Implementación de una Super Resolution GAN con aplicación a campos de velocidad de simulaciones LES

Maximiliano Bove Pugliese
Montevideo, Uruguay
mbove@fing.edu.uy

I. Motivación

En el área de la Mecánica de los Fluidos Computacional existen diversos métodos bien conocidos para modelar flujos turbulentos. A pesar de que estos responden a las ecuaciones de Navier-Stokes, hasta ahora no existen soluciones analíticas generales para las mismas. Si bien los métodos numéricos aplicados a las simulaciones CFD (*Computational Fluid Dynamics*) permiten obtener soluciones aproximadas, los recursos computacionales requeridos para realizar simulaciones numéricas directas (DNS) con Navier-Stokes para aplicaciones ingenieriles pueden ser excesivamente grandes, especialmente para flujos muy turbulentos. Algunos métodos como el LES (*Large Eddie Simulation*) se encargan de modelar solamente las escalas turbulentas de baja frecuencia (grandes vórtices) y evaluar los efectos de las pequeñas escalas sin tener que modelarlas explícitamente. Se trata de un compromiso entre costo computacional y la resolución de las escalas turbulentas presentes.

Una alternativa que permitiría economizar recursos computacionales, que se ha empezado a explorar recientemente, es la posibilidad de aumentar la resolución de las simulaciones CFD haciendo uso de Redes Neuronales Generativas Profundas. Los trabajos de Werhahn et al. [1] con su *Multi-Pass GAN*, Xie et al. [2] con *TempoGAN* o Subramaniam et al. con *TEGAN* [3] son algunos de los investigadores que han desarrollado redes generativas para este fin. Estos artículos, cuentan con enfoques similares, partiendo todos del artículo de Ledig et al. *"Photo-Realistic SingleImage Super-Resolution Using a Generative Adversarial Network"* (SRGAN) [4] y adaptándolo a las necesidades y restricciones de un sistema físico como es una simulación CFD.

Como introducción y primer abordaje en el área nos proponemos a implementar una SRGAN intentando replicar lo hecho por los autores y aplicándolo a imágenes de planos de campos de velocidad.

II. Descripción del problema

El problema a abordar consiste en implementar y aplicar una Red Generativa Antagónica que sea capaz de aumentar la resolución de imágenes de baja resolución. Las imágenes generadas, llamadas de super resolución, serán comparadas con las originales de alta resolución utilizando ciertas métricas. Para lograr este fin seguiremos a los autores del ya mencionado canónico artículo SRGAN.

II-A. Conjunto de datos disponibles

Contamos con dos datasets de imágenes. Por un lado un dataset llamado $DIV2K$[1] con 900 fotografías genéricas de resolución $(462 \times 510\text{px})$[2]. Por otro, tenemos 1609 imágenes de un plano a 90m de altura (altura de buje del aerogenerador) de la primera componente U_x del campo de velocidades de simulaciones LES alrededor de aerogeneradores (lo llamamos UxLES dataset). Estas imágenes son obtenidas a partir del vector de velocidades utilizando *colormap jet* y adecuando la escala de colores (de 2 a 12 m/s)[3] para abarcar todos los cambios de velocidad U_x presentes a lo largo del dataset que está compuesto por instantáneas de velocidad en diferente pasos temporales. Guardamos estos planos recortados tal que consistan en 64×64 puntos de información, es decir pixels.

A ambos datasets les aplicamos un *downscaling* de $4\times$. (Ver figuras 1 y 2)

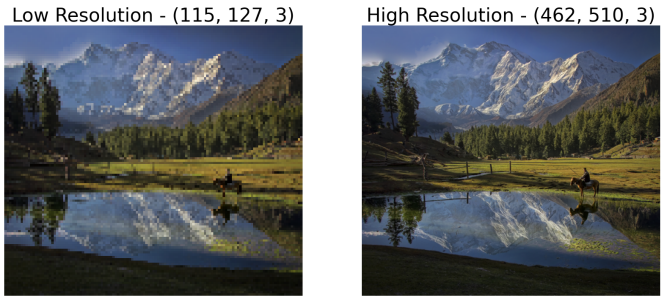

Figura 1. Ejemplo de imagen de DIV2K dataset. A la izquierda, imagen de baja resolución de 115×127 px y a la derecha imagen original de alta resolución de 462×510 px

III. Estrategia de resolución

La estrategia consiste en tomar imágenes de alta resolución (HR), aplicarles un recorte aleatorio de un tamaño definido de

[1] https://data.vision.ee.ethz.ch/cvl/DIV2K/

[2] Elegimos las versiones de resolución reducida de este dataset *Train Data Track 1 bicubic downscaling x4 (LR images)* como imágenes de alta resolución para este proyecto y así facilitarnos a la hora de entrenar la red.

