Resumen

El campo de la super-resolución de imágenes cuenta con un amplio abanico de aplicaciones. Al mismo tiempo existen limitaciones en el hardware y software desarrollado hasta la fecha. En este trabajo se presenta la teoría tanto de los métodos clásicos aplicados a la super-resolución como de las nuevas técnicas que han aparecido con el auge de los modelos de *deep learning*.

En el proyecto se introduce un nuevo conjunto de datos propuesto por la Agencia Espacial Europea (ESA) que presenta imágenes hiper-espectrales NIR y RED. A lo largo la memoria se introducen los conceptos necesarios para la comprensión del problema así como para el análisis de este tipo de imágenes. Para el problema planteado por la ESA se desarrolla uno de los métodos que ha marcado el estado del arte para imágenes RGB así como modificaciones relativas al método conocido como *Generative Adversarial Networks* (GANs).

Los resultados de la aplicación de GANs a super-resolución de imágenes hiper-espectrales, que toma el satélite PROBA-V, se comparan con un método clásico pero muy frecuentemente utilizado, la interpolación bicúbica.

Por último, también se presenta la metodología seguida para realizar el proyecto. Se explica la tecnología utilizada para que sea un punto de partida a futuros desarrollos gracias al uso de *Github* como plataforma pública y *Docker* como tecnología para agilizar el desarrollo a todos los usuarios.

Agradecimientos

Quiero agradecer a mis tutores, el profesor Jordi Muñoz y el profesor Valero Laparra por el interés mostrado, su disponibilidad y el cuidado en los detalles que han puesto durante la corrección del Trabajo de Fin de Máster.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. ¿Qué es la super–resolución?

Los algoritmos de super–resolución (SR) incrementan de forma artificial la resolución de las imágenes digitales. En la mayoría de las aplicaciones facilitan el procesado y el análisis. Dos de las principales áreas de aplicación son la mejora de calidad para la interpretación humana y la ampliación de información para el procesado automático utilizando algoritmos. La resolución de una imagen describe los detalles contenidos en esta por lo que a mayor resolución más detalles puede contener la imagen. La resolución de las imágenes digitales puede definirse de diversas maneras: resolución por pixel, resolución espacial, resolución espectral, resolución temporal y resolución radiométrica.

La idea básica para la reconstrucción de las imágenes de alta resolución a partir de imágenes de baja resolución surge del hecho de que estas últimas guardan información complementaria en sus pixeles. Esto hace posible la reconstrucción de alta resolución. Algunas áreas de investigación son las siguientes:

- Vídeo de vigilancia: Captura de fotogramas y regiones de interés (zoom) en los vídeos para la percepción humana (por ejemplo, mirar una matrícula en un vídeo), mejora de la resolución para el reconocimiento automático de objetivos (por ejemplo, tratar de reconocer la cara de un sospechoso).

- Teledetección: se proporcionan varias imágenes de la misma área para construir una imagen de resolución mejorada.

- Imágenes médicas (CT, MRI, ultrasonido, etc.): se pueden adquirir varias imágenes con calidad de resolución limitada, y la técnica SR se aplica para mejorar la resolución.

- Conversión estándar de vídeo, por ej. de la señal de vídeo NTSC a la señal HDTV.

1.2. Objetivo del proyecto

Durante el desarrollo del Trabajo de Fin de Máster plantearemos diferentes comparativas de metodologías y modelos sobre diferentes ejemplos. Nuestro principal objetivo estará centrado en el estudio y desarrollo de un modelo de SR basado en redes neuronales para imágenes satelitales. Más en concreto trabajaremos con el dataset que se

presenta en la competición de *PROBA-V Super Resolution*. PROBA-V 1.1 es un satélite de observación de la Tierra diseñado para cartografiar la cobertura terrestre y el crecimiento de la vegetación en todo el mundo.

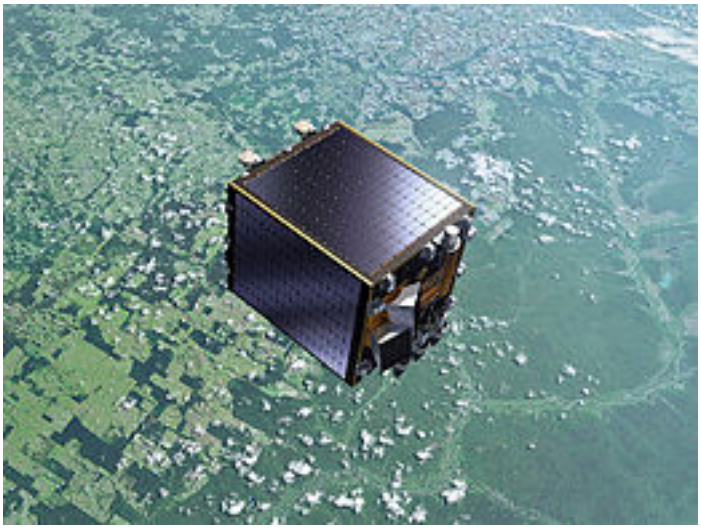

Figura 1.1: Satélite PROBA-V

Se lanzó el 6 de mayo de 2013 en una órbita polar sincrónica al Sol a una altitud de 820 km. Sus sensores de carga útil permiten una cobertura casi global (90 %) por día, proporcionando imágenes de resolución de 300 m. PROBA-V también proporciona imágenes de 'alta resolución' de 100 m, pero a una frecuencia temporal más baja, aproximadamente cada 5 días (según la ubicación).

