Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

TINTOlib: A Python library for transforming tabular data into synthetic images for deep neural networks

Jiayun Liu [a,*], David González-Fernández [a], Manuel Castillo-Cara [b,c], Raúl García-Castro [a]

[a] Universidad Politécnica de Madrid, Madrid, Spain
[b] Universidad Nacional de Educación a Distancia, Madrid, Spain
[c] Instituto de Investigación Científica, Universidad de Lima, Lima, Peru

ARTICLE INFO

Dataset link: GitHub, Zenodo, Github, Zenodo, https://oeg-upm.github.io/TINTOlib/binary.html

Keywords:
Hybrid neural networks
Synthetic images
TINTOlib
Tabular-to-image
Tabular2image

ABSTRACT

Transforming tabular data into synthetic images enables the application of vision-based deep learning models – such as Convolutional Neural Networks and Vision Transformers – to non-visual tasks. This paper presents TINTOlib, the first Python library to unify a diverse set of tabular data into synthetic image transformation methods into a cohesive, extensible framework. TINTOlib unifies parametric and non-parametric tabular to synthetic image methods within a consistent interface, lowering the barrier to apply, compare, and extend these techniques. The generated images can be directly used with vision models or integrated into Hybrid Neural Networks that combine visual and tabular branches. By addressing reproducibility, scalability, and modularity, the library simplifies experimentation and deployment of deep learning pipelines on tabular data. Illustrative results show that the use of synthetic images can achieve competitive or superior performance compared to state-of-the-art classical models in both regression and classification tasks, with outcomes varying across transformation techniques and architectural backbones. This underscores the utility of TINTOlib in bridging tabular data with vision-based deep learning via synthetic image representations.

Code metadata

Current code version	v1.0.6.1
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-25-00457
Permanent link to Reproducible Capsule	https://github.com/oeg-upm/TINTOlib
Legal Code License	Apache License. 2.0
Code versioning system used	git
Software code languages, tools, and services used	Python
Compilation requirements, operating environments & dependencies	scipy, sklearn, pandas, matplotlib, argparse, numpy, math, tqdm, seaborn, bitstring, opencv-python, pi4py
If available Link to developer documentation/manual	https://tintolib.readthedocs.io/en/latest/
Support email for questions	manuelcastilllo@dia.uned.es

1. Motivation and significance

Tabular data are widely used in areas such as healthcare, finance, and manufacturing. However, the heterogeneous and non-spatial nature of tabular data often makes traditional ensemble methods more effective, leaving the full potential of deep neural networks underutilized for structured data analysis [1,2]. In contrast, Convolutional Neural Networks (CNNs) and Vision Transformers (ViTs) have demonstrated remarkable performance on image data, where spatial relationships between features are explicit. This gap has motivated the development of methods that transform tabular data into synthetic images [3], thereby enabling structured data to benefit from powerful vision-based architectures [4,5]. A recent survey on representation learning for tabular data [6] situates tabular to synthetic image methods within the broader field. However, implementations remain fragmented across

* Corresponding author.
E-mail addresses: jiayun.liu@upm.es (J. Liu), david.gonzalezf@alumnos.upm.es (D. González-Fernández), manuelcastilllo@dia.uned.es (M. Castillo-Cara), r.garcia@upm.es (R. García-Castro).

https://doi.org/10.1016/j.softx.2025.102444
Received 1 July 2025; Received in revised form 27 October 2025; Accepted 8 November 2025
Available online 26 November 2025

Table 1
Implemented transformation methods in TINTOlib, with their respective modules and classifications.

Method	Module	Class	Type
IGTD	`TINTOlib.igtd`	`IGTD`	Parametric
REFINED	`TINTOlib.refined`	`REFINED`	Parametric
TINTO	`TINTOlib.tinto`	`TINTO`	Parametric
BarGraph	`TINTOlib.barGraph`	`BarGraph`	Non-Parametric
BIE	`TINTOlib.BIE`	`BIE`	Non-Parametric
Combination	`TINTOlib.combination`	`Combination`	Non-Parametric
DistanceMatrix	`TINTOlib.distanceMatrix`	`DistanceMatrix`	Non-Parametric
FeatureWrap	`TINTOlib.featureWrap`	`FeatureWrap`	Non-Parametric
SuperTML	`TINTOlib.supertml`	`SuperTML`	Non-Parametric

individual works and lack a reusable and standardized form, further motivating the need for unified tooling.

Tabular data INTO synthetic images library (TINTOlib) addresses this gap as the first Python library that unifies state-of-the-art tabular to synthetic image transformation methods under a single, extensible framework. It implements both parametric approaches, such as Tabular data INTO synthetic images (TINTO) [7], Image Generator for Tabular Data (IGTD) [8], and REpresentation of Features as Images with NEighborhood Dependencies (REFINED) [9], which learn optimized spatial layouts, and non-parametric approaches, such as BarGraph, DistanceMatrix, Combination [10], SuperTML [11], FeatureWrap [12], and Binary Image Encoding (BIE) [13], which directly map features into visual formats. All methods follow a consistent `fit/transform` interface, designed to be easily used as input for deep learning pipelines.

Designed for flexibility and reproducibility, TINTOlib enables researchers to generate synthetic images from tabular data and to apply them directly to CNNs, ViTs, and Hybrid Neural Networks (HyNNs) [14], which combine visual and tabular processing branches. By resolving limitations such as uncontrolled randomness and lack of modularity in existing implementations, the library reduces the entry barrier for exploring vision-based deep learning on structured data and promotes reproducible research.

TINTOlib is open source (Apache License 2.0) and is publicly available on GitHub,[1] PyPI, and Zenodo [15]. The installation details and software requirements are provided in Appendix. To support usability and reproducibility, the library includes detailed documentation[2] and a crash course repository with complete examples and notebooks.[3]

2. Software description

TINTOlib is a Python library that unifies diverse tabular to synthetic image transformation methods into a single coherent framework, allowing researchers to generate synthetic images from structured data and apply them to vision-based or hybrid deep learning models. Its core design goals are flexibility, reproducibility, and extensibility: all methods follow a standardized `fit/transform` interface, outputs are organized in a consistent format, and new transformations can be integrated through modular base classes. This section outlines its architecture and core functionalities.

2.1. Software architecture

TINTOlib adopts an object-oriented design centered around the abstract class `AbstractImageMethod`, which defines a standardized interface for all transformation methods. Each transformation method is implemented as a subclass and encapsulated in a separate module, as detailed in Table 1. These subclasses implement either parametric or non-parametric strategies for converting tabular data into 2D image representations.