[3] Si la velocidad en un punto es de 2 m/s el color en ese punto será RGB=[0,0,1] y si la velocidad es de 12 m/s el color será RGB=[1,0,0]

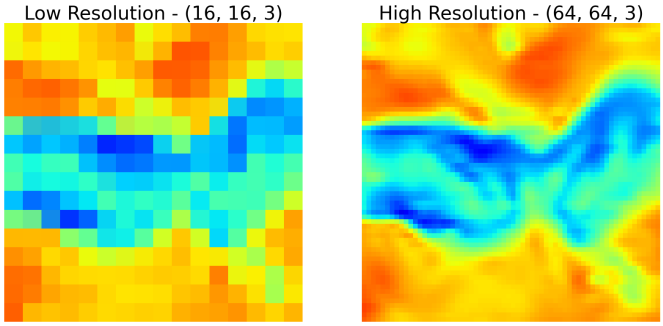

Figura 2. Ejemplo de imagen de UxLES dataset. A la izquierda, imagen de baja resolución de 16×16 px y a la derecha imagen original de alta resolución de 64×64 px

96×96 para el caso del dataset DIV2K y de 32×32 para el UxLES dataset (ya que en este último las imágenes tienen un tamaño de 64×64), disminuirle la escala para tener una imagen de baja resolución $4\times$ menor, (24×24) y normalizar tanto HR como LR (extrañamente, la normalización aplicada por los autores es diferente si la imagen es HR o LR). Luego, se procede a entrenar a un Generador y un Discriminador con estos datos con el objetivo de optimización de obtener imágenes de super resolución que el Discriminador no sepa distinguir de la original de alta resolución.

Se utiliza un *batch size* de 16, como los autores, una *learning rate* de 1e-4 y un número de épocas variable.

A continuación se realizar una síntesis de la arquitectura de la red y el tratamiento de las funcione de pérdida.

III-A. Arquitectura

En este trabajo nos proponemos implementar una Red Generativa Antagónica con el fin de generar imágenes de super resolución empleando redes residuales profundas (ResNet) con *skip-connection* y normalización de batches para contrarestar el cambio de covariable interna (*internal covariate shift*) [4].

III-A1. Redes residuales: La necesidad de utilizar Redes Residuales (ResNet) con *skip-connection* se debe a la dificultad de las capas subyacentes de un modelo de propagar la información de las capas más superficiales a las más profundas, por lo que la información se va degradando. Para resolver este problemas, las capas más profundas tienen que propagar la información de las capas superficiales directamente, es decir, con identidad de mapeo. En este sentido, una solución es conectar directamente las capas superficiales con las profundas así la información es pasada como en la función identidad (Ver Figura 4). Entonces, en las *skip-connection* el resultado de una neurona es simplemente añadido directamente a la neurona correspondiente de la capa profunda [5]

III-A2. Generador: El Generador toma como input una imagen de baja resolución, se lo envía a una CNN (Ver Figura 3) con PReLU como función de activación (a diferencia de una LeakyReLU se tiene una pendiente diferente por cada canal de salida). Luego, se definen los B bloques residuales ($B = 16$ según sugieren los autores) que consisten en una CNN, seguido de *batch normalization* (BN) con el fin de estandarizar los inputs de una capa para cada mini-batch y así estabilizar el proceso de aprendizaje y reducir el número de épocas de entrenamiento requeridas [6], seguido de una activación PReLU, CNN, BN, y finalmente una simple suma del input del bloque (aquí llega la *skip-connection*). A continuación de los B bloques residuales viene una nueva CNN, BN, y suma de la *skip-connection* proveniente del inicio de la serie de bloques residuales para asegurarse de que no se ha perdido información en el transcurso de los B bloques. El aumento de escala se realiza en dos sucesivos aumentos de factor $2\times$ al definir a continuación una CNN con una cantidad de canales de salida 4 veces mayor a la entrada[7], y luego realizando un *PixelShuffler* que distribuye ese aumento de canales en el ancho y alto del tamaño de la imagen. Es decir:

$$C \cdot 4 \times H \times W \rightarrow C \times H \cdot 2 \times W \cdot 2$$

Este paso se realiza dos veces y así obtener un aumento de cuatro veces la resolución original.

Finalmente, se agrega una última CNN con 3 canales de salida correspondientes a los colores RGB.