El objetivo de este desafío es construir imágenes de alta resolución mediante la fusión de las imágenes más frecuentes de 300 m. Este proceso, conocido como super–resolución de imágenes múltiples, ya se aplicó a satélites anteriormente: SPOT-VGT o ZY-3 TLC. Estos tienen cargas útiles que pueden tomar múltiples imágenes de la misma área durante una sola pasada del satélite y crea una condición más favorable para el uso de algoritmos de súper resolución ya que las imágenes se toman simultáneamente. Por lo tanto, la SR ya se puede incluir en el producto final de estos satélites. Sin embargo, este no es el caso de PROBA-V u otros satélites, que podrían beneficiarse de una mejora posterior a la adquisición.

Por lo tanto, los productos PROBA-V representan una manera conveniente de explorar la SR en un entorno relevante. Las imágenes proporcionadas para este desafío no se degradan artificialmente, sino que son imágenes reales, en diferentes resoluciones y con un desplazamiento espacial y temporal. Cualquier mejora en este conjunto de datos puede ser transferible a colecciones más grandes sin necesidad de desplegar sensores o satélites más costosos en lo que se refiere a los sensores y tecnología, ya que la mejora de la resolución puede ocurrir después de la adquisición. El conjunto de datos cuenta con varias imágenes de 78 ubicaciones de la Tierra. Desarrollaremos un algoritmo para fusionar las imágenes de baja resolución en una sola. El resultado será una imagen de "super resolución"que se podría comparar con su imagen de alta resolución tomada desde el satélite, PROBA-V.

La 'V' significa Vegetación, que es el foco principal de los instrumentos a bordo del satélite. Con el modelo planteado buscamos mejorar la visión de PROBA-V y aumentar la precisión en el monitorizado del crecimiento de la vegetación terrestre. Los detalles

de la competición y los datos se pueden encontrar en [2].

1.3. Estructura del proyecto

Este trabajo contiene dos grandes partes las cuales a su vez se dividen según la antigüedad de las técnicas utilizadas. En el primer bloque se presentarán los resultados y conceptos teóricos sobre los que se desarrollan las técnicas de SR estudiadas para el trabajo. Este bloque se separa en dos partes, que comprenden los capítulos 2, 3 y 4. Estos capítulos tratan sobre los métodos clásicos, los retos del campo de la SR y las técnicas de SR utilizando redes neuronales a nivel teórico, es decir, por qué se utilizan, cómo funcionan y por qué son técnicas aplicables para la resolución del problema.

El segundo bloque del trabajo es práctico. Trata sobre la aplicación de algunas de las técnicas comentadas en el primer bloque, los aspectos técnicos del problema y su relación con el conjunto de datos que se utilizan. En este bloque nos centraremos en el estudio de los datos y sus particularidades: Explicaremos qué son las imágenes RED (en la banda de frecuencias correspondiente al rojo) y NIR (infrarrojo cercano, *near infrared*), el comportamiento de los modelos y cómo adaptarlos a los datos y los resultados obtenidos por ellos. También consideraremos las posibles mejoras futuras y lineas de trabajo.

Capítulo 2

Métodos clásicos

La reconstrucción de imágenes SR ha sido una de las áreas de investigación más activas desde el trabajo seminal de Tsai y Huang [3] en 1984. En las últimas décadas se han propuesto numerosas técnicas que representan enfoques desde el dominio de la frecuencia al dominio espacial, y desde la perspectiva del procesamiento de señales a la del aprendizaje automático. Los primeros trabajos sobre SR siguieron principalmente la teoría de [3] mediante la exploración de las propiedades de cambio y *aliasing* de la transformada de Fourier. Sin embargo, estos enfoques de dominio de frecuencia están muy restringidos al modelo de observación de la imagen que pueden manejar, y los problemas reales son mucho más complicados. En la actualidad, los investigadores suelen abordar el problema principalmente en el dominio espacial, por su flexibilidad para modelar todo tipo de imágenes degradadas. Esta sección trata sobre estas técnicas, a partir de las del modelo de observación de la imagen.

El sistema de imágenes digitales no es perfecto debido a las limitaciones de hardware, ya que adquiere imágenes con varios tipos de degradaciones. Por ejemplo, el tamaño de la apertura finita causa desenfoque óptico, modelado por la función de dispersión de puntos (PSF). El tiempo finito de apertura resulta en un desenfoque por movimiento, que es muy común en los vídeos. El tamaño finito del sensor lleva a un emborronado: El píxel de la imagen esta generado por la integración sobre el área del sensor en lugar del muestreo por impulso. La densidad limitada del sensor conduce a efectos de *aliasing*, limitando la resolución espacial de la imagen lograda. Estas degradaciones se modelan total o parcialmente en diferentes técnicas de SR.

