

# **Sistema de representación mixto para detección de variedades**

Aprendizaje Automático 3 - Práctica 1

Daniel del Nuevo Montero

**Grado en Inteligencia Artificial**

Curso 2025-2026



Universidad  
Rey Juan Carlos

Escuela Técnica Superior  
Ingeniería Informática

# Índice

<b>1</b>	<b>Introducción</b>	<b>2</b>
<b>2</b>	<b>Autoencoders utilizados</b>	<b>3</b>
2.1	Entrenamiento del <i>autoencoder</i> . . . . .	3
<b>3</b>	<b>Algoritmos de <i>manifold</i> utilizados</b>	<b>5</b>
3.1	<i>Locally Linear Embeddings</i> (LLE) . . . . .	5
3.2	<i>t-Distributed Stochastic Neighbor Embedding</i> (t-SNE) . . . . .	5
<b>4</b>	<b>Proyecto de Python</b>	<b>6</b>
4.1	Diagrama de clases UML del proyecto . . . . .	6
4.2	Ejecución del proyecto . . . . .	6
<b>5</b>	<b>Conjuntos de datos utilizados</b>	<b>7</b>
5.1	MNIST . . . . .	7
5.2	Fashion-MNIST . . . . .	7
5.3	CIFAR-10 . . . . .	7
5.4	Glass-Identification . . . . .	8
<b>6</b>	<b>Experimentos</b>	<b>9</b>
6.1	Experimentos con MNIST . . . . .	9
6.1.1	<i>Embeddings</i> y su <i>trustworthiness</i> para MNIST . . . . .	9
6.1.2	Métricas temporales con MNIST . . . . .	13
6.2	Experimentos con Fashion-MNIST . . . . .	13
6.2.1	<i>Embeddings</i> y su <i>trustworthiness</i> para Fashion-MNIST . . . . .	13
6.2.2	Métricas temporales con Fashion-MNIST . . . . .	17
6.3	Experimentos con CIFAR-10 . . . . .	17
6.3.1	<i>Embeddings</i> y su <i>trustworthiness</i> para CIFAR-10 . . . . .	17
6.3.2	Métricas temporales con CIFAR-10 . . . . .	17
6.4	Experimentos con Glass Identification . . . . .	22
6.4.1	<i>Embeddings</i> y su <i>trustworthiness</i> para Glass Identification . . . . .	22
6.4.2	Métricas temporales con Glass Identification . . . . .	22

## 1 Introducción

La utilización de conjuntos de datos de altas dimensiones, como pueden ser conjuntos de imágenes, para entrenar modelos de aprendizaje automático, puede conllevar a grandes tiempos de cómputo. Como solución, se llevan años utilizando técnicas de *manifold learning* para obtener una representación de los datos en menor dimensión  $d < D$  (siendo  $D$  el número original de dimensiones) conservando hasta cierto punto ciertas propiedades sobre la estructura de los patrones originales. Entre estas técnicas, encontramos algunas como **LLE** (*Locally Linear Embeddings*, [SR03]) y **t-SNE** (*t-distributed Stochastic Neighbor Embedding*, [vdMH08]). El problema con este tipo de algoritmos es que suelen tener tiempos de ejecución demasiado altos para reducir dimensionalidades demasiado altas, lo cual tampoco solucionaría completamente el problema principal. Es por ello que, en muchos casos, una buena opción es utilizar un **autoencoder** para reducir los datos de dimensión  $D$  a  $d$ , y después aplicar un algoritmo de *manifold learning clásico* para reducir de  $d$  a  $d'$  dimensiones. Esto nos aportará:

- Mayor **eficiencia** en el entrenamiento gracias al algoritmo de **backpropagation** de la red neuronal del *autoencoder* y a utilizar *mini-batches*.
- Mayor **velocidad de inferencia** para nuevos ejemplos tras el entrenamiento (las técnicas de *manifold* clásicas utilizan técnicas como la **interpolación**).
- Mayor **robustez** en la representación final, debido a que el *autoencoder* captura características relevantes en la **capa latente**.

## 2 Autoencoders utilizados

Como ya se menciona en la sección 1, el primer paso de este sistema lo lleva a cabo un *autoencoder*. Este consiste en, básicamente, facilitar el trabajo al algoritmo de *manifold learning* clásico. Para poder llevar este proceso a cabo, se ha desarrollado una interfaz `Autoencoder` que sirve como base para implementar varios tipos de *autoencoders*. Los tipos de *autoencoders* especificados para realizar los experimentos son los siguientes:

- `LinearAutoencoder`: *autoencoder* lineal básico (sin utilizar activaciones no lineales como ReLU).
- `LinearSparseAutoencoder`: *autoencoder* lineal que incluye regularización L1 (*sparse*) en el cómputo de la pérdida.
- `DenoisingSparseAutoencoder`: *autoencoder* lineal que incluye regularizaciones L1 (*sparse*) y *denoising*.

La arquitectura especificada consiste en 3 capas lineales en el *encoder*, 32 neuronas en la capa de *embedding* y 3 capas lineales en el *decoder*.

Además, se ha implementado adicionalmente `VariationalAutoencoder`, cuya arquitectura consiste en 4 capas lineales (2 de ellas destinadas a predecir  $\mu$  y  $\log(\sigma^2)$  de la capa de *embedding*) con activaciones ReLU en dos de ellas en el *encoder* y 3 capas lineales con activaciones ReLU en el *decoder*. También implementa el método `_compute_additional_loss` para calcular el término de divergencia Kullback Leibler como pérdida adicional.

### 2.1 Entrenamiento del *autoencoder*

El entrenamiento del *autoencoder*, utiliza, por defecto, **100 epochs**, pérdida **MSE** y **Adam con minibatches** (de tamaño **32**) como optimizador.

Se entrena la red con cada *minibatch* como se observa en la figura 1.