The UML diagram in Fig. 1 illustrates the library's architecture. By enforcing inheritance from the abstract base class, TINTOlib guarantees modularity and consistency across methods, facilitating code maintenance and user extensibility.

2.2. Software functionalities

TINTOlib provides a plug-and-play framework for generating synthetic images from tabular datasets. Its functionalities are designed around two main transformation strategies:

- **Parametric methods**: TINTOlib includes three parametric transformations: IGTD [8], REFINED [9], and TINTO [4,7]. IGTD assigns each pixel to a feature and arranges them by minimizing the difference between an ideal similarity-based distance matrix and the actual pixel layout, ensuring that correlated features appear spatially close. REFINED projects features onto a 2D plane using multidimensional scaling (MDS) and then applies a hill-climbing algorithm to refine their placement, producing structured layouts where similar features remain close together and neighborhood dependencies are preserved for vision models. TINTO projects features into a 2D space using dimensionality reduction, normalizes the resulting coordinates, and maps them to pixel positions; a blurring step is then applied to enhance spatial continuity.
- **Non-parametric methods**: Six non-parametric transformations are provided: BarGraph, DistanceMatrix, Combination [10], BIE [13], FeatureWrap [12], and SuperTML [11]. These methods do not perform spatial optimization; instead, they translate features directly into visual representations. BarGraph encodes normalized variable values as bars with configurable widths and spacing. DistanceMatrix constructs square images where each pixel encodes the pairwise difference between feature values. Combination merges these two encodings into a single image, stacking a distance layer, a bar-graph layer, and a raw-value layer. BIE converts floating-point feature values into binary strings and maps the resulting bits into pixel matrices. FeatureWrap encodes features as fixed-length binary vectors. Finally, SuperTML renders feature values directly as text within an image, either using equal fonts (SuperTML-EF) or scaling font size according to feature importance (SuperTML-VF).

Each method follows a unified API with `fit()`, `transform()`, and `fit_transform()`. In parametric methods, `fit()` learns a spatial layout of feature, typically through dimensionality reduction, distance-preserving embeddings, or optimization procedures, that defines how tabular variables are mapped onto the image space. For non-parametric methods, `fit()` does not involve learning but is provided to maintain a consistent interface. This design mirrors scikit-learn's widely adopted transformation pattern, ensuring familiarity and ease of integration.

[1] https://github.com/oeg-upm/TINTOlib
[2] https://tintolib.readthedocs.io/en/latest/index.html
[3] https://github.com/oeg-upm/TINTOlib-Crash_Course

