TINTO: A Novel Framework to Convert Tidy Data into Image for Deep Learning with Convolutional Neural Networks

Manuel Castillo-Cara[*,a], Reewos Talla-Chumpitaz[b], Raúl García-Castro[a], Luis Orozco-Barbosa[c]

[a]*Universidad Politécnica de Madrid, Madrid (Spain)*
[b]*Universidad Nacional de Ingenieria, Lima (Peru)*
[c]*Albacete Research Institute of Informatics, Universidad de Castilla-La Mancha, Albacete (Spain)*

Abstract

The growing interest in the use of algorithms-based Machine Learning for predictive tasks has generated a large and diverse development of algorithms. However, it is widely known that not all of these algorithms are adapted to efficient solutions in certain datasets that are in a tidy data format. For this reason, novel techniques are currently being developed to convert tidy data into images in order to use CNNs. TINTO offers new opportunities to convert tidy data into images through the representation of characteristic pixels by implementing two-dimensional reduction algorithms: PCA and t-SNE. Our proposal also includes a blurring technique, which adds more ordered information to the image and can improve the classification task in CNNs.

Keywords: Synthetic images, Tabular data into image, Convolutional neural networks, Image blurring technique, Image generationk

1. Motivation and Significance

Nowadays, there is great interest in being able to use data for predictive purposes in any field and area of application. For this purpose, there are a large number of classical Machine Learning (ML) algorithms or based on deep neural networks such as the classic Multilayer Perceptron (MLP), or

[*]Corresponding author: Manuel Castillo-Cara

Email addresses: `manuel.castillo.cara@upm.es` (Manuel Castillo-Cara[*,]), `reewos.talla.c@uni.pe` (Reewos Talla-Chumpitaz), `r.garcia@upm.es` (Raúl García-Castro), `luis.orozco@uclm.es` (Luis Orozco-Barbosa) the more advanced Convolutional Neural Networks (CNN) [1]. For the ML and MLP case, a series of structured data, such as tabular or tidy data (here on in referred to as tidy data) format, is needed [2]; and for CNNs, the data has to be non-structured, specifically in image format [3].

In this context, one of the main contributions of this work lies in the transposition of tidy data [2] into images with a novel preprocessing technique that represents the behaviour of tidy data in the image construction process [4]. Similarly, there are many problems in which the development of an MLP for tidy data does not establish a good fit and generalisation of the model; then the use of this type of neural network for predictive purposes is discarded [5]. For this reason, a series of frameworks are being developed to convert tidy data into images in order to be able to use CNNs [6, 5, 7]. Extending the use of CNNs in tidy data problem solving would allow for improved model fitting and generalisation [8].

Therefore, the use of techniques that aim to convert tidy data into images is part of an open research area that has been developing in recent years to extend problem solving with more advanced techniques [9], e.g., CNN [5]. Consequently, TINTO software has been created for this purpose and to allow the academic and professional community to solve tidy data problems with images [8]. Moreover, TINTO is primarily used to solve classification problems in ML, i.e., binary or multiclass. This means that tidy data must have a target that is categorical, e.g., well-known datasets such as the IRIS dataset [10] or the indoor localisation dataset [11] used in the previous work as successful use-case [6].

In other words, our proposal starts by transforming tidy data, consisting of samples spatially distributed in a target, i.e., in a classical classification problem in ML, into images to enable the implementation of the classification process to be performed by a CNN [12, 13]. The proposal uses the methods and mechanisms of the articles [6, 7] as the starting point for developing the overall processing schema. In the first step of our proposed schema, two-dimensional reduction techniques [14] are used to transform tidy data into images: t-SNE [9, 15] and PCA [16]. Moreover, TINTO has added the classical painting technique known as blurring in order to represent more ordered contextual information in the resulting image and thus improve the process of feature extraction and generalisation of CNNs [17, 18].

Accordingly, the main contributions of TINTO software are presented below:

- Use of two-dimensionality reduction algorithms to convert tidy data into images, spatial distributed data readings, namely, the t-SNE and PCA algorithms.

