a weighted Gaussian noise to the entire surface. the noise_power argument is the alpha channel of the noise and must be a value between 0 and 1 (default value: 0.5)

  – `emmental`: Creates holes on the main canvas. It takes the desired number of holes (default: 2) and a seed value to initialize the random number generator. This seed ensures consistent hole placement for repeatable results.

  – `show`: Displays the result on screen.

  – `save`: Saves the result to a specified file (a filename must be provided).

- **PIL_Drawing**: To enhance the quality of graphical representations, particularly for circular shapes, we replaced OpenCV's rendering engine with Pillow. OpenCV exhibited significant rendering issues with circular shapes at low resolutions (Fig. 13). This wrapper class re-implements OpenCV's image generation methods, leveraging Pillow's graphics engine. As a result, we achieved a substantial improvement in graphical output.

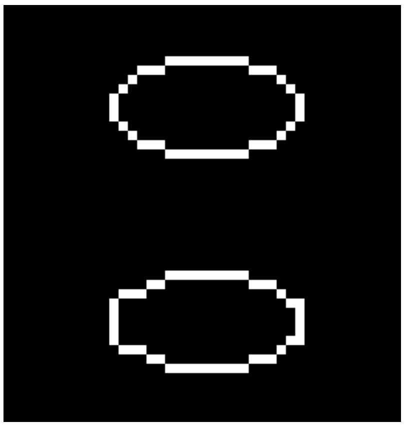


Figure 13: Ellipse (major 10px, minor 5px) from Pillow (top) and OpenCV (bottom).

- **Shape**: This class models a set of regular geometric shapes, specifically polygons inscribed within a circle. Each shape is defined by its center, radius, and the number of its sides (if the number of sides is zero, the shape is a circle).

Shapes can be personalized by adjusting their color, orientation, and morphing parameter. Morphing deforms the shape from its original state towards a circle. A morph value of 100% results in a shape with the same area as its enclosing circle.

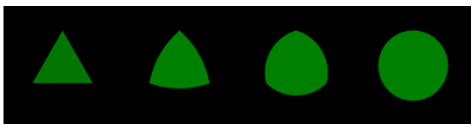


Figure 14: Triangle Morphing (from left to right morph percentage: 0%, 30%, 70%, 100%).

To get a continuous (and differentiable) deformation between the original shape and a circle, we defined $d_f \in [0, 100]$ as the deformation parameter. $d_f = 0$ corresponds to the original shape, while $d_f = 100$ leads to a complete deformation (i.e., a circle).

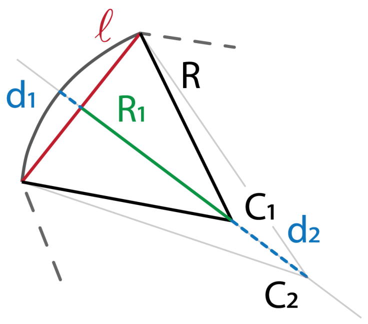


Figure 15: Deformation formula.

According to Fig. 15, for each side of the shape (of length ℓ) inscribed in the circumference of radius R and center C_1, we want to define a center C_2 such that the side is replaced by the arc of circumference with center C_2 passing through the two vertices of the original side. The center C_2 is defined by its distance d_2 from the original center C_1. If $d_2 \to \infty$, the shape is the original, while if $d_2 = 0$, $C_1 \equiv C_2$ and the shape is deformed to a circle. The formula for the derivation of d_2 is the following:

$$\begin{cases} \ell & = R \cdot \sqrt{2 - 2\cos(360/n)} \\ R_1 & = \sqrt{R^2 - (\ell/2)^2} \\ d_1 & = \frac{d_f}{100} \cdot (R - R_1) \\ d_2 & = \frac{(\ell/2)^2 - 2d_1 R_1 - d_1^2}{2d_1} \end{cases} \tag{1}$$

which leads to a differentiable deformation from the original shape to the circumscribed circumference.