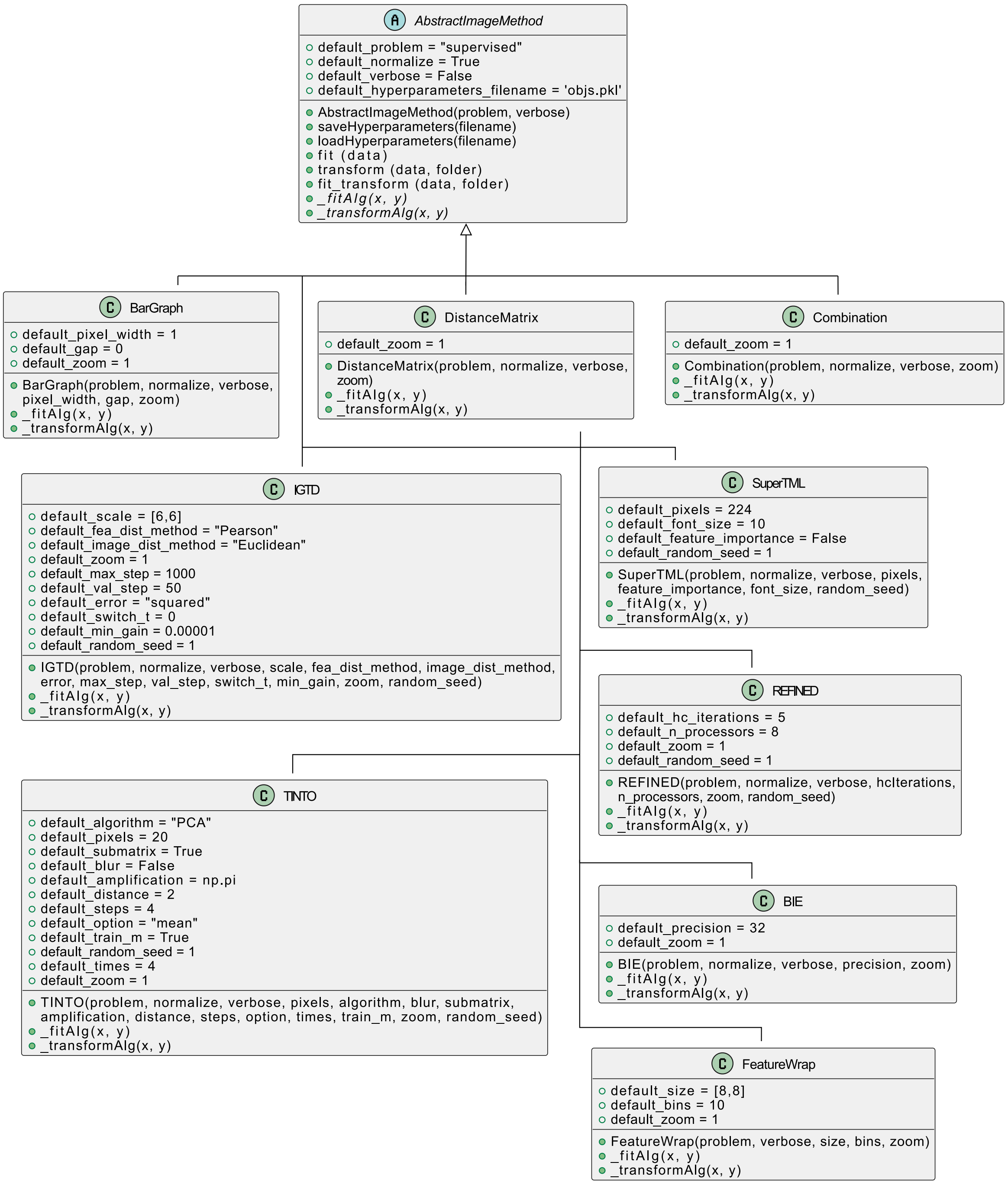

A AbstractImageMethod
default_problem = "supervised"
default_normalize = True
default_verbose = False
default_hyperparameters_filename = 'objs.pkl'
AbstractImageMethod(problem, verbose)
saveHyperparameters(filename)
loadHyperparameters(filename)
fit (data)
transform (data, folder)
fit_transform (data, folder)
_fitAlg(x, y)
_transformAlg(x, y)
C BarGraph
default_pixel_width = 1
default_gap = 0
default_zoom = 1
BarGraph(problem, normalize, verbose, pixel_width, gap, zoom)
_fitAlg(x, y)
_transformAlg(x, y)
C DistanceMatrix
default_zoom = 1
DistanceMatrix(problem, normalize, verbose, zoom)
_fitAlg(x, y)
_transformAlg(x, y)
C Combination
default_zoom = 1
Combination(problem, normalize, verbose, zoom)
_fitAlg(x, y)
_transformAlg(x, y)
C IGTD
default_scale = [6,6]
default_fea_dist_method = "Pearson"
default_image_dist_method = "Euclidean"
default_zoom = 1
default_max_step = 1000
default_val_step = 50
default_error = "squared"
default_switch_t = 0
default_min_gain = 0.00001
default_random_seed = 1
IGTD(problem, normalize, verbose, scale, fea_dist_method, image_dist_method, error, max_step, val_step, switch_t, min_gain, zoom, random_seed)
_fitAlg(x, y)
_transformAlg(x, y)
C SuperTML
default_pixels = 224
default_font_size = 10
default_feature_importance = False
default_random_seed = 1
SuperTML(problem, normalize, verbose, pixels, feature_importance, font_size, random_seed)
_fitAlg(x, y)
_transformAlg(x, y)
C REFINED
default_hc_iterations = 5
default_n_processors = 8
default_zoom = 1
default_random_seed = 1
REFINED(problem, normalize, verbose, hcIterations, n_processors, zoom, random_seed)
_fitAlg(x, y)
_transformAlg(x, y)
C TINTO
default_algorithm = "PCA"
default_pixels = 20
default_submatrix = True
default_blur = False
default_amplification = np.pi
default_distance = 2
default_steps = 4
default_option = "mean"
default_train_m = True
default_random_seed = 1
default_times = 4
default_zoom = 1
TINTO(problem, normalize, verbose, pixels, algorithm, blur, submatrix, amplification, distance, steps, option, times, train_m, zoom, random_seed)
_fitAlg(x, y)
_transformAlg(x, y)
C BIE
default_precision = 32
default_zoom = 1
BIE(problem, normalize, verbose, precision, zoom)
_fitAlg(x, y)
_transformAlg(x, y)
C FeatureWrap
default_size = [8,8]
default_bins = 10
default_zoom = 1
FeatureWrap(problem, verbose, size, bins, zoom)
_fitAlg(x, y)
_transformAlg(x, y)

Fig. 1. UML diagram of TINTOlib. All transformation methods inherit from `AbstractImageMethod`, promoting modularity and code reuse.

Most methods were either adapted from original implementations or re-engineered based on published specifications. In cases such as SuperTML and FeatureWrap, hybrid strategies combining documentation and third-party resources were used to ensure correctness and completeness.

Beyond the nine methods currently included, TINTOlib is deliberately extensible: new tabular to synthetic images techniques can be added by subclassing `AbstractImageMethod` and implementing `fit`/`transform`, while the base class manages validation, configuration, and standardized I/O. Expanding coverage to more methods is an active focus; approaches such as DeepInsight [16], Tab2Visual [17], LM-IGDT [18], TabMap [19], and TablEye [20] could be incorporated under this pattern, subject to stable specifications, licensing, and the availability of reference implementations.

2.3. Access and installation

TINTOlib is available through the Python Package Index (PyPI) and can be installed using `pip`[4]:

```
# Install TINTOlib from PyPI
pip install TINTOlib
```

Code 1 Installing TINTOlib from PyPI.

[4] https://pypi.org/project/TINTOlib/

For full access to the source code, usage examples, and documentation, refer to the GitHub repository [15,21], the official documentation,[5] and the crash-course repository with complete examples pipelines and notebooks.[6]

2.4. Example usage

All transformation methods in TINTOlib follow a unified API with `fit()` and `transform()` (see Code 2), and `fit_transform()` (see Code 3) methods.

```python
# Import the TINTO class from TINTOlib
from TINTOlib.tinto import TINTO

# Instantiate the TINTO method for a classification
    task
image_method = TINTO(problem='supervised')

# Train the transformation model on the dataset
image_method.fit(data=dataset, folder=folder)

# Apply the learned transformation to generate images
image_method.transform(folder=folder)

# The generated images will be stored in the specified
    output folder
```

Code 2 Applying the parametric transformation method (e.g. TINTO) in TINTOlib.

```python
# Import the BarGraph class from TINTOlib
from TINTOlib.barGraph import BarGraph

# Instantiate the BarGraph method for a regression
    task
image_method = BarGraph(problem='regression')

# Load and transform the dataset, saving the generated
    images
image_method.fit_transform(data=dataset, folder=folder
    )

# The transformed images will be saved in the
    specified output folder
```

Code 3 Applying the non-parametric transformation method (e.g. Bargraph) in TINTOlib.

3. Illustrative examples

This section visually demonstrates how tabular data is transformed into synthetic images using various methods available in TINTOlib. We apply these transformations to two datasets: the Boston housing dataset (Regression)[7] and the Multiple Features Dataset - Fourier variant (Mfeat-fourier, Classification)[8]

Parametric transformations such as TINTO, IGTD, and REFINED construct images by learning a spatial arrangement of features, relying on distance metrics or similarity-based embeddings. Each feature is mapped to a single pixel, so even datasets with hundreds or thousands of variables can be represented in relatively small images. This allows a compact layout that encodes a large number of features while keeping the image size manageable for vision models. As a result, these methods are particularly well suited to high-dimensional data, producing dense and structured representations. An example is shown for the Mfeat-Fourier dataset in Fig. 3.

