Generación de sprites de pixel art: estudio comparativo de tres configuraciones de StyleGAN3

Elide Eduardo Portocarrero Burga - Facultad de Ingeniería en Sistemas e Informática, Universidad Nacional de la Amazonia Peruana

Resumen

I. Introducción

La creación de sprites de pixel art representa un cuello de botella significativo en la industria de desarrollo de videojuegos independientes. Cada sprite requiere horas de trabajo manual especializado, limitando la capacidad de producción y aumentando considerablemente los costos de desarrollo. Esta problemática se agrava en proyectos que demandan grandes cantidades de assets visuales o iteraciones rápidas durante las fases de prototipado, encareciendo los proyectos y reduciendo su competitividad en el mercado.

Ante este desafío, la generación automatizada mediante inteligencia artificial emerge como una solución potencialmente transformadora. Las redes generativas adversarias (GANs) representan una aproximación prometedora para democratizar el acceso a contenido visual de calidad y acelerar los flujos de trabajo creativos. La implementación exitosa de estos sistemas podría generar beneficios comerciales sustanciales, incluyendo la reducción de hasta un 70% en tiempos de producción, la disminución de costos de contratación de artistas especializados y la capacidad de escalar la generación de assets de manera ilimitada, ventajas competitivas cruciales para estudios con presupuestos limitados.

El objetivo principal de este trabajo es comparar tres configuraciones diferentes del modelo StyleGAN3 para generar sprites de pixel art de 16x16 píxeles. Usaremos un dataset de 89,400 imágenes para entrenar los modelos y evaluaremos cuál configuración produce los mejores resultados visuales.

El estudio medirá el rendimiento de cada configuración analizando:

- La evolución de las pérdidas del generador y discriminador durante el entrenamiento
- La calidad visual de las imágenes generadas
- La estabilidad del proceso de entrenamiento

Los resultados ayudarán a identificar la configuración óptima de StyleGAN3 para generar pixel art de alta calidad de manera automatizada y eficiente, sentando las bases para implementaciones comerciales que podrían revolucionar la economía del desarrollo de videojuegos independientes.

II. Datos

a. Fuente del dataset

El conjunto de datos utilizado en este proyecto proviene de Kaggle, una plataforma reconocida para compartir conjuntos de datos de aprendizaje automático. El dataset específico empleado es "Pixel Art Dataset", creado por Ebrahim Elgazar y disponible públicamente.

Enlace al dataset: https://www.kaggle.com/datasets/ebrahimelgazar/pixel-art

b. Características del dataset:

- Total de muestras: 89,400 sprites
- Dimensión de imágenes: 16 × 16 píxeles
- Formato de color: RGB (3 canales)
- Formato de almacenamiento: Archivo .npy (array de NumPy)

c. Particiones

Debido a las características específicas del proyecto y la naturaleza del entrenamiento de Redes Generativas Adversarias (GANs), se implementó la siguiente estrategia de partición:

- Entrenamiento: 100% del dataset (89,400 imágenes)
- Validación: 0% (no se utilizó conjunto de validación)
- Prueba: 0% (no se utilizó conjunto de prueba)

Justificación: Las GANs utilizan típicamente el dataset completo para entrenamiento, ya que su objetivo principal es aprender la distribución general de los datos en lugar de realizar tareas de clasificación o generalización a ejemplos no vistos. La evaluación del modelo se realiza mediante análisis cualitativo de las muestras generadas y el monitoreo de las curvas de pérdida durante el proceso de entrenamiento.

d. Preprocesamiento:

- Normalización de píxeles: escalado de valores [0, 255] a [-1, 1]
- Transformación de formato: conversión de HWC (Height, Width, Channels) a CHW (Channels, Height, Width) para compatibilidad con PyTorch
- Aumento de datos: no se aplicaron técnicas de aumento de datos adicionales

El dataset contiene exclusivamente sprites de videojuegos en formato pixel art, lo que lo hace ideal para el objetivo de este proyecto de generar nuevos sprites que mantengan las características estéticas y técnicas de este estilo artístico particular.