III-A3. Discriminador: En el caso del Discriminador el input es la imagen de alta resolución (ya sea generada u original) y se lo pasa por una CNN con activación Leaky-ReLU, seguido de 7 bloques de CNN, BN y LeakyReLU, con sucesivos aumentos de tamaño de kernel y alternando con *stride* 1 o 2. Luego, se pasa por una red lineal con LeakyReLU, y finalmente otra líneal con activación sigmoide para devolver la probabilidad de que las dos imágenes sean idénticas según el criterio aprendido por el Discriminador.

III-B. Funciones de pérdida

En el artículo en cuestión señalan la conveniencia de apartarse de la minimización del *Mean Squared Error* (MSE) (Figura 5) (o maximización del *peak signal-to-noise ratio* (PSNR)) entre la imagen generada y la original como objetivo de optimización ya que la habilidad de estas métricas de capturar diferencias relevantes, como los detalles de textura es muy limitada por estar definidas como simples diferencias en pixels. Se propone reemplazar la función de pérdida basada en MSE por una calculada sobre el mapa de *features* de una red VGG (Figura 6).

Es decir, esta nueva función de pérdida propuesta mide las diferencias entre las *features* que genera VGG de la imagen generada frente a las de la imagen original de alta resolución. Específicamente, la pérdida VGG que se propone está basada en las capas de activación ReLU de la red de 19 capas pre entrenada VGG.

[4]Se define como el cambio en la distribución de las activaciones de la red debido al cambio en los parámetros de la red durante el entrenamiento.

[5]https://towardsdatascience.com/intuition-behind-residual-neural-networks-fa5d2996b2c7

[6]https://machinelearningmastery.com/batch-normalization-for-training-of-deep-neural-networks/

[7]Recordemos que aumentar al doble el tamaño de una imagen, corresponde a cuadruplicar la cantidad de pixels

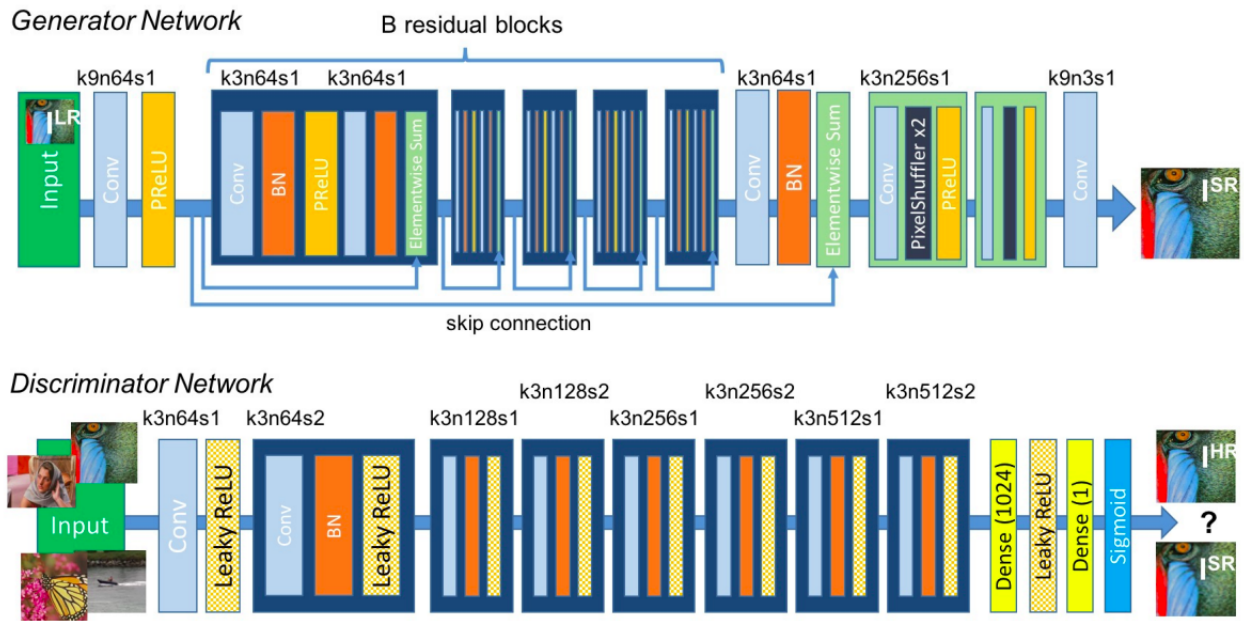

Figura 3. Arquitectura del Generador y Discriminador con tamaño de kernel (k), número de *features* (n) y *stride* (s)

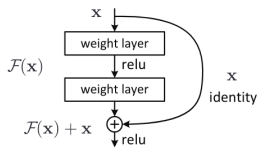

Figura 4. Esquema de un bloque residual. Notar como la *skip connection* contribuye a traer la función identidad a las capas más profundas

$$l^{SR}_{MSE} = \frac{1}{r^2WH}\sum_{x=1}^{rW}\sum_{y=1}^{rH}(I^{HR}_{x,y} - G_{\theta_G}(I^{LR})_{x,y})^2$$

