TP 5 - S.I.A - Grupo 1

Integrantes:

Burgos, Jose (61525)
Matilla, Juan Ignacio (60459)
Curti, Pedro (61616)
Panighini, Franco (61258)

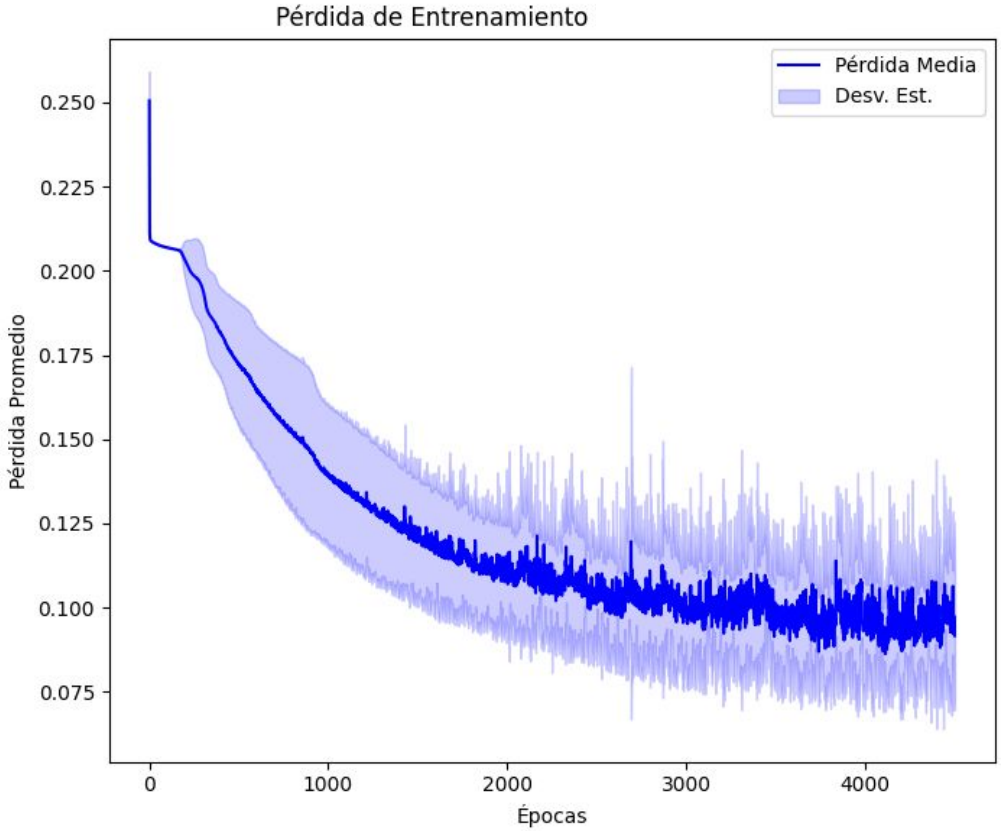
Autoencoder

Primer intento

Estrategias aplicadas

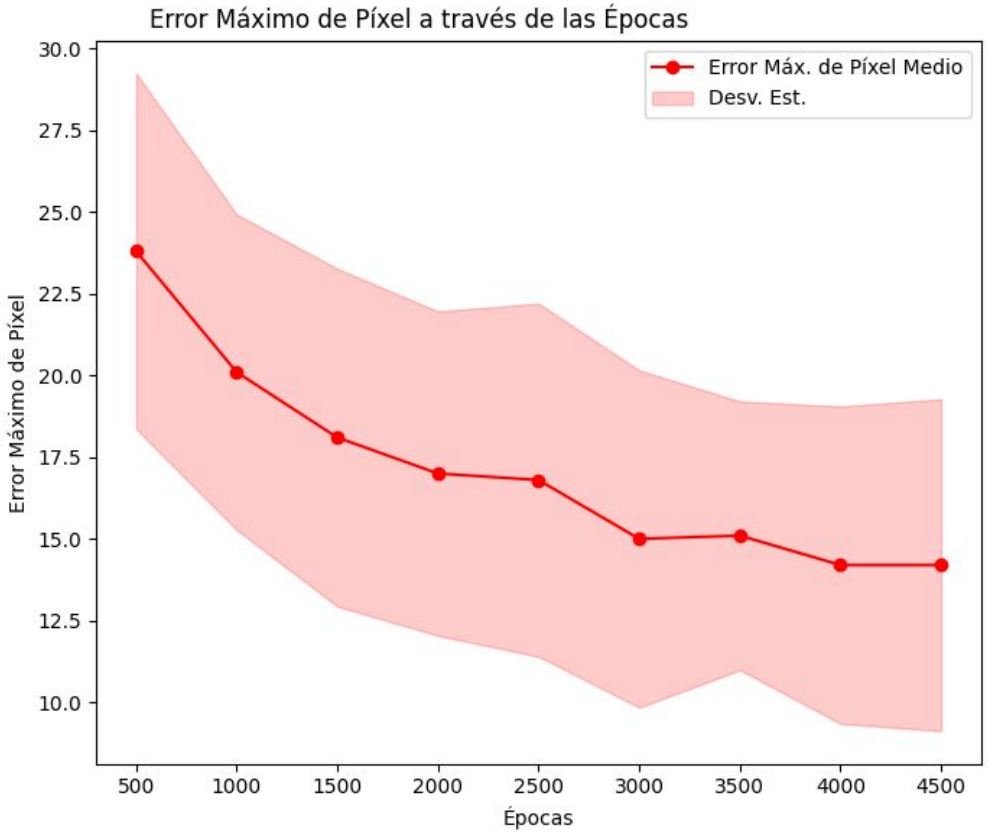
- Uso de una arquitectura de 35 - 15 - 2 - 15 - 35
- Entrenado por 4500 epochs
- Learning rate: 0.1
- Función de activación sigmoide
- MLP Vanilla

Resultados - MSE



10 runs

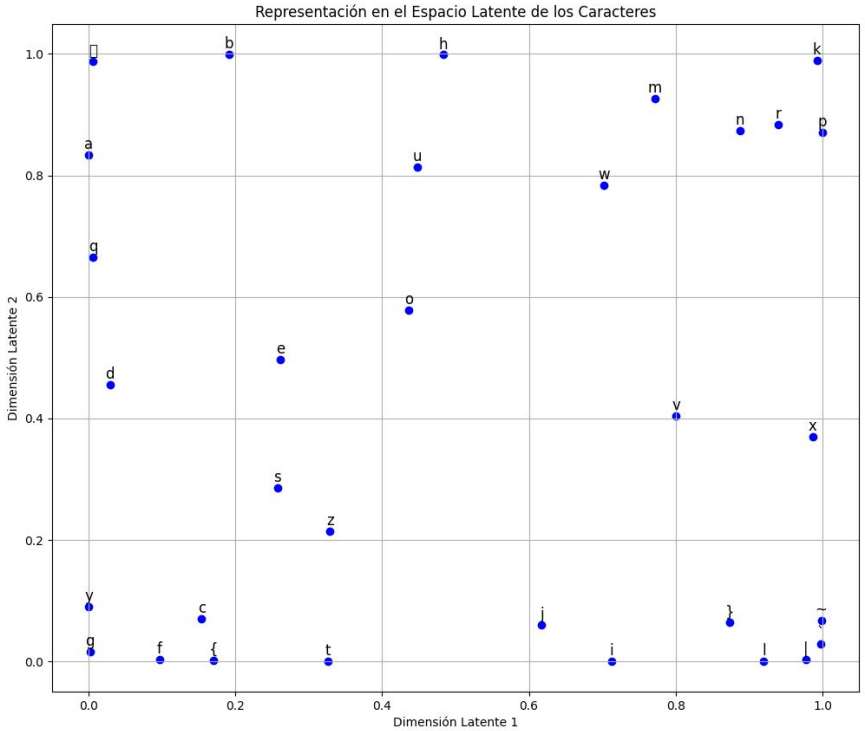
Resultados - Max pixel error



10 runs

Mejor run con máximo
5 pixeles de error

Espacio latente

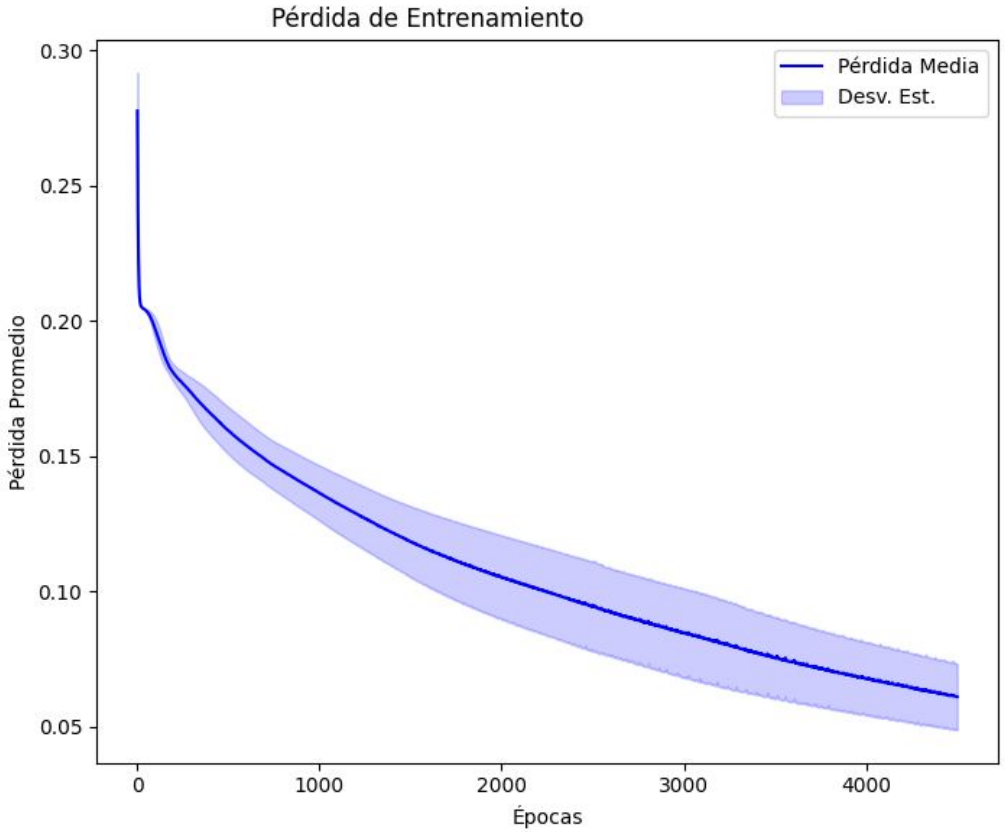


Adam

Estrategias aplicadas

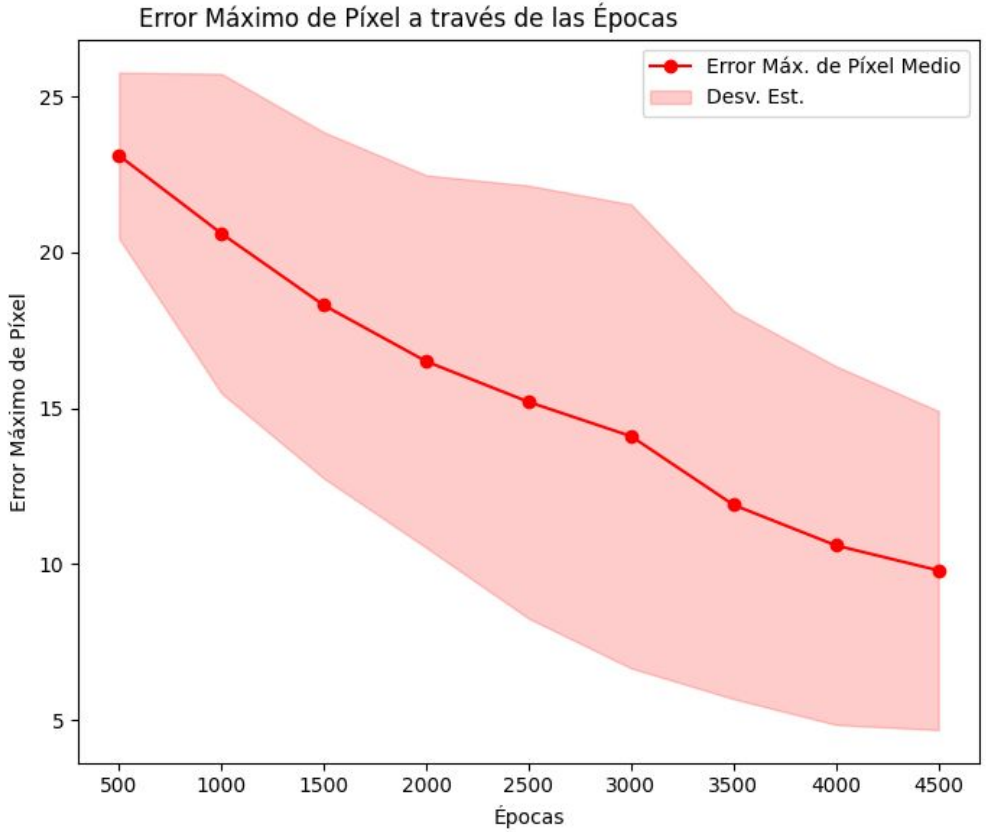
- Uso de una arquitectura de 35 - 15 - 2 - 15 - 35
- Uso de optimizador Adam
- Learning rate y betas usados como la recomendación de Adam
- Entrenado por 4500 epochs
- Función de activación sigmoide