- Se añade ruido a los datos mediante el método `_add_noise`, que cada clase que herede de `Autoencoder` implementará según su propósito.
- Se añade un término de pérdida (L1, KL, etc.) que, al igual que ocurre con `_add_noise`, se implementará según su propósito.

```

def _train_batch(
    self,
    x_batch: torch.Tensor,
    optimizer: torch.optim.Optimizer
) -> None:
    # Ponemos a cero los gradientes de los parámetros al comenzar
    # a entrenar con un batch
    optimizer.zero_grad()

    # Añadimos ruido al batch
    x_batch_tilde = self._add_noise(x_batch)

    # Ejecutamos el forward de la red
    z, recon = self._forward(x_batch_tilde)

    # Obtenemos la pérdida (error de reconstrucción)
    loss = self.loss_fn(recon, x_batch)

    # Calculamos la pérdida total (p. ej. si hay regularización)
    total_loss = loss + self._compute_additional_loss(x_batch, z, recon)

    # Ejecutamos el backward de la red
    total_loss.backward()

    optimizer.step()
    return total_loss.item()

```

Figura 1: Snippet del método `_train_batch`

### 3 Algoritmos de *manifold* utilizados

El sistema está implementado de forma que funcione con los dos algoritmos de *manifold learning* que se requieren en el enunciado, es decir, **LLE** [SR03] y **t-SNE** [vdMH08].

#### 3.1 *Locally Linear Embeddings* (LLE)

El algoritmo LLE busca preservar la topología de los datos originales al proyectarlos en un espacio de menor dimensión, haciendo que las proyecciones de patrones que se encuentran cerca en el espacio original también lo estén. Para lograr esto, primero se representa cada patrón como una combinación ponderada de sus  $k$  vecinos más cercanos y después se utilizan las ponderaciones para calcular la nueva representación.

#### 3.2 *t-Distributed Stochastic Neighbor Embedding* (t-SNE)

A diferencia de LLE, t-SNE crea un patrón al azar en el espacio reducido por cada patrón en el espacio original. Después, calcula un valor de **similitud** entre cada par de patrones, para lo cual emplea una **función de coste**. Por último, se ejecuta un proceso iterativo en que se emplea el **descenso del gradiente** de la función de coste, haciendo que los patrones similares en el espacio original también lo sean en el espacio reducido.

## 4 Proyecto de Python

#### 4.1 Diagrama de clases UML del proyecto

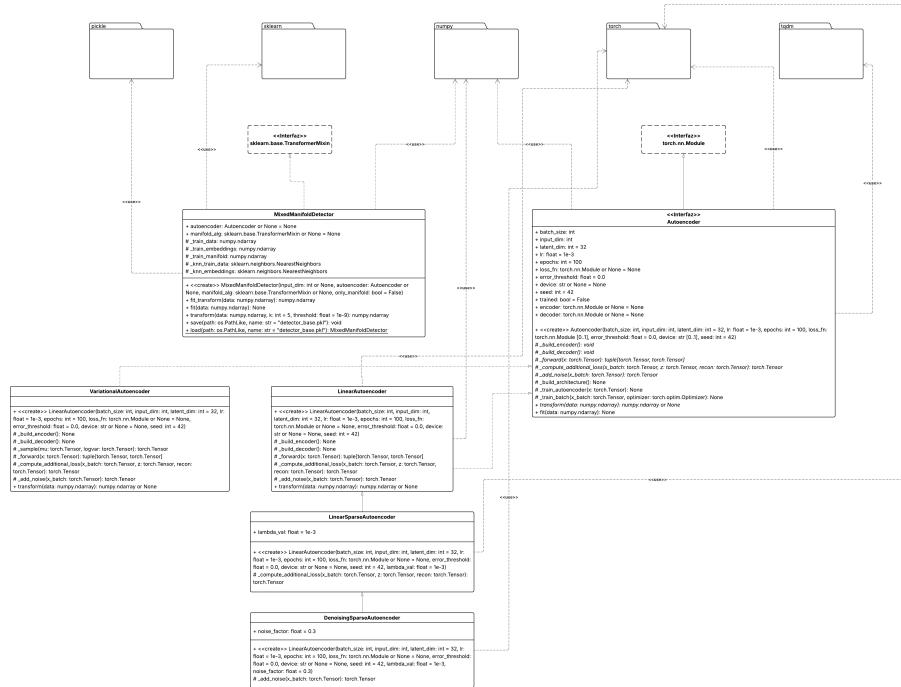


Figura 2: Diagrama de clases del proyecto

El diagrama de clases 2 muestra cómo está estructurado el proyecto (carpeta `src/`) y cómo se relaciona tanto la clase principal `MixedManifoldDetector` como el resto de clases. Para realizarlo, se ha hecho uso de la herramienta LucidChart.

## 4.2 Ejecución del proyecto

Podemos encontrar las instrucciones para ejecutar el proyecto en el fichero `README.md` de la raíz del proyecto. Contamos con varias opciones, como utilizar `venv`, `conda` o `docker`.

## 5 Conjuntos de datos utilizados

Los *datasets* utilizados para recoger métricas y realizar pruebas sobre este sistema son los siguientes:

### 5.1 MNIST

Consiste en un conjunto de 70.000 imágenes de  $28\text{px} \times 28\text{px}$  que puede separarse en un subconjunto de 60.000 imágenes de entrenamiento y 10.000 de *test*. Cada imagen corresponde a un dígito (*label*) del 0 al 9 [LCB98]. Se utiliza mnist\_784

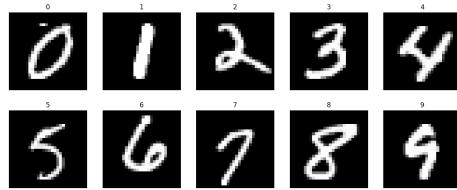


Figura 3: Muestra de cada clase de MNIST

de OpenML.