Figura 5. Expresión de la función de pérdida *Mean Squared Error* (MSE)

Además, se tiene la función de pérdida esencial al entrenamiento del Generador, que busca engañar al Discriminador haciendole creer que la imagen de super resolución que genera es la imagen real. Para ello se maximiza (en la práctica se minimiza el negativo) la evaluación del Discriminador de la imagen generada (Figura 7). En el artículo sugieren sumar la pérdida antagónica junto con la VGG. Se podría suponer que la pérdida MSE es directamente sustituida por VGG, pero

$$l^{SR}_{VGG/i.j} = \frac{1}{W_{i,j}H_{i,j}}\sum_{x=1}^{W_{i,j}}\sum_{y=1}^{H_{i,j}}(\phi_{i,j}(I^{HR})_{x,y} - \phi_{i,j}(G_{\theta_G}(I^{LR}))_{x,y})^2$$

Figura 6. Expresión de la función de pérdida basada en las capas de activación de una red VGG19.

encontramos mejores resultados si sumamos las 3, usando un factor de escala (dado por los autores) para la pérdida VGG y la antagónica.

$$l^{SR}_{Gen} = \sum_{n=1}^{N} -\log D_{\theta_D}(G_{\theta_G}(I^{LR}))$$

Figura 7. Expresión de la pérdida antagónica *Adversarial Loss* esenciales a una GAN, con el fin de maximizar las probabilidades que la imagen generada sea una imagen real, para la salida del Discriminador.

A modo esquemático se muestran dos figuras (8 y 9) de los valores de la función de pérdida a medida que aumentan las épocas. Si bien el entrenamiento que se muestra en las Figuras se realizó con el dataset UxLES, no hay diferencias cualitativas y apenas cuantitativas entre la evolución de la *loss* al entrenar con el dataset DIV2K.

Se tomaron como inspiración y referencia algunos repositorios públicos de GitHub con implementaciones de SRGAN en PyTorch. [8] [9] [10]

La implementación de este trabajo se puede encontrar en este repositorio: https://github.com/maxibove13/SRGAN

IV. Evaluación Experimental

IV-A. *Procedimiento de entrenamiento y validación*

Para evaluar correctamente al modelo implementado y descrito en la sección anterior y hacer un uso eficiente de los datos disponibles, utilizamos validación cruzada de tipo *k-fold*.

[8] https://github.com/leftthomas/SRGAN
[9] https://github.com/aitorzip/PyTorch-SRGAN
[10] https://github.com/aladdinpersson/Machine-Learning-Collection

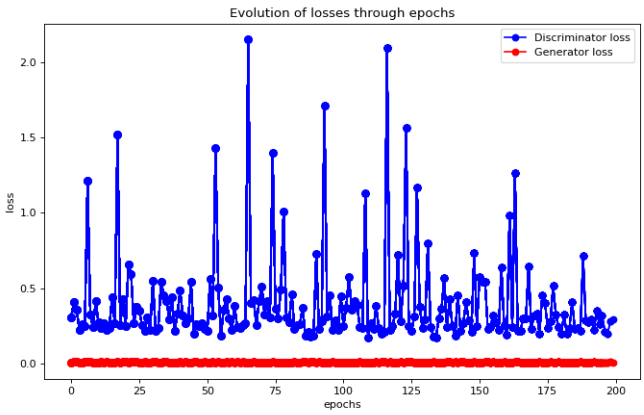

Figura 8. Pérdidas del Discriminador y el Generador a medida que aumentan las épocas.

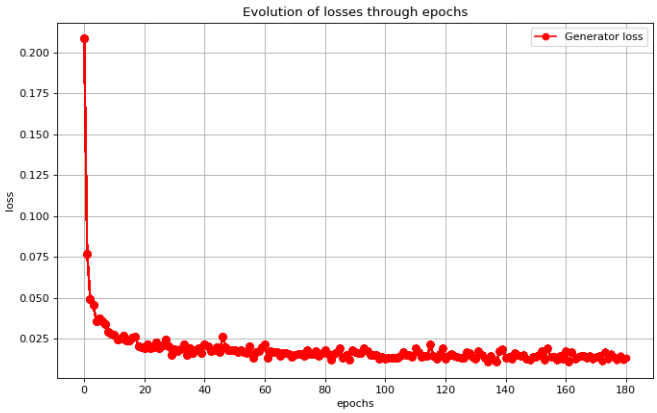

Figura 9. Pérdida del Generador a medida que se aumentan las épocas. Recordemos que esta es la suma de la pérdida VGG, la MSE y la adversaria.