References

[1] Ahmed Shihab Albahri, Ali Mohammed, Mohammed Abdulraheem Fadhel, Alhamzah Alnoor, Noor S. Baqer, Laith Alzubaidi, Osamah Shihab Albahri, Abdullah Hussein Alamoodi, Jinshuai Bai, Asma Salhi, José I. Santamaría, Chun Ouyang, Ashish Gupta, Yuantong Gu, and Muhammet Deveci. A systematic review of trustworthy and explainable artificial intelligence in healthcare: Assessment of quality, bias risk, and data fusion. *Inf. Fusion*, 96:156–191, 2023.

[2] Claudia Caudai, Antonella Galizia, Filippo Geraci, Loredana Le Pera, Veronica Morea, Emanuele Salerno, Allegra Via, and Teresa Colombo. Ai applications in functional genomics. *Computational and Structural Biotechnology Journal*, 19:5762 – 5790, 2021.

[3] Giulio Del Corso, Roberto Verzicco, and Francesco Viola. Sensitivity analysis of an electrophysiology model for the left ventricle. *Journal of The Royal Society Interface*, 17(171):20200532, 2020.

[4] Giulio Del Corso, Eva Pachetti, Rossana Buongiorno, Ana Carolina Rodrigues, Danila Germanese, M Antonietta Pascali, José Almeida, Nuno Rodrigues, Manolis Tsiknakis, Nickolas Papanikolaou, et al. Radiomics-based reliable predictions of side effects after radiotherapy for prostate cancer. In *2024 IEEE International Symposium on Biomedical Imaging (ISBI)*, pages 1–4. IEEE, 2024.

[5] Gianluca Carloni, Eva Pachetti, and Sara Colantonio. Causality-driven one-shot learning for prostate cancer grading from mri. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 2616–2624, 2023.

[6] Giulio Del Corso, Danila Germanese, Maria Antonietta Pascali, Serena Bardelli, Armando Cuttano, Fabrizia Festante, Andrea Guzzetta, Lucia Rocchitelli, and Sara Colantonio. Facial landmark identification and data preparation can significantly improve the extraction of newborns' facial features. In *2024 IEEE 18th International Conference on Automatic Face and Gesture Recognition (FG)*, pages 1–7. IEEE, 2024.

[7] Giulio Del Corso, Danila Germanese, Claudia Caudai, Giada Anastasi, Paolo Belli, Alessia Formica, Alberto Nicolucci, Simone Palma, Maria Antonietta Pascali, Stefania Pieroni, et al. Adaptive machine learning approach for importance evaluation of multimodal breast cancer radiomic features. *Journal of Imaging Informatics in Medicine*, pages 1–10, 2024.

[8] Wenchong He and Zhe Jiang. A survey on uncertainty quantification methods for deep neural networks: An uncertainty source perspective. *ArXiv*, abs/2302.13425, 2023.

[9] Eyke Hüllermeier and Willem Waegeman. Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods. *Machine learning*, 110(3):457–506, 2021.

[10] Loic Matthey, Irina Higgins, Demis Hassabis, and Alexander Lerchner. dsprites: Disentanglement testing sprites dataset. https://github.com/deepmind/dsprites-dataset/, 2017.

[11] Anas El Korchi and Youssef Ghanou. 2d geometric shapes dataset – for machine learning and pattern recognition. *Data in Brief*, 32, 2020.

[12] Nathan Michlo, Richard Klein, and Steven James. Overlooked implications of the reconstruction loss for vae disentanglement. *arXiv preprint arXiv:2202.13341*, 2022.

[13] Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29:141–142, 2012.

[14] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *ArXiv*, abs/1708.07747, 2017.

[15] David Klindt, Lukas Schott, Yash Sharma, Ivan Ustyuzhaninov, Wieland Brendel, Matthias Bethge, and Dylan Paiton. Towards nonlinear disentanglement in natural data with temporal sparse coding. *arXiv preprint arXiv:2007.10930*, 2020.

[16] Giulio DEL CORSO et al. Uncertainty analysis of biological systems: towards a digital twin of the human heart. 2022.

[17] Tom Schimansky. Customtkinter, a modern and customizable python ui-library based on tkinter. `https://customtkinter.tomschimansky.com/`, 2024.

[18] Michaël Baudin, Anne Dutfoy, Bertrand Iooss, and Anne-Laure Popelin. Open turns: An industrial software for uncertainty quantification in simulation. *arXiv preprint arXiv:1501.05242*, 2015.