La entrada del sistema de imágenes son escenas naturales continuas bien aproximadas como una banda-limitada de señales. Estas señales pueden estar contaminadas por las turbulencias atmosféricas antes de llegar al sistema de imágenes. Muestreando imágenes más allá de la frecuencia de Nyquist generaría la imagen de SR que desearíamos, sin embargo en nuestro sistema existen problemas como el movimiento de la cámara dando lugar a múltiples *inputs* correlados pero que pueden estar desplazados entre ellos lo que puede incurrir en un emborronado óptico o en un emborronado de movimiento. Además, estas imágenes emborronadas son re-escaladas mediante submuestreo (*down-sampling*) en los sensores en píxeles, por la integral de la imagen que cae en cada área del sensor. Estas imágenes muestreadas se ven afectadas por el ruido del sensor y el ruido de filtrado de color. Las imágenes del sistema subyacente son, borrosas, diezmadas y ruidosas representaciones de la escena original y real.

Denotemos X o $\mathcal{X}$ como la imagen HR deseada, es decir, la imagen digital muestreada por encima de la frecuencia de muestreo de Nyquist de la escena continua limitada

en banda. Y_k o $\mathcal{Y}_k$ es la observación $k-esima$ de baja resolución (*low-resolution*) (LR) desde la cámara. Supongamos que la cámara captura K fotogramas LR de X. Las observaciones LR están relacionadas con la escena X de *high-resolution* (HR) por:

$$Y_k = \mathcal{D}_k \mathcal{H}_k \mathcal{F}_k X + \mathcal{V}_k$$

donde $\mathcal{F}_k$ codifica la información de movimiento de la imagen k, $\mathcal{H}_k$ los efectos de emborronado, $\mathcal{D}_k$ es el operador de *down-sampling* y $\mathcal{V}_k$ es un término de ruido del modelado. Este sistema de ecuaciones linear puede ser reescrito como:

$$\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_k \end{bmatrix} = \begin{bmatrix} D_1 H_1 F_1 \\ D_2 H_2 F_2 \\ \vdots \\ D_k H_k F_k \end{bmatrix} X + V$$

o equivalentemente

$$Y' = MX + V' \tag{2.1}$$

donde Y' y V' son la concatenación de Y_k y V_k.

Tenemos que las matrices $\mathcal{F}_k$, $\mathcal{H}_k$, $\mathcal{D}_k$ son *sparse* y por lo tanto M también. El sistema de ecuaciones es por lo tanto mal definido (*ill-posed*). Es más, como es un sistema real estas matrices son desconocidas y se aproximan mediante las imágenes LR que tenemos. Por lo tanto, la regularización previa adecuada para la imagen de alta resolución es siempre deseable y, a menudo, incluso crucial. A continuación, se presentan algunas técnicas básicas de SR propuestas en la literatura.

2.1. SR en el dominio de frecuencias

El trabajo pionero en SR de Tsai y Huang [3] l relaciona imágenes de alta resolución con múltiples imágenes desplazadas de baja resolución mediante una formulación del dominio de la frecuencia basado en las propiedades de *shift* y *aliasing* de la transformada discreta y continua de Fourier.

Sea $x(t_1, t_2)$ una escena de alta resolución. La translación global da K imágenes desplazadas $x_k(t_1, t_2) = x(t_1 + \Delta_{k_1}, t_2 + \Delta_{k_2})$ con $k = 1, 2, \cdots, K$ y donde Δ_{k_1} y Δ_{k_2} son movimientos arbitrarios pero conocidos. La transformada continua de Fourier (CFT) de la escena está dada por $\mathcal{X}(u_1, u_2)$ y las imágenes desplazadas por $\mathcal{X}_k(u_1, u_2)$. Entonces por las propiedades de desplazamiento de la CFT estas imágenes desplazadas pueden ser escritas como:

$$\mathcal{X}_k(u_1, u_2) = \exp[j2\pi(\Delta_{k_1} u_1 + \Delta_{k_2} u_2)\mathcal{X}(u_1, u_2)] \tag{2.2}$$

Las imágenes desplazadas están muestreadas con períodos T1 y T2 dados por las observaciones de baja resolución $y_k[n_1, n_2] = x_k(n_1 T_1 + \Delta_{k_1}, n_2 T_2 + \Delta_{k_2})$ con $n_1 = 0, 1, 2, \ldots, N_1 - 1$ y $n_2 = 0, 1, 2, \ldots, N_2 - 1$. Denotamos la transformada discreta de Fourier (DFTs) de estas imágenes de baja resolución como $\mathcal{Y}_k[r_1, r_2]$. Las CFTs de las imágenes desplazadas están relacionadas con sus DFTs por la propiedad de *aliasing*

$$\mathcal{Y}_k[r_1, r_2] = \frac{1}{T_1 T_2} \sum_{m_1=-\infty}^{\infty} \sum_{m_2=-\infty}^{\infty} \mathcal{X}_k \left(\frac{2\pi}{T_1}(\frac{r_1}{N_1} - m_1), \frac{2\pi}{T_2}(\frac{r_2}{N_2} - m_2) \right) \tag{2.3}$$

Asumiendo que $\mathcal{X}(u_1, u_2)$ está limitado en banda, $|\mathcal{X}(u_1, u_2)| = 0$ para $|u_1| > (N_1\pi)/T_1$, $|u_2| > (N_2\pi)/T_2$, combinando las ecuaciones (2.2), (2.3) relacionamos los coeficientes de la DFT de $y_k[n_1, n_2]$ con las muestras de la CFT desconocida de $x(t_1, t_2)$ en la matriz de la forma:

$$\mathcal{Y}' = \Phi\mathcal{X}' \tag{2.4}$$

donde $\mathcal{Y}'$ es un vector columna de $K \times 1$ dimensiones con el k^{th} elemento siendo el coeficiente de la DFT de $\mathcal{Y}_k[r_1, r_2]$. $\mathcal{X}'$ es un vector columna $N_1N_2 \times 1$ conteniendo las muestras de los coeficientes desconocidos de la CFT de $x(t_1, t_2)$, y donde Φ es una matriz $K \times N_1N_2$ relacionando $\mathcal{Y}'$, $\mathcal{X}'$. La ecuación (2.4) define un conjunto lineal de ecuaciones a partir de las cuales podemos resolver y obtener $\mathcal{X}'$, y a partir de esta podemos obtener las imágenes reconstruidas utilizando la DFT inversa.