- Use of the blurring painting technique to introduce contextual information to the image, which can improve the classification process.
- Use CNNs models for binary or multiclass classification problems in classical ML as the TINTO software allows the conversion of Tidy Data into images.

2. Software Description

This section describes the software architecture and specifications for TINTO framework.

2.1. Software Architecture

One of the first translations to be performed is to convert tidy data into image data format, i.e., the corresponding sample (row) in the sampling phase (see Table 1). The reader can learn more about the mathematical foundations of the transformation process and the characteristics of the sample dataset used in this example by consulting the scientific article [6] and by downloading the dataset from Zenodo [11]. The following considerations should be taken into account when creating the images:

- The input dataset must be in CSV, taking into account the tidy data format [2].
- The target (variable to be predicted), Y, should be set as the last column of the dataset. Therefore, the first columns will be the features, X.
- All data must be in numerical data type. TINTO does not accept data in string or any other non-numeric data type.
- TINTO will create as many folders with their corresponding images in each folder as there are targets. For example, in this case, we have 15 different targets, so TINTO will create 15 different folders, one for each class.

Therefore, Figure 1 depicts the tidy data to image translation process. As can be seen, tidy data are converted into image data through a two-dimensional space in X and Y. This translation to two-dimensional space enables us to build an image of characteristic pixels for each sample of the dataset, i.e., each data sample is converted to a two-dimensional space.

As seen in Figure 1, the pipeline for converting each sample in tidy data format into a two-dimensional space consists of five main processing tasks:

Table 1: Structure of the dataset in tidy data format. The first 5 columns would be the features (Fe) and the last column would be the target. The first row is the heading of each column.

Fe01	**Fe02**	**Fe03**	**Fe04**	**Fe05**	**Target**
-65	-61	-74	-73	-67	1
-60	-57	-83	-62	-69	2
-66	-70	-78	-63	-73	3
...	...	...	...	...	...
-58	-66	-71	-73	-69	14
-60	-62	-73	-69	-57	15

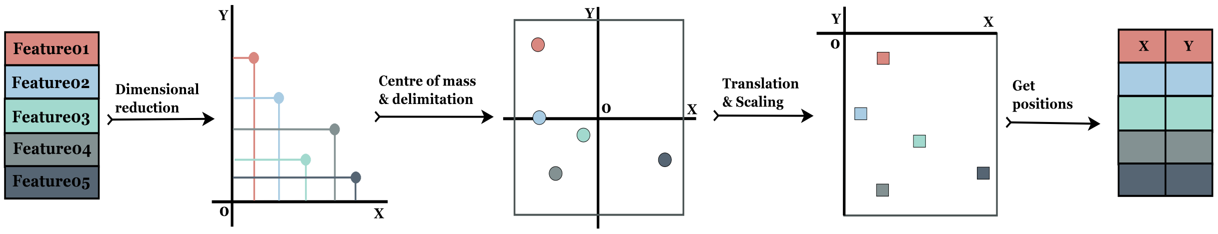


Figure 1: Process for obtaining the feature coordinates from the transpose matrix. It can be seen that the tidy data are converted into a two-dimension matrix through the different phases with their respective techniques, i.e., delimitation, translation, and so on.

1. Data dimensionality reduction: The initial matrix is transposed. Note that each feature is represented with a different colour for presentation purposes, although the resulting figure will consist of two channels, i.e., black and white. This task makes use of a dimensionality reduction algorithm. In this case, TINTO explored two such algorithms: PCA and *t*-SNE.
2. Centre of mass and delimitation: Having obtained the coordinates, the centre of gravity of the points is determined, and the area is subsequently delimited.
3. Scaling and pixel positions. The matrix is transposed, scaled and the values are rounded to integer values.
4. Characteristic pixel positions: The values obtained would be the positions of the characteristic pixels for the creation of the image pattern.

The authors from [6] provide a formal specification of the steps to obtain the positions of the characteristic pixels from the initial tidy data. In the following paragraphs, we describe the main software specifications in which TINTO has been coded and analyse them.