Non-parametric methods such as BarGraph, DistanceMatrix, SuperTML, FeatureWrap, Combination, and BIE map feature values directly to visual components without performing any spatial optimization. While this direct mapping makes them simple to compute and easy to interpret, the image size grows in proportion to the number of features. These methods therefore generate clear and interpretable visualizations for datasets with a small number of variables, such as the Boston housing dataset in Fig. 2, but they scale poorly for high-dimensional data. In datasets with more features, such as Mfeat-Fourier (Fig. 3), non-parametric approaches would lead to larger or cluttered images, which reduces interpretability and increases computational cost for downstream vision models.

Figs. 2 and 3 illustrate synthetic images generated by each method.

3.1. Experimental setup

This section describes the experimental protocol used to illustrate TINTOlib's usability and to enable fair, reproducible comparisons across transformations and model families; the setup is illustrative rather than an exhaustive benchmark. We therefore evaluate two representative tasks: regression (Boston) and classification (Mfeat–fourier). In each case, data are split into train/validation/test with a 70/15/15 ratio using the same random seed across all models to ensure identical splits. Categorical variables are one-hot encoded and numerical variables are min–max scaled; this preprocessing is applied consistently to all models, including the classical baselines.

We compare tabular and vision architectures as well as hybrids. Classical baselines are XGBoost, LightGBM, and CatBoost. Deep learning baselines include an MLP (tabular), a CNN, and a ViT (vision). Hybrid models (HyCNN and HyViT) combine an image branch (CNN or ViT) with an MLP branch on the raw tabular features. All model families are tuned with Optuna hyperparameter searches; to ensure comparability, we use identical search spaces across tasks, models, and transformation methods, and the MLP branch in hybrids shares the standalone MLP search space. Best configurations are selected based on validation performance.

Transformation method hyperparameters are not tuned and remain fixed across both tasks to showcase default usability, with the sole exception of FeatureWrap, which requires task-specific sizing to control output image dimensions. For robustness, the best configuration identified by Optuna for each setting is re-run with five independent seeds; we report mean ± standard deviation. Accuracy is used for classification and RMSE for regression. Fig. 4 illustrates the hybrid configuration (image branch plus tabular MLP).

We emphasize that the two datasets used were selected to illustrate TINTOlib's flexibility with different tabular characteristics: Boston has fewer features, resulting in sparser synthetic images (see Figs. 2), while Mfeat-Fourier is middle-dimensional with complex feature interactions, resulting in rich image structures (see Fig. 3). These settings are not intended as definitive benchmarks, but rather as accessible and contrasting examples.

3.2. Experimental results

We organize results in two steps: (i) for each task, we vary the transformation method and backbone (CNN, ViT, and their hybrids) to assess how pairings behave (Tables 2 and 3); and (ii) we situate the best vision/hybrid configurations against strong classical baselines (Table 4). All scores are reported as mean ± std over five seeds.

[5] https://tintolib.readthedocs.io/en/latest/
[6] https://github.com/oeg-upm/TINTOlib-Crash_Course
[7] https://www.openml.org/search?type=data&sort=runs&id=531&status=active
[8] https://www.openml.org/search?type=data&sort=runs&id=14&status=active

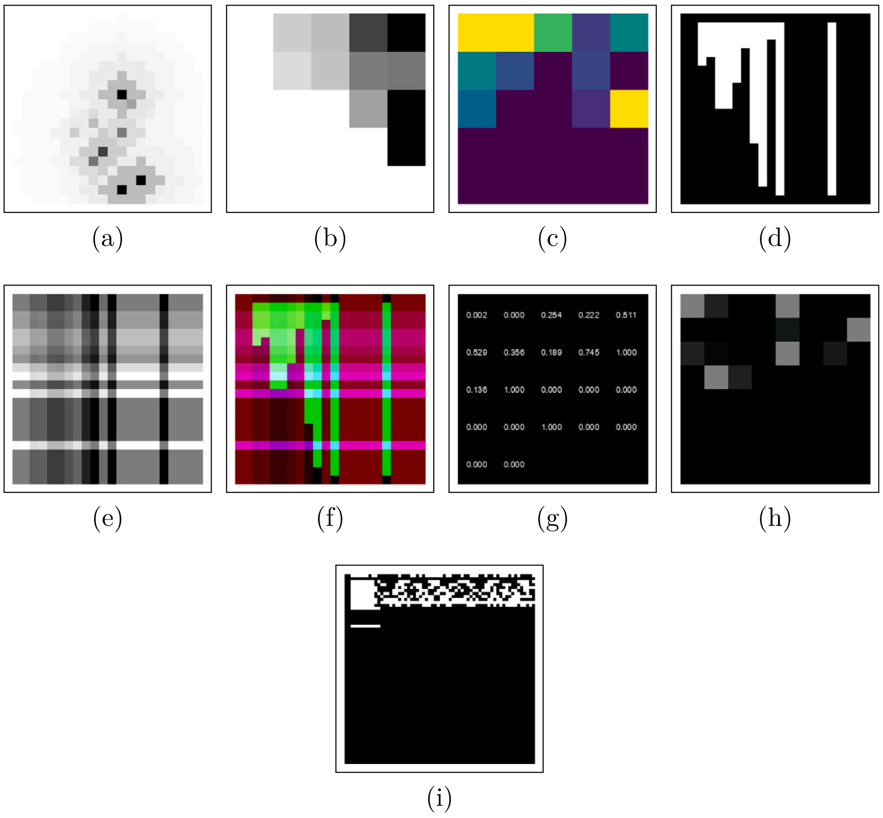


Fig. 2. Visual representations of synthetic image transformations applied to the Boston housing dataset: (a) TINTO; (b) IGTD; (c) REFINED; (d) BarGraph; (e) DistanceMatrix; (f) Combination; (g) SuperTML; (h) FeatureWrap; and (i) BIE.