### 5.2 Fashion-MNIST

Al igual que MNIST (5.1), contiene 70.000 imágenes de  $28\text{px} \times 28\text{px}$  que puede separarse en 60.000 imágenes para entrenamiento y 10.000 para *test*. En este caso, cada imagen corresponde a una prenda de ropa (*label*) codificada como un dígito del 0 al 9 [XRV17]. Se utiliza Fashion-MNIST de OpenML.

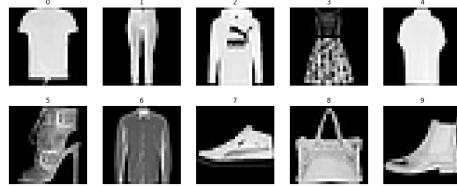


Figura 4: Muestra de cada clase de Fashion-MNIST

### 5.3 CIFAR-10

Está formado por un total de 60.000 imágenes de  $32\text{px} \times 32\text{px}$  que puede separarse en 50.000 imágenes de entrenamiento y 10.000 de *test*. Cada imagen corresponde a una clase (*label*) codificada como un entero de 0 a 9 [Kri09]. Se utiliza cifar\_10\_small de OpenML que, cabe recalcar, tiene **20.000 patrones** muestreados aleatoriamente CIFAR-10 original (que tiene 60.000 patrones como ya se especificó previamente).



Figura 5: Muestra de cada clase de CIFAR-10

#### 5.4 Glass-Identification

Este *dataset* recoge parámetros (*refractive index*, *sodium*, etc.) de 214 muestras de cristales. Cada instancia tiene asociado un entero del 0 al 6, correspondiendo cada uno a un tipo de cristal (*label*) [Ger87]. Se utiliza Glass-Classification de OpenML.

## 6 Experiments

Para realizar el *testing* del proyecto se especifica realizar las combinaciones entre los siguientes elementos:

- Autoencoder:
  - **LinearAutoencoder**: *autoencoder* lineal con 3 capas lineales en el *encoder*, una capa de 32 neuronas para el *embedding* y 3 capas lineales en el *decoder*.
  - **LinearSparseAutoencoder**: *autoencoder* lineal con regularización *sparse*.
  - **DenoisingSparseAutoencoder**: *autoencoder* lineal con regularizaciones *denoising* y *sparse*.
  - **VariationalAutoencoder**: *autoencoder* variacional con regularización KL.
- Algoritmo de *manifold learning*:
  - LLE (con 2 componentes).
  - t-SNE (con 2 componentes).
- Conjunto de datos: se utilizan los 4 conjuntos de datos explicados en la sección 5 para crear subconjuntos de datos de 3 tamaños:
  - Subconjunto pequeño: 300 patrones para *train* y 50 para *test*.
  - Subconjunto mediano: 1000 patrones para *train* y 200 para *test*.
  - Subconjunto grande: 5000 patrones para *train* y 1000 para *test*.

Cabe recalcar que, para el conjunto Glass Identification, al tener únicamente 214 patrones, en lugar de crear 3 subconjuntos de distintos tamaños, se utilizan diferentes permutaciones aleatorias del conjunto completo. Adicionalmente, se realizan las pruebas mencionadas utilizando el detector con únicamente los algoritmos de *manifold learning*. De este modo, contamos con un total de 120 experimentos con subconjuntos de datos y ascendería a 160 si ejecutásemos las pruebas con los conjuntos de datos completos (implementadas también).

### 6.1 Experiments con MNIST

#### 6.1.1 *Embeddings* y su *trustworthiness* para MNIST

En los subconjuntos pequeños, podemos observar que la combinación de *autoencoder* variacional con cualquiera de los algoritmos de *manifold learning* obtiene malos resultados, mientras que tanto el resto de combinaciones como los algoritmos de *manifold* por sí solos obtienen valores de *trustworthiness*  $> 0.7$ . En cuanto a los medianos, se observa un comportamiento muy similar por parte de los detectores con *autoencoder* variacional, con valores de *trustworthiness* cercanos a 0.5. El resto de combinaciones obtienen resultados mejores que con los