2.2. Image Rendering Techniques

As the procedure used was to convert tidy data into images, two rendering techniques were developed as case studies to be evaluated: one with and one without blurring of the characteristic pixel.

Case 1: Characteristic pixels without blurring

In this case, TINTO fills in the data A_{scale} at their corresponding characteristic pixel positions in the matrix M. The matrix M is then converted into an image. This is done for each A_{scale} sample. All other values of the matrix M are zeros, so they are not significant in this case.

For the creation of images from the tidy data, the characteristic pixel position and the scale generated by the normalisation are used. Figure 3(a) shows an example of the results without the blurring technique.

Case 2: Characteristic pixels with blurring

This procedure was created empirically, based on the technique of blurring used in the plastic arts, specifically in drawing and painting. This technique is often used to soften and extend strokes so that the transition from intense to faint is uniform. Then, by making an analogy with the characteristic pixels as spots on a canvas, these spots can be blurred so that they cover more area without losing the intensity of the original characteristic pixel.

Therefore, the use of the blurring technique of the characteristic pixels is proposed to enlarge their area. To do this, the thickness, number, and intensity of the strokes must be defined. Each stroke would be a circumference around the pixel and the previous stroke with a thickness called *distance*, represented as r; and the number of strokes or circles would be denoted as *steps*, represented as C_x, see Figure 2. Moreover, the intensity of this stroke or circumference would be determined by the encompassed area of the circumference, with the centre (the characteristic pixel) being the initial intensity, represented as P. Additionally, the intensity fades as you move away from the characteristic pixel.

Finally, by using this technique, it is possible to determine the step size in which the blurring of two characteristic pixels overlaps. In this case, TINTO experimented with two possible solutions for the representation of the scene with overlapping pixels:

- **Average value**: An alternative to the above is to average the intensity of both, being the one already in the matrix and the new one (see Figure 3(b)).

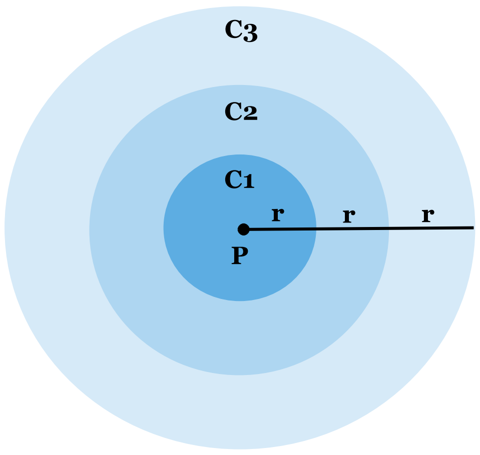
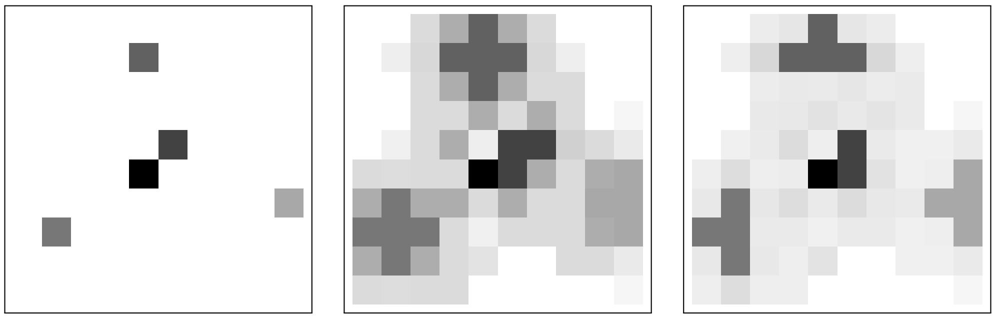


Figure 2: Representation of the blurring technique, where r would be the distance, P the localisation of the characteristic pixel, and the colours would represent the intensity for each stroke or circumference.

