# SIA - TP5

Autoencoders

# Ejercicio 1.a

# Autoencoder

# Problemática

El objetivo es diseñar un autoencoder que permita aprender un dataset provisto para luego poder generar nuevos elementos que podrían pertenecer a ese conjunto

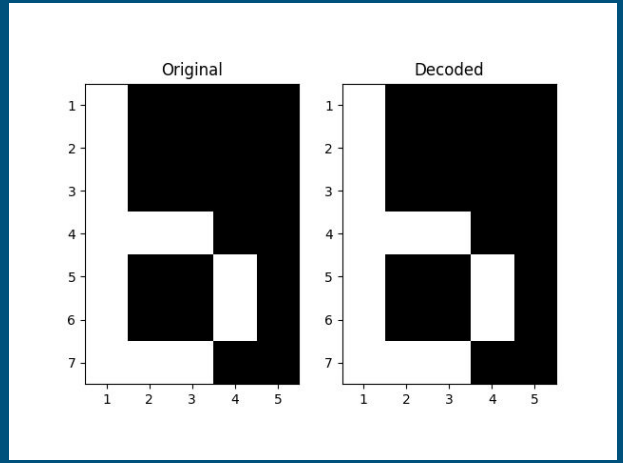
- Arquitectura del autoencoder
- Exploración del espacio latente
- Capacidad de generar
- Implementación de un Denoising Autoencoder

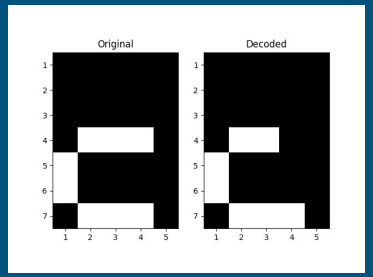
# Resultados (I) - Entrenamiento

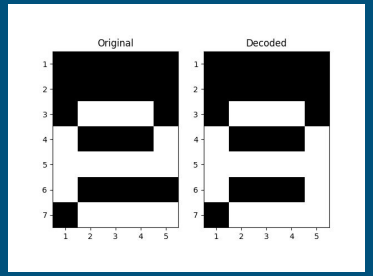
- Arquitectura: [35, 10, 2, 10, 35]
- Thetas: Sigmoid + Tanh final
- Épocas: 5000
- Eta: 0.01
- Error en píxeles totales: 14
- Caracteres totales reconocidos: 28/32
- Optimizador: Adam

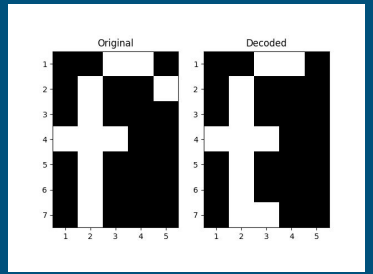
**Caracteres con error en más de un pixel:**

- x (3)
- f (2)
- k (2)
- g (2)

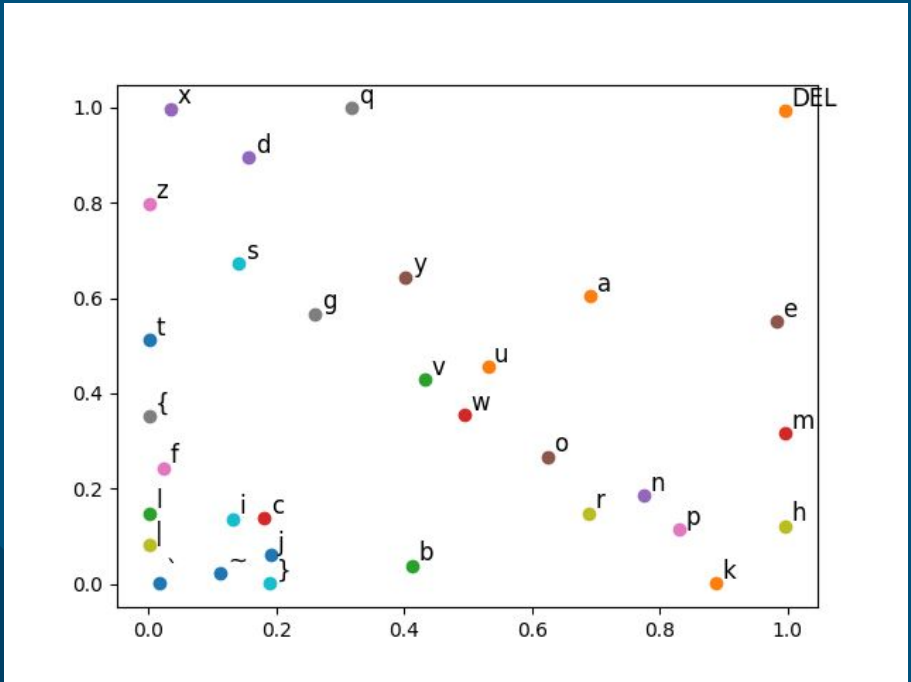


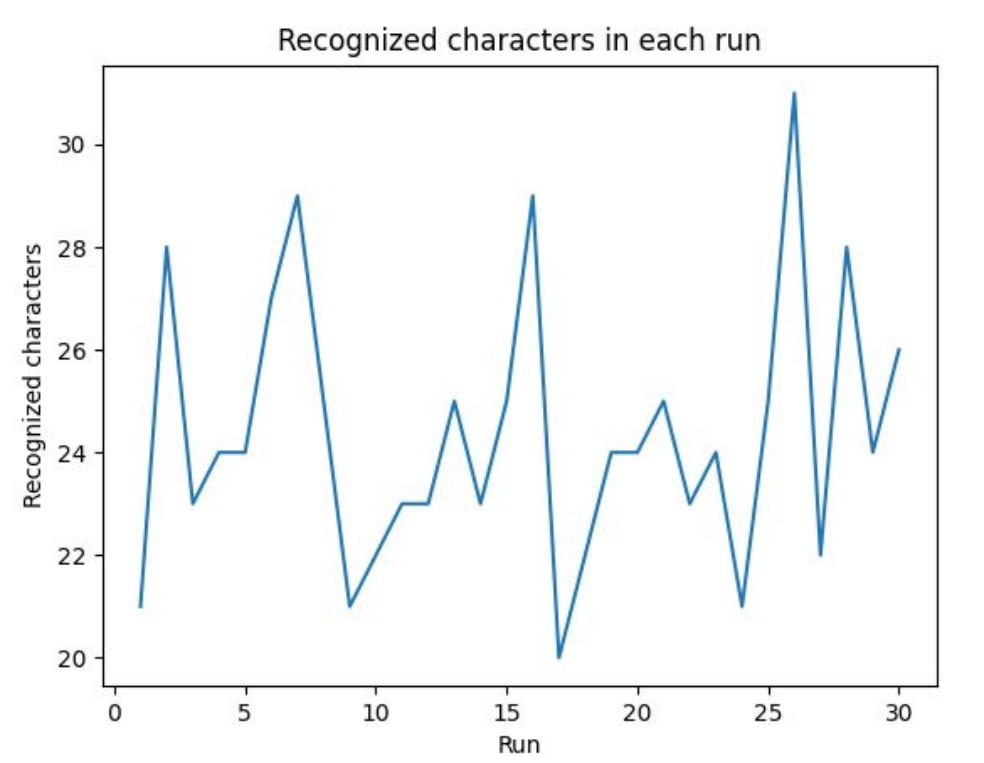



# Resultados (I) - Espacio Latente



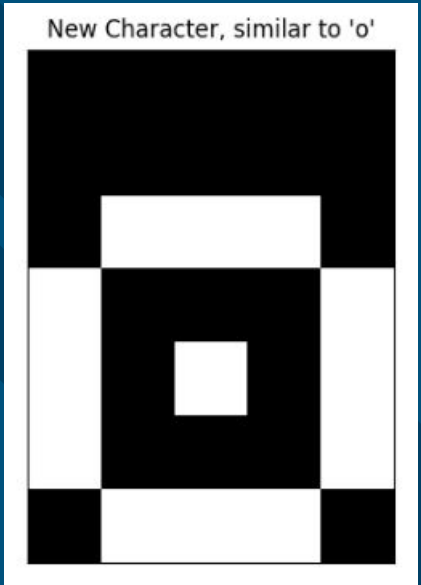

[35, 10, 2, 10, 35]