Resultados - MSE



10 runs

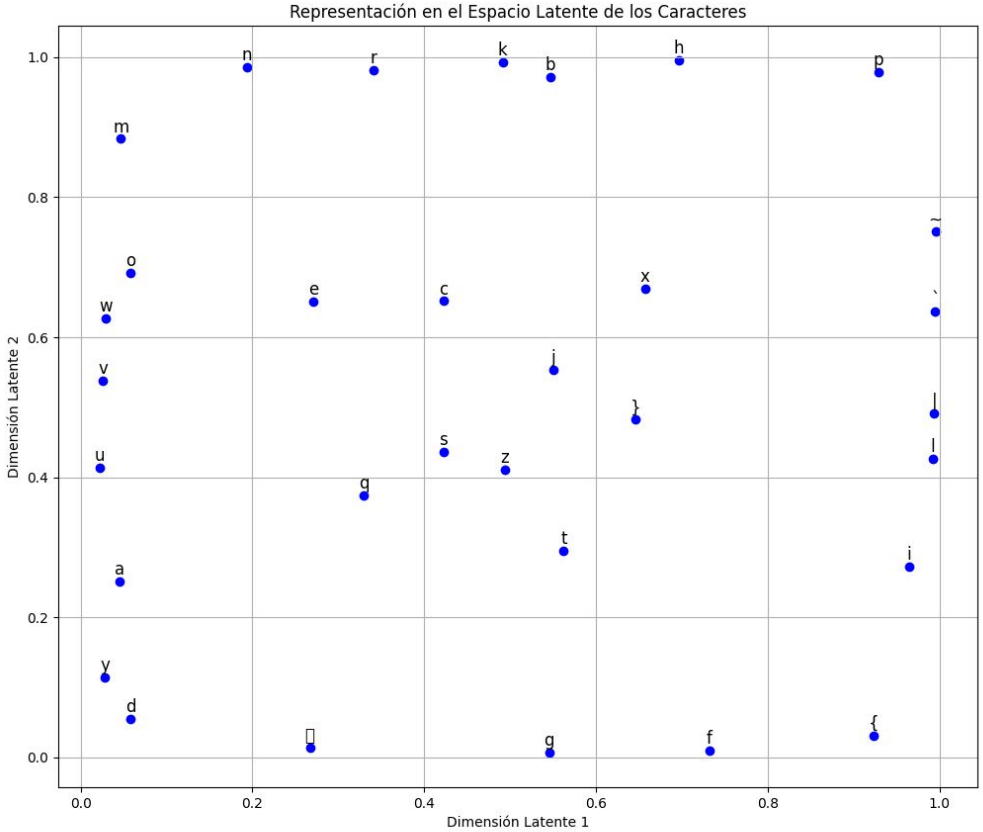
Resultados - Max pixel error



10 runs

Mejor run con máximo
5 pixeles de error

Espacio latente

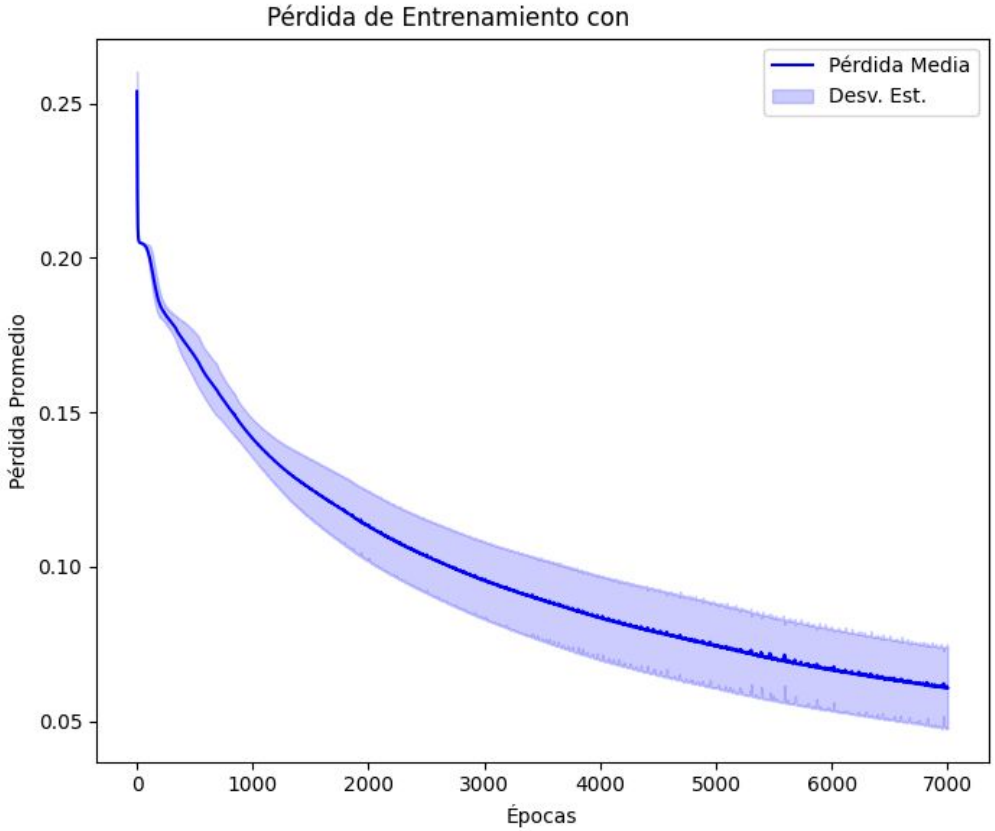


Xavier

Estrategias aplicadas

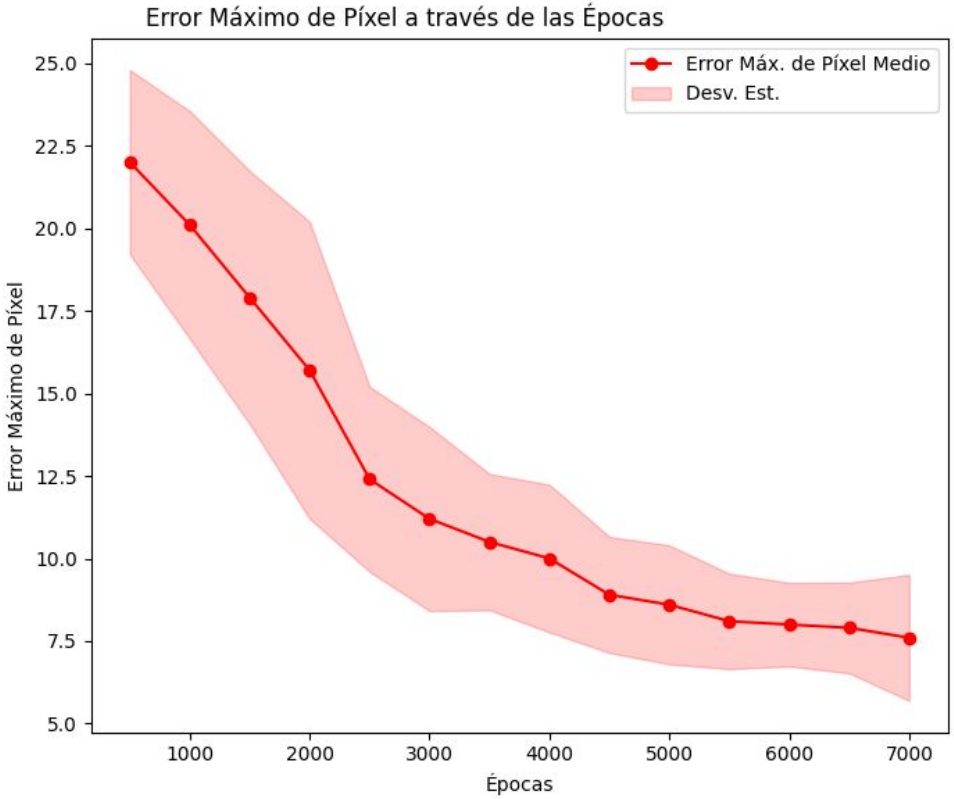
- Uso de una arquitectura de 35 - 15 - 2 - 15 - 35
- Uso de optimizador Adam
- Learning rate y betas usados como la recomendación de Adam
- Uso de Xavier para inicialización de pesos
- Entrenado por 4500 epochs
- Función de activación sigmoide