- **Maximum**: The highest scoring value pixel of the overlapping pixels is chosen, i.e., if the value of the pixel in the matrix is lower than the value of the incoming pixel, it is replaced; otherwise, it is kept (see Figure 3(c)).



(a) Without blurring. (b) Blurring with maximum value. (c) Blurring with average value.

Figure 3: Image samples generated by TINTO using PCA.

2.3. Software Specifications

Having seen the steps in which TINTO performs a conversion from tidy data into images, this section will describe the main implementation design with the algorithmic specifications. Note that this section will expose the most relevant Python code of TINTO framework, therefore all the Python code and its functionality can be found in the repository [19].

A first function in TINTO has the purpose of being able to create the square delimitation of the resulting image, i.e., return the values, coordinates and vertices (see Alg. 1).

```
def square(coord):
    m = np.mean(coord, axis=0).reshape((1,2))
    coord_new = coord - m
    dista = (coord_new[:,0]**2+coord_new[:,1]**2)**0.5
    maxi = math.ceil(max(dista))
    vertices = np.array([[-maxi,maxi],[-maxi,-maxi],[maxi,-maxi],[maxi,maxi]])
    coord_new = coord_new - vertices[0]
    vertices = vertices - vertices[0]
    return coord_new, vertices
```

Algorithm 1: Square Delimitation

On the other hand, a function is also created to obtain the coordinates of the matrix and fill in the positions of the features (see Alg. 2). Note that a conditional is created if the features are grouped in the same position.

```
def m_imagen(coord, vertices, filename, pixeles=24):
    size = (pixeles, pixeles)
    matriz = np.zeros(size)
    coord_m = (coord/vertices[2,0])*(pixeles-1)
    coord_m = np.round(abs(coord_m))
    for i,j in zip(coord_m[:,1],coord_m[:,0]):
        matriz[int(i),int(j)] = 1
    if(np.count_nonzero(matriz!=0)!=coord.shape[0]):
        return coord_m, matriz, True
    else:
        return coord_m, matriz, False
```

Algorithm 2: Coordinates in the matrix

Finally, one of the main functions of TINTO is defined, which is to be able to add more ordered contextual information to the image through the classical painting technique called blurring (see Alg. 3). This function develops the following main steps: (i) Take the coordinate matrix of the characteristic pixels; and (ii) Create the blurring according to the number of steps taken in a loop.

```
def blurring(matriz, coordinate, distance=0.1, steps=3,
    amplification=np.pi, option='maximum'):
    x = int(coordinate[1])
    y = int(coordinate[0])
    core_value = matriz[x,y]
    for p in range(steps):
        r_actual = int(matriz.shape[0]*distance*(p+1))
```

```python
        intensity=min(amplification*core_value/(np.pi*r_actual**2),core_value)
        lim_inf_i = max(x-r_actual-1,0)
        lim_sup_i = min(x+r_actual+1,matriz.shape[0])
        lim_inf_j = max(y-r_actual-1,0)
        lim_sup_j = min(y+r_actual+1,matriz.shape[1])
        for i in range(lim_inf_i, lim_sup_i):
            for j in range(lim_inf_j, lim_sup_j):
                if((x-i)**2 + (y-j)**2 <= r_actual**2):
                    if(matriz[i,j]==0):
                        matriz[i,j]=intensity
                    elif(x!=i and y!=j):
                        if(option=='mean'):
                            matriz[i,j]=(matriz[i,j]+intensity)/2
                        elif(option=='maximum'):
                            matriz[i,j]=max(matriz[i,j],intensity)
    return matriz
```

Algorithm 3: Blurring specification

3. Illustrative Examples

In this section we will present the results that TINTO has in the generation of images given as input a dataset in tidy data format. Therefore, first the basic commands to launch the script in the terminal are exposed, and then the generated images are shown.

3.1. Command Execution

Before visualising the images generated according to a dataset as shown in Table 1, the basic guidelines for running TINTO in the command terminal are outlined. Note that the data must be in tidy data format, all samples must be of numerical type, and the last column must be the target [2].