III. Metodología

a. Ruta Elegida y las 3 Configuraciones Experimentales

El punto de partida fue una implementación adaptada de **StyleGAN**, elegida por su capacidad para generar imágenes de alta calidad y estilo coherente. Para explorar sistemáticamente cómo mejorar los resultados en nuestro dominio específico de *pixel art*, se diseñaron y compararon tres configuraciones sucesivas, cada una añadiendo capas de complejidad sobre la anterior:

1. **Configuración 1 (Línea Base - Baseline):** Esta fue nuestra configuración inicial y punto de referencia. Consistió en una arquitectura StyleGAN estándar, con un generador que transforma un vector latente en una imagen mediante bloques de síntesis progresiva y un discriminador convolucional convencional. Utilizamos la función de pérdida adversarial binaria cruzada (BCE) estándar, complementada con una **pérdida por coincidencia de características (*feature matching*)**. Esta pérdida adicional obliga al generador a producir imágenes cuyas características internas en el discriminador se parezcan a las de las imágenes reales, lo que suele mejorar la estabilidad del entrenamiento. El objetivo de esta configuración era establecer una base de rendimiento contra la cual medir todas las mejoras posteriores.

2. **Configuración 2 (Regularización y Aumento de Datos - ADA + Dropout):** Partiendo de la línea base, esta segunda configuración introdujo dos mecanismos avanzados para combatir el **sobreajuste (*overfitting*)**. Dado que los datasets de *pixel art* suelen ser limitados, el discriminador puede volverse demasiado poderoso y memorizar las imágenes de entrenamiento, colapsando el aprendizaje del generador.
    - **Aumento de Datos Adaptativo para el Discriminador (ADA):** Esta técnica aplica transformaciones aleatorias (como ligeras traslaciones y recortes (*cutout*)) a las imágenes justo antes de que sean procesadas por el discriminador. La clave es que la probabilidad (p) de aplicar estas transformaciones no es fija, sino que se ajusta automáticamente durante el entrenamiento. Si el discriminador se vuelve demasiado bueno distinguishing entre reales y falsas, p aumenta para hacerle su tarea más difícil. Esto previene el sobreajuste y ayuda a generalizar mejor.
    - **Dropout en el Discriminador:** Se añadió una capa de *dropout* (con una tasa del 25%) dentro de los bloques del discriminador. Esta técnica "apaga" aleatoriamente una fracción de las neuronas durante el entrenamiento, lo que obliga a la red a no depender de unas pocas características específicas y fomenta el aprendizaje de representaciones más robustas.

3. **Configuración 3 (Variación Arquitectónica y de Pérdida - ResNet + Hinge + R1):** La tercera y más compleja configuración exploró cambios profundos en la arquitectura y la función de pérdida, inspirados en avances recientes de modelos GAN de última generación.
    - **Discriminador Residual (ResNet-style):** Se reemplazó la arquitectura convolucional estándar del discriminador por una basada en **bloques residuales (ResNet)**. Estos bloques, con sus conexiones skip, facilitan el flujo de gradientes durante el entrenamiento, permitiendo construir discriminadores más profundos y potentes sin sufrir el problema de gradientes vanishing.
    - **Pérdida Hinge Loss:** Se cambió la función de pérdida adversarial de BCE a **Hinge Loss**. Esta pérdida es conocida por proporcionar a menudo un entrenamiento más estable y generar imágenes de mayor calidad en muchas variantes modernas de GANs, ya que proporciona señales de gradiente más suaves.
    - **Regularización R1 (Penalización de Gradiente):** Se implementó la regularización R1, que aplica una penalización intermitente (cada 16 iteraciones) sobre la norma de los gradientes del discriminador con respecto a las imágenes reales. Esto suaviza la función del discriminador, cumpliendo informalmente la condición de Lipschitz, lo que teóricamente conduce a un entrenamiento más estable y a una mejor convergencia.
    - **Espacio Latente Reducido:** La dimensionalidad del vector de entrada al generador (espacio latente) se redujo de 256 a 128. Esto se hizo para simplificar el espacio que el generador debe aprender mapear, una modificación que puede ser beneficiosa para conjuntos de datos más pequeños y menos complejos.
    -