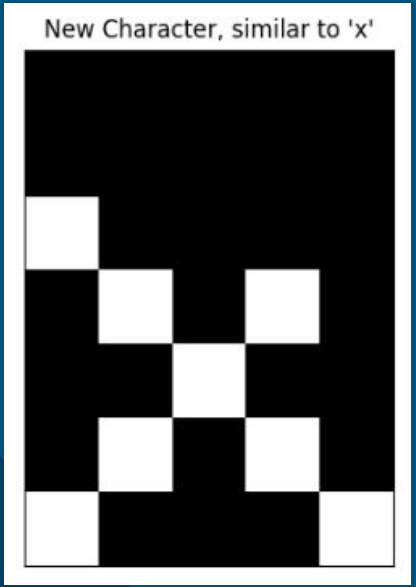
# Resultados (II) - Entrenamiento

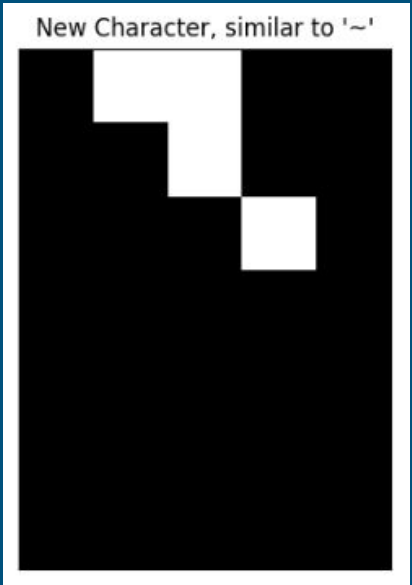


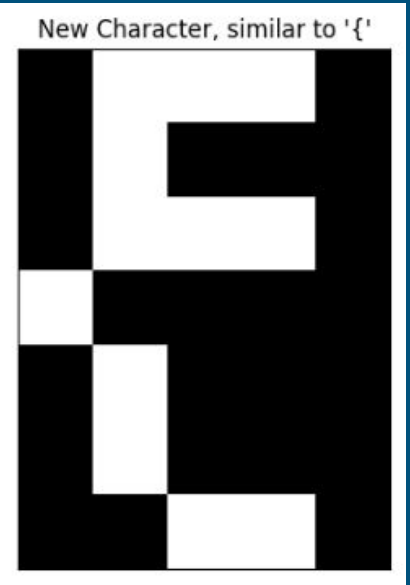
# Generando un nuevo elemento

## Offset de +0.1: (x+0.1, y+0.1)



# Conclusiones (I)

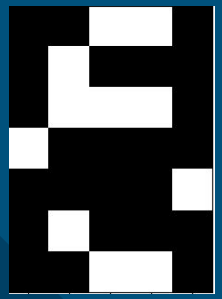
- ➔ No se encontró empíricamente un conjunto de parámetros que logre llevar al autoencoder a reconocer los 32 caracteres la mayoría de las veces.
- ➔ Poder graficar el espacio latente en 2D o 3D facilita nuestra comprensión. Mostrándonos rápidamente outliers o caracteres similares.
- ➔ El poder de generación no es tan potente sin la “conquista del espacio latente”. ¡Ver más adelante!

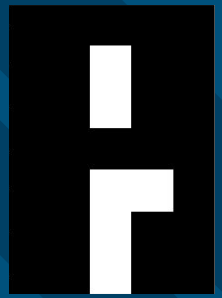
# Ejercicio 1.b

# Denoising Autoencoder

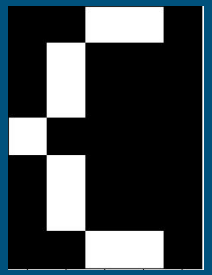
# Entrenamiento

1. **Entrada**: Caracter ruidoso

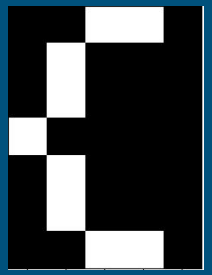




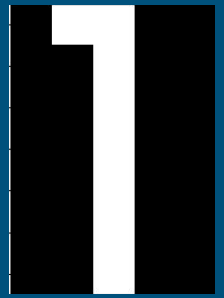
2. **Error**: Salida vs. Caracter sin ruido



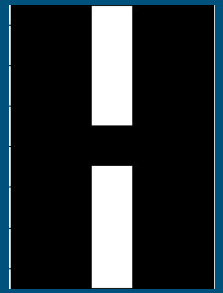
vs



✅



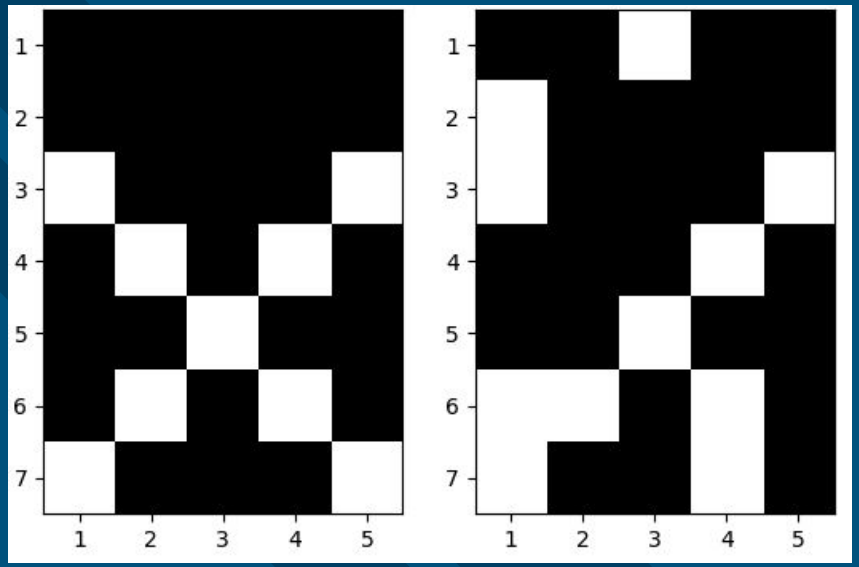
vs

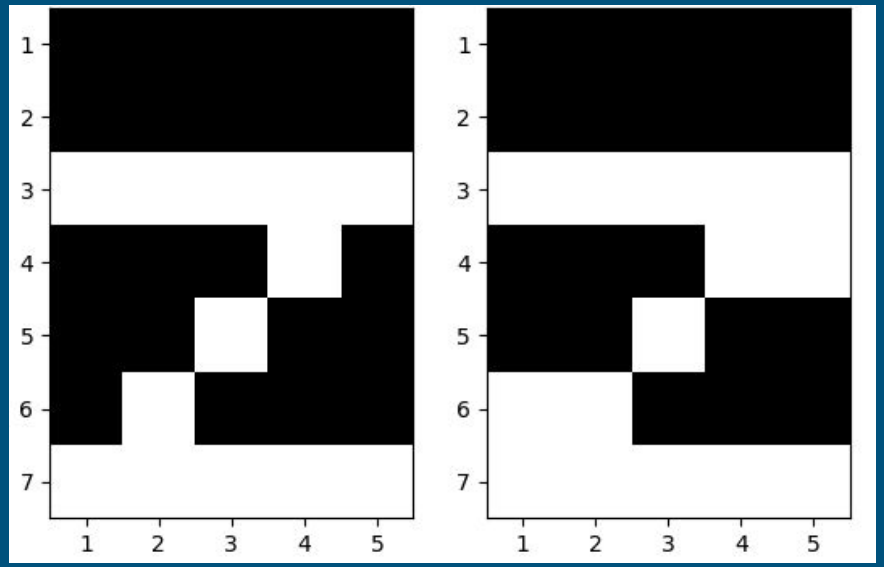


❌

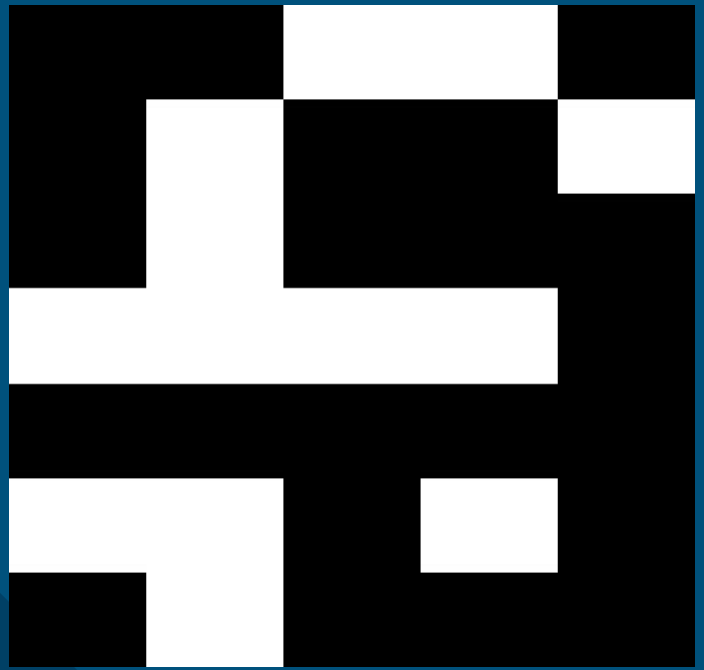
# Ruido: Salt & Pepper

- ➔ Ejemplo con 10% probabilidad de Salt y Pepper
- ➔ Letra ‘X’ y ‘Z’