Toda la formulación presentada para la reconstrucción SR asume un entorno libre de ruido y con la traslación global de modelos con parámetros conocidos. El proceso de *downsampling* se asume que es por muestreo por impulso sin un sensor de efectos de *blurreado* modelado. Siguiendo esta línea de trabajo se han desarrollado algoritmos de diversas complejidades. Kim et al.[4] introducen la regularización de Tikohonov, un modelo local de estimación de movimientos dividiendo las imágenes por bloques solapados y estimando el movimiento para cada bloque local individualmente. También se han desarrollado algoritmos donde para la recuperación y estimación del movimiento hace uso de algoritmos *expectation-maximization* EM simultáneos. Estas aproximaciones son computacionalmente eficientes pero limitadas en sus habilidades para trabajar con modelos más complejos.

2.2. Interpolación y aproximaciones no iterativas

Para superar las dificultades y las debilidades de los métodos de SR en el dominio de la frecuencia se han presentado a lo largo de los años muchas aproximaciones en el dominio espacial [5, 6]. Como las imágenes HR y LR están relacionadas por el sistema lineal no denso (2.1) se pueden aplicar una diversidad de estimadores para el problema de SR. Por ejemplo, *maximum likelihood*, *Maximum a Posteriori* y la *proyección en conjuntos convexos*. Vamos a presentar el modelo de SR más sencillo en el dominio del espacio en analogía a la aproximación en el dominio de la frecuencia.

Asumimos que H_k es espacialmente invariante y lineal (LSI) y es igual para las K imágenes restantes, denotadas H. Supongamos F_k considerando únicamente modelos simples de movimiento como traslación y rotación. Entonces H y F_k conmutan y obtenemos:

$$Y_k = D_kF_kHX + V_k = D_kF_kZ \text{ para } k = 1, 2, \ldots, K \tag{2.5}$$

lo que motiva el planteamiento de un método no iterativo basado en la interpolación y restauración. Nos encontramos pues con tres estadios:

- Muestreo de las imágenes de baja resolución, LR.
- Interpolación no uniforme para obtener Z.
- *Emborronado y eliminación del ruido.*

Gráficamente podemos entender el proceso de como muestra la Figura 2.1 obtenida de [7].

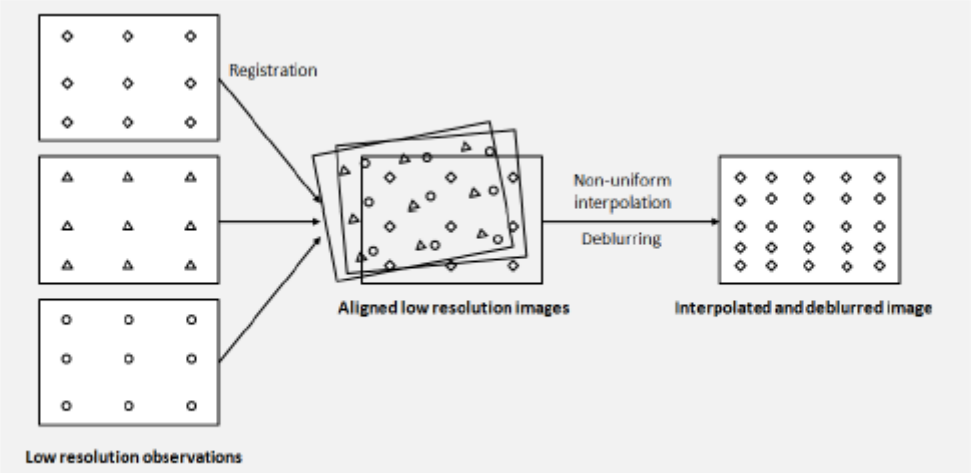

Figura 2.1: Aproximación a la super-resolución mediante la interpolación

Las imágenes de baja resolución LR son alineadas por un algoritmo de muestreo de imágenes hasta la precisión de subpixel. Estas imágenes son colocadas entonces en un *grid* donde mediante métodos no uniformes de interpolación se rellenan los píxeles faltantes en las imágenes de HR obteniendo Z. Finalmente, Z es emborronada y se utiliza cualquier algoritmo clásico de deconvolución para la eliminación del ruido y obtener X.

Estos métodos son intuitivos, simples y computacionalmente eficientes [8] asumiendo modelos de observación simples. Sin embargo, la aproximación no garantiza un óptimo en la estimación. Cualquier pequeño error de muestreo puede fácilmente propagarse al procesado posterior. Además, sin el *prior* de las imágenes HR para la adecuada regularización, estos métodos necesitan un tratamiento especial de las observaciones para reducir el *aliasing*.