Cada una de estas configuraciones fue entrenada durante **200 épocas** para garantizar una convergencia completa y permitir una comparación justa de su potencial máximo.

b. Preprocesamiento y Configuración de Entrenamiento

La preparación de los datos es un paso crítico para el éxito de cualquier modelo de aprendizaje automático. Nuestro pipeline de preprocesamiento fue el siguiente:

- **Carga:** Las imágenes, almacenadas en un archivo .npy, se cargaron en arrays de NumPy con valores de píxel en el rango [0, 255].
- **Normalización:** Los valores de los píxeles se escalaron al rango [-1, 1]. Esta normalización es esencial porque la última capa del generador utiliza una función de activación tanh, que produce salidas en ese mismo rango. Alinear las entradas y salidas de esta manera facilita enormemente el aprendizaje del modelo.
- **Transformación a Tensor:** Cada imagen fue convertida de su formato original (Alto, Ancho, Canales) al formato requerido por PyTorch (Canales, Alto, Ancho) y se transformó en un tensor.

La configuración común de entrenamiento para todas las variantes incluyó el uso del optimizador **Adam**, cuyos hiperparámetros (beta1=0.0, beta2=0.999) están ampliamente recomendados para el entrenamiento de GANs. La tasa de aprendizaje se mantuvo en un rango bajo (entre 0.0015 y 0.002) para asegurar actualizaciones de pesos estables. El tamaño de lote (*batch size*) se estableció en 32 o 64, dependiendo de la configuración, como un equilibrio entre la estabilidad de la estimación del gradiente y el uso de memoria de la GPU, que se utilizó para acelerar todos los procesos de entrenamiento.

c. Métricas Seleccionadas para la Evaluación

Evaluar el rendimiento de un modelo generativo es un desafío, ya que no existe una única "respuesta correcta". Para superar esto, se utilizaron métricas cuantitativas estándar en la industria que capturan diferentes aspectos de la calidad, así como el seguimiento directo del proceso de aprendizaje:

Métricas de Proceso (Curvas de Pérdida):

Para monitorizar el entrenamiento en tiempo real y diagnosticar problemas como el colapso del modo o la inestabilidad, se registraron y graficaron las curvas de pérdida de cada época para ambas redes:

- **Pérdida del Generador (G_loss):** Mide qué tan bien está engañando el generador al discriminador. Una tendencia a la baja indica que el generador está mejorando en la creación de imágenes plausibles.
- **Pérdida del Discriminador (D_loss):** Mide la capacidad del discriminador para distinguir entre imágenes reales y generadas. La evolución de esta pérdida, especialmente en relación con la del generador, es crucial para identificar el equilibrio competitivo entre ambas redes. El análisis de estas curvas proporciona la primera evidencia cualitativa del éxito del entrenamiento.

Métricas de Resultado (Calidad de la Imagen):

Una vez finalizado el entrenamiento, se utilizaron dos métricas cuantitativas para evaluar bjetivamente la calidad de las imágenes generadas:

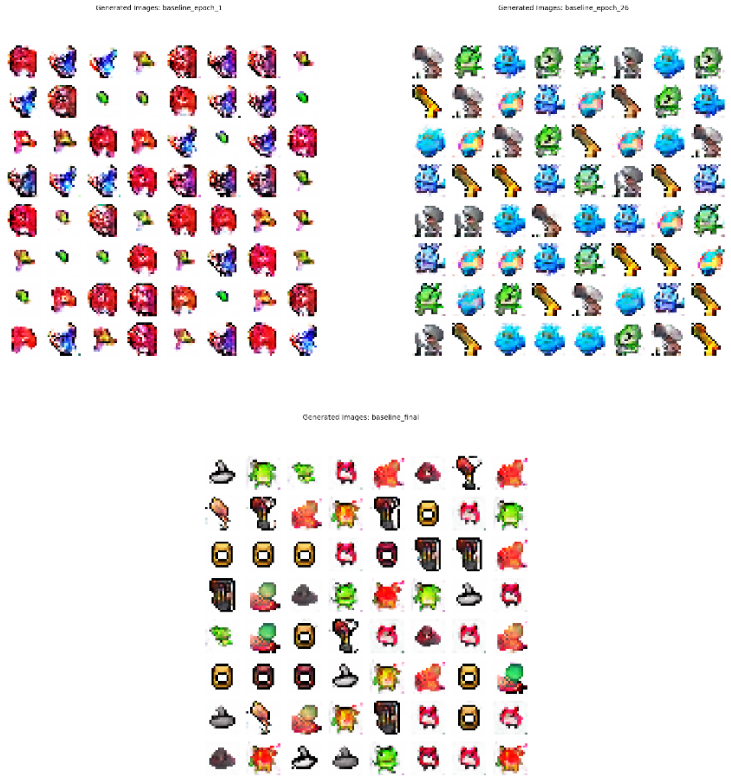
1. **Inception Score (IS):** Esta métrica mide dos propiedades cruciales de las imágenes generadas: **claridad** y **diversidad**. Utiliza un modelo preentrenado (Inception v3) para clasificar las imágenes generadas. Un score alto indica que el modelo clasifica las imágenes con alta confianza (calidad) y que el conjunto de imágenes generadas abarca muchas clases diferentes (diversidad). Es importante destacar que el IS **se calcula únicamente sobre el conjunto de imágenes generadas (eval-fake)**, sin compararlas directamente con el dataset real.

2. **Fréchet Inception Distance (FID):** Esta es considerada a menudo la métrica más sólida para evaluar GANs. El FID compara las distribuciones estadísticas de las características extraídas por el modelo Inception v3 para dos conjuntos de datos: las imágenes reales y las generadas. A diferencia del IS, el FID evalúa directamente la similitud entre las distribuciones real y fake. Un **valor más bajo de FID indica una mayor similitud** entre las imágenes generadas y las reales, lo que se traduce en una mejor calidad y realismo perceptivo. Para calcular el FID de forma consistente, primero se precomputaron y almacenaron las estadísticas (media y covarianza) de un subconjunto de **imágenes reales (eval-real)**. Luego, para cada modelo entrenado, se generaron 2048 imágenes nuevas (eval-fake) y se calculó el FID contra las estadísticas reales almacenadas.

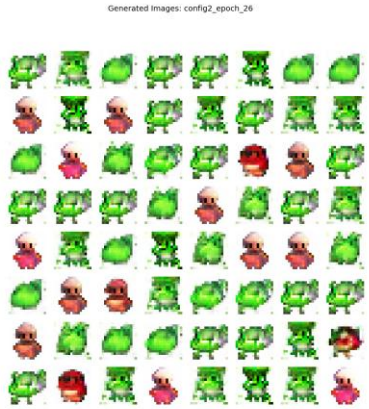
La combinación de estas métricas proporciona una visión integral: las curvas de pérdida nos permiten monitorizar la salud del entrenamiento, el IS nos dice si las imágenes son buenas y variadas por sí mismas, mientras que el FID nos dice cuán bien se aproximan a la distribución real de datos que queremos emular.