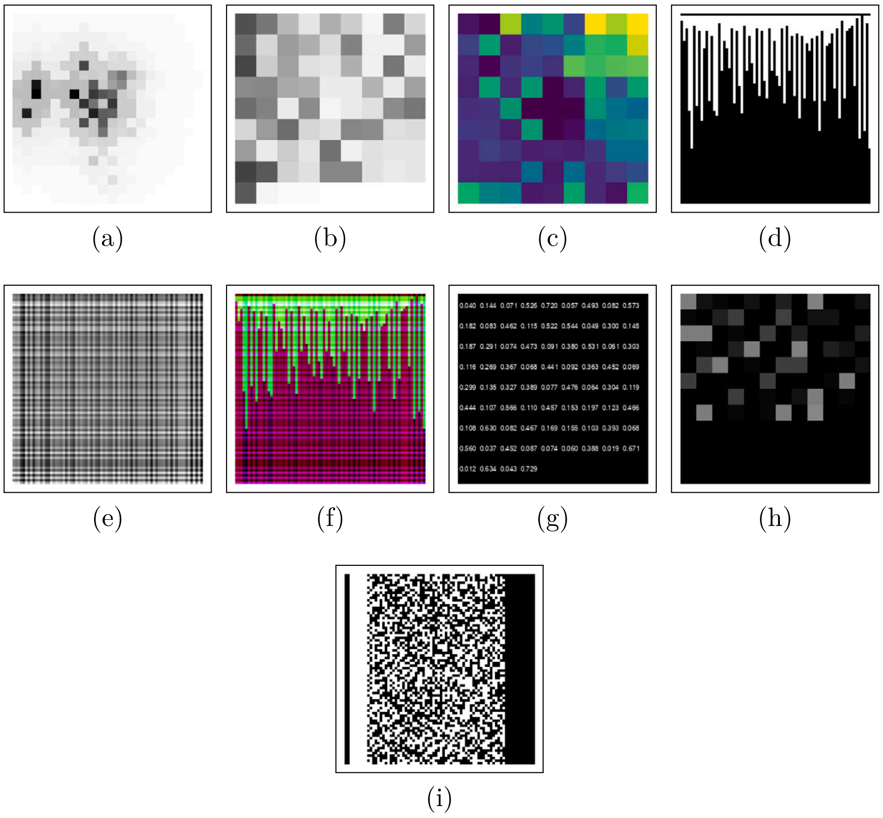
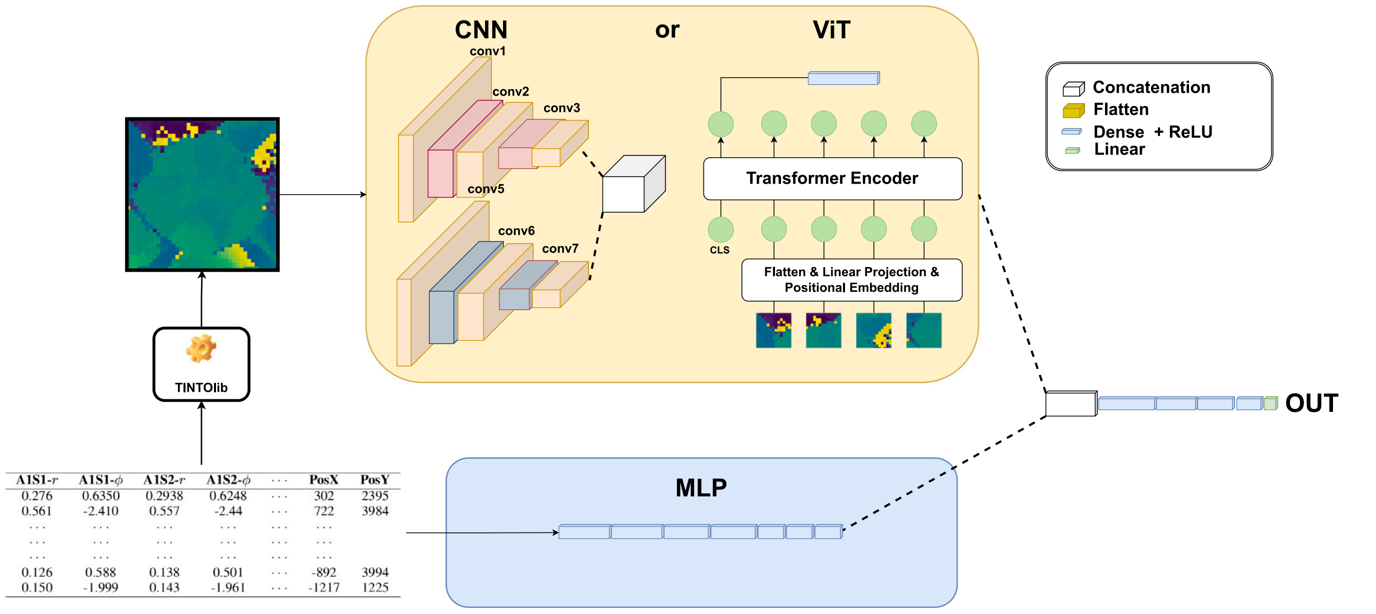


Fig. 3. Visual representations of synthetic image transformations applied to the Multiple Features Dataset: Fourier: (a) TINTO; (b) IGTD; (c) REFINED; (d) BarGraph; (e) DistanceMatrix; (f) Combination; (g) SuperTML; (h) FeatureWrap; and (i) BIE.



$\mathbf{A1S1}$-r	$\mathbf{A1S1}$-ϕ	$\mathbf{A1S2}$-r	$\mathbf{A1S2}$-ϕ	$\cdots$	**PosX**	**PosY**
0.276	0.6350	0.2938	0.6248	$\cdots$	302	2395
0.561	-2.410	0.557	-2.44	$\cdots$	722	3984
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	
0.126	0.588	0.138	0.501	$\cdots$	-892	3994
0.150	-1.999	0.143	-1.961	$\cdots$	-1217	1225

Fig. 4. A simplified illustration of the HyNN architecture, where CNN or ViT process synthetic images, while MLP handles tabular data.

Table 2
Results using the Regression (RMSE ↓) task on the Boston dataset. Values are reported as mean ± std over 5 seeds. Best results per transformation are shown in bold.

Transformation	ViT	ViT+MLP	CNN	CNN+MLP
IGTD	3.1423 ± 0.4393	**2.7182 ± 0.2550**	3.3517 ± 0.3704	3.8601 ± 0.4261
REFINED	2.8878 ± 0.0754	2.7975 ± 0.1112	2.6553 ± 0.3108	**2.4389 ± 0.1630**
TINTO	3.4201 ± 0.2915	**2.9124 ± 0.2221**	3.3786 ± 0.4393	3.0586 ± 0.5522
BarGraph	**2.5199 ± 0.1961**	2.5593 ± 0.2469	2.6941 ± 0.2354	2.8532 ± 0.2366
DistanceMatrix	3.3126 ± 0.4841	2.9739 ± 0.2240	**2.7075 ± 0.0844**	2.7680 ± 0.0921
Combination	**2.6335 ± 0.1637**	2.7065 ± 0.1354	2.8055 ± 0.4631	3.0653 ± 0.8553
BIE	5.7790 ± 0.7448	**3.7024 ± 0.4825**	6.1717 ± 0.3183	3.6898 ± 0.2622
FeatureWrap	6.6278 ± 0.3493	**3.0425 ± 0.0895**	6.2074 ± 0.4733	3.3606 ± 0.3171
SuperTML	6.1095 ± 0.4264	**3.2711 ± 0.7654**	4.7482 ± 0.2940	3.3525 ± 0.5113

Table 2 highlights the benefits of hybrid models in regression tasks. These configurations often outperform their vision-only counterparts, particularly when the synthetic image fails to preserve numerical precision. However, some powerful standalone backbones such as ViT remain competitive. These observations reinforce the recommendation to explore multiple combinations of transformations and model types.