Resultados - MSE



10 runs

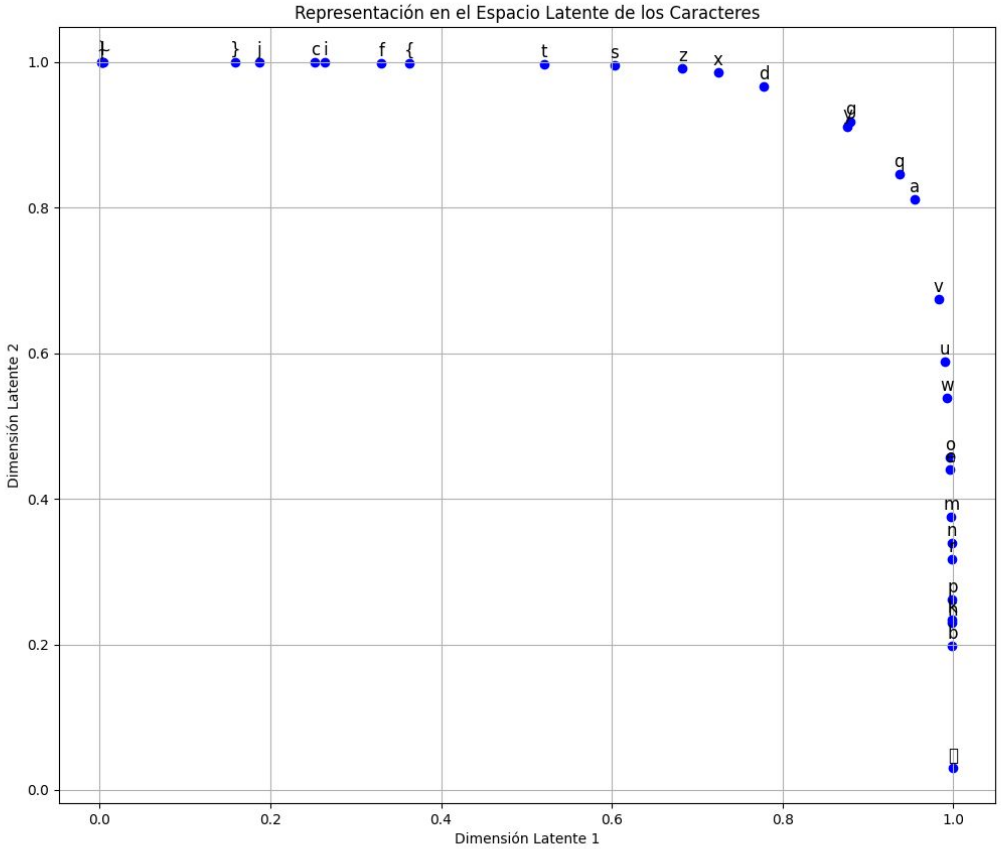
Resultados - Max pixel error



10 runs

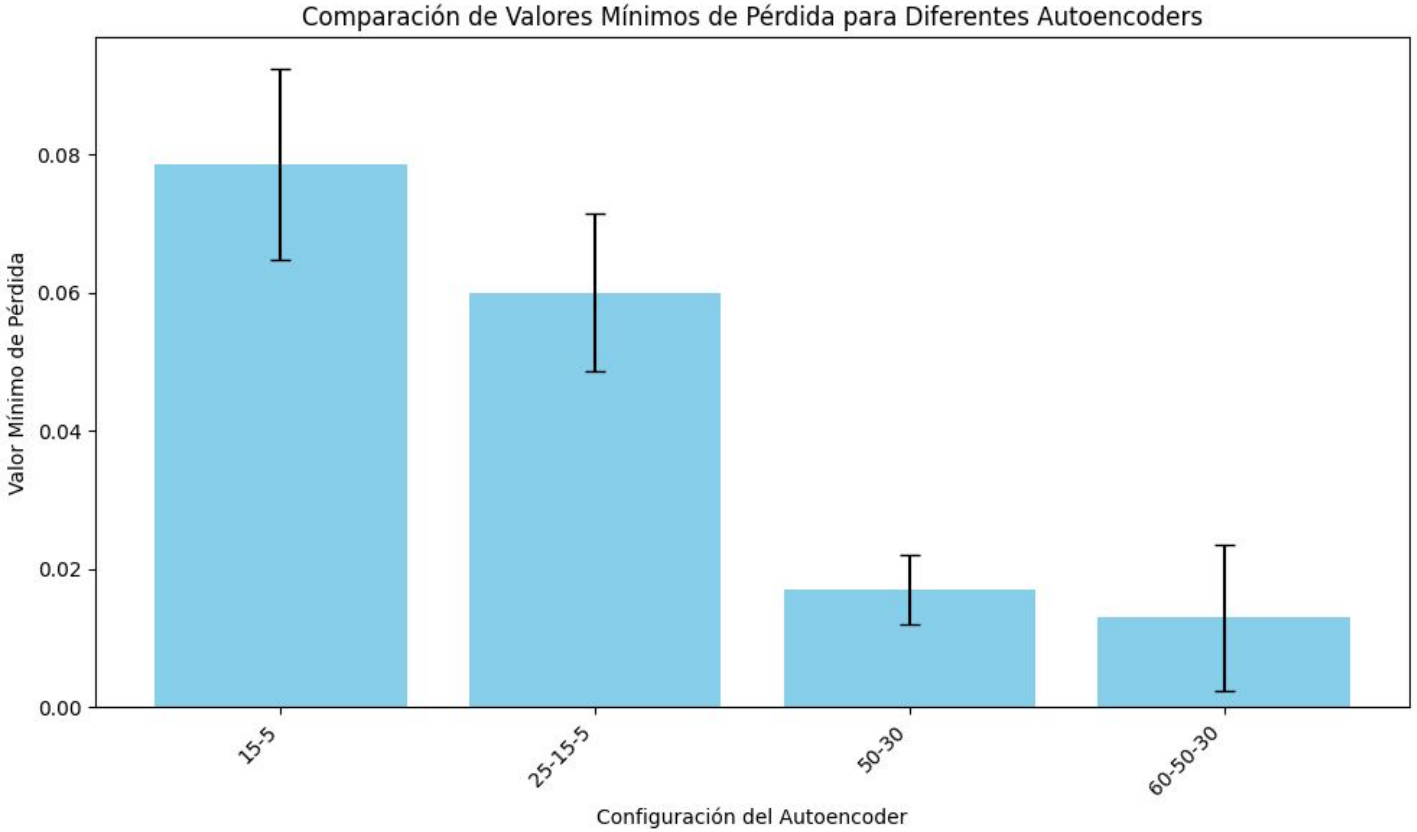
Mejor run con máximo
5 pixeles de error

Espacio latente

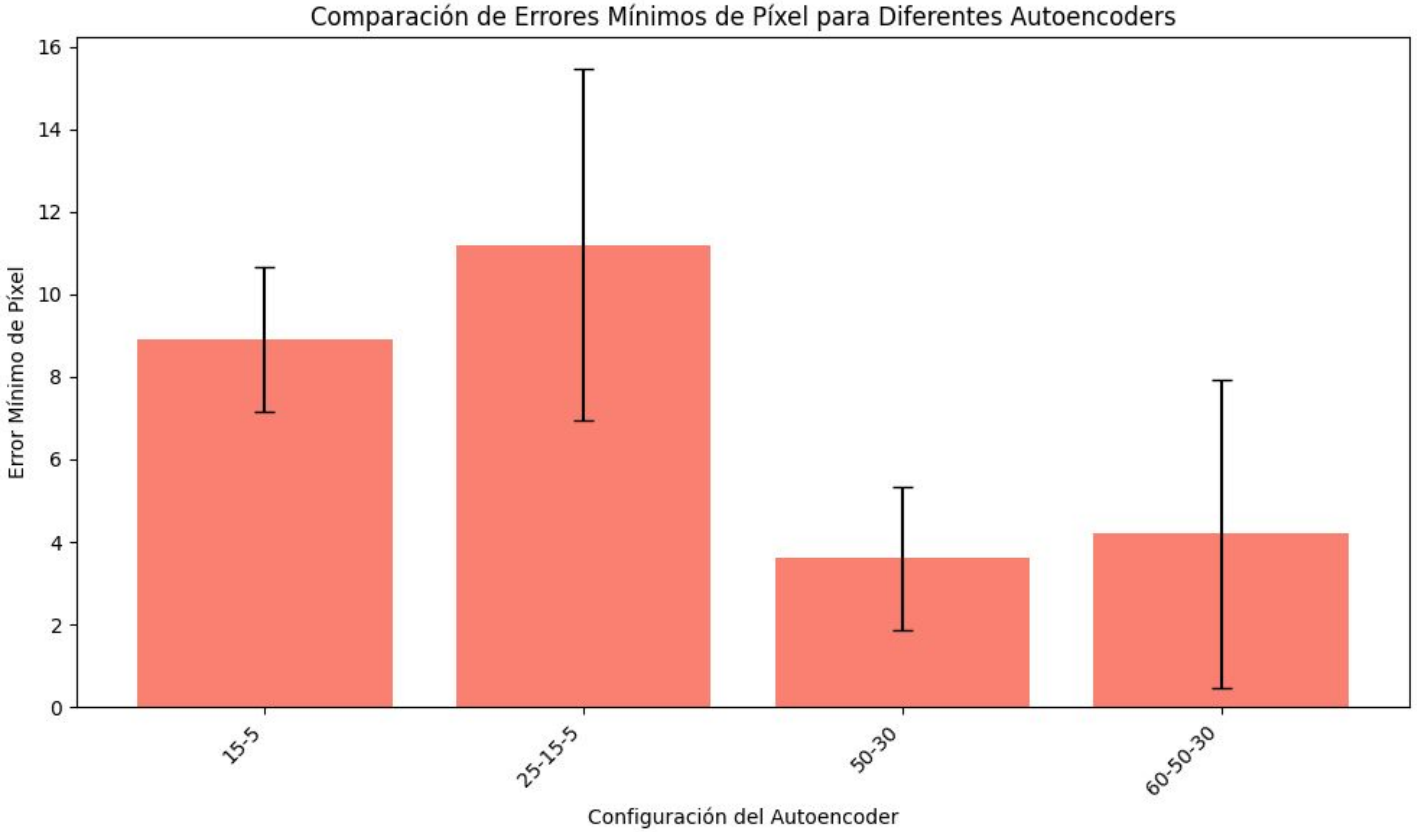


Diferentes arquitecturas de layers

Loss para diferentes arquitecturas



Max pixel error para diferentes arquitecturas

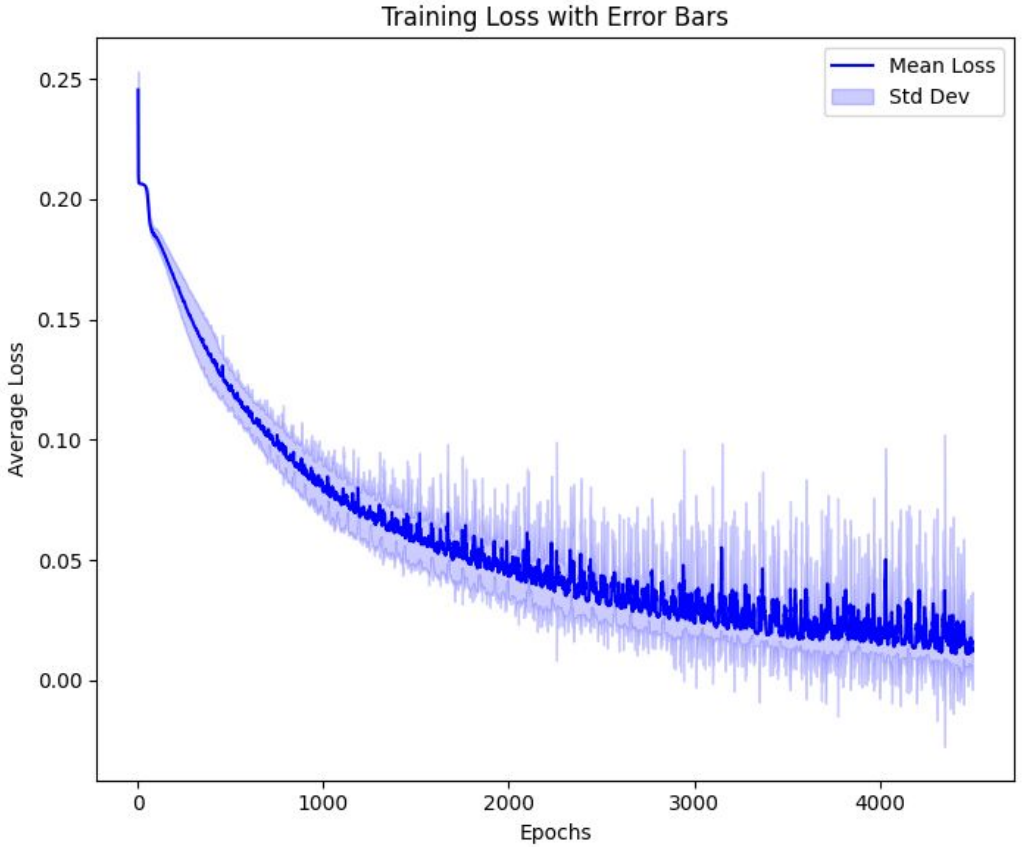


Winning Solution

Estrategias aplicadas

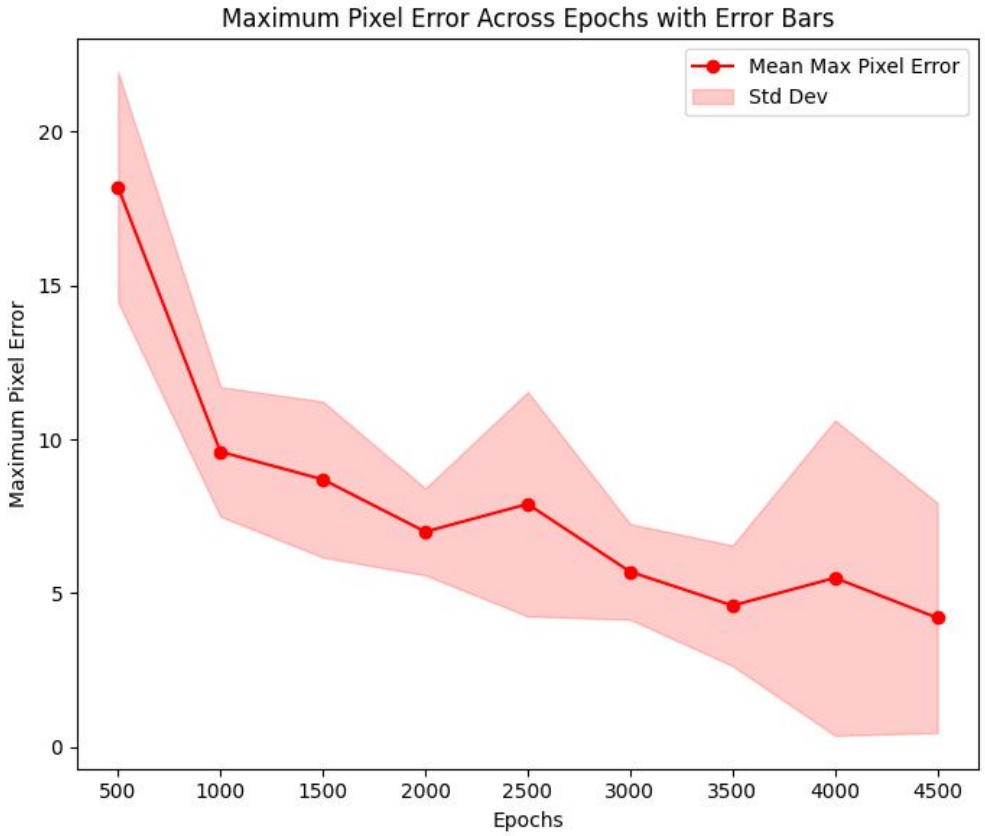
- Uso de optimización de Adam
- Learning rate y betas usados como la recomendación de Adam
- Uso de inicialización de pesos con Xavier
- Uso de una arquitectura de 35 - **60 - 50 - 30** - 2 - 30 - 50 - 60 - 35
- Función de activación sigmoide
- Entrenado por 4500 epochs