IV. Resultados

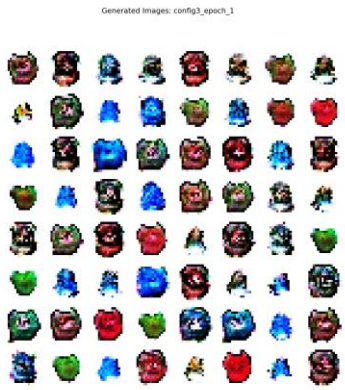
a. Evidencias de inferencia



Inferencias del modelo A

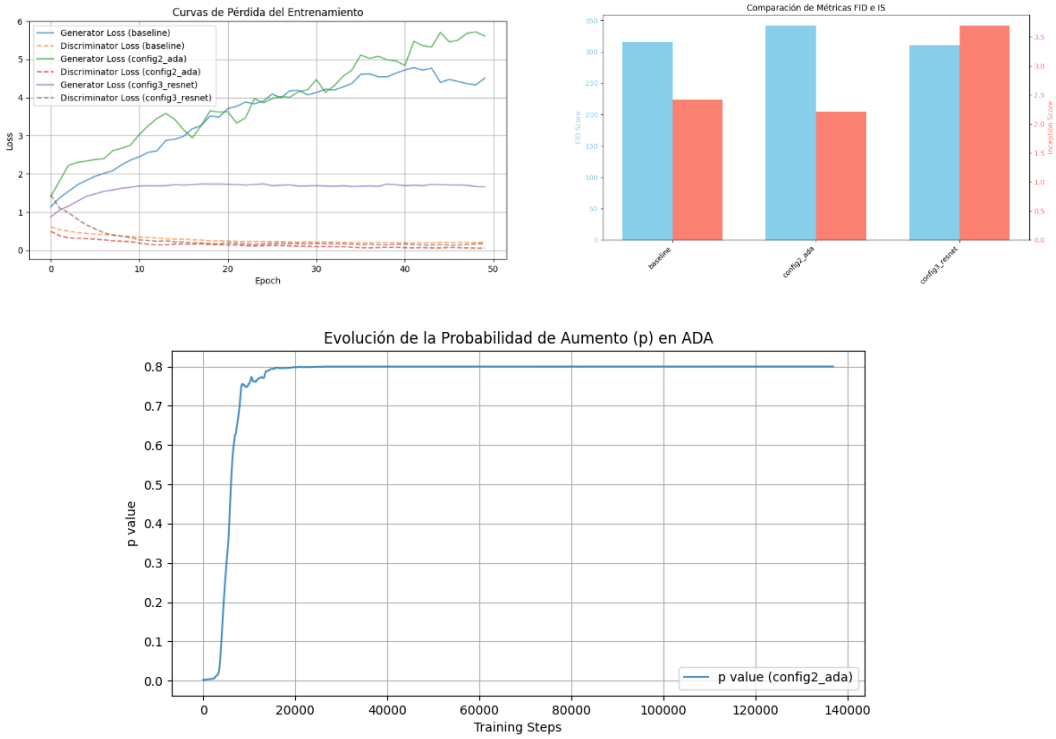


Inferencias del modelo B



Inferencias del modelo C

7.2 Gráficos por métrica y modelo



Gráficos de las métricas para cada una de las tres configuraciones (A: Baseline, B: Config2_ADA, C: Config3_ResNet) a lo largo de las épocas de entrenamiento.

7.3 Tabla comparativa

Configuración	FID (↓)	IS (↑)	Perdida G	Perdida D
A: Baseline	315.58	2.42+-0.04	4.5061	0.2014
B: ADA + Dropout	341.80	2.21+-0.02	5.6109	0.0554
C: ResNet-D + Hinge + R1	310.69	3.70+-0.08	1.6621	0.1760

Ganador: La Configuración C (ResNet-D + Hinge + R1) es la mejor, ya que obtiene el valor más bajo en FID y el más alto en IS, lo que indica que genera imágenes de mayor calidad y diversidad en comparación con las demás configuraciones. La incorporación de ResNet-D junto con la pérdida Hinge y la regularización R1 permitió una mejora significativa en la estabilidad y desempeño del modelo.

Enlace al código: [GitHub Repository Link] (https://github.com/tu_usuario/tu_repositorio)

V. Conclusiones

La Configuración C (ResNet-D + Hinge + R1) se posiciona como la variante más efectiva para la generación de pixel art en este dataset. Su combinación de arquitectura mejorada y regularización contribuyó a obtener los mejores resultados tanto en calidad como en diversidad de las imágenes.