On the other hand, Table 3 shows that hybrid architectures also tend to enhance classification performance, particularly for transformation methods that on their own yield weaker results. This benefit is especially evident when the MLP head compensates for the limitations of the vision model in extracting meaningful patterns from synthetic images. However, for certain transformations, strong visual backbones like ViT can achieve top performance without requiring hybridization, demonstrating their robustness in high-dimensional classification tasks. In the Mfeat-Fourier dataset, the large number of numerical features is associated with rich, densely populated synthetic images (Fig. 3), in contrast to the sparser representations observed in low-dimensional datasets such as Boston (Fig. 2).

Therefore, adding an MLP head often helps, particularly for weaker encodings, but the effect is not universal, and we recommend trying multiple (encoding, backbone, MLP) combinations per dataset. Notably, in some settings the best results are achieved by a hybrid model, whereas in others a plain backbone suffices. These experiments are in two representative settings; we provide additional results and ongoing analyses [15,22].

Table 4 summarizes performance across model families. Several vision and hybrid configurations are competitive with classical baselines such as XGBoost and CatBoost. This reinforces that performance gains are driven by the specific transformation–architecture pairing, not by any universally superior model.

Overall, our two tasks illustrate that performance depends strongly on the transformation–architecture pairing rather than any single universally best choice. Hybrids are not uniformly superior, though they can yield the best outcome in some cases (e.g., HyCNN+REFINED on Boston). Note that these findings are illustrative and not universally generalizable. They are intended to demonstrate TINTOlib's functionality. Moreover, it is important to consider that most transformation methods are accompanied by specific guidelines in their original publications regarding the most suitable neural architectures or training setups. For example, BIE [13] recommends using pretrained CNNs to enhance final model performance.

4. Impact

TINTOlib addresses the current fragmentation of tabular data into synthetic images methods: many transformations exist, but their implementations are scattered, inconsistently specified, and hard to reuse. To our knowledge, TINTOlib is the first unified and extensible Python library designed explicitly for flexibility and reproducibility in this space. It provides a consistent transformation style using the `fit/transform` API across both parametric and non-parametric transformations, a standardized output layout, and reference pipelines and notebooks that let researchers generate synthetic images from tabular data and apply them directly to CNNs, ViTs, and hybrid vision–tabular models easily.

Beyond its core functionality, TINTOlib is engineered for reproducibility and reuse. The distribution includes deterministic seeding

Table 3
Results using the Classification (Accuracy ↑) task on the Mfeat-Fourier dataset. Values are reported as mean ± std over 5 seeds. Best results per transformation are shown in bold.

Transformation	ViT	ViT+MLP	CNN	CNN+MLP
TINTO	**0.8700 ± 0.0170**	0.8667 ± 0.0147	0.8647 ± 0.0077	0.8447 ± 0.0214
IGTD	**0.8313 ± 0.0130**	0.8207 ± 0.0162	0.7987 ± 0.0251	0.7980 ± 0.0117
REFINED	0.8073 ± 0.0136	**0.8080 ± 0.0244**	0.8047 ± 0.0093	0.7973 ± 0.0043
DistanceMatrix	0.8213 ± 0.0222	**0.8373 ± 0.0119**	0.8100 ± 0.0180	0.8013 ± 0.0080
BarGraph	**0.8253 ± 0.0183**	0.8240 ± 0.0258	0.8067 ± 0.0268	0.8173 ± 0.0132
Combination	**0.8360 ± 0.0179**	0.8207 ± 0.0155	0.8040 ± 0.0083	0.8120 ± 0.0096
SuperTML	0.7427 ± 0.0182	**0.8120 ± 0.0177**	0.6613 ± 0.0475	0.6920 ± 0.0581
FeatureWrap	0.6747 ± 0.0229	**0.8160 ± 0.0068**	0.5940 ± 0.0185	0.8013 ± 0.0110
BIE	0.6780 ± 0.0210	**0.8053 ± 0.0112**	0.4813 ± 0.1171	0.5187 ± 0.3822

Table 4
Performance comparison of different model architectures on the test split. Best results are shown in bold.

Model	Classification (Accuracy ↑) – Mfeat-Fourier	Regression (RMSE ↓) – Boston
XGBoost	0.8420 ± 0.0096	2.7749 ± 0.1665
CatBoost	0.8587 ± 0.0096	2.7001 ± 0.1062
LightGBM	0.8573 ± 0.0103	2.8055 ± 0.1009
MLP	0.8200 ± 0.0122	2.8280 ± 0.1311
CNN	0.8647 ± 0.0077 (TINTO)	2.6553 ± 0.3108 (REFINED)
HyCNN	0.8447 ± 0.0214 (TINTO)	**2.4389 ± 0.1630** (REFINED)
ViT	**0.8700 ± 0.0167** (TINTO)	2.5199 ± 0.1961 (BarGraph)
HyViT	0.8667 ± 0.0147 (TINTO)	2.5593 ± 0.2469 (BarGraph)

and easy to use hyperparameters. The architecture is deliberately open and future-proof: new transformations can be added by subclassing the provided abstract base classes, inheriting validation, logging, and I/O. In this way, TINTOlib not only lowers the barrier to applying, comparing, and reusing existing methods, but also offers the community a practical path to contribute additional transformation techniques under a single and maintainable framework.