TINTO has a parameter to be able to visualise all the different options for the generation of images:

```
$ python tinto.py -h
```

where,

- `python`: Python run command.
- `tinto.py`: TINTO Python script.
- `-h`: Show the TINTO parameters and exit.

Therefore, the following command shows what the basic execution would look like with the default values, i.e., the generation of images with the characteristic pixels (no blurring), PCA as the dimensionality reduction algorithm, and the image size of $20x20$ pixels:

```
$ python tinto.py "dataset.csv" "folder"
```

where,

- `"dataset.csv"`: Set the path where the dataset is located in CSV format.
- `"folder"`: Set the path where the folder with the images will be created.

Finally, a more complex example of image generation is shown below:

```
$ python tinto.py "dataset.csv" "folder" -B -alg t-SNE \
 -oB maximum -px 30 -sB 5
```

where,

- `-B`: Create images with the blurring technique.
- `-alg t-SNE`: Select t-SNE as the dimension reduction algorithm.
- `-oB maximum`: Create the images with maximum value of overlapping pixel.
- `-px 30`: Create images with a size of $30x30$ pixels.
- `-sB 5`: Expand to 5 pixels the blurring.

3.2. Image Generation

In the following, TINTO will present the samples generated using PCA and t-SNE in each dataset, with and without the blurring technique. The same seed was used in all evaluations for the separation of train and test data, i.e., for the same dataset, the same conditions are used, varying only the use of blurring and the type of overlapping. Note that we can observe the images generated by the script. An important aspect is that it will create as many folders as different targets that have the dataset where each folder will contain the images generated by each sample.

Therefore, Figures 3 and 4 show the generation of three sample images for the different cases under study, for PCA and t-SNE, respectively. These sample images are clear examples of the resulting images that can be the input to CNN.

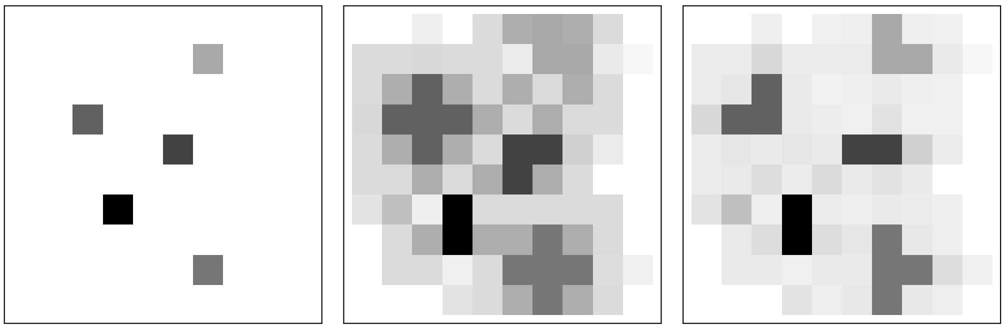

(a) Without blurring. (b) Blurring with maximum value. (c) Blurring with average value.

Figure 4: Image samples generated by TINTO using t-SNE.

4. Impact

As has been discussed in the previous sections, TINTO is a novel software that has the main purpose of converting tidy data into images. Accordingly, TINTO creates the possibility of being able to develop CNN-based models to solve classification problems in classical ML in which either there were solutions with very little convergence of the models or, as was the case in many cases, they could not be solved due to poor generalisation and model fitting. In fact, the following major impacts are expected:

- TINTO is an open source software that can be easily extended or modified.
- TINTO has an object-orientated structure consisting of multiple Python modules that allow for easier debugging and reuse of codes through inheritance.
- TINTO provides a framework for the conversion of tidy data format into images that can be used by academia or businesses.This saves the time needed for coding and manipulating data, and thus finding direct solutions by extending tidy data format solutions to CNNs.
- TINTO allows to convert tidy data into images mainly for classification problems in classical ML. Hence, TINTO allows us to extend the search for solutions of complex problems to CNNs when classical ML algorithms or MLP did not generalise or did not approach an acceptable solution.
- TINTO uses well-documented dimensionality reduction algorithms such as PCA and t-SNE in the image construction process. Note that these

are not the only algorithms, and more of these algorithms or other techniques can be added and studied to represent characteristic pixels in images. With TINTO framework, this could be done very quickly and easily by adding a few additional lines of code.