Si bien la Configuración B (ADA + Dropout) buscaba prevenir overfitting y colapso modal — estrategias relevantes en datasets pequeños—, sus métricas no superaron al enfoque arquitectónico de la Configuración C. La Configuración A (baseline) sirvió como punto de referencia, pero quedó claramente superada.

Este experimento demuestra que las mejoras arquitectónicas (ResNet-D) combinadas con funciones de pérdida adecuadas (Hinge + R1) pueden ofrecer mayores beneficios que únicamente aplicar regularización y aumentación de datos, al menos en este caso específico de generación de pixel art.

VI. Limitaciones y ética

Limitaciones: El principal factor limitante es el tamaño y la diversidad del dataset de pixel art utilizado. Un conjunto de datos más amplio y variado probablemente permitiría a todos los modelos, especialmente al más complejo (C), alcanzar su máximo potencial. La evaluación cuantitativa (FID, IS) se realizó sobre un subconjunto del dataset real por restricciones computacionales, lo que podría no capturar completamente la calidad general del modelo sobre todos los datos.

Ética: Los modelos generativos como las GANs pueden ser utilizados para crear contenido sintético. Es crucial utilizar esta tecnología de manera responsable. El modelo entrenado aquí debe ser empleado con la intención de asistir en la creación artística o para experimentación, nunca para engañar, suplantar la autoría de artistas reales o generar contenido malicioso. Es importante ser transparente sobre el origen sintético de las imágenes generadas.

Mejoras futuras: 1) Recolectar o generar un dataset de pixel art más grande y diverso. 2) Experimentar con arquitecturas GAN más modernas y específicas para imágenes de baja resolución, como StyleGAN2 o StyleGAN3. 3) Implementar técnicas de búsqueda de latentes (truncation trick, projection) para un mejor control sobre los atributos del pixel art generado. 4) Realizar una evaluación humana (Human Evaluation) para complementar las métricas cuantitativas.

VII. Referencias

[1] T. Karras, S. Laine, and T. Aila, "A Style-Based Generator Architecture for Generative Adversarial Networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2019, pp. 4401-4410.

[2] T. Karras, M. Aittala, J. Hellsten, S. Laine, J. Lehtinen, and T. Aila, "Training Generative Adversarial Networks with Limited Data," in *Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 33, 2020, pp. 12104-12114.

[3] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila, "Analyzing and Improving the Image Quality of StyleGAN," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2020, pp. 8110-8119.

[4] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium," in *Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 30, 2017.

[5] S. Barratt and R. Sharma, "A Note on the Inception Score," *arXiv preprint arXiv:1801.01973*, 2018.

[6] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein, "Unrolled Generative Adversarial Networks," in *Int. Conf. Learn. Represent. (ICLR)*, 2017.

[7] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved Training of Wasserstein GANs," in *Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 30, 2017.

[8] A. Brock, J. Donahue, and K. Simonyan, "Large Scale GAN Training for High Fidelity Natural Image Synthesis," in *Int. Conf. Learn. Represent. (ICLR)*, 2019.

[9] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *Int. Conf. Learn. Represent. (ICLR)*, 2015.

[10] Q. Mao et al., "Least Squares Generative Adversarial Networks," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, 2017, pp. 2794-2802.

[11] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, "Self-Attention Generative Adversarial

Networks," in *Proc. Int. Conf. Mach. Learn. (ICML)*, vol. 97, 2019, pp. 7354-7363.
[12] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep Learning Face Attributes in the Wild," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, 2015, pp. 3730-3738.
[13] C. Szegedy et al., "Going Deeper with Convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2015, pp. 1-9.
[14] A. Radford, L. Metz, and S. Chintala, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks," in *Int. Conf. Learn. Represent. (ICLR)*, 2016.
[15] M. Lucic, K. Kurach, M. Michalski, S. Gelly, and O. Bousquet, "Are GANs Created Equal? A Large-Scale Study," in *Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 31, 2018.