Resultados - MSE



10 runs

Resultados - Max pixel error

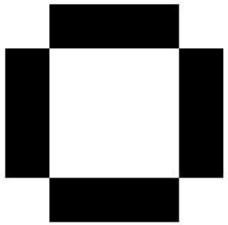


10 runs

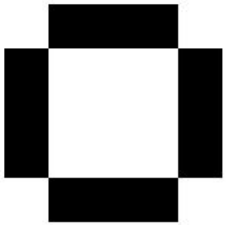
Mejor run con máximo
1 pixel de error

Ejemplos de reconstrucción

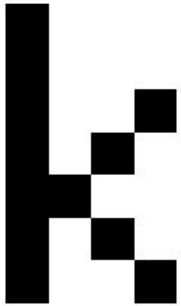
Original



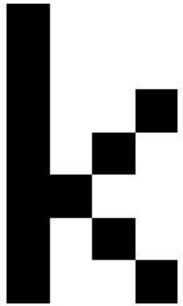
Reconstructed



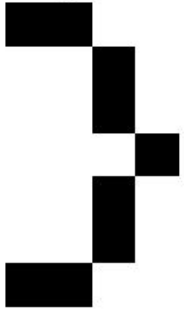
Original



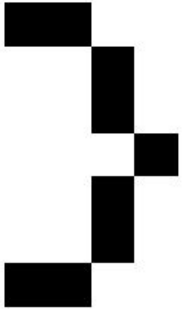
Reconstructed



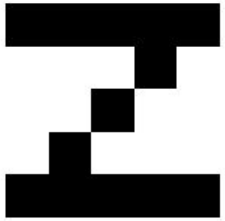
Original



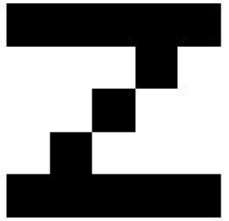
Reconstructed



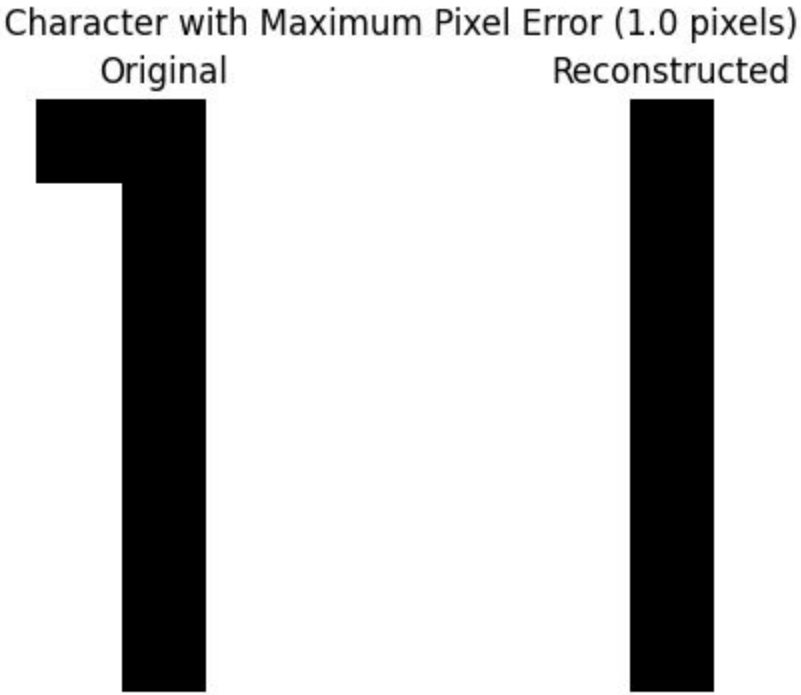
Original



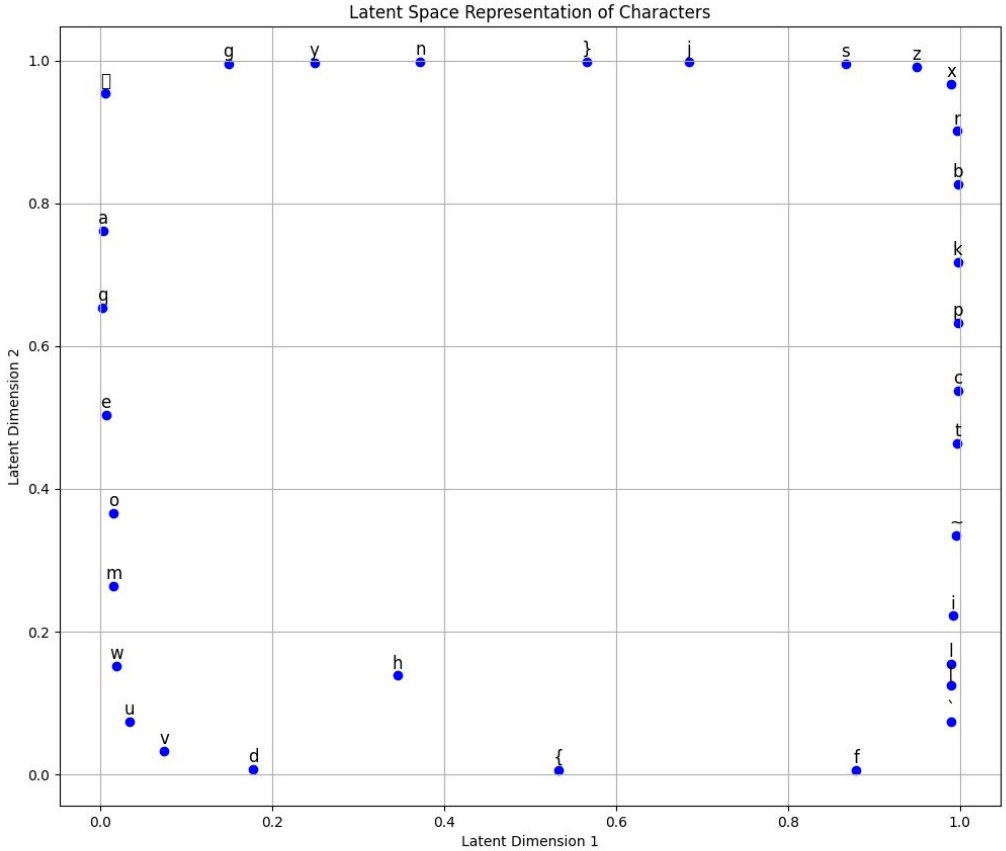
Reconstructed



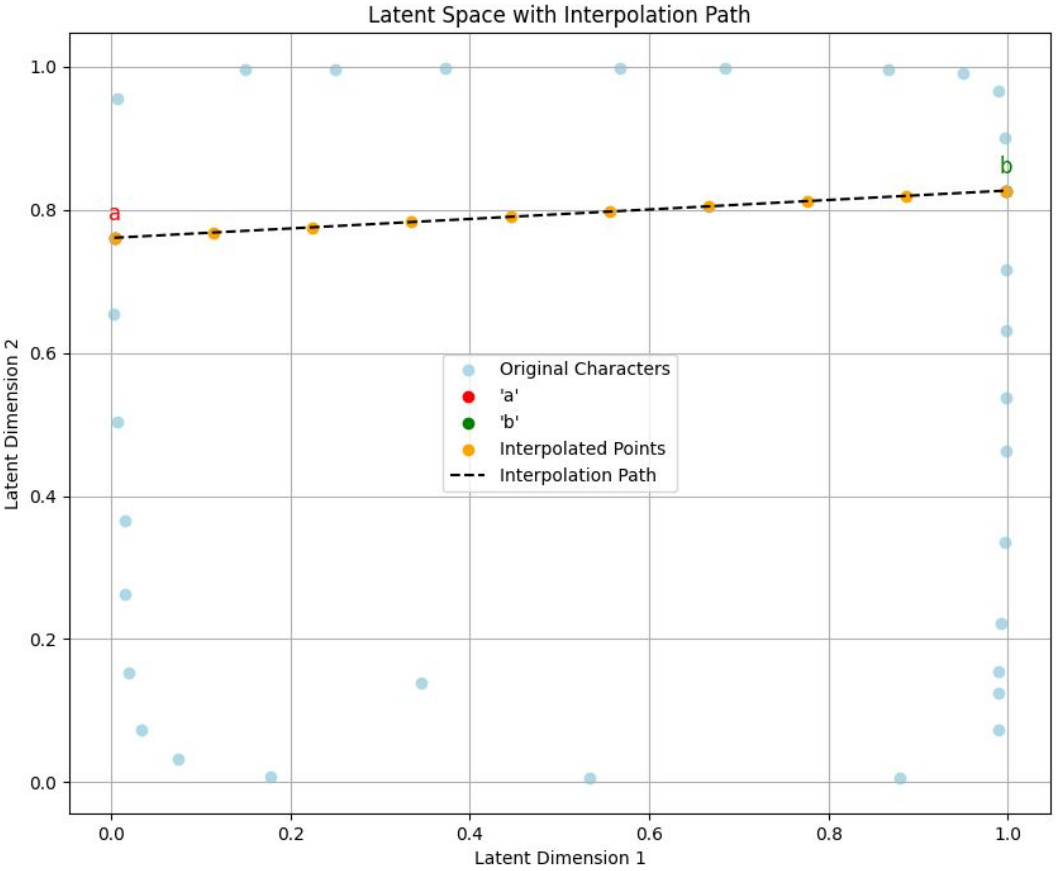
Peor compresión (mejor run)

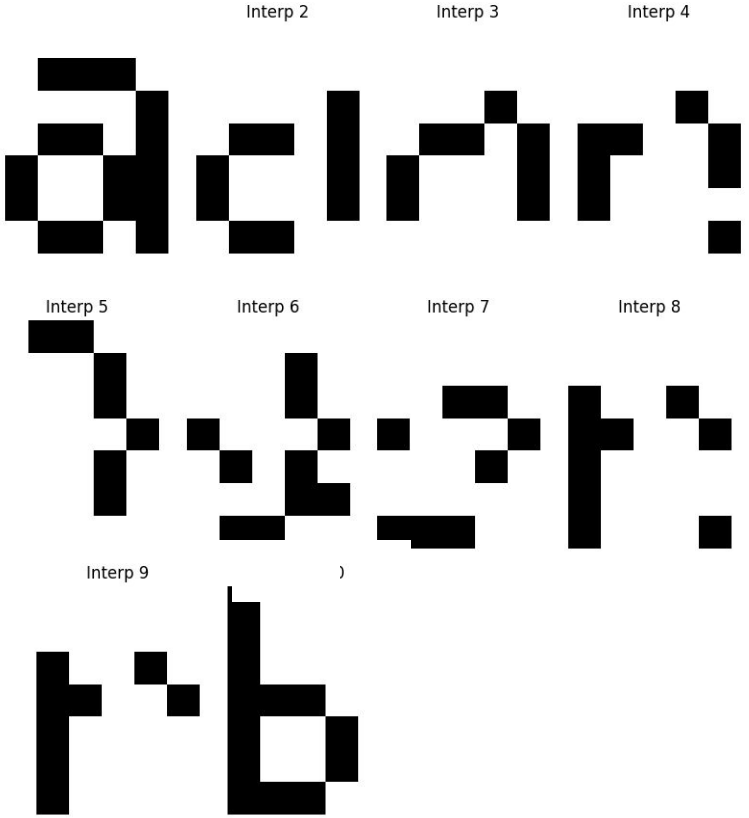


Espacio latente



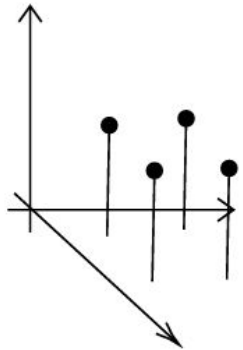
Generación de nuevo caracter





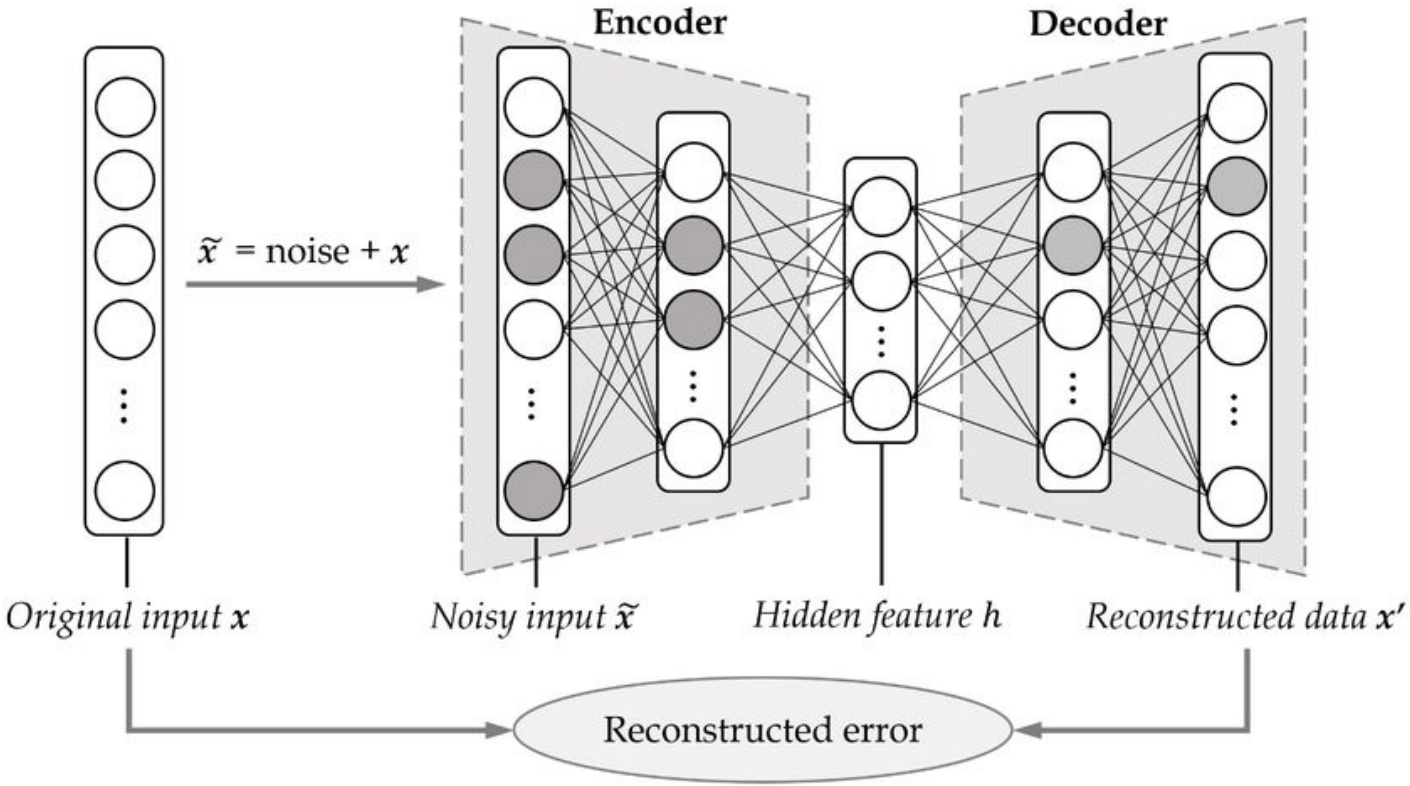
Conclusiones

- Comportamiento muy estocástico para lograr un max pixel error de 1
- Xavier acelera fuertemente el aprendizaje y la capacidad de generalización ya que empieza de un punto mucho más alto a entrenar
- Adam mejora mucho más los pasos de minimización de la función de costo
- No logra generar nuevas letras “buenas” porque tiene el espacio latente en forma de torres:



Denoising Autoencoder

Funcionamiento

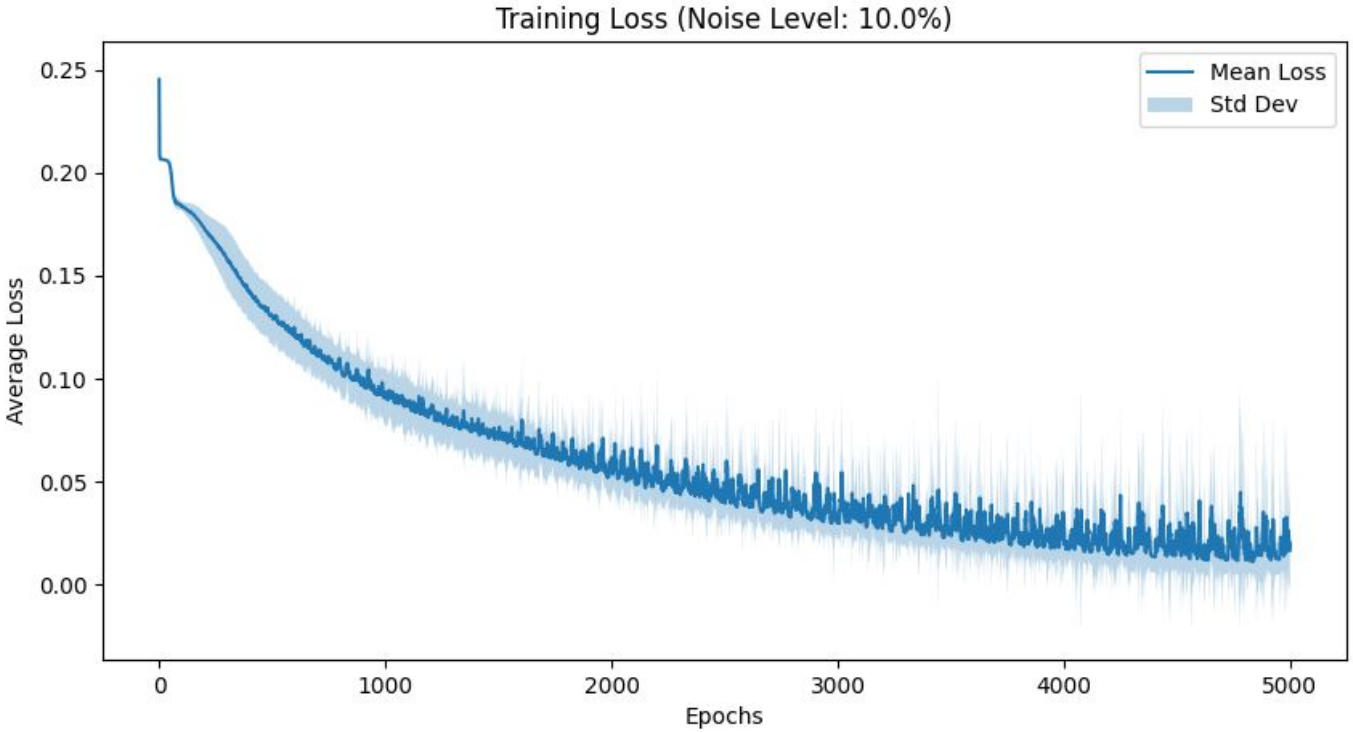


Mismas estrategias aplicadas

- Uso de optimización de Adam
- Learning rate y betas usados como la recomendación de Adam
- Uso de inicialización de pesos con Xavier
- Uso de una arquitectura de 35 - **60 - 50 - 30** - 2 - 30 - 50 - 60 - 35
- Entrenado por 4500 epochs