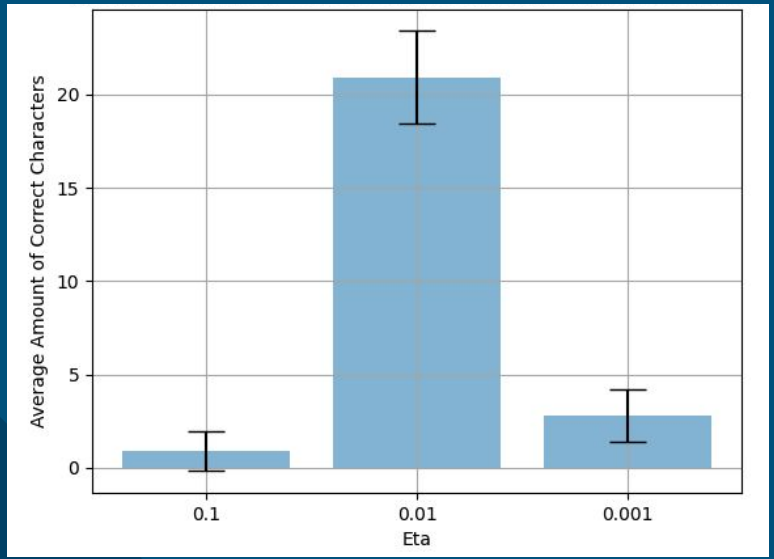



# Entrenamiento
# La entrada a través de las épocas



# Elección de Tasa de Aprendizaje

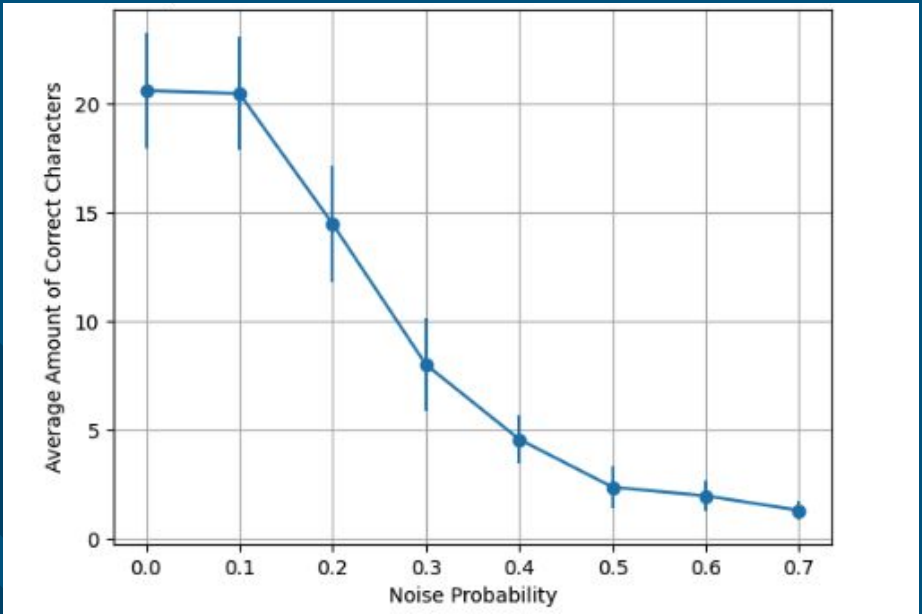
- Queremos una tasa que en tiempo razonable logre buenos resultados
- Cantidad de épocas: 3000 (10s por corrida)
- Promedio de ✨30✨ Corridas
- Probabilidad Salt & Pepper 10%




➔ ***Tomamos eta $10^{-2}$***

# # de Caracteres Correctos Vs. Nivel de Ruido

- Nivel de Ruido afectado por probabilidad de Salt & Pepper
- Cantidad de épocas: 3000 (10s por corrida)
- Tasa de Aprendizaje: $10^{-2}$
- Promedio de ✨30✨ Corridas

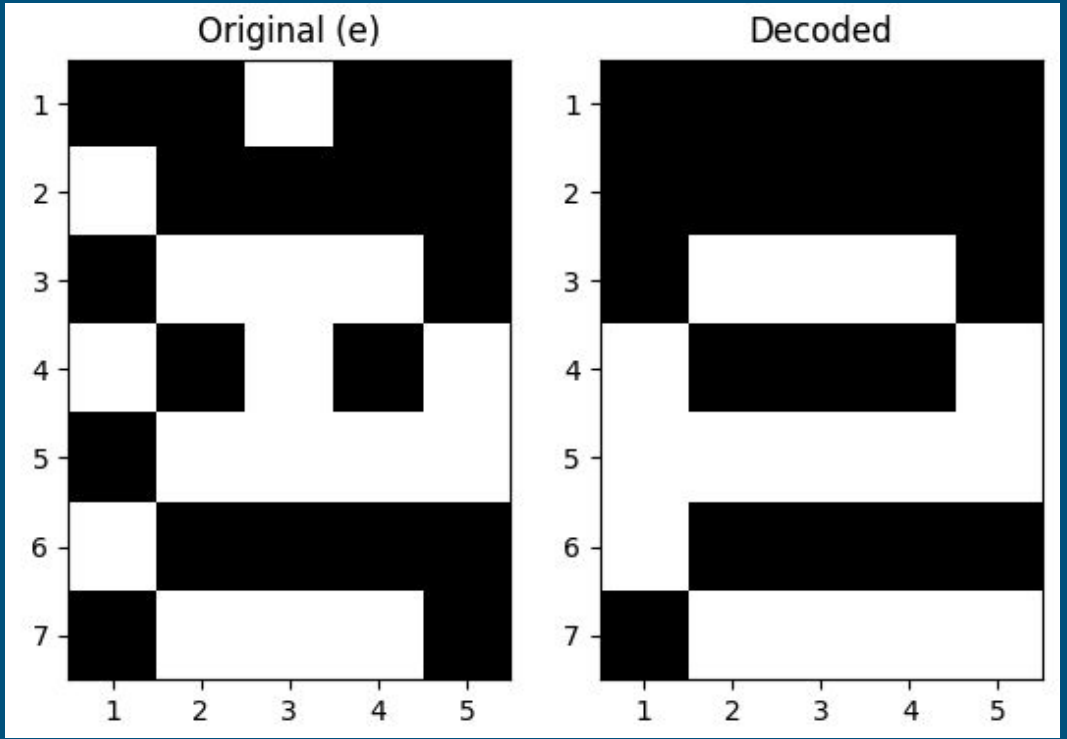


# Conclusiones (I)

- ➔ Aumentar la probabilidad de S&P por un nivel superior al 20% puede provocar que los elementos muten a algo irreconocible.
- ➔ Lo mismo para cualquier otra función de ruido que provoque lo mismo, analizadas en TPs anteriores.
- ➔ Se observó un buen nivel de reconocimiento hasta 10% de ruido.