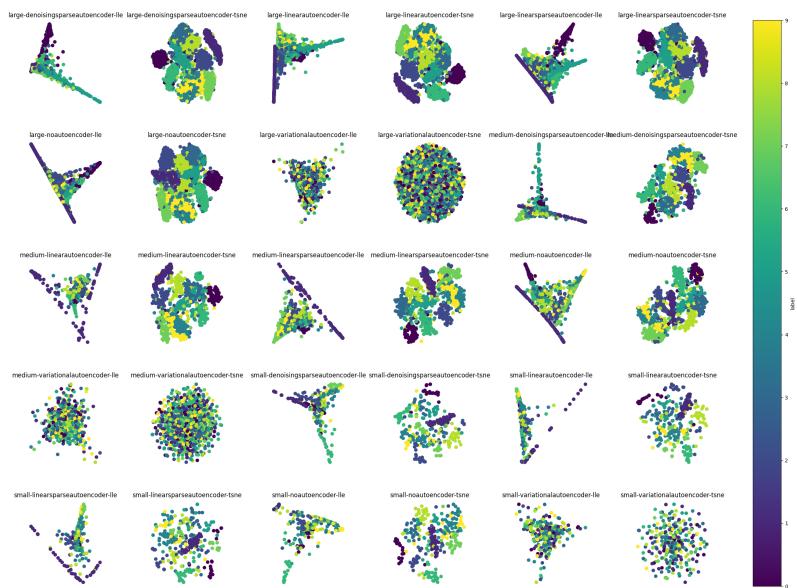


Figura 6: *Embeddings* de los subconjuntos de entrenamiento de MNIST

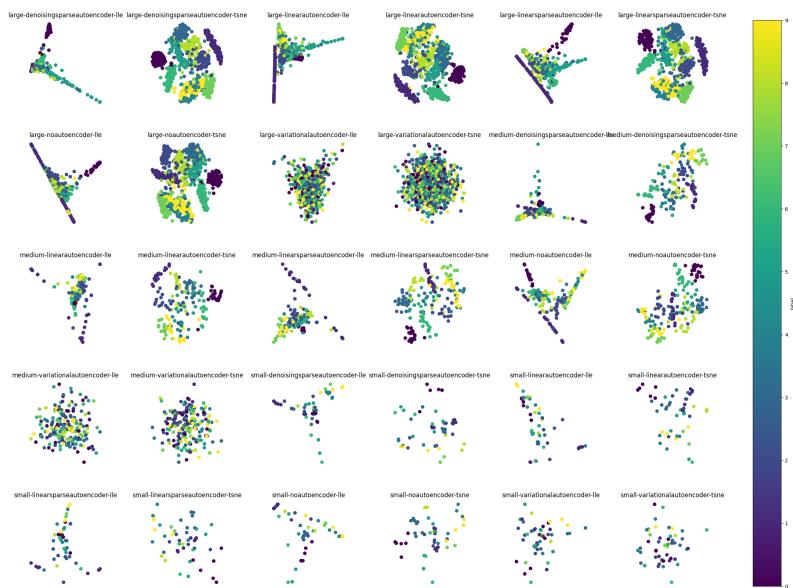


Figura 7: *Embeddings* de los subconjuntos de *test* de MNIST

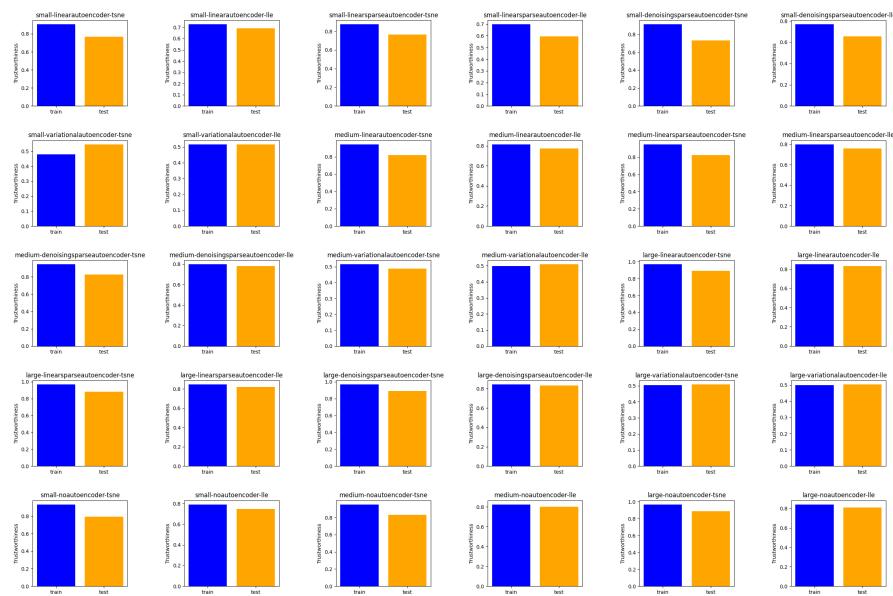


Figura 8: *Trustworthiness* de los subconjuntos de *test* de MNIST

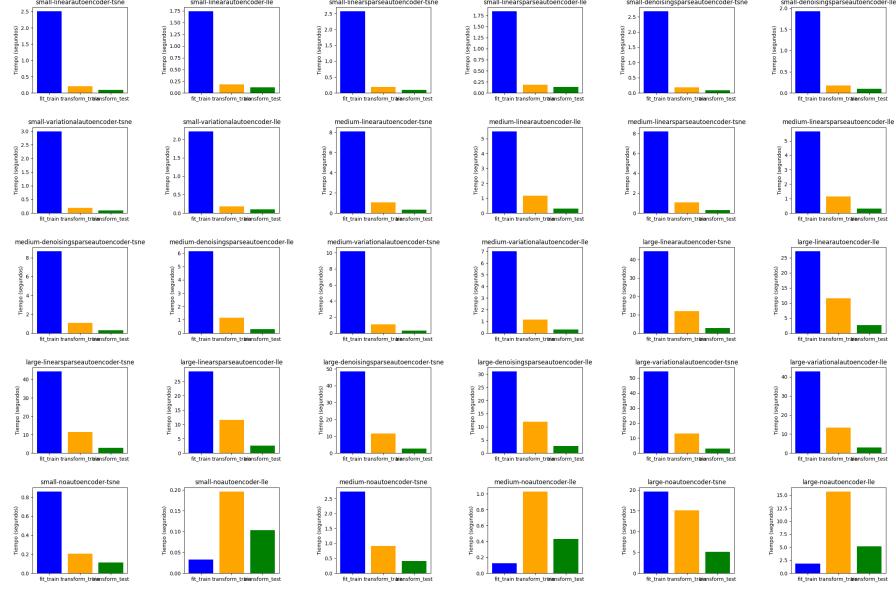


Figura 9: Tiempos de entrenamiento y de transformación para los subconjuntos de *test* de MNIST

subconjuntos pequeños. En los subconjuntos grandes, se observa una mejora sustancial, destacando el detector compuesto por un *denoising sparse autoencoder* y t-SNE.

### 6.1.2 Métricas temporales con MNIST

Como es de esperar, los tiempos aumentan conforme lo hace el tamaño del subconjunto de datos. Debido a su relación *trustworthiness*-tiempo, destacaría el detector compuesto por el *linear sparse autoencoder* y t-SNE.

## 6.2 Experimentos con Fashion-MNIST

### 6.2.1 *Embeddings* y su *trustworthiness* para Fashion-MNIST

Generalmente, se observa un mejor rendimiento en los diferentes subconjuntos muestreados de Fashion-MNIST en comparación con MNIST, tanto en los detectores con *autoencoder* como en los algoritmos de *manifold learning* por sí solos.