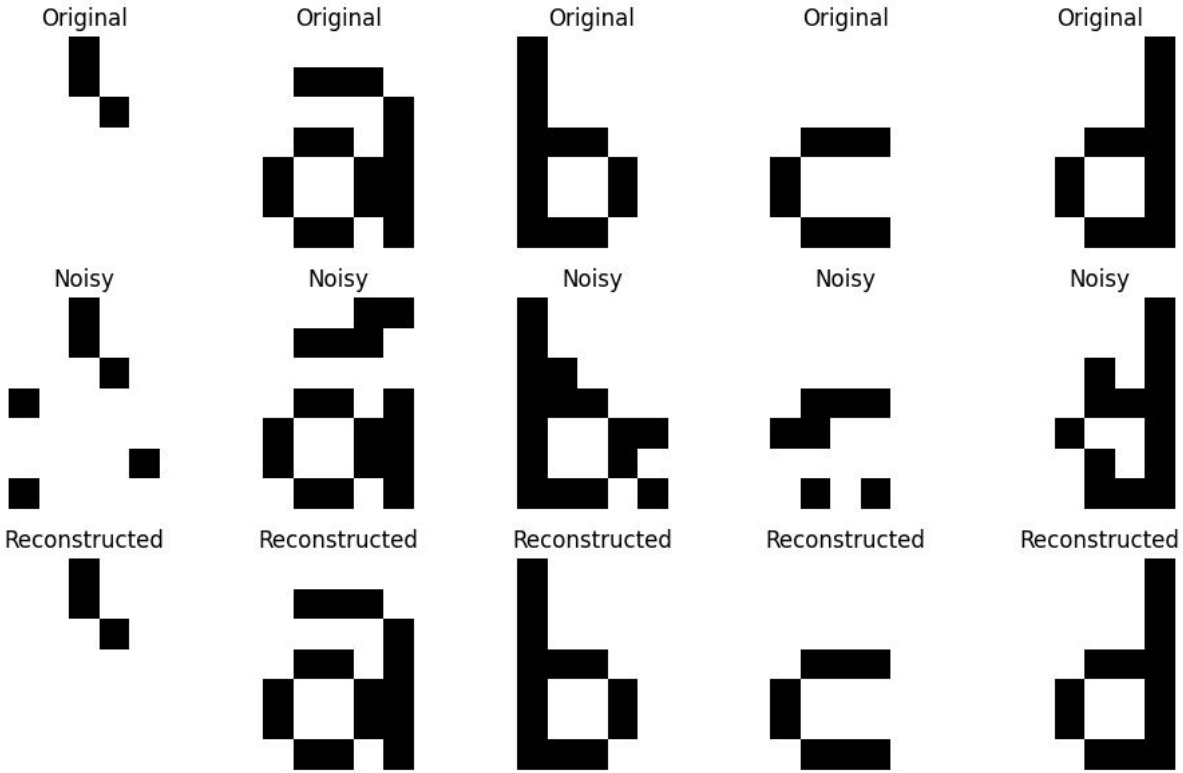
10% de ruido

MSE por epoch

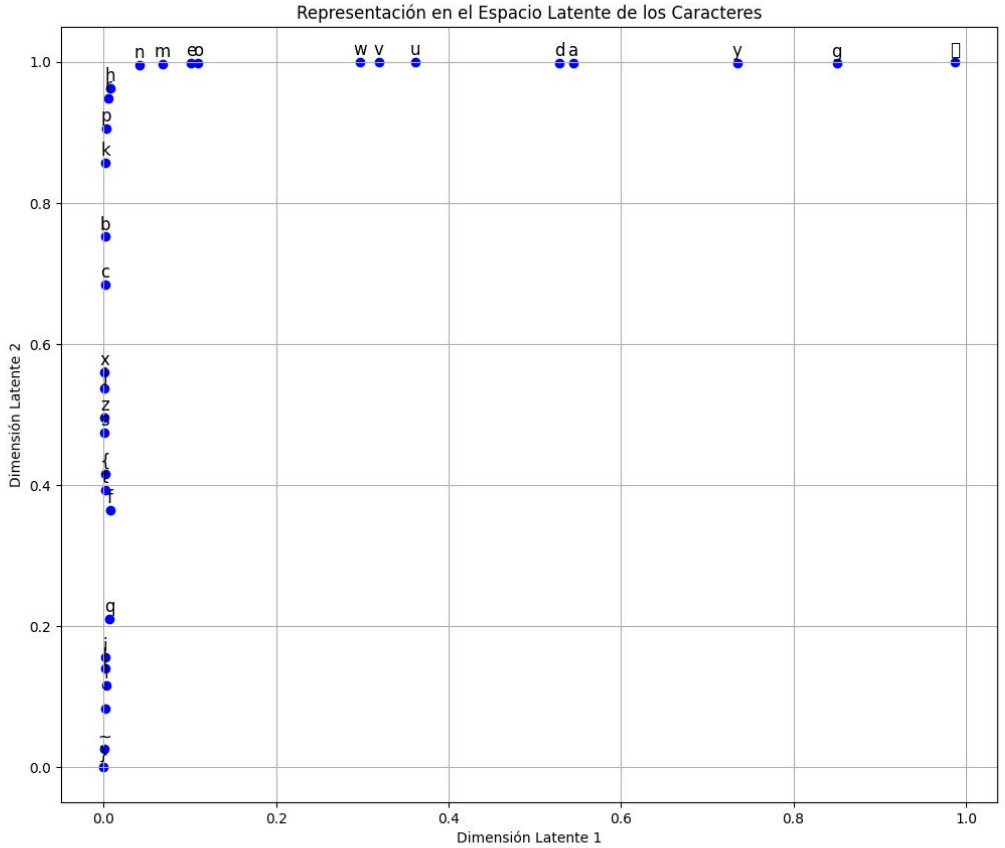


10 runs
Average minimum loss:
0.009570 +- 0.000041

Algunos resultados

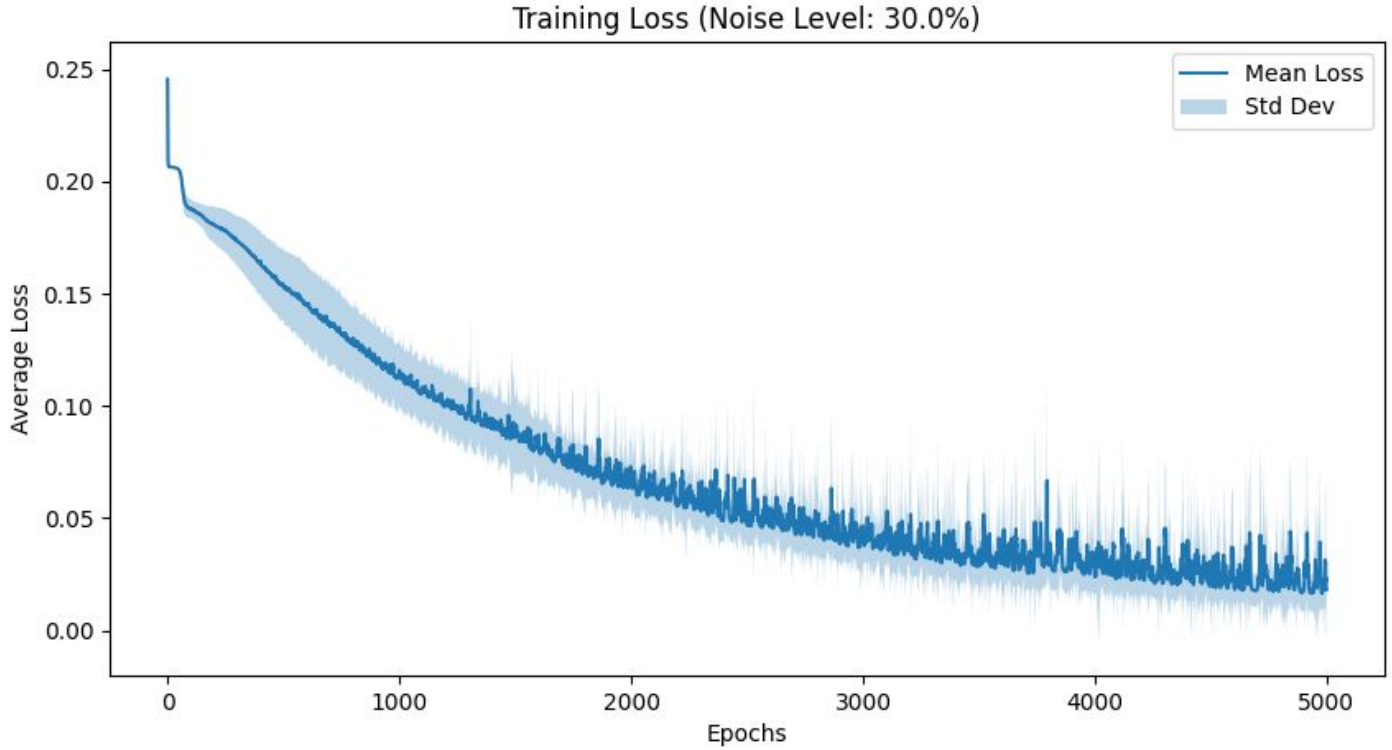


Espacio Latente



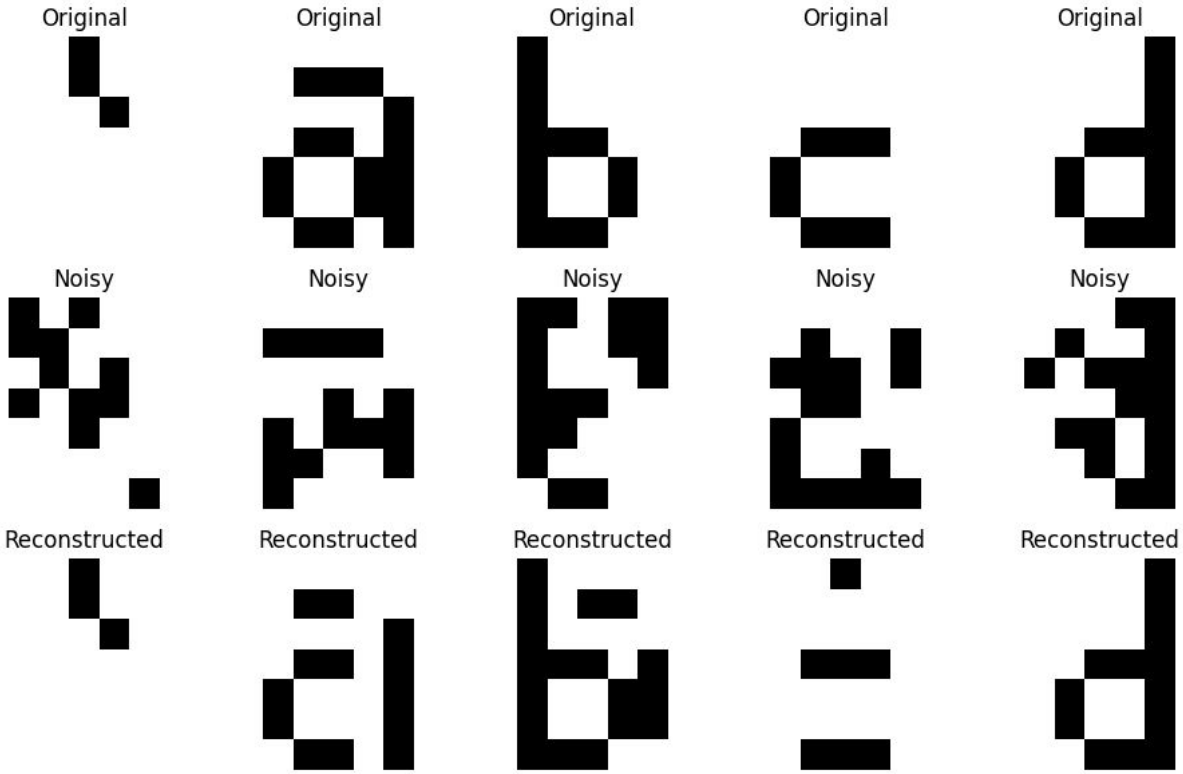
30% de ruido

MSE por epoch

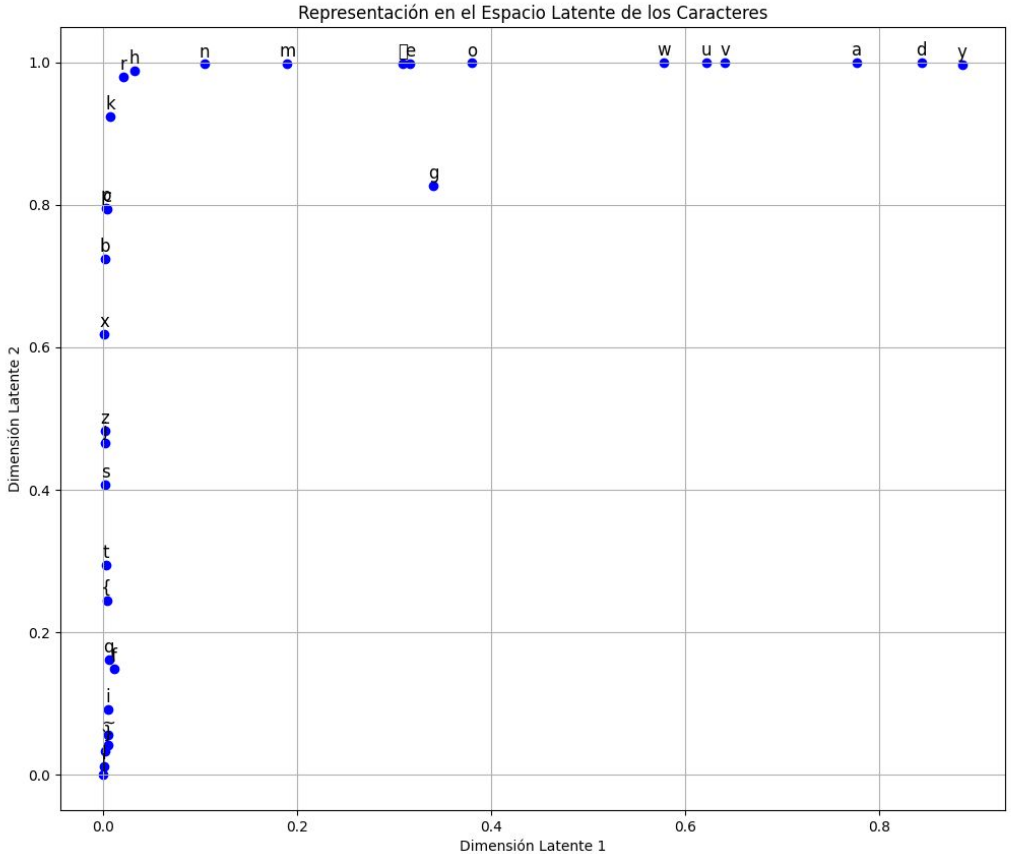


10 runs
Average minimum loss:
0.014443 +- 0.000030

Algunos resultados

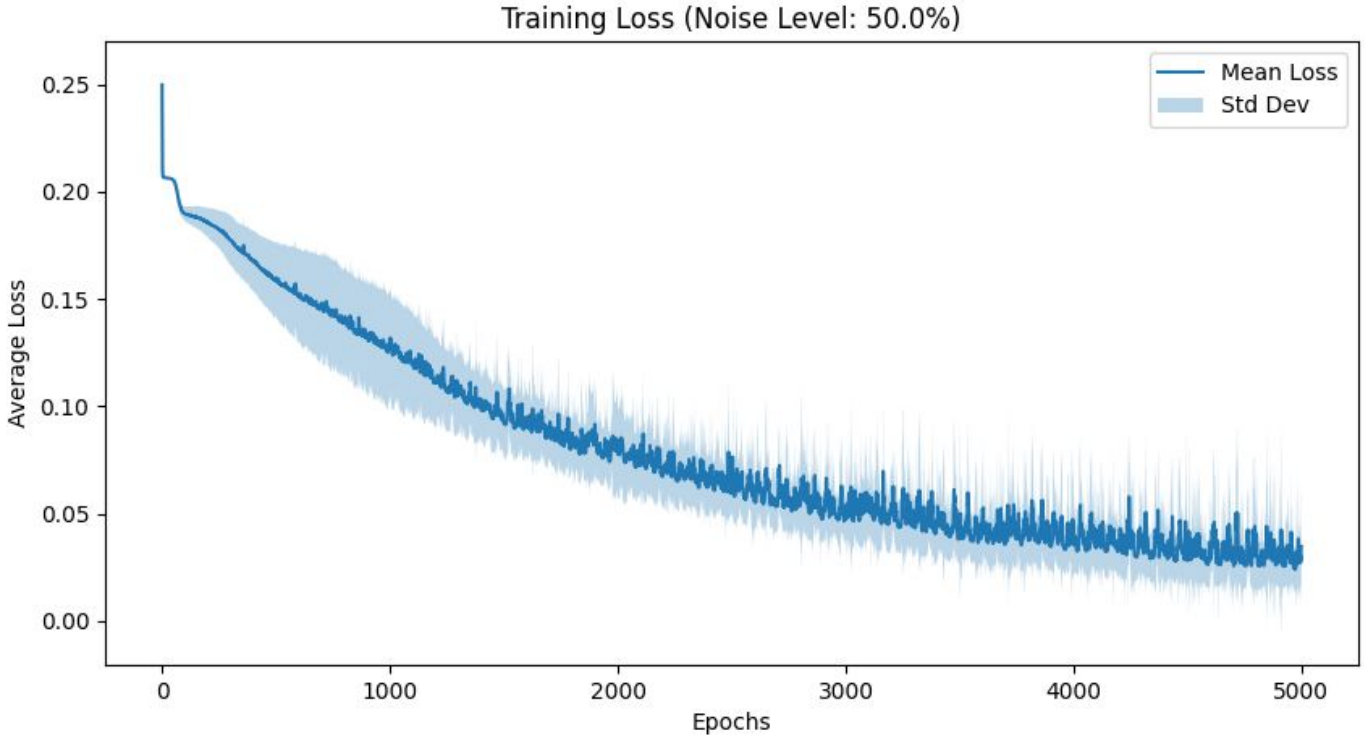


Espacio Latente



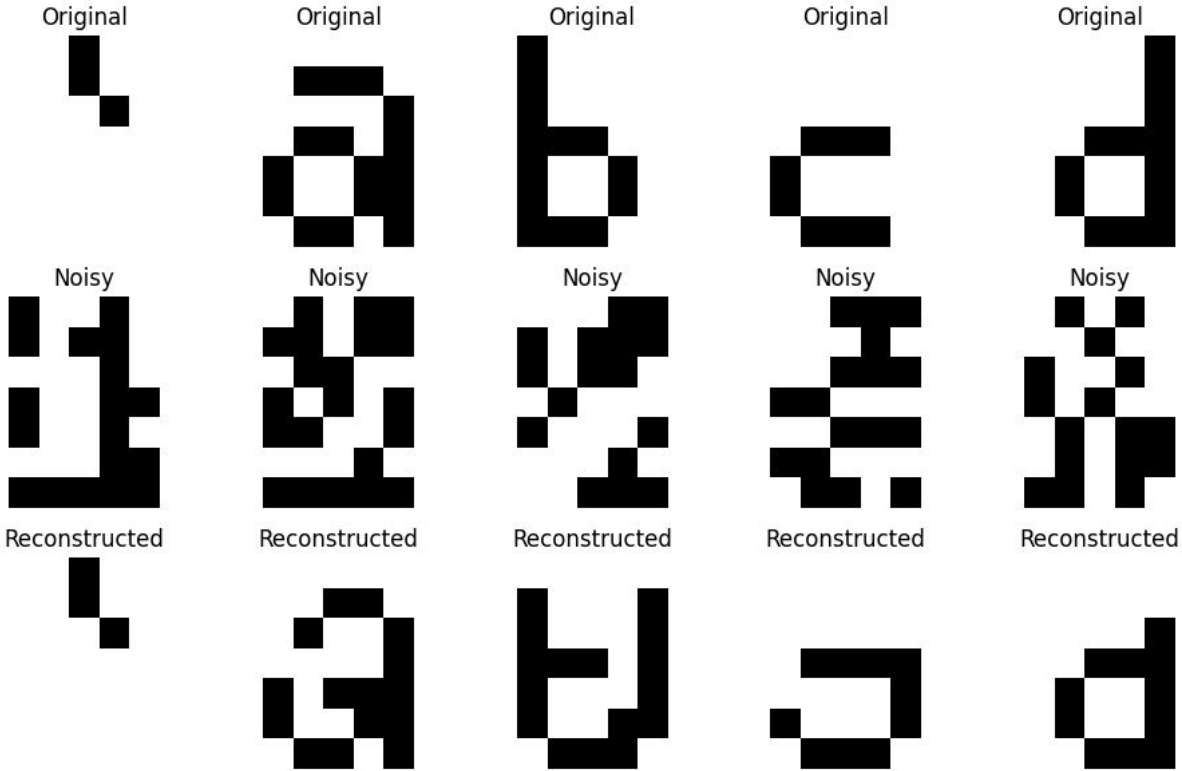
50% de ruido

MSE por epoch

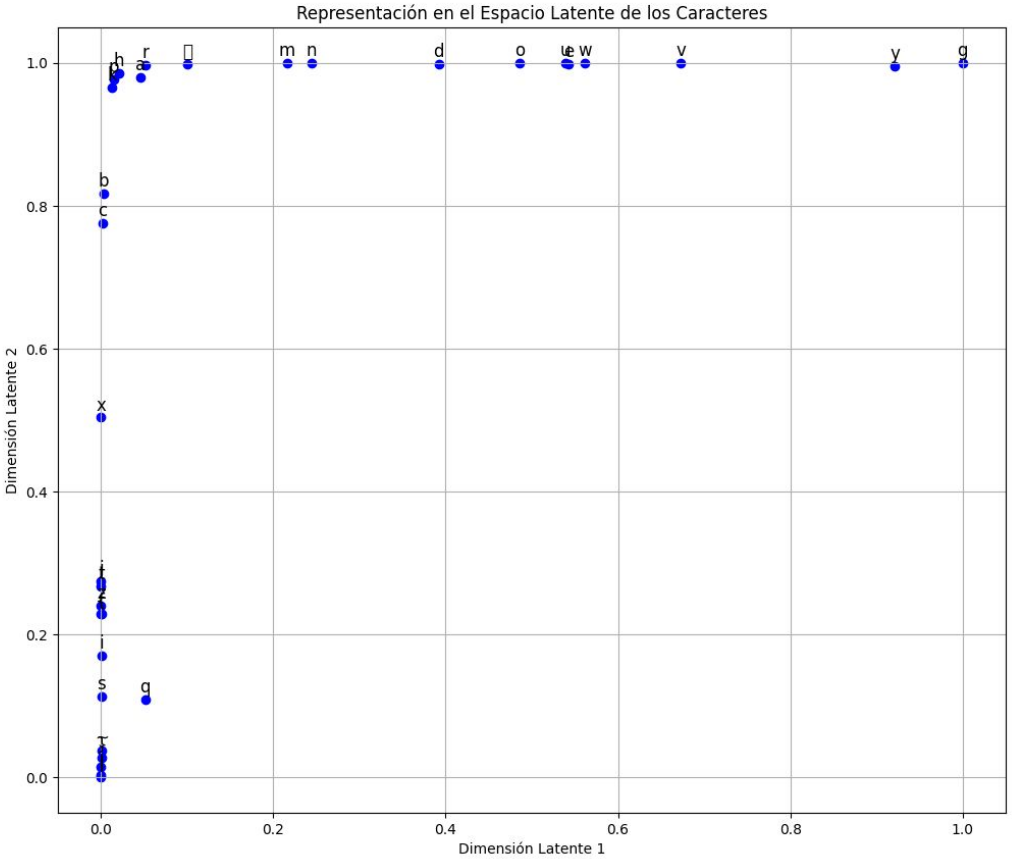


10 runs
Average minimum loss:
0.022765 +- 0.000097

Algunos resultados



Espacio Latente

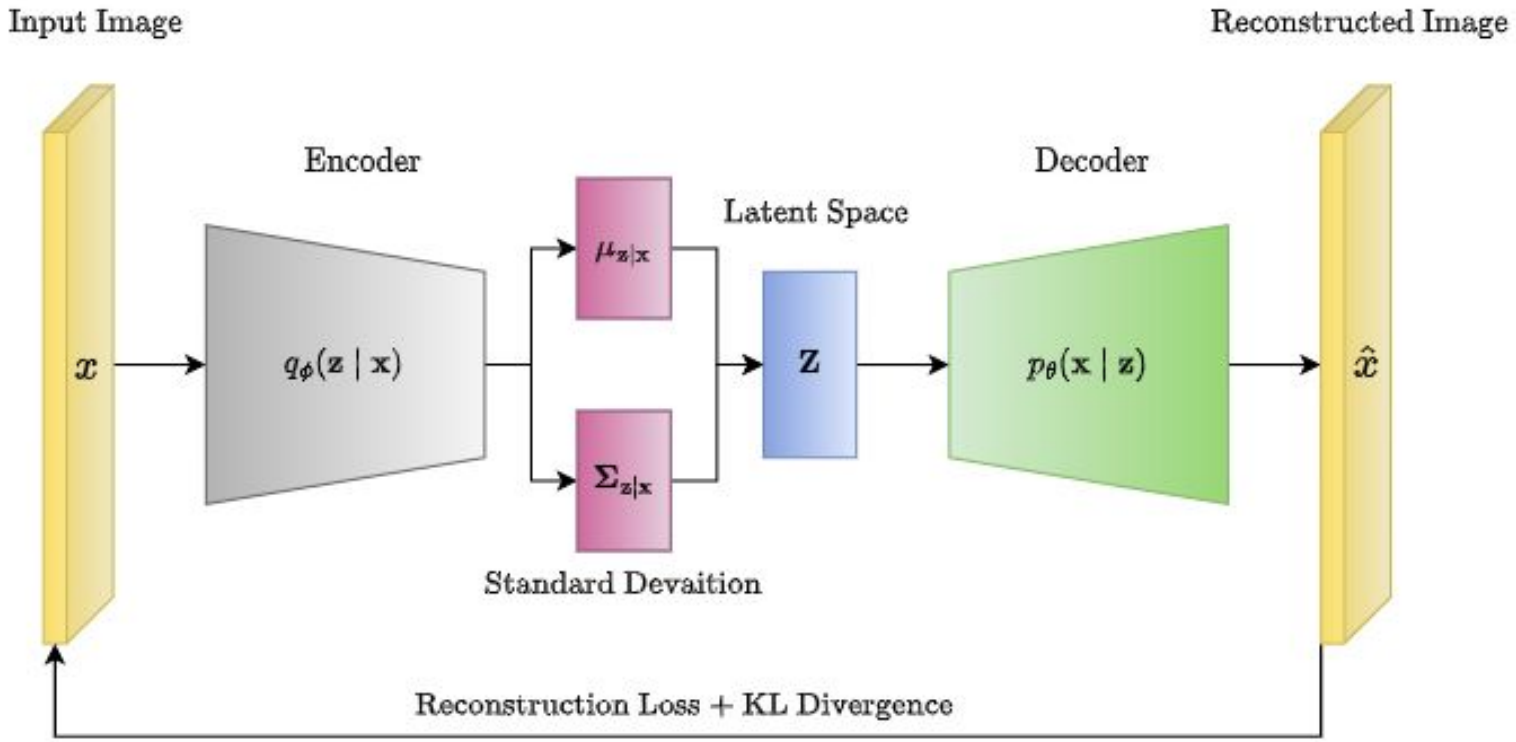


Conclusiones