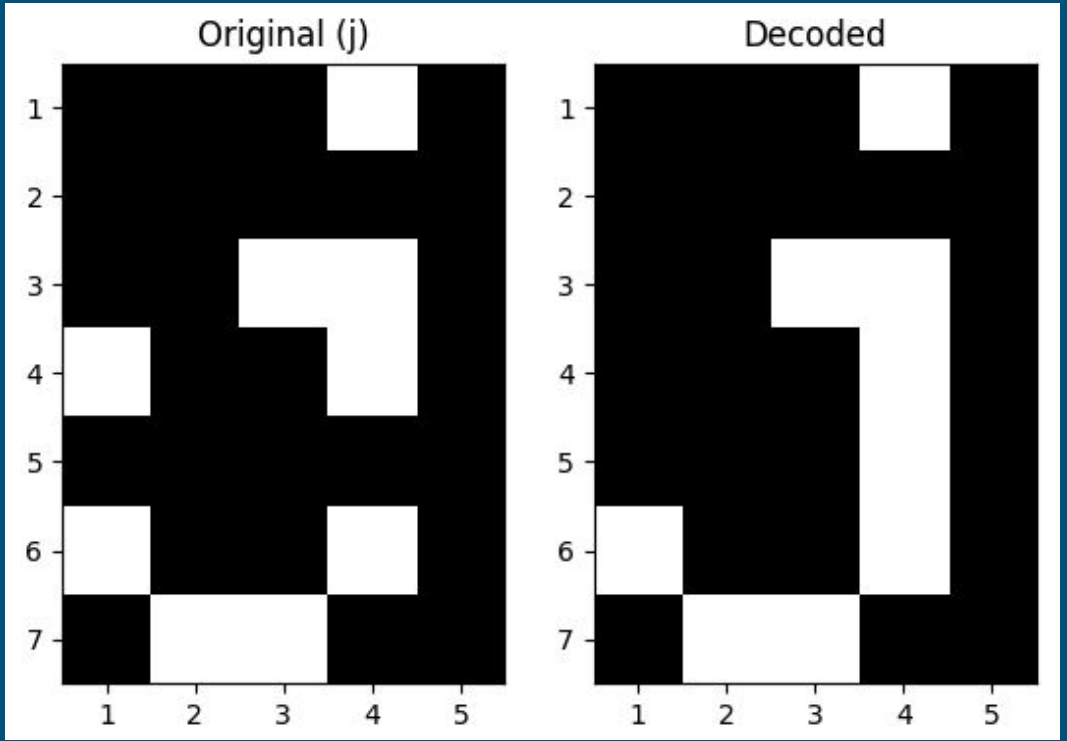
# Capacidad de Denoising

- Salt & Pepper 10%
- Cantidad de épocas: 9000
- Tasa de Aprendizaje: $10^{-2}$



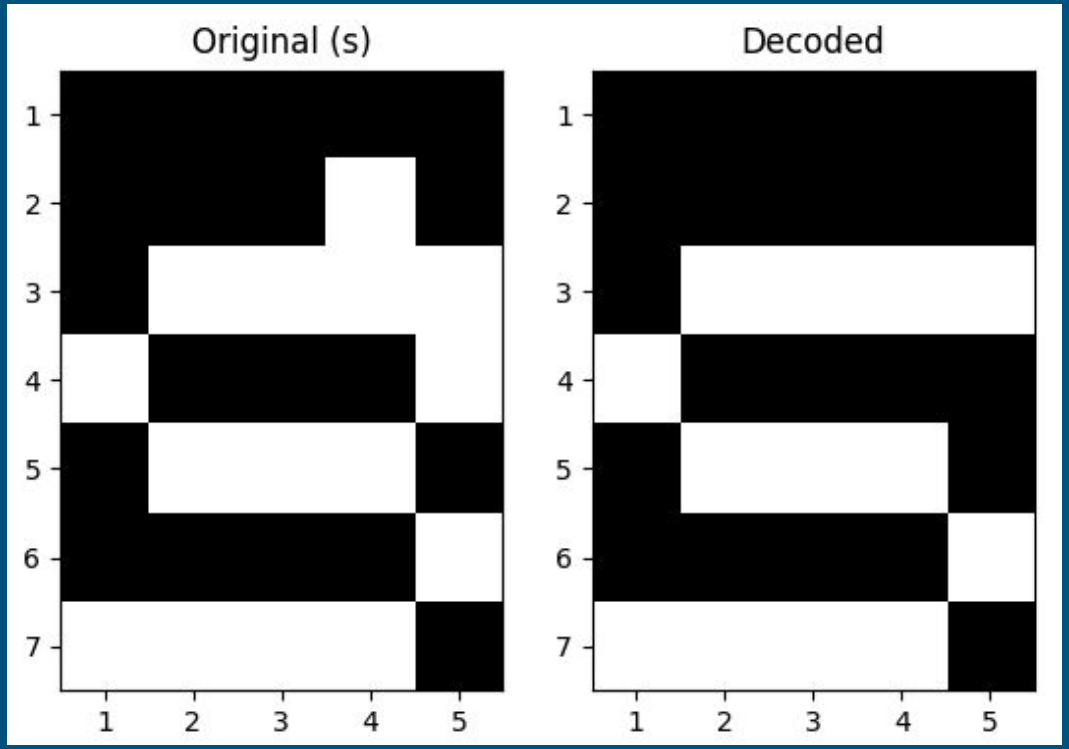
# Capacidad de Denoising

- Salt & Pepper 10%
- Cantidad de épocas: 9000
- Tasa de Aprendizaje: $10^{-2}$



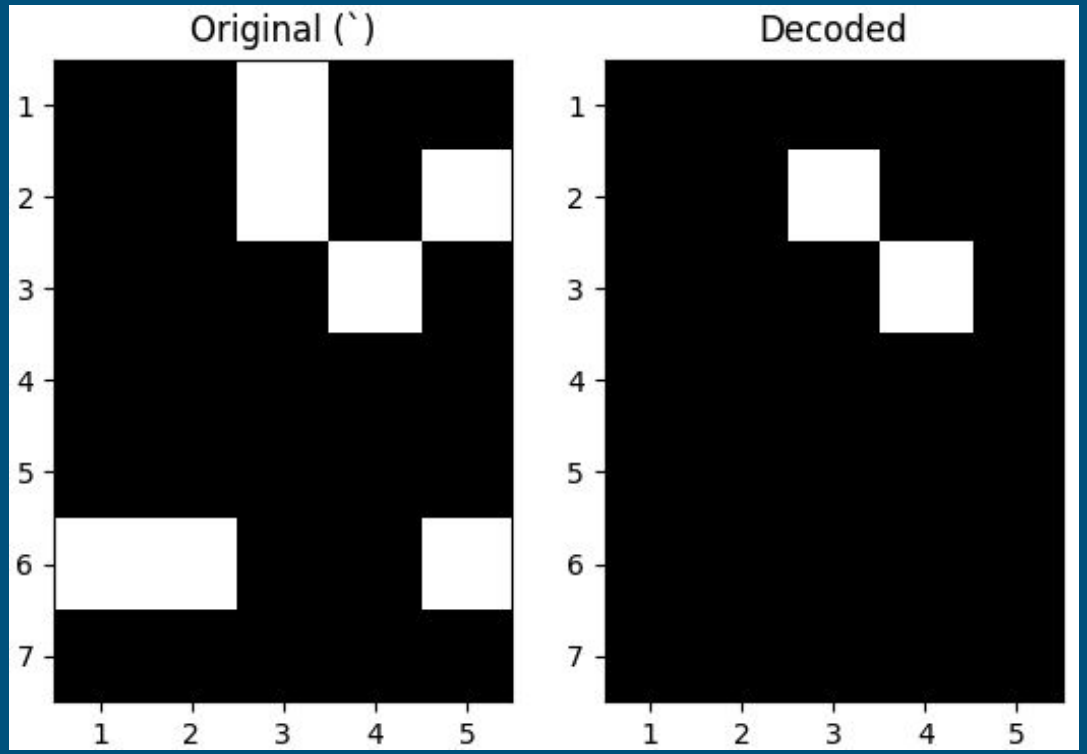
# Capacidad de Denoising

- Salt & Pepper 10%
- Cantidad de épocas: 9000
- Tasa de Aprendizaje: $10^{-2}$



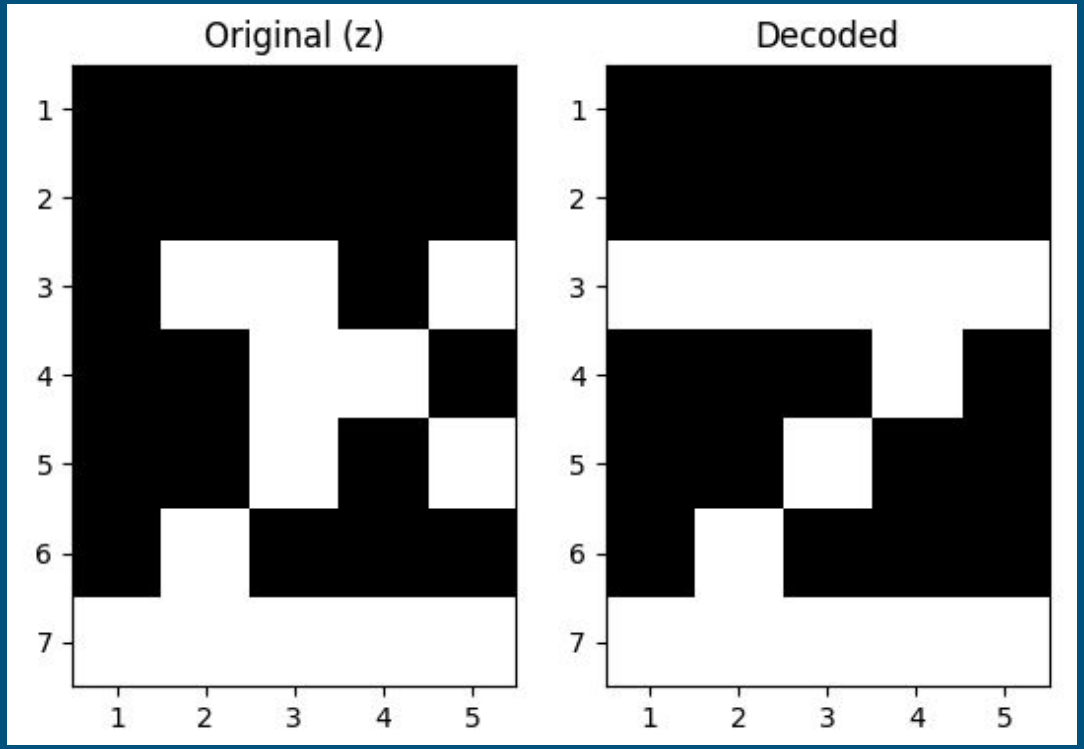
# Capacidad de Denoising

- Salt & Pepper 10%
- Cantidad de épocas: 9000
- Tasa de Aprendizaje: $10^{-2}$



# Capacidad de Denoising

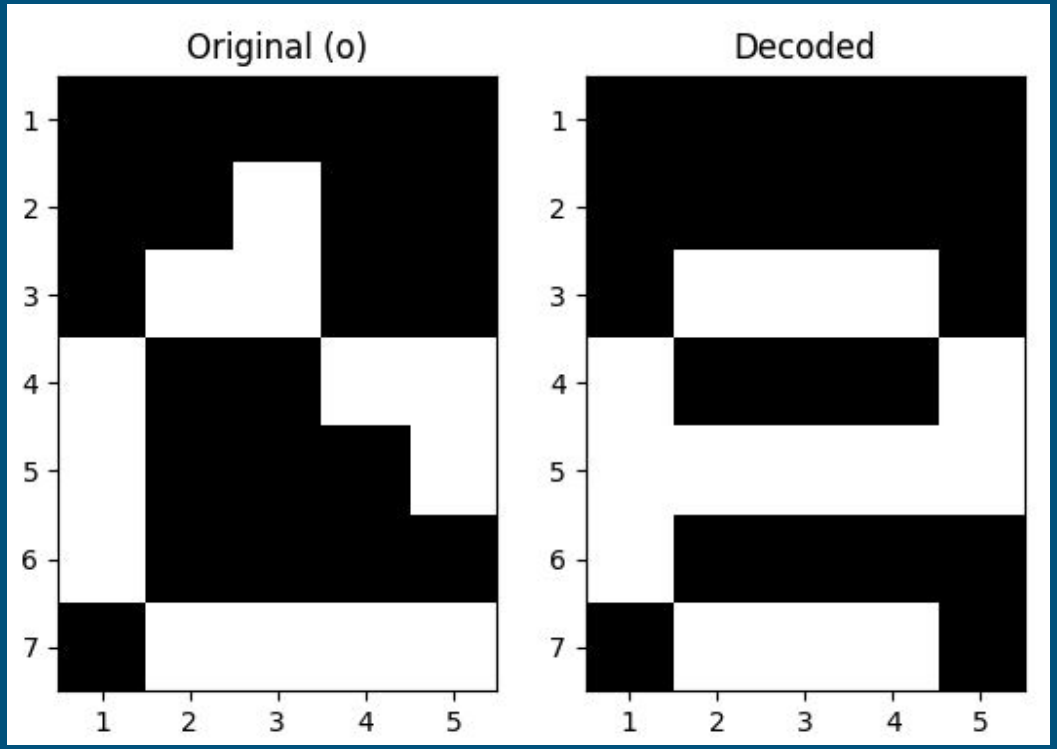
- Salt & Pepper 10%
- Cantidad de épocas: 9000
- Tasa de Aprendizaje: $10^{-2}$



# Capacidad de Denoising

- Salt & Pepper 10%