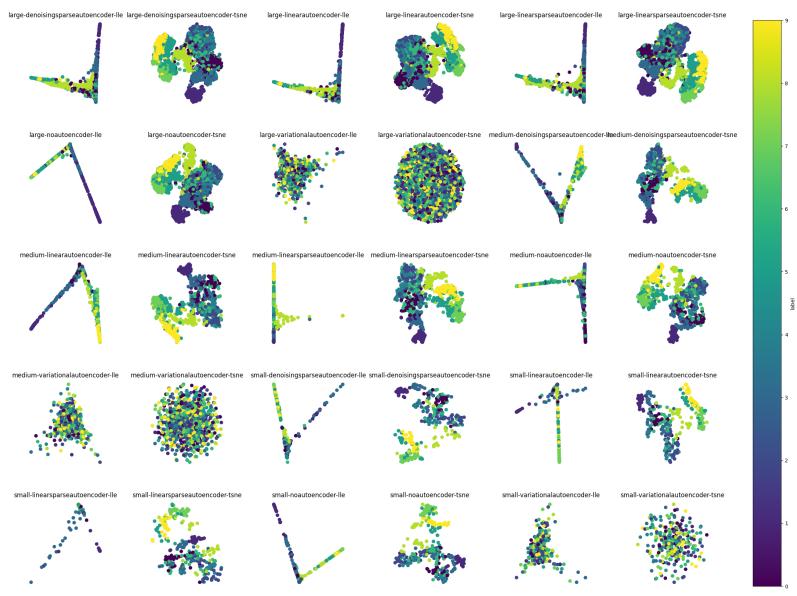


Figura 10: *Embeddings* de los subconjuntos de entrenamiento de Fashion MNIST

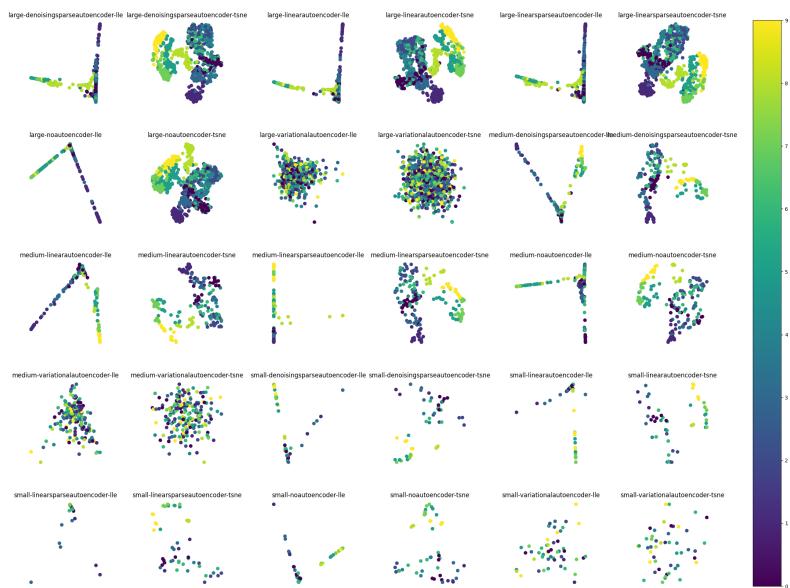


Figura 11: *Embeddings* de los subconjuntos de *test* de Fashion MNIST

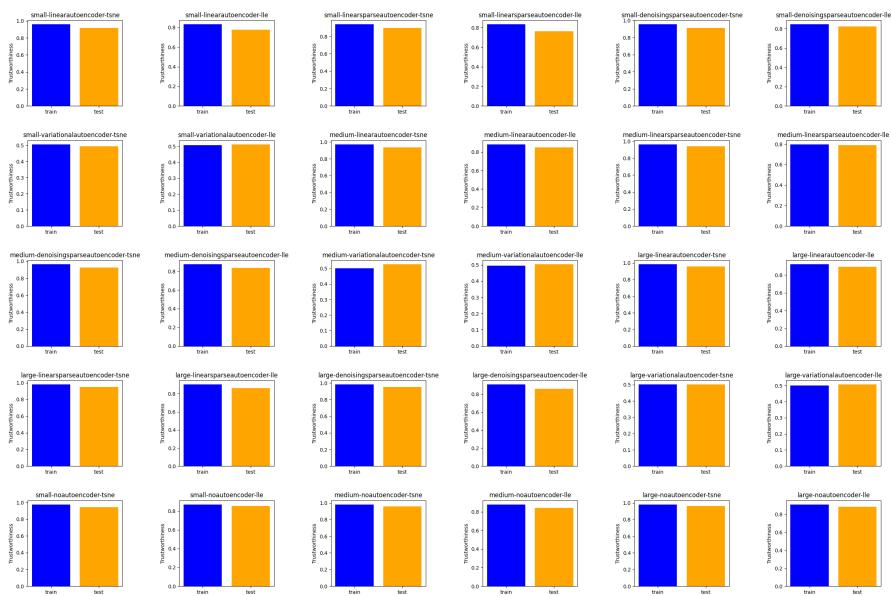


Figura 12: *Trustworthiness* de los subconjuntos de *test* de Fashion MNIST

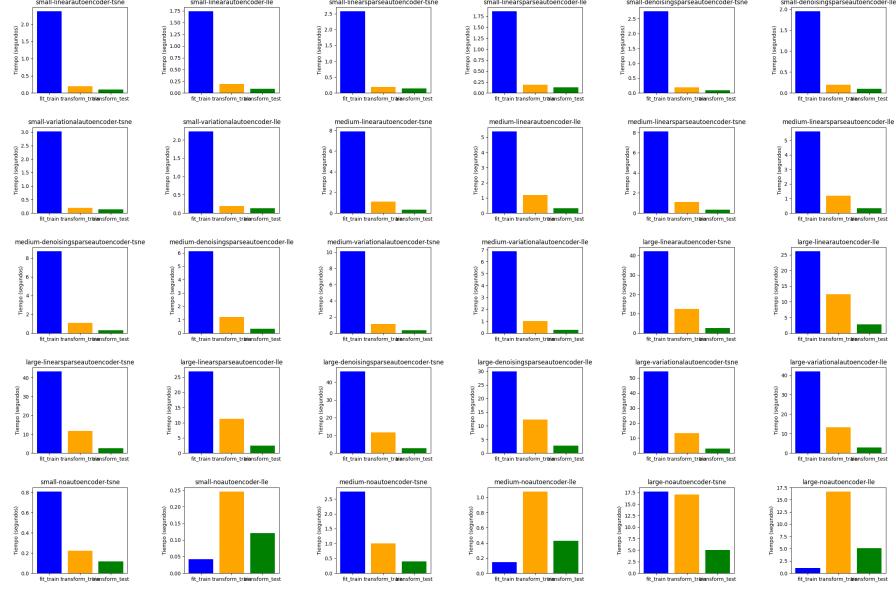


Figura 13: Tiempos de entrenamiento y de transformación para los subconjuntos de *test* de Fashion MNIST

### 6.2.2 Métricas temporales con Fashion-MNIST

Los tiempos de *fit* y *transform* son similares a los obtenidos con MNIST, aunque ligeramente inferiores. En este caso, destacaría la combinación del *linear sparse autoencoder* con LLE ya que tiene tiempos muy bajos y valores de *trustworthiness* de  $\approx 0.85$  con el subconjunto más grande.

## 6.3 Experimentos con CIFAR-10

### 6.3.1 *Embeddings* y su *trustworthiness* para CIFAR-10

Se observa que los valores de *trustworthiness* obtenidos con CIFAR-10 son bastante similares, manteniéndose en un rango de 0.6 a 0.8 para todas las combinaciones y subconjuntos.

### 6.3.2 Métricas temporales con CIFAR-10

Los tiempos, en este caso, tienden a ser mayores, debido a que este conjunto de datos tiene un mayor número de características, ya que cuenta con los tres