2.3. Aproximación estadística a la super-resolución

Las aproximaciones estadísticas se acercan a la SR mediante un enfoque estocástico hasta la reconstrucción óptima. Las imágenes HR y los movimientos a través de los *inputs* de baja resolución pueden ser entendidos como variables aleatorias. Sea $\mathcal{M}(\sigma, h)$ la matriz de degradación definida por el vector de movimiento σ y el kernel de emborronado h, el problema de SR puede ser representado en un marco completamente Bayesiano

$$\begin{aligned} X &= \arg\max_X P(X|Y') \\ &= \arg\max_X \int_{\sigma,h} P(X, \mathcal{M}(\sigma,h)|Y')d\sigma \\ &= \arg\max_X \int_{\sigma,h} \frac{P(Y'|X, \mathcal{M}(\sigma,h))P(X, \mathcal{M}(\sigma,h))}{P(Y')}d\sigma \\ &= \arg\max_X \int_{\sigma,h} P(Y'|X, \mathcal{M}(\sigma,h))P(X)P(\mathcal{M}(\sigma,h))d\sigma \end{aligned} \tag{2.6}$$

Aquí X y $\mathcal{M}(\sigma, h)$ son estadísticamente independientes. Entonces $P(Y'|X, M(\sigma, h))$ es una vecindad de datos, $P(X)$ es el prior de las imágenes de alta resolución HR y $P(\mathcal{M}(\sigma, h))$ es el prior de la estimación del movimiento. V' en la ecuación (2.1) habitualmente se mantiene para un ruido aditivo asumiendo que este tiene media cero y es un vector *random* siguiendo una distribución Gaussiana. Entonces,

$$P(Y'|X, \mathcal{M}(\sigma, h)) \propto \exp\left\{-\frac{1}{2\sigma^2}\|Y' - \mathcal{M}(\sigma, h)X\|^2\right\} \tag{2.7}$$

P(X) típicamente está definido con una distribución de Gibbs en su forma exponencial.

$$P(X) = \frac{1}{Z}\exp\{-\alpha A(X)\} \tag{2.8}$$

donde $A(X)$ es una función potencial no negativa y Z es un factor de normalización. La formulación de la ecuación (2.6) puede ser complicada y difícil de evaluar debido a la integración sobre las estimaciones del movimiento. Si $\mathcal{M}(\sigma, h)$ está dado o estimado de antemano, la ecuación (2.6) puede ser simplificada como

$$\begin{aligned} X &= \arg\max_X P(Y'|X, M)P(X) \\ &= \arg\min_X \|Y' - MX\|^2 + \lambda A(X) \end{aligned} \tag{2.9}$$

donde λ engloba la varianza del ruido y de α en la ecuación (2.9), balanceando la consistencia de los datos y el impacto del prior de la imagen HR. Así pues, la ecuación (2.9) es conocida como la formulación *Maximum a Posteriori* (MAP) para SR donde M se asume conocido. Las aproximaciones estadísticas varían sobretodo en el tratamiento de la matriz de degradación $\mathcal{M}(\sigma, h)$, el término $P(X)$ y los métodos de inferencia estadística a través de la (2.6).

Capítulo 3

Desafíos en la SR

El campo de la super-resolución, que cuenta con amplias aplicaciones en la vida real, tiene fuertes limitaciones a nivel de hardware y software a diferentes escalas dependiendo del problema que se busca resolver. Conocer estas limitaciones es un punto importante para el desarrollo de aplicaciones para la super-resolución ya que son puntos clave para ofrecer un producto mejorado. En este capítulo se presentan los desafíos más importantes dentro de la super-resolución.

3.1. Muestreo de la imágenes

El muestreo de las imágenes es un punto crítico para el éxito del problema con SR. La importancia del muestreo es debido a lo comentado previamente sobre el contexto de un problema mal condicionado (*ill-conditioned*). El problema puede llegar a ser mucho más complicado cuando las observaciones son imágenes de baja resolución con fuertes efectos de *aliasing*. Este tipo de errores son visualmente más molestos que el emborronado de las imágenes, efecto del resultado de la interpolación de las imágenes LR. Además, el hecho de la limitación a la hora de obtener registros puede llevar a un problema de *overfitting* de los modelos dando lugar a una cuestión de falta de generalización [9].

3.2. Eficiencia computacional

Otra de las principales dificultades está relacionada con la necesidad de computación intensiva debido al gran número de incógnitas y parámetros a ajustar tanto en los métodos clásicos como en los modernos. Todo ello puede requerir procesos extensos de manipulación de matrices o de optimización de gran número de parámetros. Las aplicaciones reales siempre demandan eficiencia y velocidad para poder tener una utilidad práctica. Dentro de este apartado es interesante ver los avances gracias a las técnicas de paralelización y al uso de GPU y hardware más potentes. Se puede mejorar la eficiencia de los modelos sin tener que recurrir a métodos de aproximación o entrenamientos menos intensivos de los modelos que podrían desembocar en un peor rendimiento del software implementado [9].