- Cantidad de épocas: 9000
- Tasa de Aprendizaje: $10^{-2}$

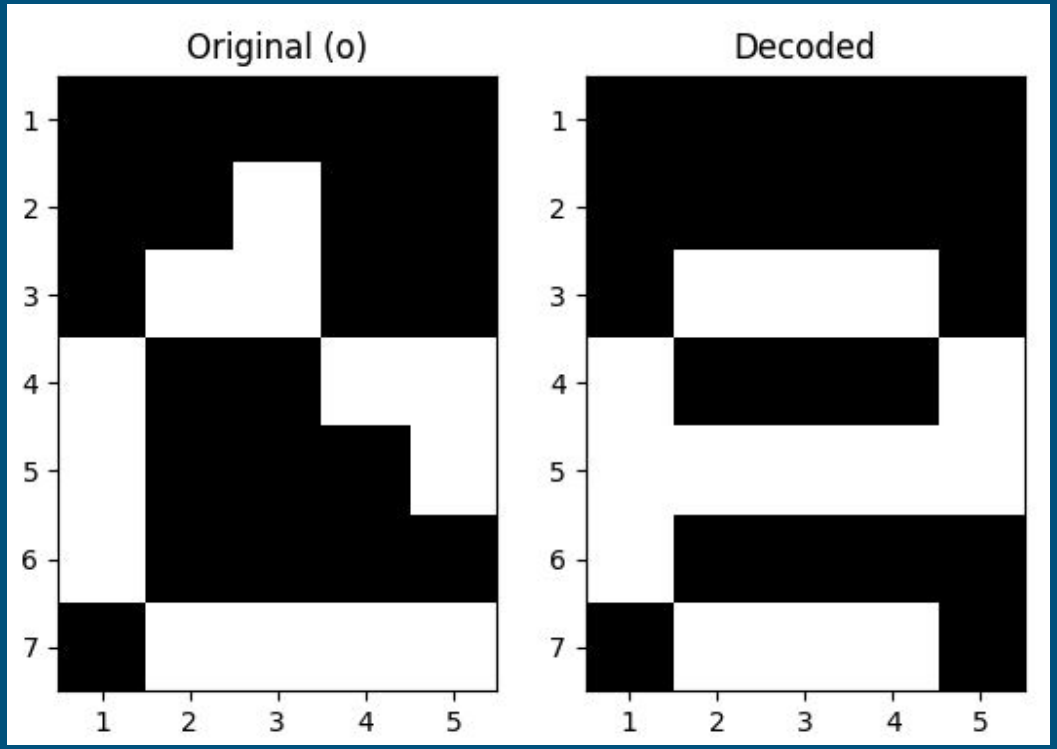
Se equivoca de letra 😞



# Capacidad de Denoising

- Salt & Pepper 10%
- Cantidad de épocas: 9000
- Tasa de Aprendizaje: $10^{-2}$

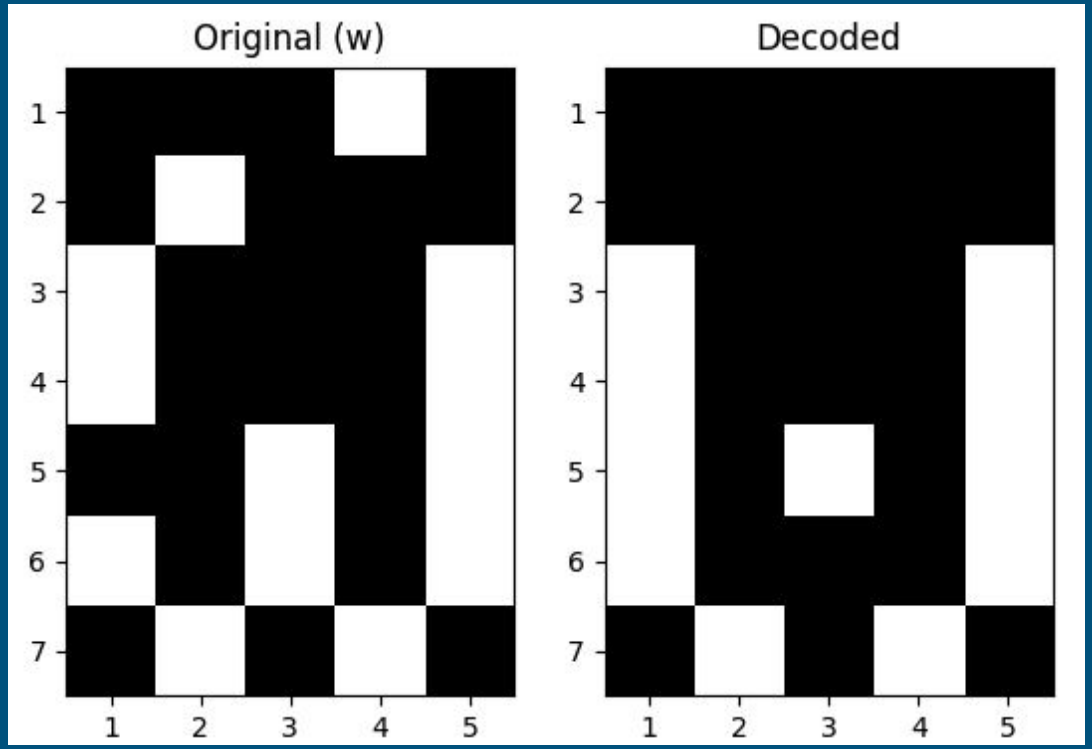
Se equivoca de letra 😞



# Capacidad de Denoising

- Salt & Pepper 10%
- Cantidad de épocas: 9000
- Tasa de Aprendizaje: $10^{-2}$

Casi acierta 👊



# Conclusiones (II)

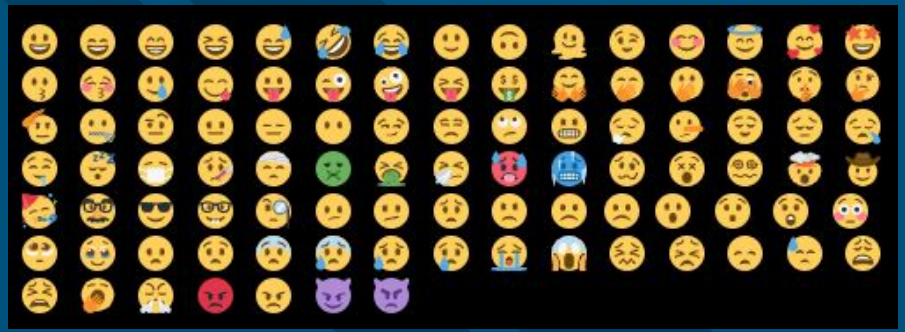
- ➔ Se observaron buenos resultados cuando los caracteres parecen preservar “la forma” al ser afectados por el ruido.
- ➔ El autoencoder es capaz de reconstruir caracteres con gran nivel de ruido (caso letra e).
- ➔ El autoencoder sufre algunos errores, que se podrían corregir con un cambio de parámetros.