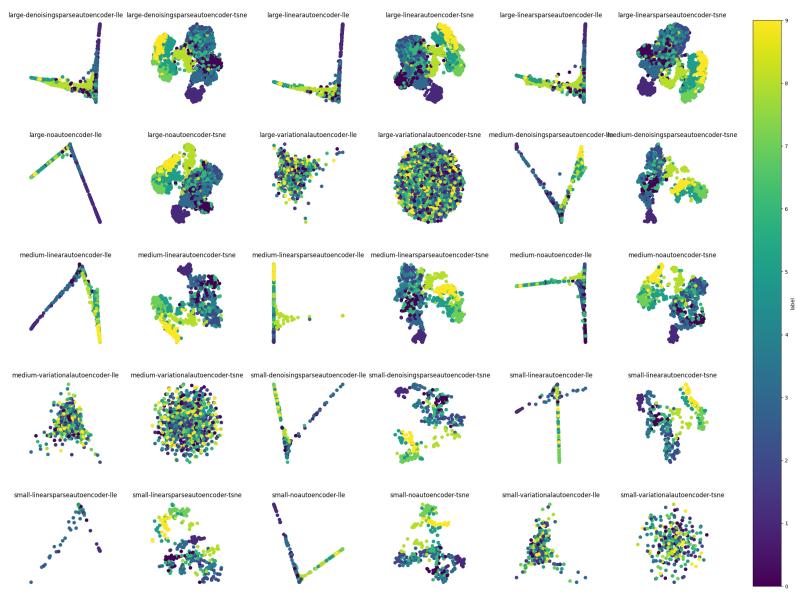


Figura 14: *Embeddings* de los subconjuntos de entrenamiento de CIFAR-10

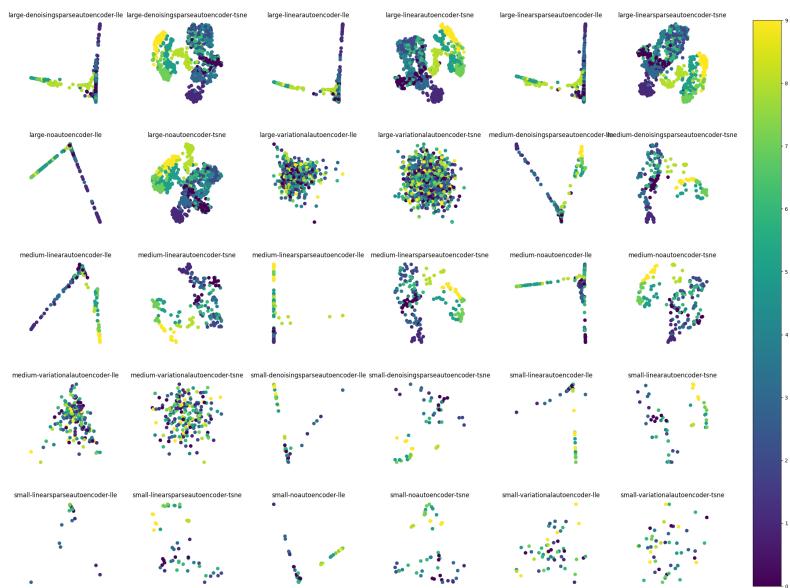


Figura 15: *Embeddings* de los subconjuntos de *test* de CIFAR-10

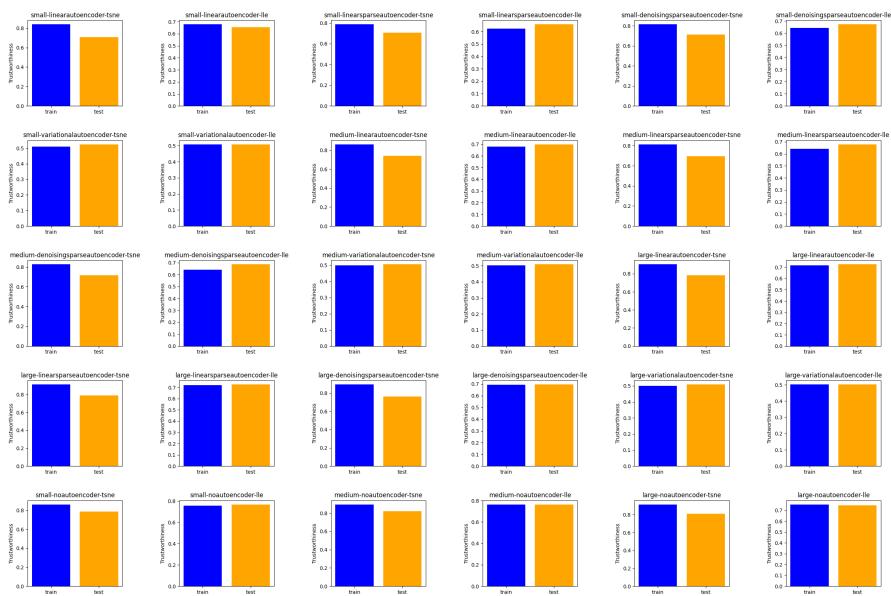


Figura 16: *Trustworthiness* de los subconjuntos de *test* de CIFAR-10

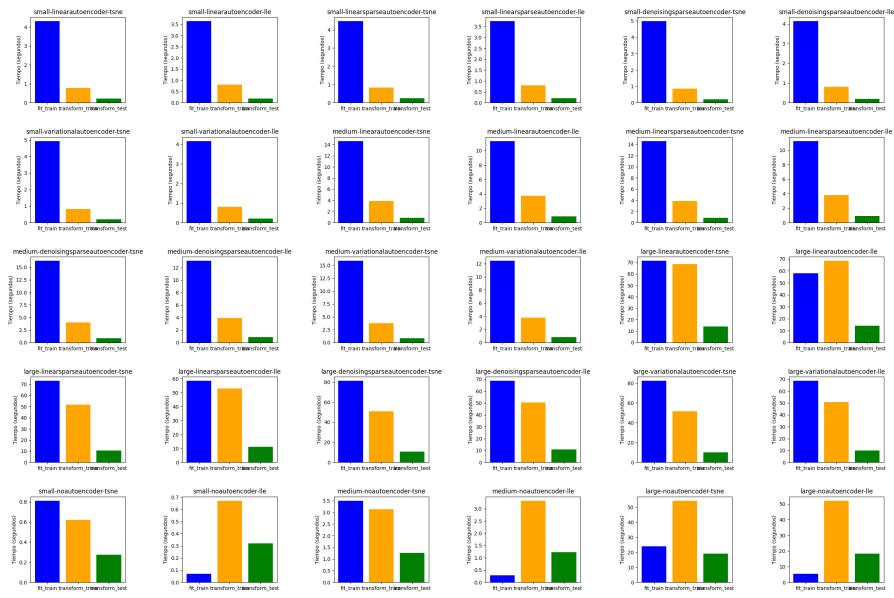


Figura 17: Tiempos de entrenamiento y de transformación para los subconjuntos de *test* de CIFAR-10

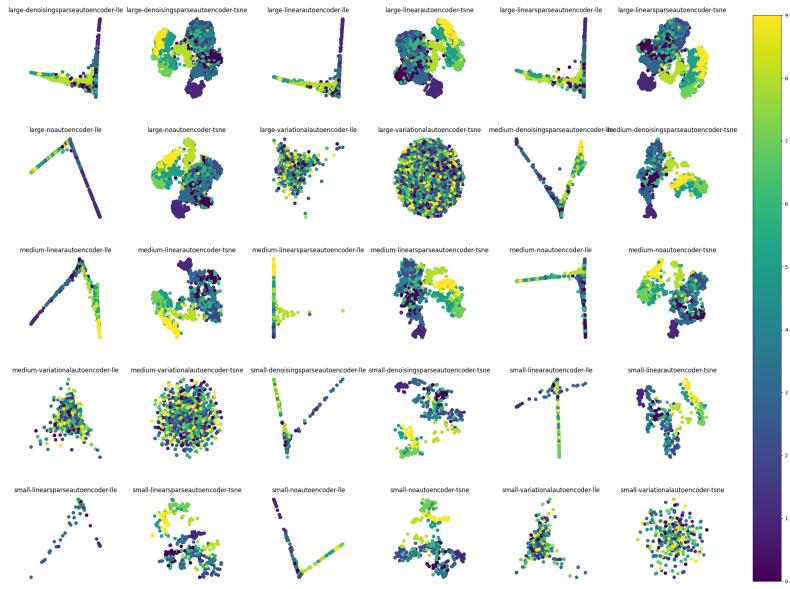


Figura 18: *Embeddings* de los subconjuntos de entrenamiento de Glass Identification

canales de color de las imágenes (aplanados).

## 6.4 Experimentos con Glass Identification

### 6.4.1 *Embeddings* y su *trustworthiness* para Glass Identification

Vemos que los mejores valores de *trustworthiness* se obtienen, de nuevo, con los detectores que combinan un *linear sparse autoencoder* con un algoritmo de *manifold learning*, destacando el *denoising sparse autoencoder* con t-SNE.

### 6.4.2 Métricas temporales con Glass Identification

Los tiempos son mucho más bajos (generalmente < 1 segundo) debido al reducido tamaño del conjunto de datos y al bajo número de características.