3.3. Robustez de los modelos

Los modelos tradicionales son vulnerables a la presencia de *outliers* debido a los errores de movimiento, modelos poco precisos de emborronado, ruido, objetos en movimiento, etc. Estos errores no pueden ser tratados ni tomados como el ruido Gaussiano habitual, por lo que ciertas asunciones sobre errores y sus reconstrucciones sobre los espacios, por ejemplo $\mathcal{L}_2$, no pueden ser tomadas como válidas [9].

3.4. Límites a la eficiencia de los algoritmos

Las tareas de SR son muy difíciles debido a varios efectos y se ha de entender muy bien el problema para tratar de resolver cada uno de estos puntos. Hay que tener en cuenta factores como el modelado de los errores, factores de zoom, número de frames o imágenes con las que contamos, etc. Además, son necesarias medidas particulares para el problema y no es una buena idea trabajar con las medidas habituales como el MSE ya que pueden no ser adecuadas para que los modelos aprendan correctamente. Incluso resultados con MSE mejores y peores pueden no tener diferencias visuales apreciables [9].

Capítulo 4

SR y deep learning

A lo largo de las próximas secciones y debido al carácter de este trabajo de fin de máster se obviarán las explicaciones básicas sobre las redes neuronales o modelos de deep learning. El objetivo es que el lector pueda centrarse en detalles no tan conocidos relacionados con las características del desarrollo de una GAN. Así pues, se entiende que aquel lector que empiece este capítulo tiene conocimiento del funcionamiento de una neurona, que una red neuronal es un aproximador universal o que se entrena mediante un proceso de optimización llamado *backpropagation*. Para revisar dicho conocimiento básico se puede acudir a los capítulos 1 a 7 de [10]. Así pues, me centraré únicamente en los modelos y métodos utilizados en el proyecto.

A pesar de la existencia de diversas estructuras y modelos como puede ser las *SR dense networks* (SRDN), *SR convolutional networks* (SRCN), los módulos de atención para la SR (SRRAM), etc. en este proyecto se ha optado por el uso de las GAN para la super-resolución, y durante este capítulo y el capítulo 5 se explicará el funcionamiento de estas, cómo construirlas y cómo se pueden aplicar a los problemas de SR.

4.1. CNN

Las redes convolucionales se presentan como el estado del arte en muchas aplicaciones de *computer vision*. Será una de las herramientas principales del trabajo por lo que este capítulo presentará sus características y su funcionamiento.

Las redes convolucionales son un tipo de arquitectura formada principalmente por capas de convoluciones donde los filtros se encargan de encontrar las características que contienen más información de los *inputs*.

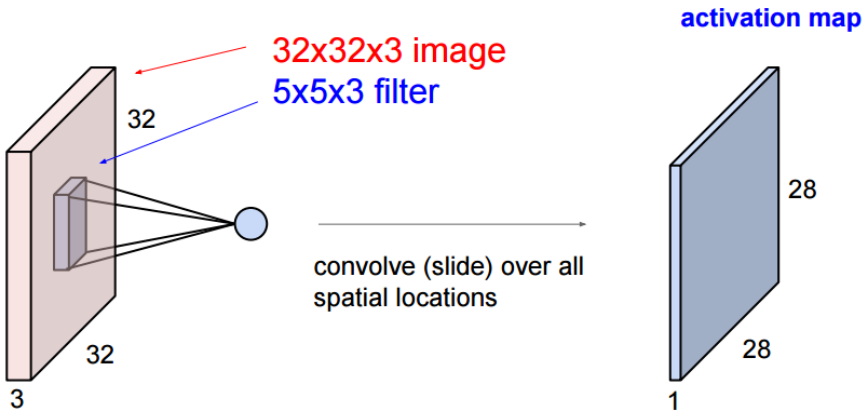

Figura 4.1: Ejemplo de convolución

Es importante entender que esta operación no se restringe a imágenes de 1 o 3 canales, ni siquiera a *inputs* de tipo matricial. La operación convolución puede aplicarse a cualquier tipo de *inputs* y se define como una operación lineal que combina dos señales de la siguiente manera:

$$F(x,y) * G(x,y) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} F(i,j) * G(x-i, y-j) \tag{4.1}$$

Como se ve en la Figura 4.1, en las redes convolucionales el input de las capas es filtrado por una matriz que recibe el nombre de kernel. Este es el que define exactamente lo que la operación convolución va a filtrar y puede producir respuestas muy claras cuando encuentra la característica adecuada.

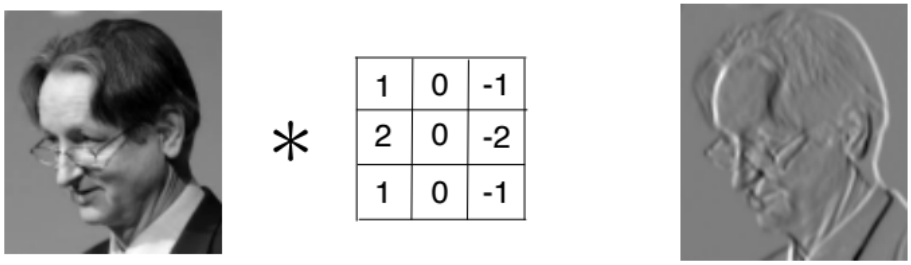

Figura 4.2: Ejemplo de aplicación del filtro de Sobel.