IV-A1. validación cruzada k-fold: Por defecto, dividiríamos los datos en dos partes, *training* y *testing*; luego, comprobaríamos, bajo alguna métrica, si el modelo entrenado con los datos de *training* performa bien en los datos de *testing*. El problema con este enfoque es que existe un riesgo de $overfitting$ en los datos de *testing* ya que estos son usados para modificar los hiperparámetros. De este modo, conocimiento del conjunto de datos de *testing* puede filtrarse al modelo y las métricas de evaluación ya no reportarían correctamente sobre la generalidad del modelo. Para solucionar este problema, se suele dividir el conjunto de datos en tres partes *training*, *testing* y *validation*. El entrenamiento se realiza en el conjunto de *training*, la evaluación y tuneo de hiperparámetros en el conjunto de *validation* y finalmente se mide la performance del modelo en el conjunto de *testing*. Sin embargo, al partir los datos disponibles en tres conjuntos, se reduce considerablemente el número de muestras que pueden ser usadas para entrenar el modelo.

Es por ello que optamos por usar validacación cruzada con la técnica *k-fold*. De esta forma, no necesitamos un conjunto de *validation* ya que un subconjunto del conjunto de *training* será usado para evaluar el modelo en cada instancia.

IV-A2. Procedimiento: Se procede entonces a dividir el *dataset* en $89\,\%$ para *training* y $11\,\%$ para *testing* final. El conjunto de *training* se dividide en $k = 5$ subconjuntos[11]. Luego se realiza el entrenamiento de la red k veces, apartando del entrenamiento un subconjunto distinto de los k existentes cada vez. De esta forma se tienen k instancias de entrenamiento con una métrica asociada por cada uno, la cual se promedia y se toma en cuenta para ir realizando los ajustes de hiperparámetros del modelo: *learning rate*, *batch size*, número de épocas. Finalmente, cuando se logran unas métricas aceptables, o no se percibe mayor capacidad de mejora, se procede a entrenar la versión final del modelo con toda la información de entrenamiento (sin dividir en *k-folds*). Se evalúa la performance del modelo con los datos de *testing*.

IV-B. Métricas

Siguiendo a los autores se utilizan las métricas *peak signal-to-noise ratio* (PSNR) y *structural similarity index measure* (SSIM).

PSNR es una expresión para el ratio entre la máxima energía posible de una señal y el ruido que afecta a su representación fidedigna. Se utiliza habitualmente como medida cuantitativa de la calidad de la reconstrucción en el ámbito de compresión de imágenes. Se mide en decibeles (dB) y tiende a infinito cuando las imágenes son idénticas

En cambio, SSIM [5] es un método que busca identificar la información estructural de una escena y así identificar las diferencias entre la información extraída a partir de una referencia y de una muestra a evaluar. Esta métrica extrae tres características claves de una imagen: luminisencia, contraste y estructura. Esta métrica es adimensionada y tiende a 1 cuando las imágenes son idénticas.

IV-C. Resultados

IV-C1. DIV2K dataset: Durante el proceso de validación del modelo se probó con diferentes números de épocas, para comprobar, como es esperable que al aumentar las mismas se mejora la performance de la red hasta cierto punto pasado el cual ya no trae ventajas. En la Figura 6 se muestran resultados de SSIM y PSNR para diferentes épocas.

Se experimentó con diferentes *learning rates* y *batch sizes* aunque se obtuvieron peores resultados que con los valores sugeridos por los autores del artículo ($lr = 1e-4, batch-size = 16$) por lo que se mantuvieron los valores en la versión final del modelo.

Se calculó PSNR y SSIM para el conjunto de datos de *testing* obteniendo una media de PSNR de 21.07dB y SSIM de 0.62, si bien bastante por debajo de lo obtenido por los autores (21.15dB/0.69) son resultados aceptables y más aún teniendo en cuenta que se entrenó la red solamente con 800 imágenes, mientras que Ledig et al. utilizan 350 mil imágenes.

[11]La elección de $k = 5$ se hizo por simplicidad y recomendaciones heurísticas en algunos sitios.