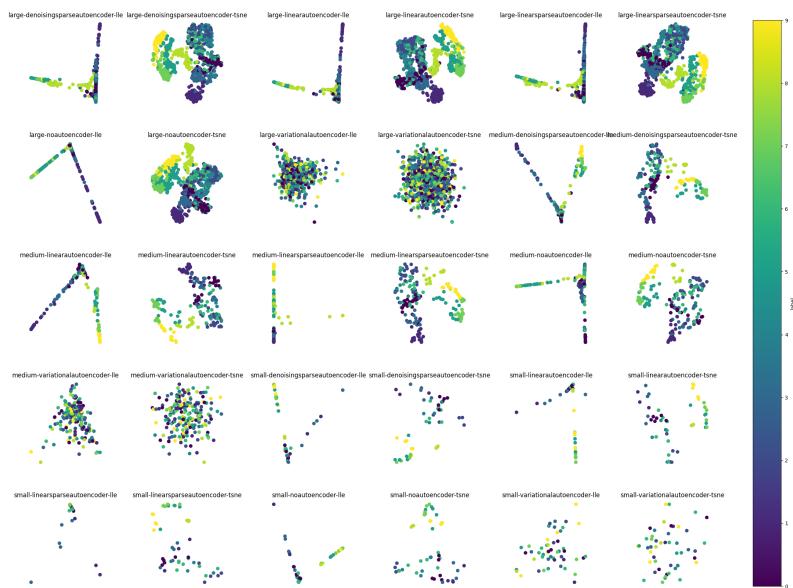


Figura 19: *Embeddings* de los subconjuntos de *test* de Glass Identification

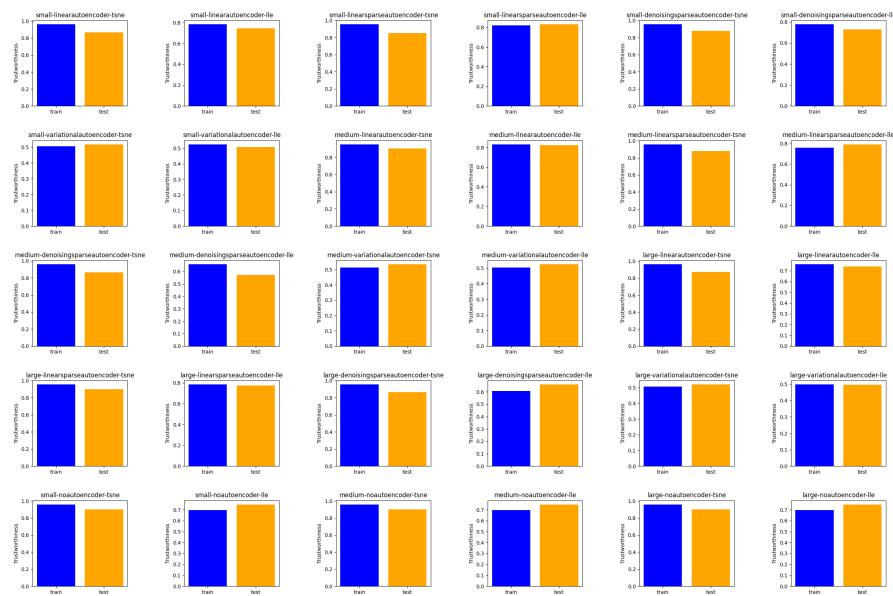


Figura 20: *Trustworthiness* de los subconjuntos de *test* de Glass Identification

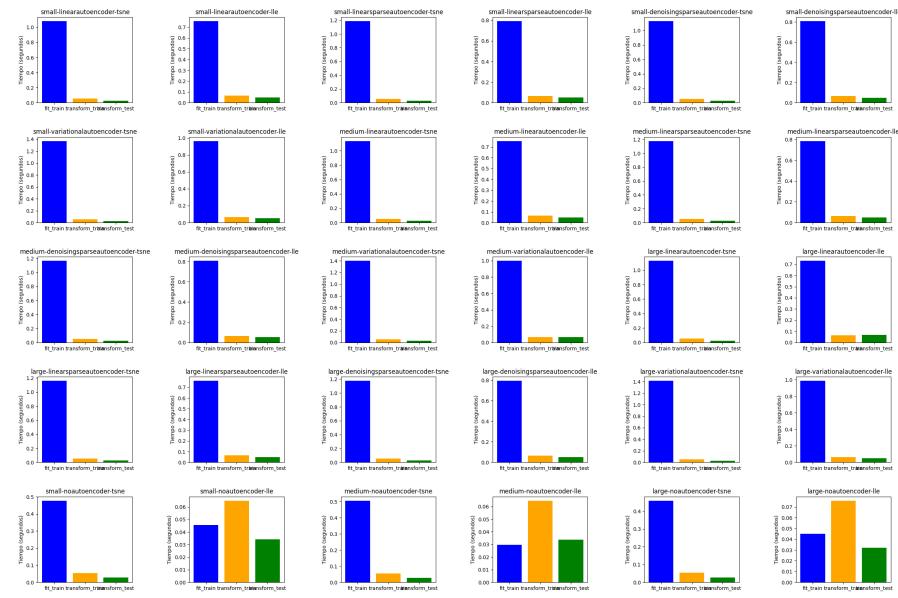


Figura 21: Tiempos de entrenamiento y de transformación para los subconjuntos de *test* de Glass Identification

## Referencias

- [Ger87] B. German. Glass Identification. UCI Machine Learning Repository, 1987. DOI: <https://doi.org/10.24432/C5WW2P>.
- [Kri09] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.
- [LCB98] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. The mnist database of handwritten digits. <http://yann.lecun.com/exdb/mnist/>, 1998.
- [SR03] Lawrence K. Saul and Sam T. Roweis. Think globally, fit locally: Unsupervised learning of low dimensional manifolds. *Journal of Machine Learning Research*, 4(Jun):119–155, 2003.
- [vdMH08] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.
- [XRV17] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:1708.07747, 2017. cs.LG.