- TINTO also presents the classical painting technique called Blurring, which allows contextual information to be added to the image in an orderly manner. This allows to improve, in many cases, the classification process and, therefore, the generalisation and fitting in CNNs [6].
- TINTO performs the training and validation process of image conversion in a fairly acceptable time. For example, for the Iris dataset [11] it takes about 10 minutes, and for the data used as an example in this paper [11] it takes 15 minutes. Note that datasets with large amounts of samples can take hours, this is due to the training process of the dimensionality reduction algorithms, i.e., in this case t-SNE and PCA.
- TINTO is among the first software created to convert tidy data into images and is based on [7] and is extended and tested as a use-case in [6]. Moreover, another software that performs this task is the one presented in [5]. To the authors' knowledge, there are no further relevant publications and/or code on this open research line. TINTO is left at the disposal of the community to further develop the research line.

5. Conclusions and Open Challenges

This paper explored as its main work the development of software, called TINTO, that allows the conversion of tidy data into images. This conversion of structured data into unstructured data allows the use of CNNs to solve complex problems where classical ML algorithms or a customised MLP had shortcomings in fitting and generalisation.

To this end, two-dimensionality reduction algorithms such as t-SNE and PCA were evaluated. In this context, two cases of image generation were tested, namely: (i) using the characteristic pixels; and (ii) applying the blurring painting technique.

Finally, one of the main points to develop is to be able to set up more dimensionality reduction algorithms and to study the impact of each of them on the generation of images. Additionally, one of the main challenges that should be studied is the impact on the generation of patterns based on the types of data and the distribution of these data in a dataset.

CRediT authorship contribution statement

Manuel Castillo-Cara: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Software, Visualization, Writing – original draft, Writing – review & editing. **Reewos Talla-Chumpitaz**: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Software, Validation, Visualization. **Raúl García-Castro**: Formal Analysis, Investigation, Methodology, Funding acquisition, Supervision, Writing – original draft, Writing – review & editing. **Luis Orozco-Barbosa**: Formal Analysis, Investigation, Methodology, Supervision, Writing – original draft, Writing – review & editing.

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 scientific article.

Data availability

This scientific experiment contains the following extended data:

- Dataset used in this analysis [11].
- TINTO Python code [19]

Acknowledgments

The research leading to these results has been funded by the European Commission H2020 project "Construction-phase diGItal Twin mOdel (COGITO)", Grant No. 958310; by the Spanish Ministry of Science, Education and Universities, the European Regional Development Fund and the State Research Agency, Grant No. PID2021-123627OB-C52; and by the CYTED, Grant No. 520rt0011. Manuel Castillo-Cara is supported by a postdoctoral research grant (María Zambrano) of the European Union-NextGenerationEU programme and the Spanish Ministry of Universities.

References

[1] J. C. Rangel, J. Martínez-Gómez, C. Romero-González, I. García-Varea, M. Cazorla, Semi-supervised 3d object recognition through cnn labeling, Applied Soft Computing 65 (2018) 603–613. `doi:10.1016/j.asoc.2018.02.005`.

[2] H. Wickham, Tidy data, Journal of Statistical Software 59 (10) (2014) 1–23. `doi:10.18637/jss.v059.i10`.

[3] L. Vasquez-Espinoza, M. Castillo-Cara, L. Orozco-Barbosa, On the relevance of the metadata used in the semantic segmentation of indoor image spaces, Expert Systems with Applications 184 (2021) 115486.

[4] B. Sun, L. Yang, W. Zhang, M. Lin, P. Dong, C. Young, J. Dong, Supertml: Two-dimensional word embedding for the precognition on structured tabular data, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, 2019, pp. 0–0.