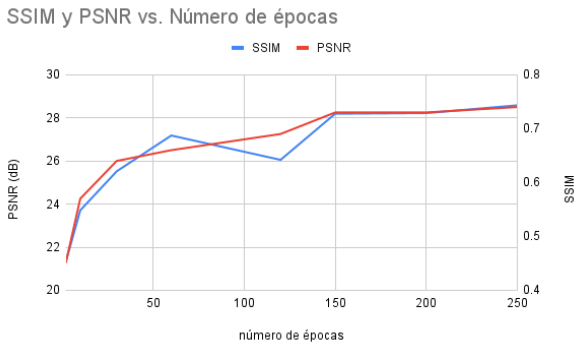

Figura 10. Resultados

En la Figura 11 se muestra un ejemplo de una imagen del dataset de *testing* en sus diferentes versiones de baja, super y alta resolución. Con el fin de ver la distribución de las métricas calculadas, se muestra en la Figura 12, histogramas de PSNR y SSIM para este conjunto de imágenes. Ver al final del documento la Figura 15 para mayores comparaciones visuales.

Figura 11. Ejemplo de imagen del dataset de *testing* en sus versiones Low Resolution, Super Resolution y High Resolution

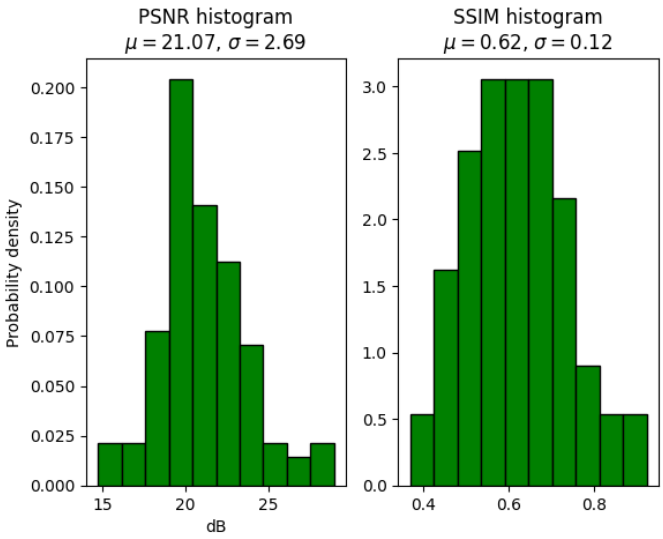

Figura 12. Histogramas de las métricas PSNR y SSIM aplicadas al conjunto de imágenes de *testing* comparando la imagen generada de super resolución con la original de alta resolución. DIV2K dataset

IV-C2. UxLES dataset: A continuación, presentamos los resultados de evaluar el modelo entrenado con el conjunto de imágenes UxLES que correspondían a planos de la primera componente del campo de velocidades. Se obtuvieron mejores resultados que con el anterior dataset (ver Figura 13 para una comparación visual de los resultados), probablemente debido a que las imágenes son más parecidos en su estructura, detalle y color, a que el dataset es más extenso (en este caso entrenamos con 1482 imágenes), y a que las imágenes en sí no cuentan con tantos detalles como en una fotografía genérica. PSNR aplicado al conjunto de testing de 177 imágenes dio 21.86 dB en promedio, y la media del SSIM resulta en un 0.84, ver Figura 14. Ver al final del documento la Figura 16 para mayores comparaciones visuales.

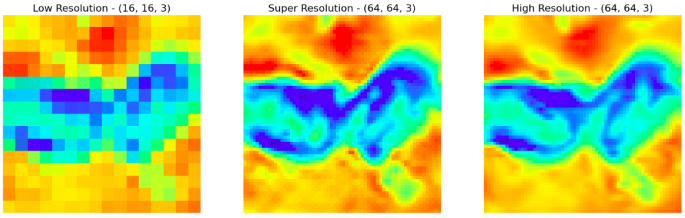

Figura 13. Ejemplo de imagen del dataset de *testing* en sus versiones Low Resolution, Super Resolution y High Resolution

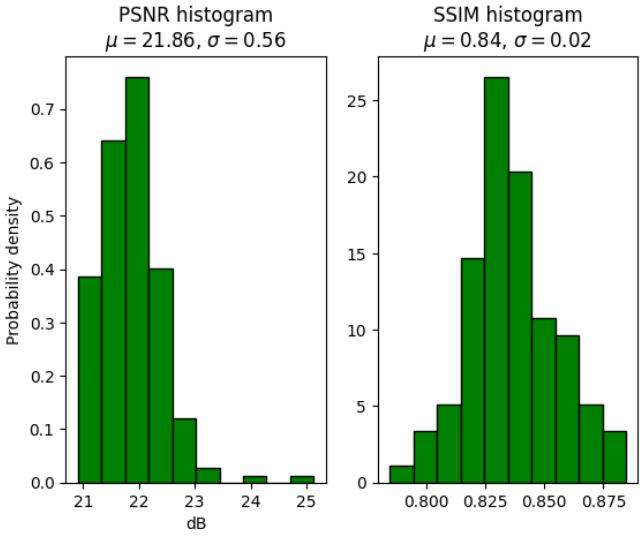

Figura 14. Histogramas de las métricas PSNR y SSIM aplicadas al conjunto de imágenes de *testing* comparando la imagen generada de super resolución con la original de alta resolución. UxLES dataset

REFERENCIAS

[1] M. Werhahn, Y. Xie, M. Chu, N. Thuerey, “A Multi-Pass GAN for Fluid Flow Super-Resolution”, 2019.