- A medida que aumenta el ruido:
  - Aparecen más reconstrucciones incorrectas o no esperadas.
  - Aumenta el Loss
  - En el espacio latente se superponen caracteres y no se logra crear una buena separación. Esto indica que no logra discernir correctamente los caracteres.

Variational Autoencoder

Funcionamiento



Emojis :)

Arquitectura

- Input size: 400
- Hidden layers: **256 - 128**
- Dimensiones de espacio latente: **2**
- Optimización con Adam
- Learning rate y betas usados como la recomendación de Adam
- Inicialización de pesos con Xavier

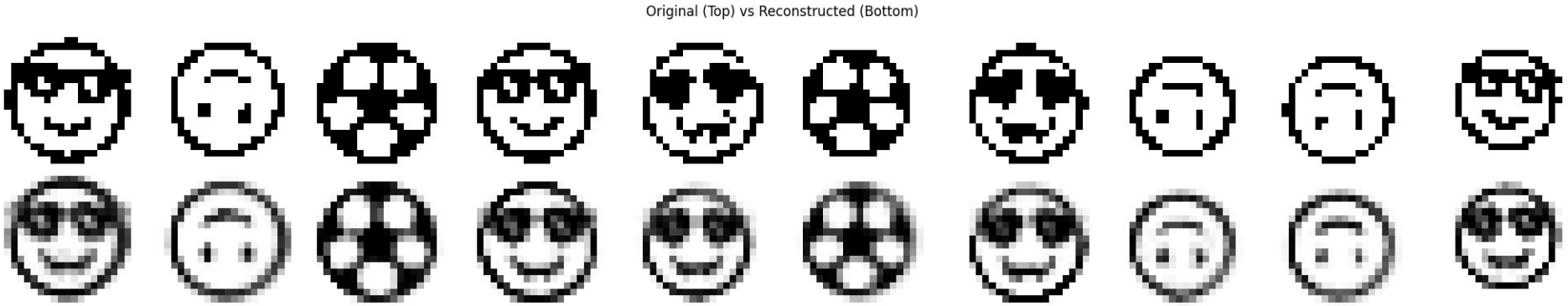
Emojis elegidos

Binarización, rotación y traslación

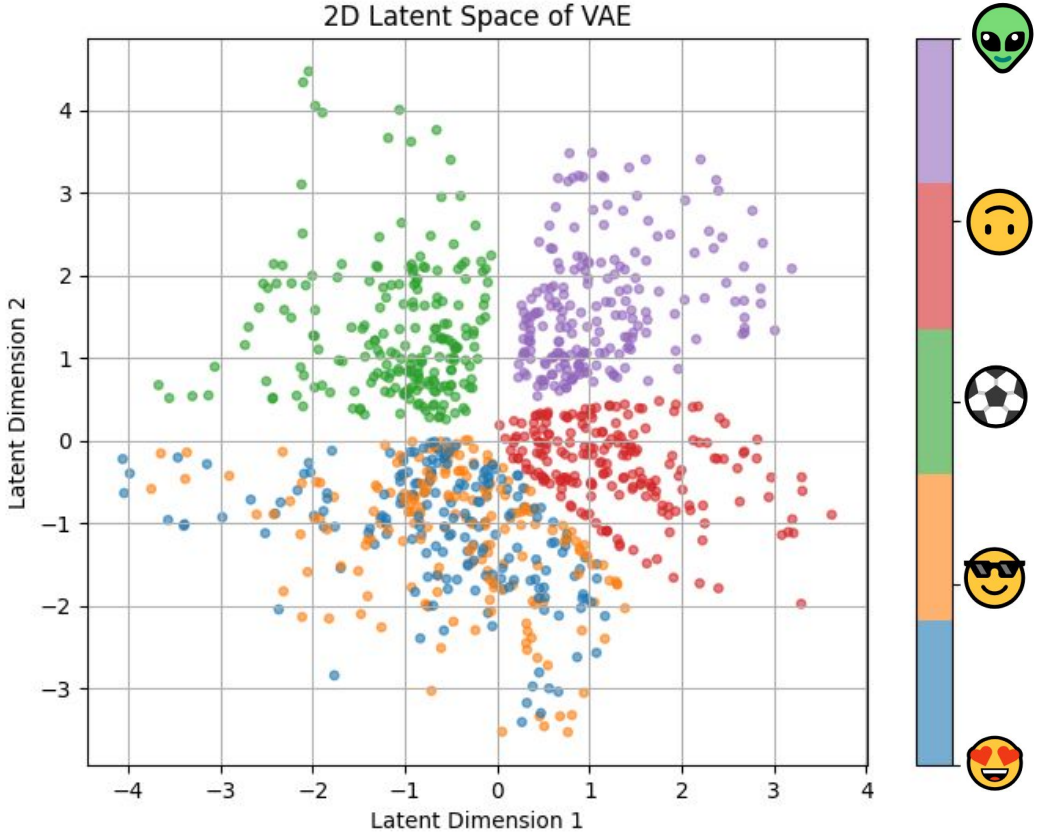
- 20x20 pixels binarizada
- 800 “samples” por emoji
- 4000 emojis totales