[5] Y. Zhu, T. Brettin, F. Xia, A. Partin, M. Shukla, H. Yoo, Y. A. Evrard, J. H. Doroshow, R. L. Stevens, Converting tabular data into images for deep learning with convolutional neural networks, Scientific reports 11 (1) (2021) 1–11.

[6] R. Talla-Chumpitaz, M. Castillo-Cara, L. Orozco-Barbosa, R. García-Castro, A novel deep learning approach using blurring image techniques for bluetooth-based indoor localisation, Information Fusion 91 (2023) 173–186. `doi:10.1016/j.inffus.2022.10.011`.

[7] A. Sharma, E. Vans, D. Shigemizu, K. A. Boroevich, T. Tsunoda, Deepinsight: A methodology to transform a non-image data to an image for convolution neural network architecture, Scientific reports 9 (1) (2019) 1–7.

[8] M. I. Iqbal, M. S. H. Mukta, A. R. Hasan, S. Islam, A dynamic weighted tabular method for convolutional neural networks, IEEE Access (2022).

[9] L. Van der Maaten, G. Hinton, Visualizing data using t-sne., Journal of machine learning research 9 (11) (2008).

[10] R. Fisher, UCI machine learning repository, [Online: accessed 10-jan-2023] (1950).
URL `https://archive.ics.uci.edu/ml/datasets/iris`

[11] M. Castillo-Cara, Bluetooth indoor localization dataset, [Online: accessed 10-jan-2023] (Mar. 2022). `doi:10.5281/zenodo.6343083`.
URL `https://doi.org/10.5281/zenodo.6343083`

[12] L. v. d. Maaten, G. Hinton, Visualizing data using t-sne, Journal of machine learning research 9 (Nov) (2008) 2579–2605.

[13] V. Borisov, T. Leemann, K. Seßler, J. Haug, M. Pawelczyk, G. Kasneci, Deep neural networks and tabular data: A survey, arXiv preprint arXiv:2110.01889 (2021).

[14] E. J. Menvouta, S. Serneels, T. Verdonck, direpack: A python 3 package for state-of-the-art statistical dimensionality reduction methods, SoftwareX 21 (2023) 101282. doi:10.1016/j.softx.2022.101282.

[15] M. Wattenberg, F. Viégas, I. Johnson, How to use t-sne effectively, Distill (2016). doi:10.23915/distill.00002.

[16] D. Kim, D. Joo, J. Kim, Tivgan: Text to image to video generation with step-by-step evolutionary generator, IEEE Access 8 (2020) 153113–153122. doi:10.1109/ACCESS.2020.3017881.

[17] M. S. Farid, A. Mahmood, S. A. Al-Maadeed, Multi-focus image fusion using content adaptive blurring, Information fusion 45 (2019) 96–112.

[18] M. Delbracio, I. Garcia-Dorado, S. Choi, D. Kelly, P. Milanfar, Polyblur: Removing mild blur by polynomial reblurring, IEEE Transactions on Computational Imaging 7 (2021) 837–848.

[19] M. Castillo-Cara, R. G. Castro, L. O. Barbosa, TINTO: A novel algorithm for converting tidy data into synthetic image (Dec. 2022). doi:10.5281/zenodo.7520959.

Current code version

Nr.	**Code metadata description**	**Please fill in this column**
C1	Current code version	v1.1.0
C2	Permanent link to code/repository used for this code version	`https://github.com/oeg-upm/TINTO/releases/tag/v1.1.0`
C3	Permanent link to Reproducible Capsule	`https://github.com/oeg-upm/TINTO`
C4	Legal Code License	Apache License, 2.0
C5	Code versioning system used	git
C6	Software code languages, tools, and services used	Python
C7	Compilation requirements, operating environments & dependencies	scipy, sklearn, pandas, matplotlib, argparse, numpy, math, pickle
C8	If available Link to developer documentation/manual	`https://github.com/oeg-upm/TINTO`
C9	Support email for questions	jcastillo@fi.upm.es

Table 2: Code metadata: TINTO software-specific information