# Ejercicio 2

# Variational Autoencoder

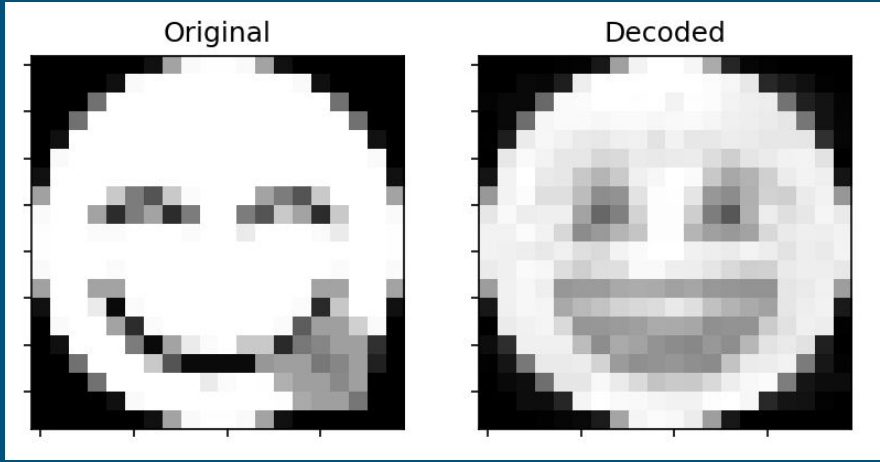
# EMOJIS 🤠

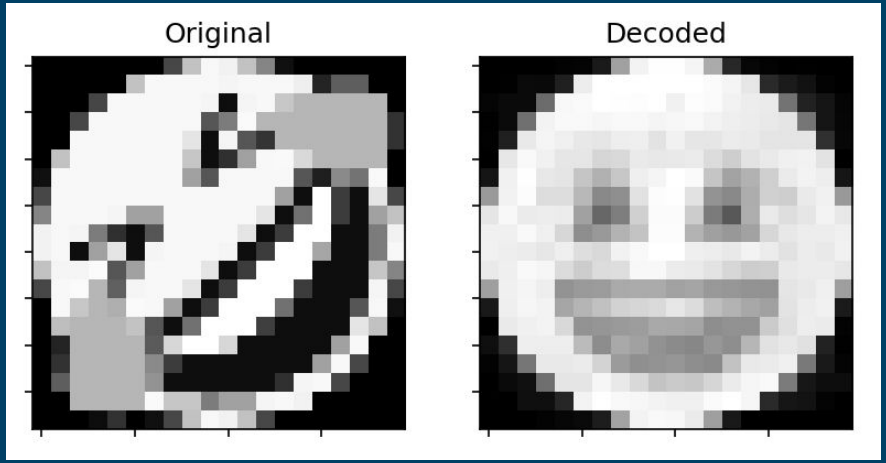
- ➔ Imágenes de 20x20 en escala de grises, normalizados en el rango [0,1].
- ➔ Dataset de 97 emojis para experimentar.
- ➔ Experimentamos con distintas arquitecturas de red y cantidad de datos para intentar encontrar la forma en la que el autoencoder los aprenda.
- ➔ Corridas particulares. Para una misma configuración se puede obtener convergencia y no convergencia debido a la aleatoriedad del problema.

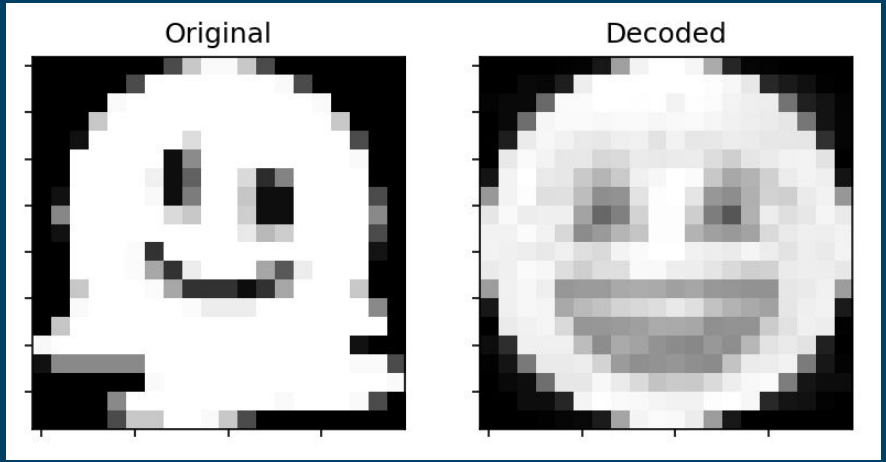


# Primer Intento:

- ➔ Dataset de 20 emojis
- ➔ Arquitectura: 400-50-10-50-400
- ➔ Aprendizaje: Adam con η=0.01
- ➔ 3000 épocas
- ➔ *¡Todos los emojis son iguales!*

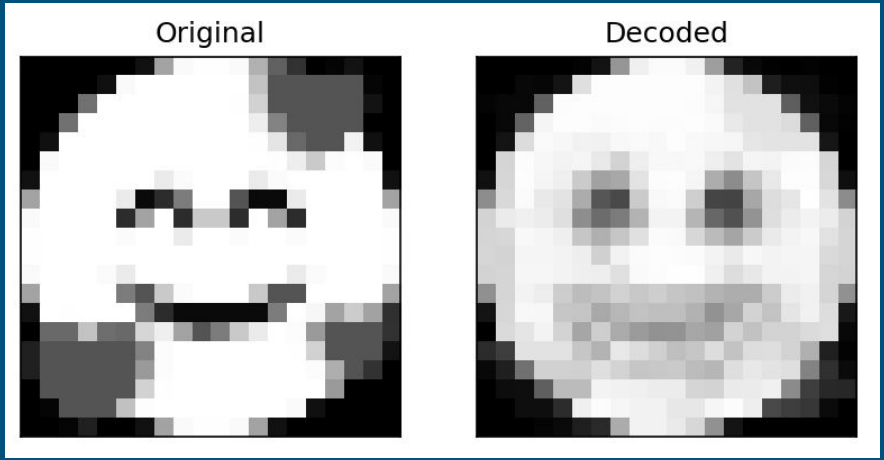


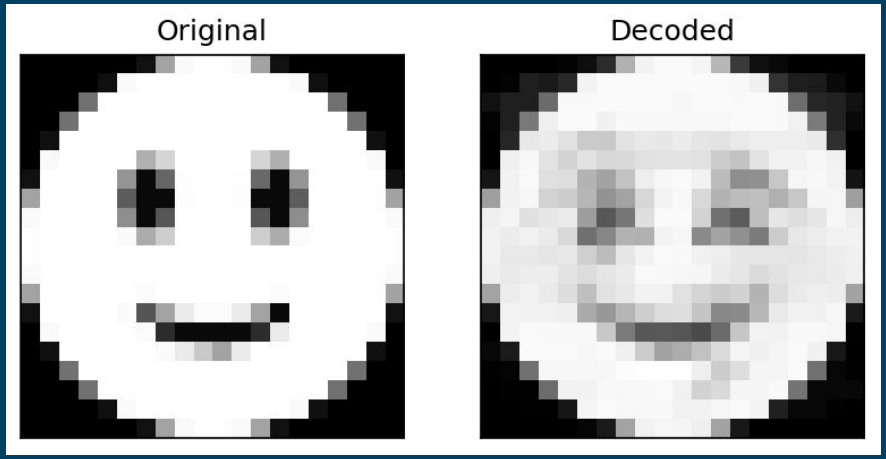



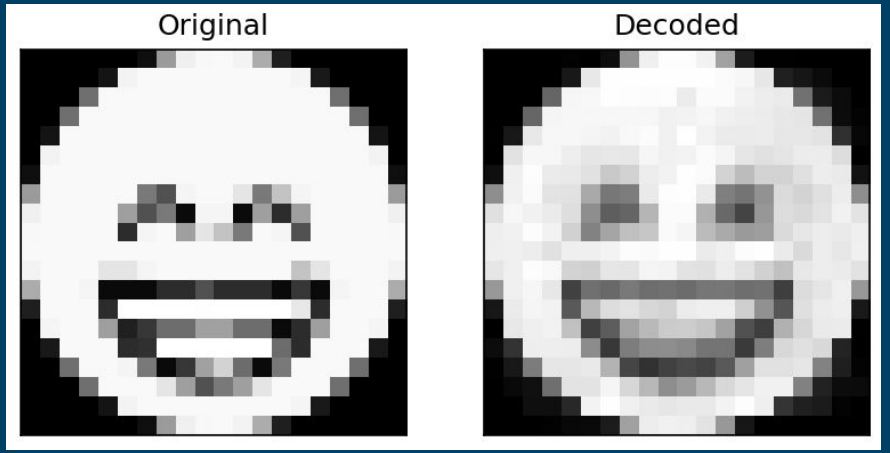



# Segundo Intento:

- ➔ Dataset de 20 emojis
- ➔ 400-258-126-2-126-258-400
- ➔ Aprendizaje: Adam con η=0.01
- ➔ 3000 épocas
- ➔ *¿Espacio latente pequeño?*