The empirical results in this paper are illustrative rather than exhaustive: Table 4 situates vision-based and hybrid models enabled by TINTOlib alongside strong classical baselines on one classification and one regression task. Outcomes vary by dataset, transformation, and backbone; in several settings, vision-based and hybrid models are competitive with gradient boosting methods. Synthetic images can reorganize and highlight visual patterns derived from tabular structure, but they may lose numerical precision or explicit inter-variable relationships that the MLP branch can preserve. This complementarity makes hybrids a promising direction, particularly since the MLP branch could be replaced or extended with other deep learning models [4,5]. Note that some transformation methods may perform better when applying the architectural recommendations provided in their original publications, e.g., BIE [13] advises the use of pretrained models to boost final accuracy.

5. Conclusion and future work

TINTOlib unifies diverse tabular to synthetic image methods under a reusable framework and demonstrates how these representations can be used with CNNs, ViTs, and hybrid vision–tabular models.

The illustrative experiments presented in this work show that TINTOlib makes it straightforward to compare multiple methods and architectures within a unified framework, and that vision-based and hybrid approaches can be competitive with strong classical baselines on both regression and classification tasks. However, these results should not be interpreted as universally generalizable. Performance depends heavily on the dataset characteristics, transformation strategy, and backbone architecture. While hybrid models often offer improvements – especially when combining strong MLP heads with less effective visual encodings – they are not always superior. Moreover, several transformation methods come with specific guidelines in their original publications regarding suitable neural architectures or training procedures.

Future development will focus on extending the library with both new and existing transformation methods, subject to licensing and specification stability. In parallel, we plan to explore and adopt widely used interpretability techniques from computer vision, enabling a deeper understanding of how models leverage synthetic images, identifying which features drive predictions, and ensuring transparency and trustworthiness in applications where tabular data are critical.

CRediT authorship contribution statement

Jiayun Liu: Writing – review & editing, Writing – original draft, Visualization, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **David González-Fernández:** Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Manuel Castillo-Cara:** Writing – review & editing, Writing – original draft, Visualization, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Raúl García-Castro:** Writing – review & editing, Writing – original draft, Visualization, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The research leading to these results has been funded by grant PID2023-150515OB-I00 from the Spanish Government, partially covered with funds from the Recovery and Resilience Facility (RRF), and by the Madrid Government (Comunidad de Madrid-Spain) under the Multiannual Agreement with the Universidad Politécnica de Madrid in the Excellence Programme for University Teaching Staff, in the context of the V PRICIT (Regional Programme of Research and Technological Innovation). The authors gratefully acknowledge the Universidad Politécnica de Madrid for providing computing resources on the IPTC-AI. Innovation Space AI Supercomputing Cluster.

Appendix. Software specifications

TINTOlib is openly available to the research community to promote transparency, reproducibility, and collaborative development. The library is hosted on GitHub[9] and Zenodo [15], providing full access to the source code, documentation, and example use cases. Additionally,

[9] https://github.com/oeg-upm/TINTOlib

it is published on the Python Package Index (PyPI),[10] enabling easy installation and integration with existing Python workflows.

The software is distributed under an open-source license (Apache License 2.0), which allows users to freely use, modify, and distribute the library, ensuring flexibility for academic and industrial applications. By combining access through GitHub and PyPI, and providing detailed documentation, TINTOlib ensures accessibility and ease of use for practitioners and researchers alike. Furthermore, ongoing updates and community contributions are encouraged to continuously enhance the library's capabilities.

A.1. Computational details

TINTOlib is compatible with all versions of Python starting from 3.7. The library can be installed and executed on various operating systems, including Linux, MacOS, and Windows, but some specific requirements must be met for certain transformation methods.

Specific requirements for transformation methods

- REFINED: This method relies on `mpi4py`, which enables parallel computation using MPI (Message Passing Interface). However, `mpi4py` requires administrative permissions to utilize multiple processors, making it incompatible with platforms like Google Colab.
  - Installation on Linux: Ensure that the MPI environment is set up before installing `mpi4py`. Run the following commands:

    ```
    sudo apt-get install python3
    sudo apt install python3-pip
    sudo apt install python3-mpi4py
    ```

    Once MPI is installed, use:

    ```
    pip install mpi4py
    ```

  - Installation on MacOS and Windows: Direct installation of `mpi4py` is supported. Use the following command:

    ```
    pip install mpi4py
    ```

- SuperTML: This method relies on the **MS Sans Serif** font to generate text-based synthetic images. On systems like Windows, this font is available by default. However, for Linux and MacOS, additional steps are required to install the necessary fonts:
  - Linux: Install the Microsoft Core Fonts package by running the following command:

    ```
    sudo apt install ttf-mscorefonts-
    installer
    ```

  - MacOS: Download and install the Microsoft Core Fonts package using tools like Homebrew:

    ```
    brew tap homebrew/cask-fonts
    brew install --cask font-microsoft
    -sans-serif
    ```

  On Google Colab, installing additional fonts is not permitted due to administrative restrictions. Consequently, the SuperTML method may not work as intended in this environment without further customization.

[10] https://pypi.org/project/TINTOlib/

A.2. General dependencies for TINTOlib

The library requires the following Python packages, which support the implementation of various transformation methods:

- `TINTOlib` (core library)
- `numpy` (numerical computations)
- `pandas` (data manipulation)
- `scikit-learn` (data preprocessing and feature selection)
- `matplotlib` (visualizations)
- `tensorflow` or `pytorch` (deep learning frameworks)
- `keras` (deep learning framework)
- `mpi4py` (for REFINED, as detailed above)
- `keras_preprocessing` (data preprocessing utilities for Keras)
- `tifffile` (handling TIFF image formats)
- `tqdm` (progress bar visualization)
- `seaborn` (advanced data visualization)
- `bitstring` (manipulation of binary strings)
- `opencv-python` (computer vision utilities)

A.3. Recommendations

After installing TINTOlib and its dependencies, it is highly recommended to restart the `Jupyter Notebook` kernel to ensure that all packages are properly loaded and functional.

A.4. Note on google colab compatibility

While TINTOlib supports most methods seamlessly in Google Colab, certain limitations should be considered:

- The **REFINED** method cannot run in Colab due to the lack of administrative permissions required for parallel processing with `mpi4py`.
- The **SuperTML** method may require additional font installations, which are restricted in Colab.

By addressing these considerations, TINTOlib ensures robust functionality across different environments and enables researchers to leverage its capabilities for transforming tabular data into synthetic images.

Data availability

The complete Python source code of the TINTOlib library is available at GitHub and Zenodo [15]. The repository containing the Python code for the experiments reported in this manuscript, including the Jupyter notebooks for the two datasets used, is accessible at Github and Zenodo [21].