[2] Y. Xie, E. Franz, M. Chu, N. Thuerey, “tempoGAN: A Temporally Coherent, Volumetric GAN for Super-resolution Fluid Flow”, 2018.

[3] A. Subramaniam, M. L. Wong, R. D. Borker, S. Nimmagadda, S. K. Lele, “Turbulence Enrichment using Physics-informed Generative Adversarial Networks ”, 2020.

[4] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A, Tejani, J. Totz, Z. Wang, W. Shi, “Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network”, 2016.

[5] Wang Z, Bovik AC, Sheikh HR, Simoncelli EP. Image quality assessment: from error visibility to structural similarity. IEEE Trans Image Process. 2004 Apr;13(4):600-12. doi: 10.1109/tip.2003.819861. PMID: 15376593.

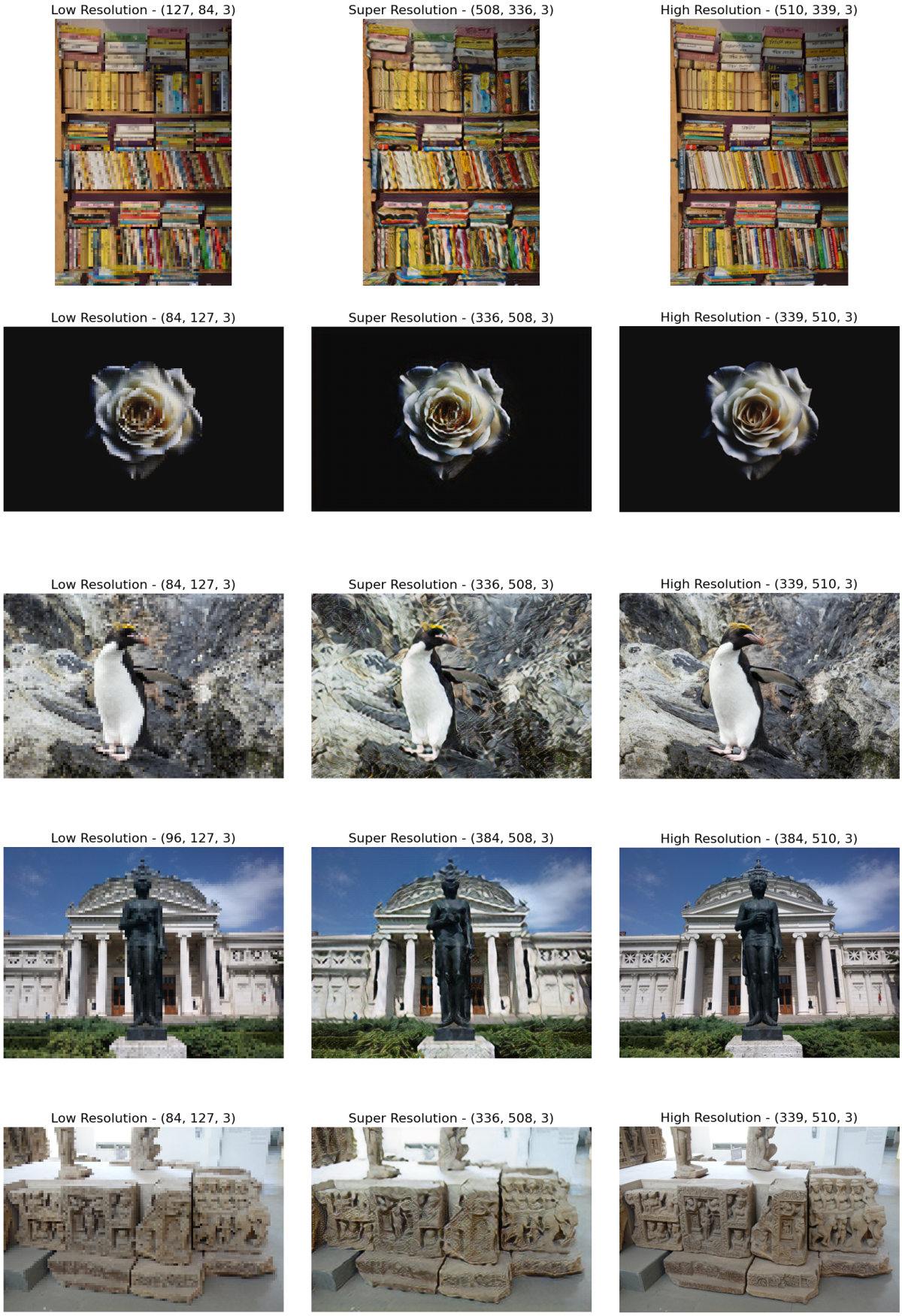

Figura 15. Resultados visuales de evaluar al Generador entrenado con el dataset DIV2K