Reconstrucciones a partir de un input

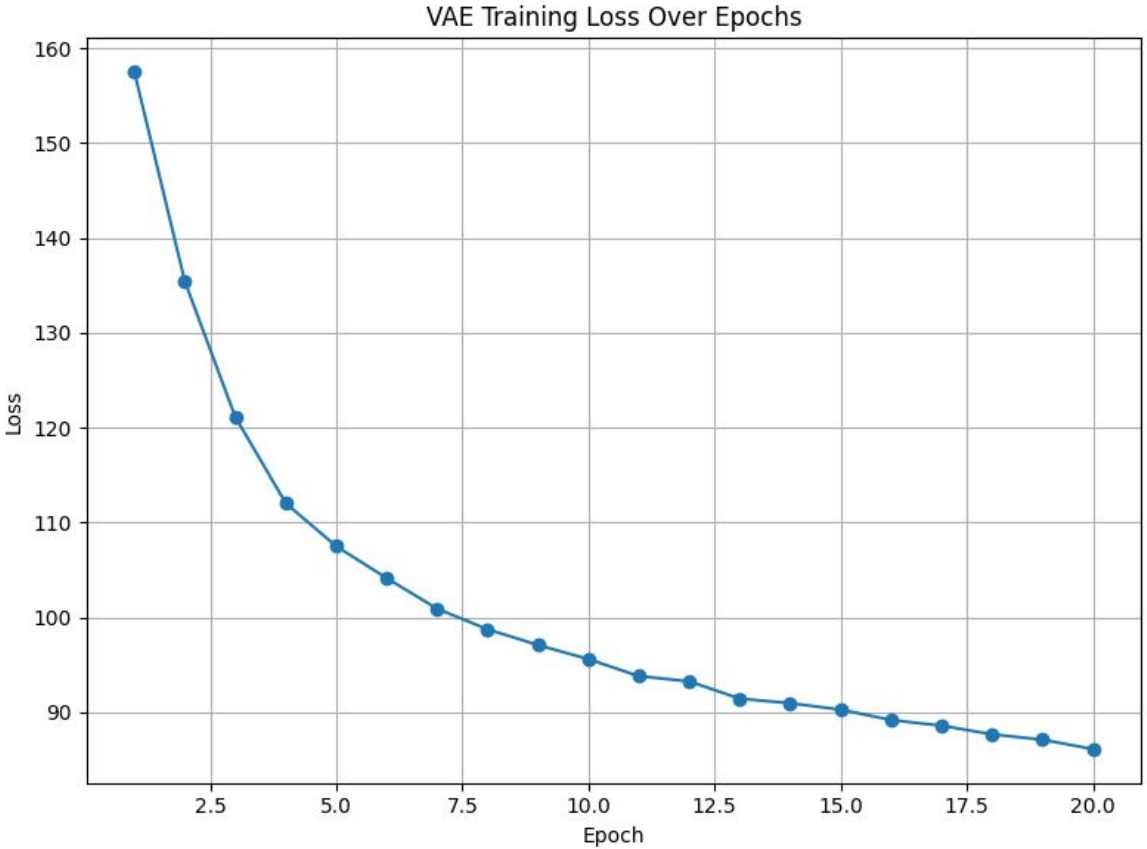
Estos inputs **no** entrenaron a la red



Espacio latente

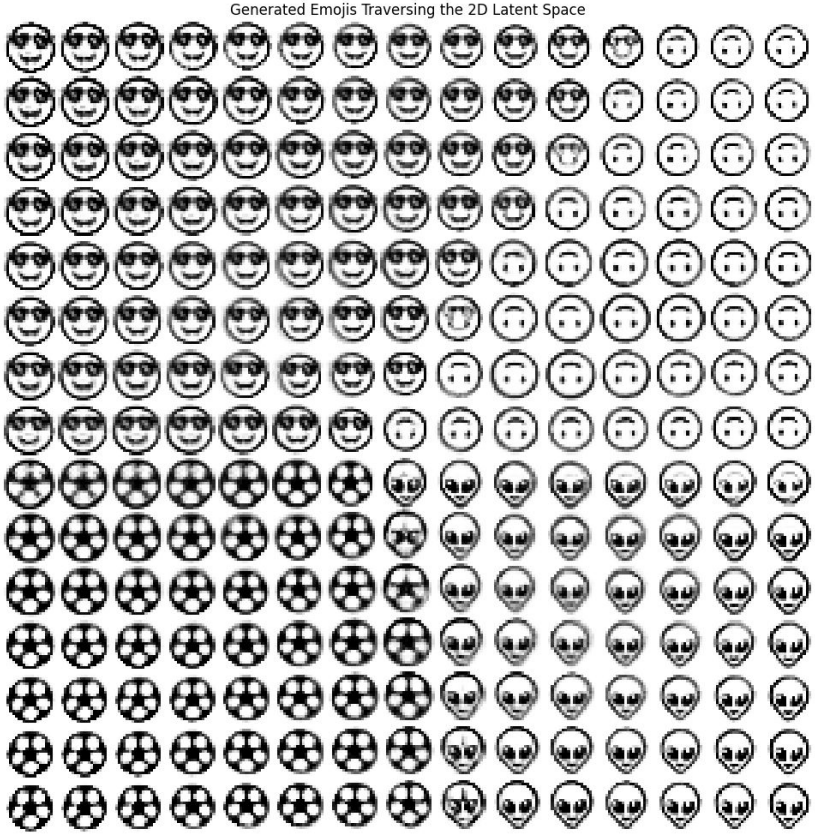


Loss vs epochs



tiempo de entrenamiento: 16min

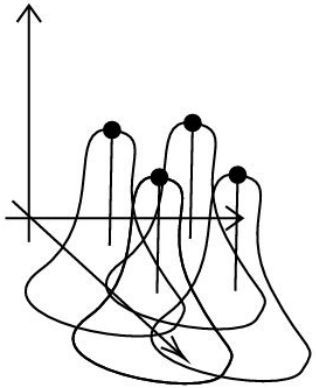
Emojis a través del espacio latente



Nuevos emojis

Conclusiones

- A medida que avanza sobre el espacio latente los emojis se va convirtiendo progresivamente de uno a otro
- En los clusters agrupa a los parecidos en la misma zona debido a su similitud
- Logra generar nuevos emojis “buenos” porque el espacio latente está estructurado y regularizado, permitiendo muestreos coherentes:

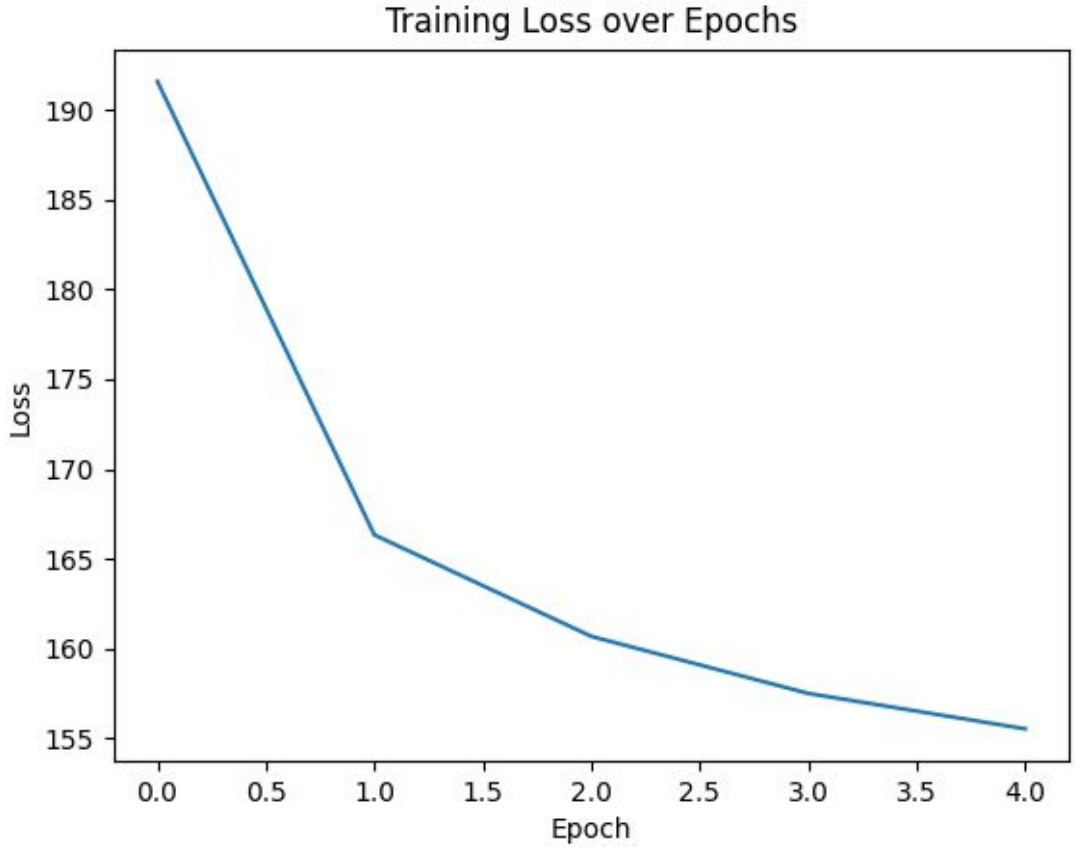


MNIST

Arquitectura

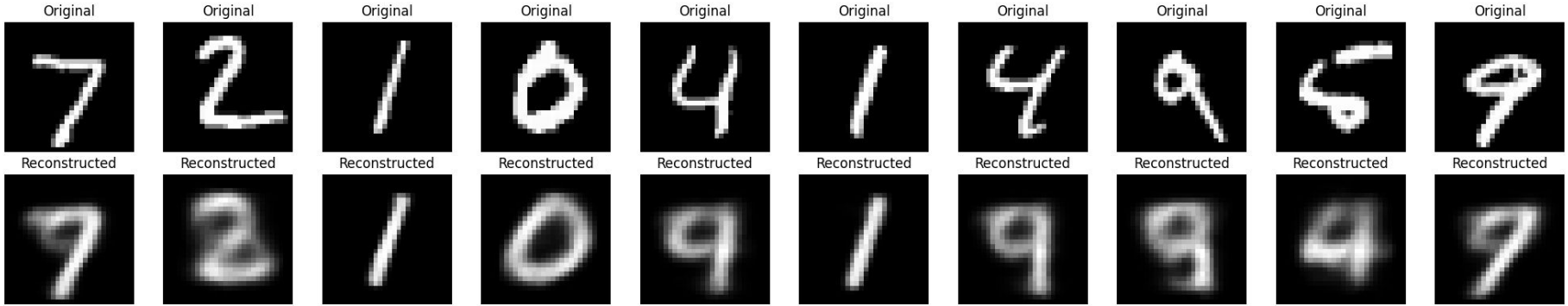
- Input size: 784
- Hidden layers: **512 - 256**
- Dimensiones de espacio latente: **2**
- 5 epochs
- 30k inputs del dataset
- 25 min tiempo de entrenamiento

Binary cross entropy a lo largo de las épocas

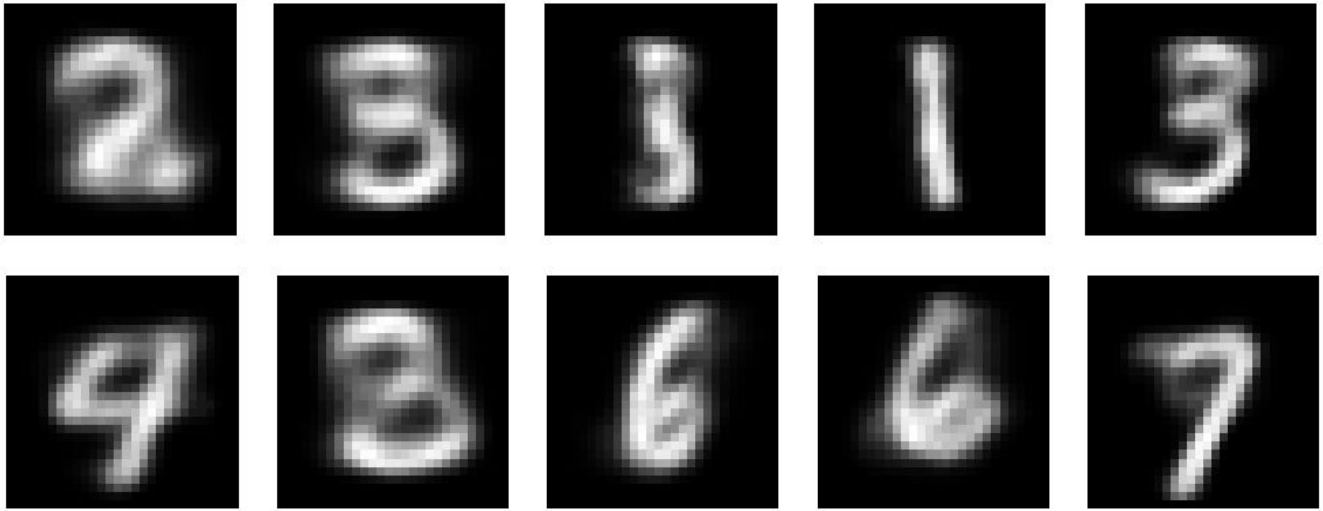


Algunos resultados

Estos inputs **no** entrenaron a la red



Dígitos generados



Conclusiones

- Lograr aprender los patrones pero con poca definición, al igual que los números nuevos que genera
- Son demasiados datos y no logra converger su aprendizaje para tan poca cantidad de dimensiones del espacio latente

Variational Autoencoder con TPU

Problema de performance

Como el dataset de MNIST es muy grande, al correrlo local con solo CPU se dificulta entrenar todo el dataset con una buena cantidad de épocas ya que tarda demasiado.