En la Figura 4.2 podemos observar el resultado de aplicar un filtro Sobel a una imagen de Geoffrey Hinton. En este caso el filtro de Sobel actúa como un kernel 3×3. Existen miles de posibles filtros diseñados para detectar patrones sobre las imágenes. El filtro de Sobel es adecuado para buscar los bordes de las imágenes. En el entrenamiento de las redes, inicialmente los pesos de la matriz kernel son aleatorios, aunque se pueden inicializar preentrenados. Así pues, estos pesos se van actualizando desde la inicialización aleatoria hasta construir los filtros óptimos en cada capa para el conjunto de datos que se introduce como *input*.

Como ya hemos dicho la red CNN se compondrá principalemente por capas convolucionales y los principales hiperparámetros para controlar estas capas de convolución son los siguientes:

- Kernel size (K): Indica cómo de grandes van a ser las ventanas con las que se va a realizar la convolución. Las ventanas pequeñas son generalmente mejores.
- Stride (S): Indica cuántos píxeles movemos al kernel para realizar la siguiente convolución. Habitualmente se fija $S = 1$ con el propósito de no variar el tamaño del input de capa a capa pero S puede tomar valores mayores.
- Zero padding (pad): La cantidad de ceros a añadir en los bordes de la imagen. Utilizar *padding* permite a los kernels filtrar cada punto de la imagen incluyendo los bordes de esta.
- Número de filtros (F): La cantidad de filtros que cada una de las capas va a tener. Esto controla el número de características que la capa convolucional va a estar buscando.

4.2. GANs: estructura y funcionamiento

Originalmente en [11] se presentaba la Generative Adversarial Network (GAN) como un método para estimar modelos generativos mediante un proceso *adversario* en el cual se entrenaban simultáneamente dos modelos: Un modelo generativo, G, que captura la distribución de los datos, y un modelo discriminativo, D, que estima la probabilidad de que una muestra provenga del modelo G o de la muestra real de datos. En este contexto diríamos que D se encarga de estimar $P(y|x)$, es decir, la probabilidad de y condicionado a x. Cabe notar que los modelos, a pesar de conocer x a priori, no tienen ninguna información sobre las distribuciones marginales de x e y. Así pues, el modelo D se utiliza como un *mapping* que busca predecir si los *inputs* que reciben siguen la distribución $P(y|x)$ real. Por otro lado, G se encarga de generar los inputs de D tratando de estimar $P(y, x)$, es decir, la probabilidad conjunta. Determinar $P(y|x)$ puede llegar a ser relativamente fácil ya que no hay que realizar suposiciones sobre la distribución marginal de x o y. Gráficamente podemos observarlo en la Figura 4.3 donde observamos un conjunto de datos en el que se encuentran dos tipos de imágenes. Mientras que G ha de tomar un papel de modelo generativo y ha de tratar de reproducir muestras con una distribución similar o igual a la gráfica superior, D se encarga de descifrar si dadas las muestras estas pertenecen o no a la distribución real.

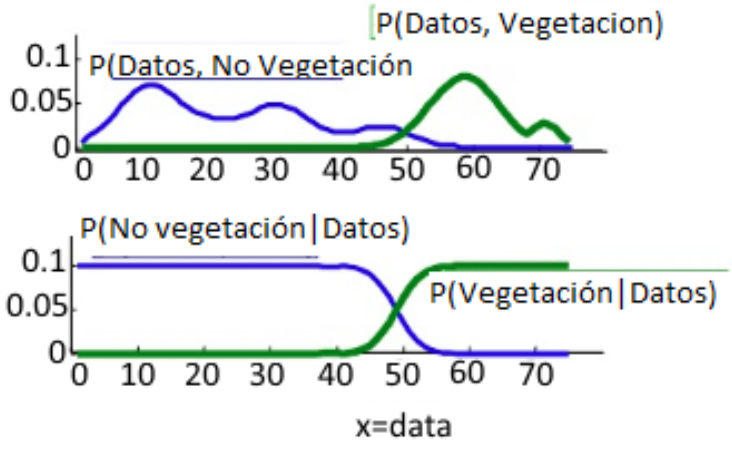

Figura 4.3: Objetivo de las GAN.

Este tipo de modelos tiene una capacidad fascinante para aprender de grandes y complejas distribuciones de datos, incluso con un número relativamente pequeño de datos. Sin embargo, a pesar de que en los últimos años han representado el estado del arte en muchas aplicaciones relacionadas con los modelos generativos, su entrenamiento puede llegar a ser extremadamente complejo.

4.2.1. Generador

El generador es una función del tipo

$$G : z \longrightarrow \hat{x} \tag{4.2}$$

donde z habitualmente esta tratado como un vector de ruido tal que $z \in \mathbb{R}^m$. Sin embargo, esto no tiene que ser necesariamente así y depende de la aplicación que estemos desarrollando. En el caso de la SR, z se compone de todo el conjunto de imágenes de LR. Así pues, queda $G(z) = \hat{x} \in \mathbb{R}^d$, donde G es el generador.

Dependiendo de la aplicación, la construcción de G puede variar. En visión por computación, por su extensión y sus resultados, en muchos de los problemas se utilizan las arquitecturas CNN [12] como se comentaba en la sección 4.1. Las redes muy profundas pueden llegar a ser muy difíciles de entrenar debido a la cantidad de parámetros a optimizar. Sin embargo, en múltiples ocasiones se ha presentado su capacidad para aumentar sustancialmente la precisión de los resultados [13, 14]. Para incrementar la eficiencia de dichas redes se utiliza *batch normalization* que permite acelerar el proceso de entrenamiento. Sin embargo, utilizar *batch normalization* puede provocar inestabilidad en el entrenamiento y un impacto negativo en los modelos entrenados.