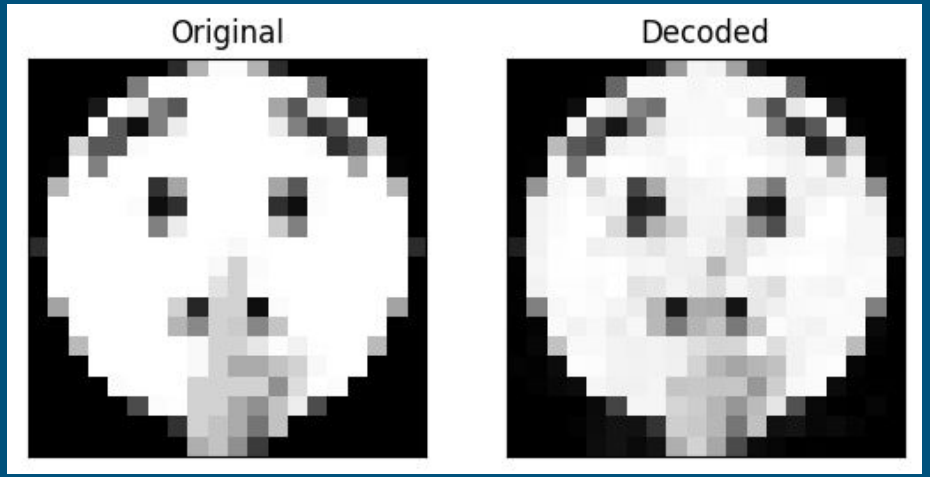


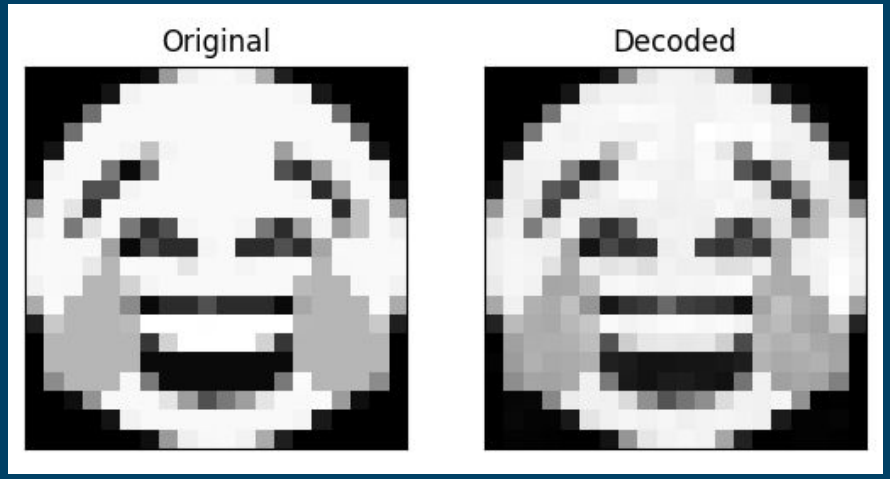



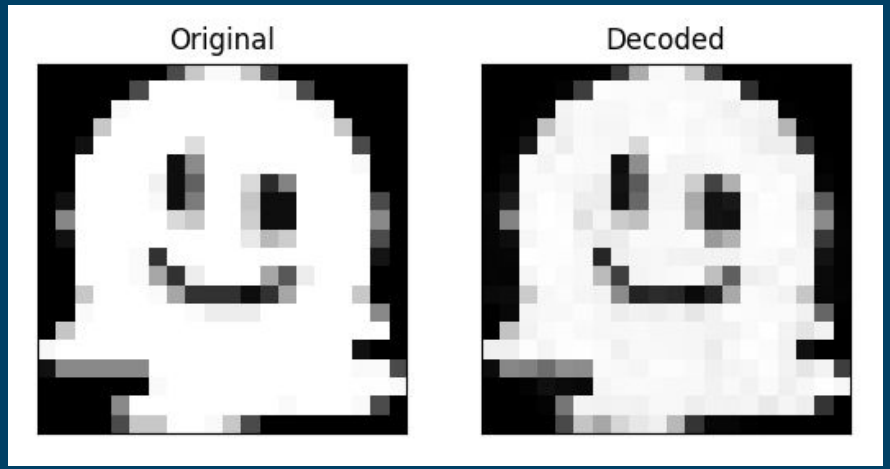



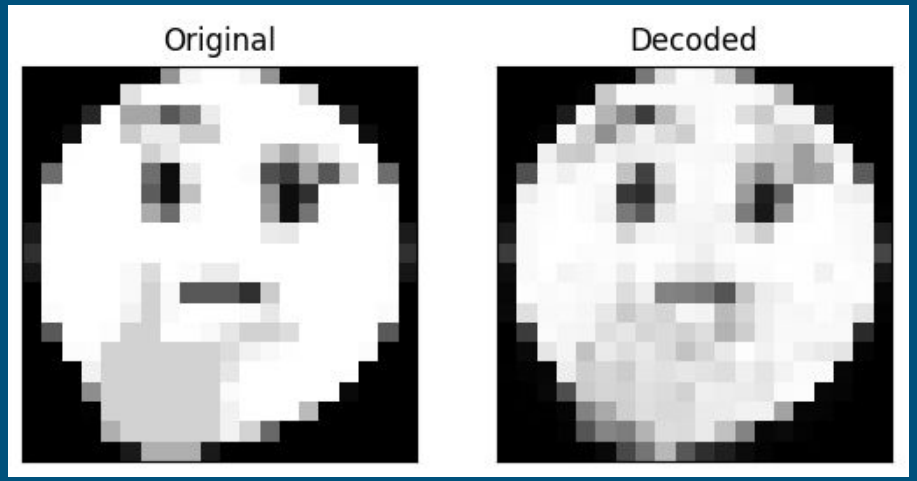
# Tercer Intento:

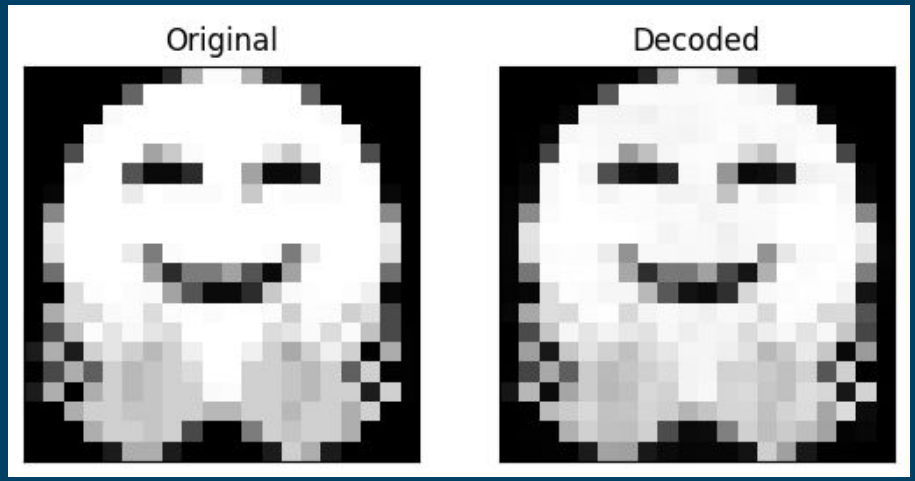
- ➔ Dataset de 55 emojis
- ➔ 400-300-200-20-200-300-400
- ➔ Aprendizaje: Adam con η=0.001
- ➔ 1000 épocas
- ➔ Ahora sí 👍