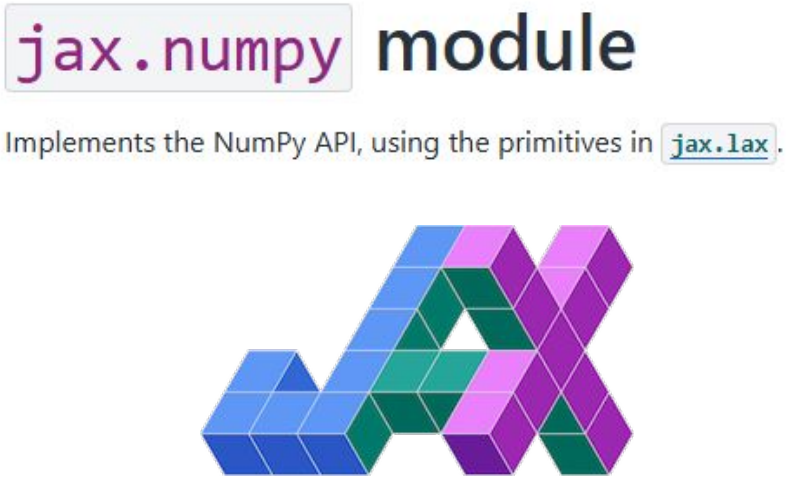
TPU (Tensor Processing Unit)

Una TPU es un acelerador de hardware desarrollado por Google, diseñado específicamente para realizar operaciones de aprendizaje automático.

Google Colab brinda de manera gratuita el uso de una TPU de prueba.

JAX como API para usar la TPU

Por simplicidad utilizamos JAX para poder delegar la complejidad del cálculo y uso de una TPU



Tiempos

Arquitectura: input size: 784, hidden layers: 512-256, latent size: 2

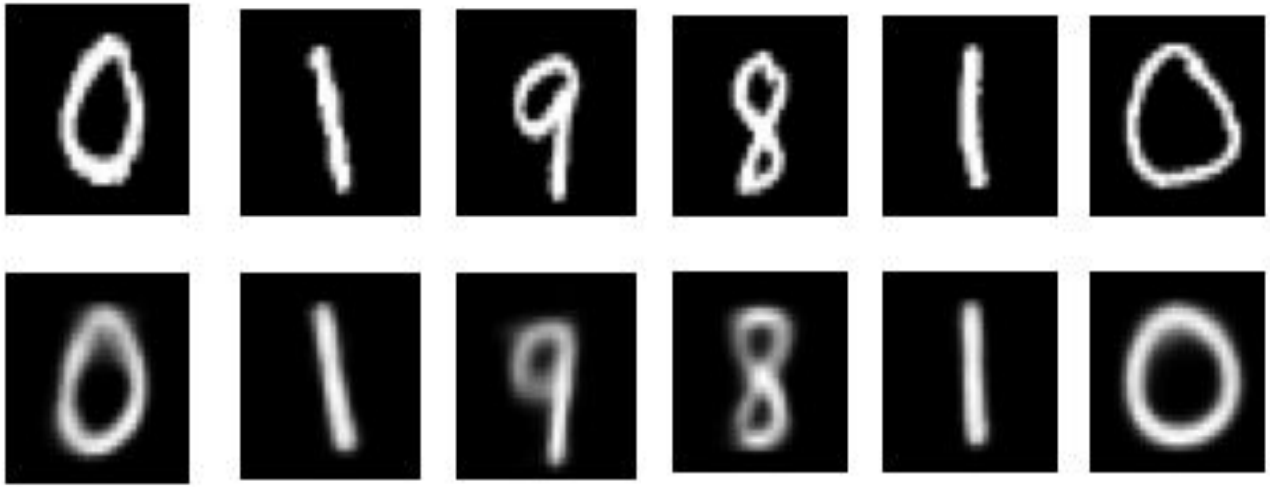
Sin TPU:

5 epochs, 30k inputs del dataset: 25 min

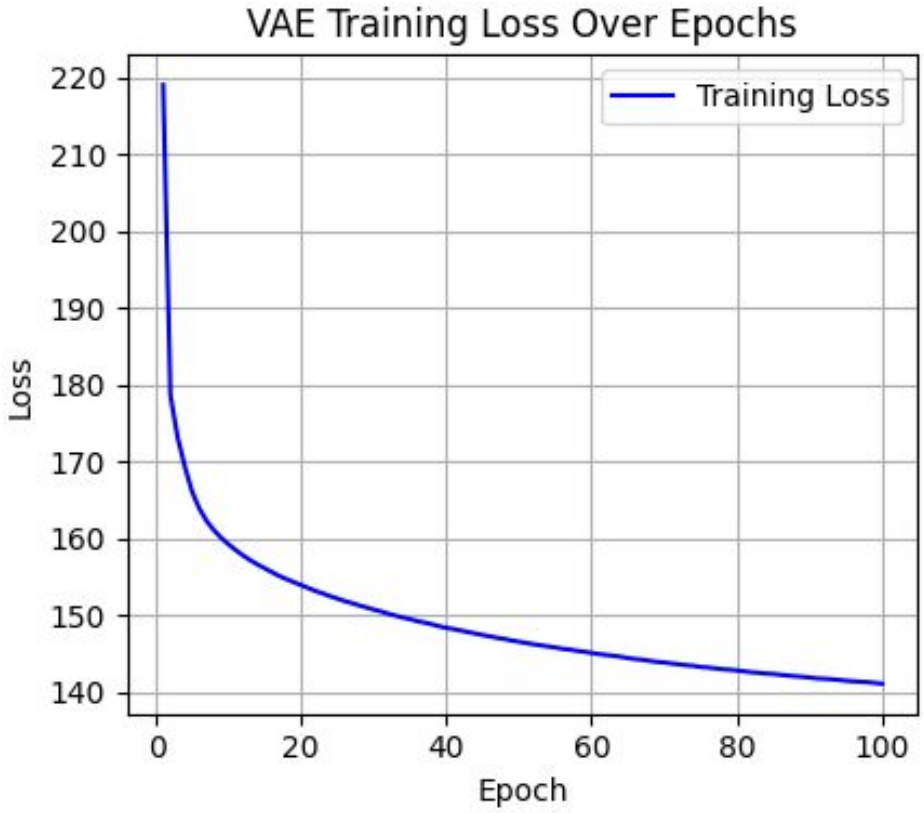
Con TPU:

100 epochs, 60k inputs del dataset: **51s**

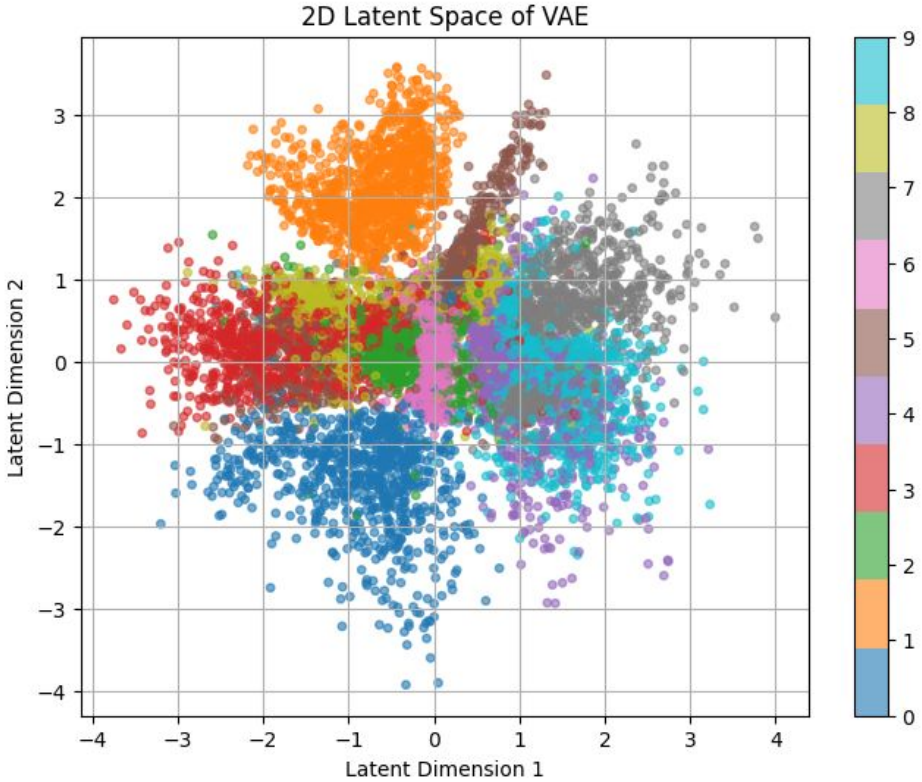
Algunos resultados



Binary cross entropy a lo largo de las épocas

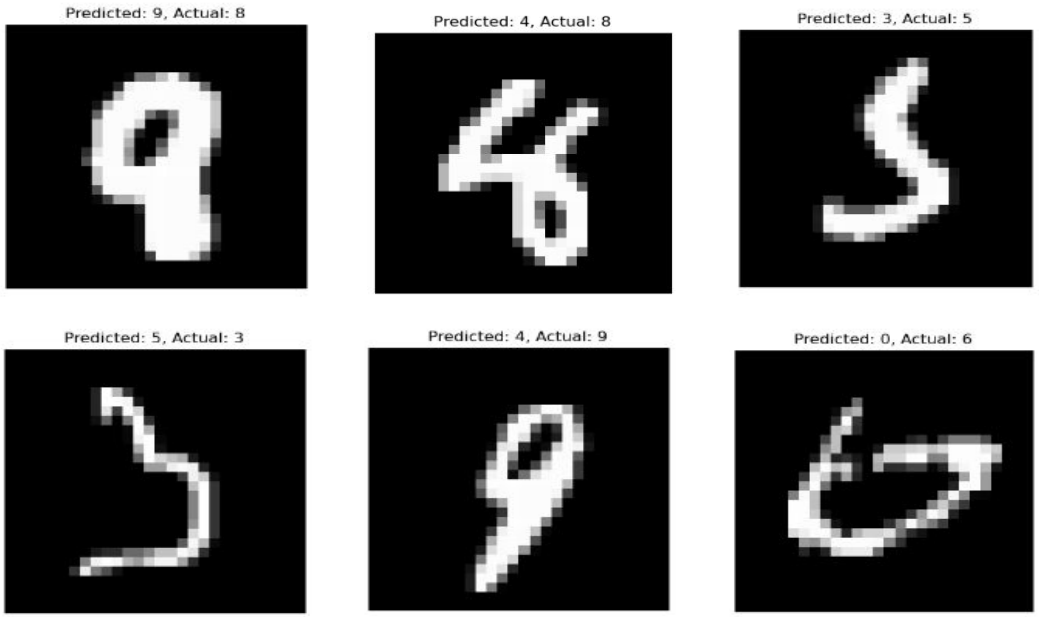


Espacio latente

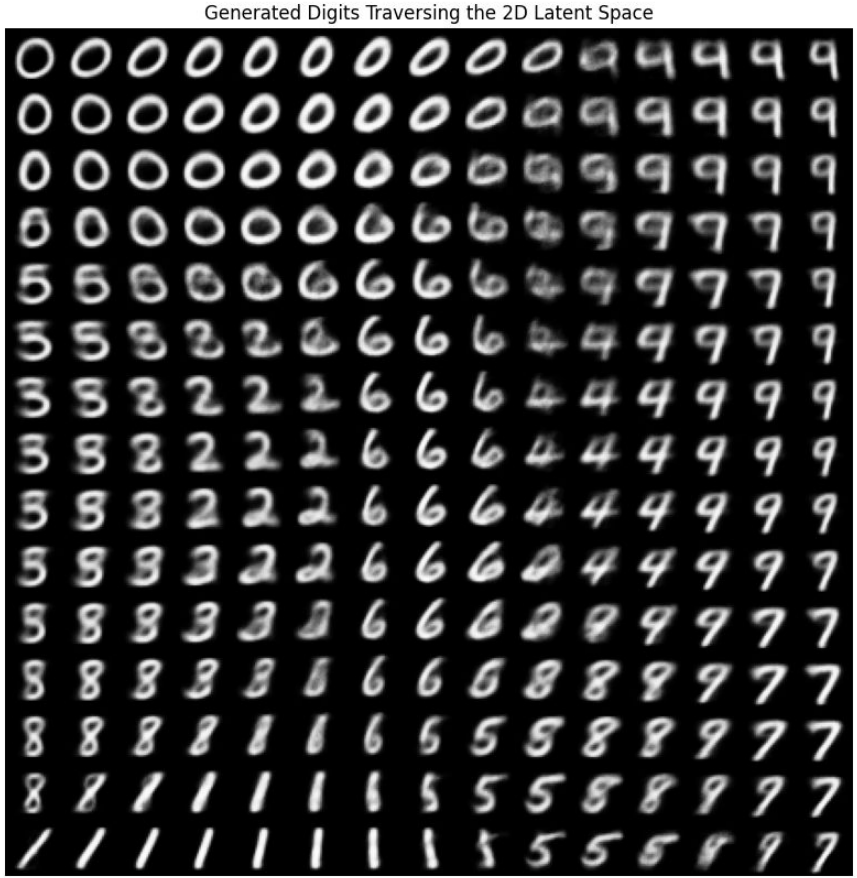


Flashback TP3

- Se confunde entre números que son “parecidos”
  - 4 y 9,
  - 5 y 8,
  - 9 y 7,
  - 5 y 3



Dígitos generados en el espacio latente



Conclusión

- El uso de una TPU acelera abismalmente el tiempo de entrenamiento

Gracias por su atención :)