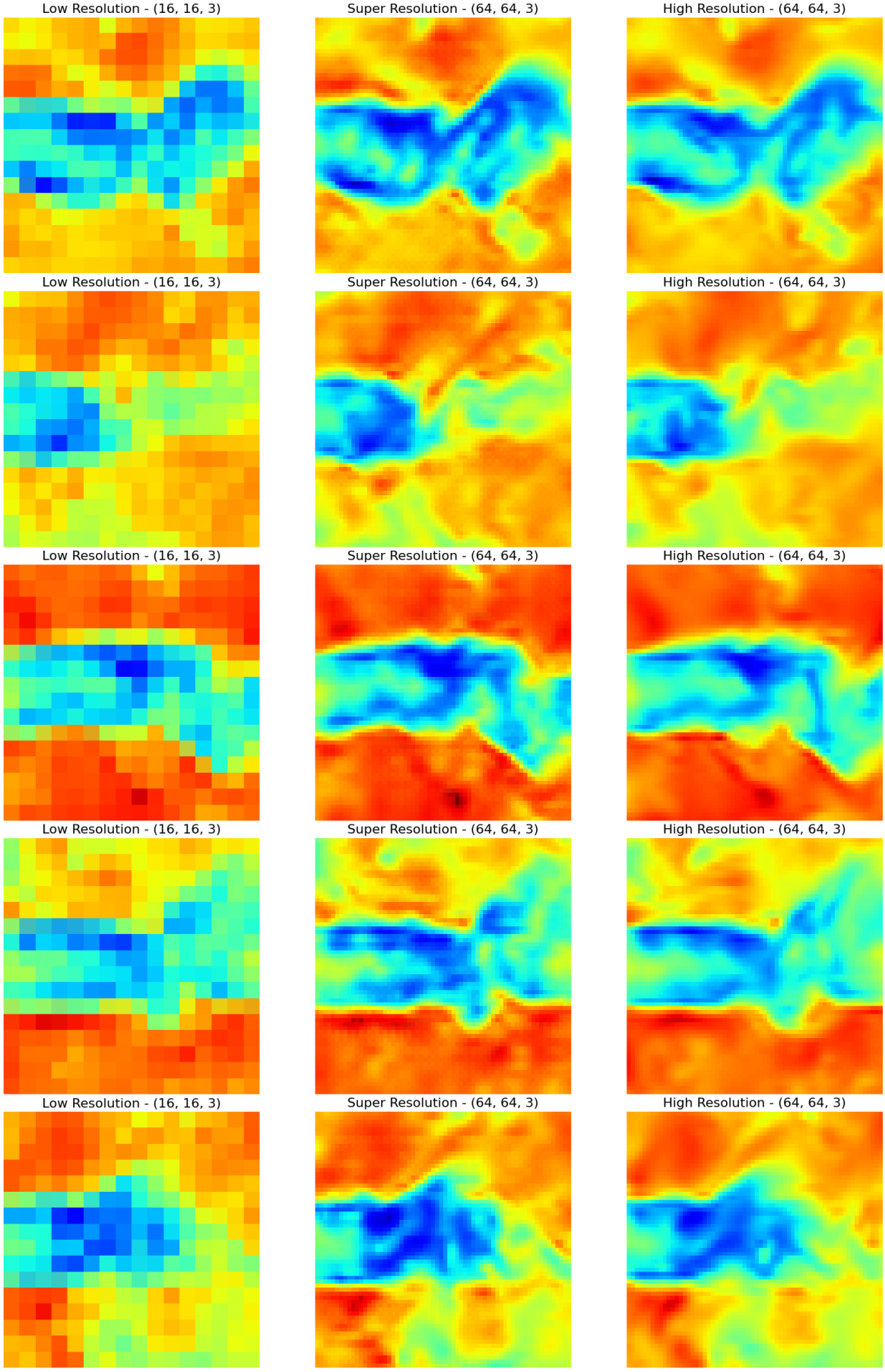

Figura 16. Resultados visuales de evaluar al Generador entrenado con el dataset UxLES