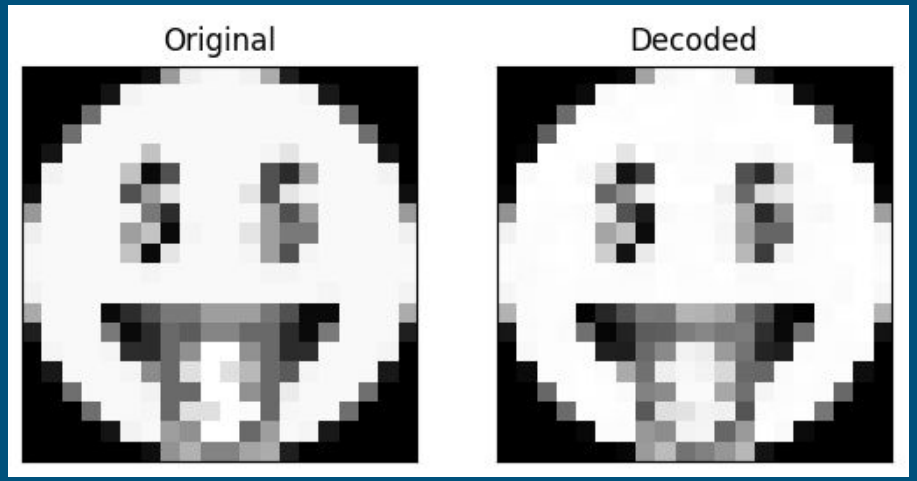


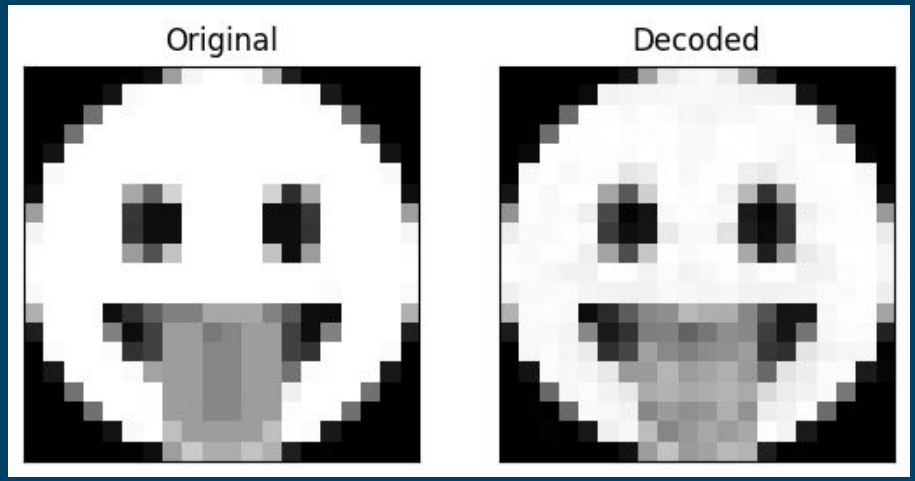




Original
Decoded


Original
Decoded


Original
Decoded


Original
Decoded

# Implementación

**Tamaños y θ de las capas:**

**Encoder:**

400 → 300 [ReLU]
300 → 200 [ReLU]
200 → 100 [ReLU]

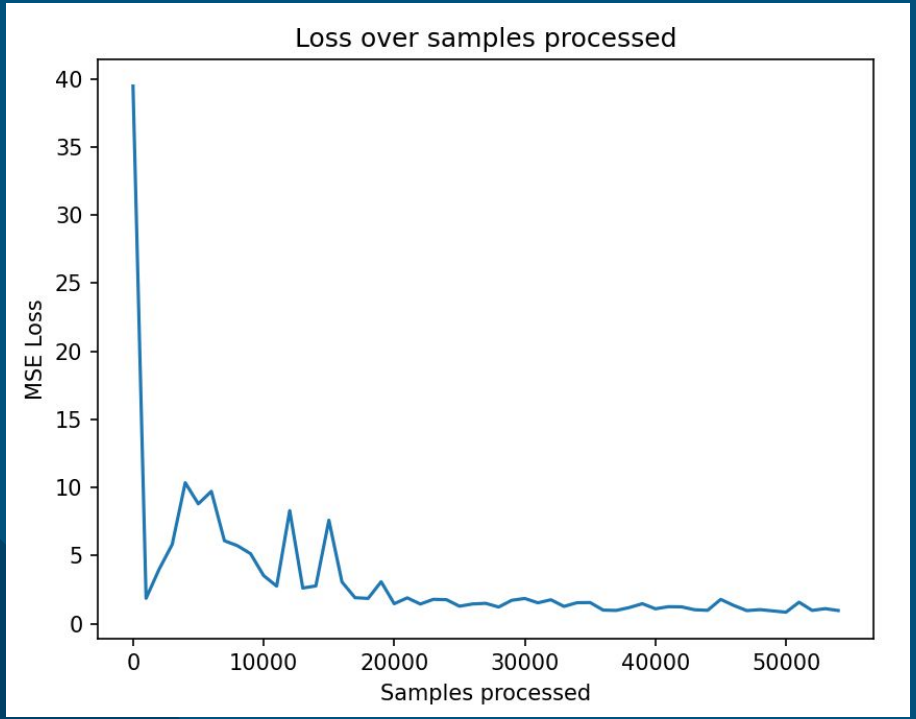
**Latent Space (Mean, LogVar):**

100 → 20 [Identity]

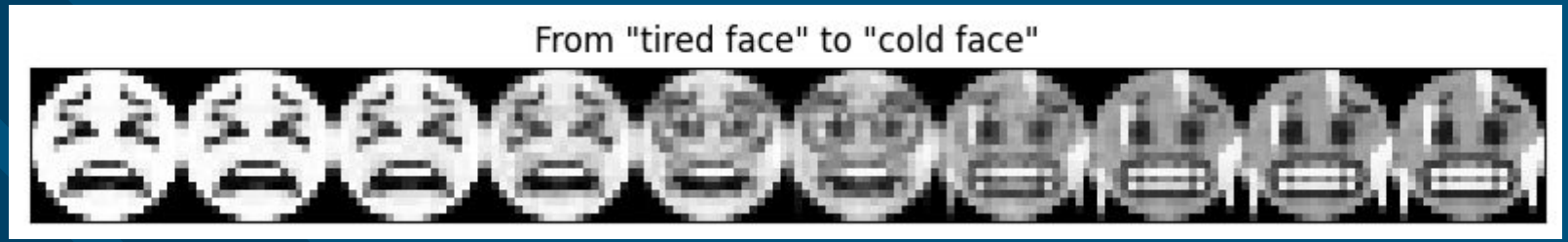
**Decoder:**

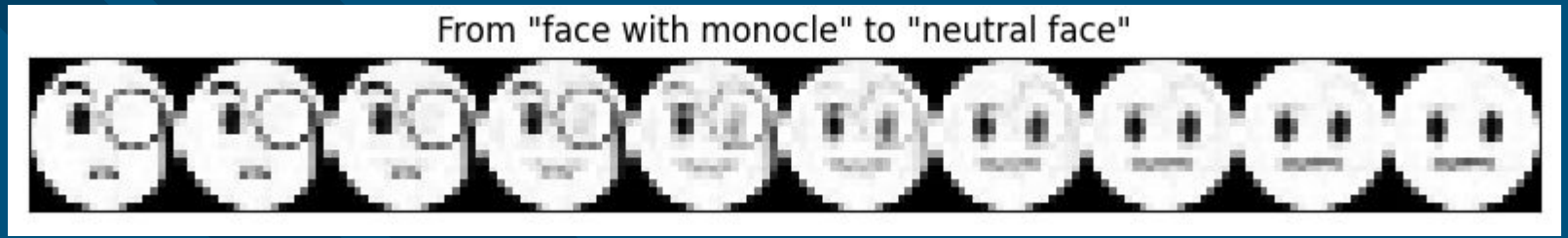
20 → 100 [ReLU]
100 → 200 [ReLU]
300 → 400 [Sigmoid]

# Loss Function (MSE)



# Interpolando en el espacio latente

# Conclusiones sobre los autoencoders

1. La representación latente en un **autoencoder normal no tiene una estructura probabilística**, lo que limita su capacidad para generar nuevas muestras o realizar interpolación suave en el espacio latente.

2. Los **autoencoders normales son más sencillos y rápidos de entrenar** en comparación con los VAE, ya que no requieren el cálculo de los parámetros latentes y la función de pérdida de reconstrucción es más directa.

3. Los **autoencoders normales son eficientes en la compresión de datos y la eliminación de ruido,** mientras que **los VAEs agregan una estructura probabilística en la representación latente, lo que les permite generar nuevas muestras y realizar interpolaciones suaves.**

# Conclusiones sobre los autoencoders

**Tienen DEMASIADOS USOS**

**Denoising Autoencoder** → recuperar la imagen original a partir de una ruidosa.

**Sparse Autoencoder** → Feature Learning

**Deep Autoencoder** → Representaciones más complejas

**Contractive Autoencoder** → Menor sensibilidad a la entrada, representaciones más simples y crudas

**Variational Autoencoder** → Mayor control en el espacio latente, sampleo con distribución

# Conclusiones sobre la implementación

1. Como vimos en anteriores entregas, es importante ajustar correctamente la arquitectura y los parámetros del VAE para obtener buenos resultados. **Esto incluye el número de capas y neuronas en el codificador y el decodificador, la dimensión del espacio latente y las funciones de activación utilizadas.**

2. Se busca mejorar el **feature extraction** en el VAE para capturar características relevantes de los datos y generar representaciones latentes significativas.

3. Si el **espacio latente es demasiado pequeño**, puede resultar en una **incapacidad** del modelo para capturar la **variabilidad de las imágenes** y, en consecuencia, generar imágenes similares.

4. Se usó sigmoid en la última capa del decodificador, y se normalizó de tal manera la escala de grises, para trabajar con el mismo rango ([0,1]). **Algo que hay que tener en cuenta.**

# ¿Preguntas?