In addition, a dedicated benchmark webpage is maintained[11] [22], where results and comparative analyses across a growing set of datasets, transformation methods, and neural architectures are continuously updated. This page aims to foster reproducibility and transparency while providing researchers with a reference point for further exploration and evaluation in this area.

References

[1] Shwartz-Ziv R, Armon A. Tabular data: Deep learning is not all you need. Inf Fusion 2022;81:84–90. http://dx.doi.org/10.1016/j.inffus.2021.11.011.

[2] Borisov V, Leemann T, Seßler K, Haug J, Pawelczyk M, Kasneci G. Deep neural networks and tabular data: A survey. IEEE Trans Neural Netw. Learn Syst 2024;35(6):7499–519. http://dx.doi.org/10.1109/TNNLS.2022.3229161.

[11] https://oeg-upm.github.io/TINTOlib/binary.html

[3] Van Breugel B, Van Der Schaar M. Position: Why tabular foundation models should be a research priority. In: Salakhutdinov R, Kolter Z, Heller K, Weller A, Oliver N, Scarlett J, Berkenkamp F, editors. Proceedings of the 41st international conference on machine learning, 235 of proceedings of machine learning research. PMLR; 2024, p. 48976–93.

[4] Talla-Chumpitaz R, Castillo-Cara M, Orozco-Barbosa L, García-Castro R. A novel deep learning approach using blurring image techniques for bluetooth-based indoor localisation. Inf Fusion 2023;91:173–86. http://dx.doi.org/10.1016/j.inffus.2022.10.011.

[5] Lara-Abelenda FJ, Chushig-Muzo D, Peiro-Corbacho P, Gómez-Martínez V, Wägner AM, Granja C, Soguero-Ruiz C. Transfer learning for a tabular-to-image approach: A case study for cardiovascular disease prediction. J Biomed Informat. 2025;165:104821. http://dx.doi.org/10.1016/j.jbi.2025.104821.

[6] Jiang J-P, Liu S-Y, Cai H-R, Zhou Q, Ye H-J. Representation learning for tabular data: A comprehensive survey. 2025, arXiv:2504.16109.

[7] Castillo-Cara M, Talla-Chumpitaz R, García-Castro R, Orozco-Barbosa L. Tinto: Converting tidy data into image for classification with 2-dimensional convolutional neural networks. 22, 2023, 101391. http://dx.doi.org/10.1016/j.softx.2023.101391, SoftwareX.

[8] Zhu Y, Brettin T, Xia F, Partin A, Shukla M, Yoo H, Evrard YA, Doroshow JH, Stevens RL. Converting tabular data into images for deep learning with convolutional neural networks. Sci Rep 2021;11. http://dx.doi.org/10.1038/s41598-021-90923-y.

[9] Bazgir O, Zhang R, Dhruba SR, Rahman R, Ghosh S, Pal R. Representation of features as images with neighborhood dependencies for compatibility with convolutional neural networks. Nat Commun 2020;11. http://dx.doi.org/10.1038/s41467-020-18197-y.

[10] Sharma A, Kumar D. Classification with 2-D convolutional neural networks for breast cancer diagnosis. Sci Rep 2022;12. http://dx.doi.org/10.1038/s41598-022-26378-6.

[11] Sun B, Yang L, Zhang W, Lin M, Dong P, Young C, Dong J. SuperTML: Two-dimensional word embedding for the precognition on structured tabular data. In: 2019 IEEE/CVF conference on computer vision and pattern recognition workshops. CVPRW, IEEE Computer Society; 2019, p. 2973–81.

[12] Li Z, Qin Z, Huang K, Yang X, Ye S. Intrusion detection using convolutional neural networks for representation learning. In: Liu D, Xie S, Li Y, Zhao D, El-Alfy E-S M, editors. Neural information processing, springer international publishing. Cham; 2017, p. 858–66.

[13] Briner N, Cullen D, Halladay J, Miller D, Primeau R, Avila A, Basnet R, Doleck T. Tabular-to-image transformations for the classification of anonymous network traffic using deep residual networks. IEEE Access 2023;11:113100–13. http://dx.doi.org/10.1109/ACCESS.2023.3323927.

[14] Castillo-Cara M, Martínez-Gómez J, Ballesteros-Jerez J, García-Varea I, Orozco-Barbosa L, García-Castro R. MIMO-Based Indoor Localisation with Hybrid Neural Networks: Leveraging Synthetic Images from Tidy Data for Enhanced Deep Learning. IEEE J Sel Top Signal Process 2025;19(3):559–71. http://dx.doi.org/10.1109/JSTSP.2025.3555067.

[15] Liu J, Fernández DG, Castillo-Cara M, Castro RG. Tintolib: A python library for transforming tabular data into synthetic images for deep neural networks. 2025, http://dx.doi.org/10.5281/zenodo.17153509.

[16] Sharma A, Vans E, Shigemizu D, Boroevich KA, Tsunoda T. DeepInsight: A methodology to transform a non-image data to an image for convolution neural network architecture. Sci Rep 2019;9.

[17] Mamdouh A, El-Melegy M, Ali S, Kikinis R. Tab2visual: Overcoming limited data in tabular data classification using deep learning with visual representations. 2025, URL http://arxiv.org/abs/2502.07181 arXiv:2502.07181.

[18] Gómez-Martínez V, Lara-Abelenda FJ, Peiro-Corbacho P, Chushig-Muzo D, Granja C, Soguero-Ruiz C. LM-IGTD: a 2D image generator for low-dimensional and mixed-type tabular data to leverage the potential of convolutional neural networks. 2024, arXiv:2406.14566.

[19] Yan R, Islam MT, Xing L. Interpretable discovery of patterns in tabular data via spatially semantic topographic maps. Nat Biomed Eng 2025;9(4):471–82. http://dx.doi.org/10.1038/s41551-024-01268-6.

[20] eon Lee S, Lee S-C. Tableye: seeing small tables through the lens of images. 2023, arXiv:2307.02491.

[21] Liu J, Castillo-Cara M, Castro RG. Tintolib: A python library for transforming tabular data into synthetic images for deep neural networks - examples. 2025, http://dx.doi.org/10.5281/zenodo.17154092.

[22] Liu J, Castillo-Cara M, García-Castro R. Tintolib benchmark results. 2025, URL https://oeg-upm.github.io/TINTOlib/binary.html. (Accessed 20 October 2025).