Otra de las técnicas para construir G de una manera más eficiente es introducir el concepto de *residual block* [15] y las *skip-connections* [16]. La idea principal se mantiene respecto al resto de arquitecturas habituales. Cada una de las capas de la red es alimentada por la salida de la capa anterior. Sin embargo, cada una de las capas o bloques residuales pueden alimentar a las capas o bloques siguientes a $2 - K$ capas o bloques de distancia. La Figura 4.4 muestra un ejemplo gráfico de cómo funcionaría un bloque residual donde la entrada X no solo alimenta a la capa inicial del bloque si no que también colabora de manera aditiva a la salida del bloque.

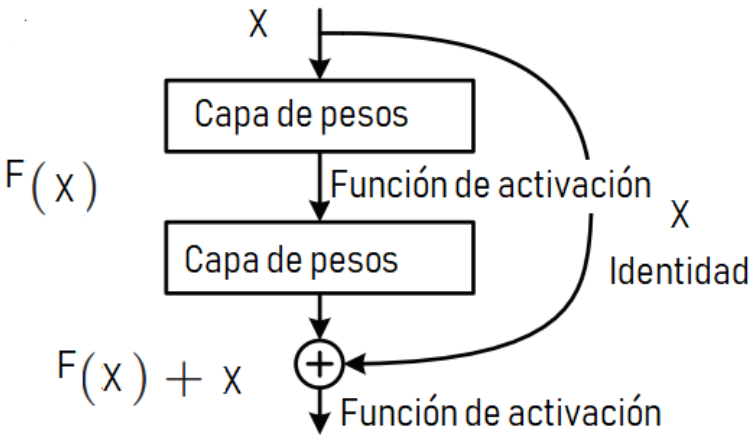

Figura 4.4: Ejemplo de bloque residual con *skip-conection*.

Como decíamos, entrenar redes muy profundas puede ser una tarea complicada y pueden aparecer efectos como el *vanishing gradient*, que provoca que no haya una

actualización de las capas debido a que el gradiente es próximo a cero. Los bloques residuales y las *skip-connections* colaboran a que, en una red profunda, se actualicen los pesos de todas las capas ya que el error no se retropropaga únicamente capa a capa o bloque a bloque, si no que también se retropropaga mediante los saltos de $2 - K$ que hayamos definido. Esto permite a la red entrenar de manera adecuada con redes muy profundas consiguiendo aumentar la precisión y evitando el riesgo de que los pesos de la red no se actualicen correctamente.

Así pues, G es un modelo generativo que trata de predecir $P(y, x)$. Sin embargo, tiene una conexión directa con las arquitecturas más clásicas del *deep learning*, redes CNN, como puede ser la clasificación de imágenes. La principal diferencia viene dada por las etiquetas ya que G tiene como objetivo la construcción de una muestra y no la predicción de una etiqueta en concreto.

4.2.2. Discriminador

Siendo estrictamente formales, el discriminador D es una función

$$D : \hat{X} \longrightarrow Y \tag{4.3}$$

donde $\hat{X}$ representa una muestra de $\mathbb{R}^d$ y Y es un escalar siendo normalmente $Y \in [0, 1]$ ya que nos encontramos en un contexto supervisado con una etiqueta *real or fake*. Dentro del contexto de las GAN, $\hat{X}$ es la estimación de $P(y|x)$ $\forall x \in \hat{X}, y \in Y$. Además Y es el conjunto de valores reales de la muestra, es decir, la probabilidad de que dadas las muestras generadas, estas pertenezcan al conjunto de datos reales.

La arquitectura de D, aunque similar a la de G, está pensada para problemas de aprendizaje supervisado. La red CNN, junto con los avances y mejoras presentados en la subsección 4.2.1, puede continuar utilizándose o bien plantearse una arquitectura diferente. Esto dependerá del problema que se trate. En muchas de las aplicaciones de visión por computación las arquitecturas se mantendrán como redes convolucionales y simplemente se añaden capas densas conectadas a las convolucionales al final para transformar la salida a un problema supervisado. Este problema puede ser de clasificación (habitualmente) o de regresión, que es menos común en este tipo de métodos.

4.2.3. Función de pérdidas (*loss*)

Una vez definidos D y G estos participan en un proceso iterativo de minimización de la función *loss*

$$V(G, D) = \min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_z}[\log(1 - D(G(z)))] \tag{4.4}$$

De esta manera la función *loss* se minimiza cuando $D(x) = 1$ y $D(G(z)) = 0$. Esto básicamente implica que el discriminador es capaz de detectar perfectamente las imágenes reales pero las generadas (*fake*) son tan similares que este es incapaz de diferenciarlas. Esta ecuación es equivalente a minimizar la divergencia de Jensen-Shannon (JS) entre las distribuciones, como se describe en [11]. Dicha distribución es una versión simétrica y suavizada de la divergencia de Kullback-Leibler (KL), lo que también le otorga las características de distancia o métrica.

$$D_{JS}(P||Q) = \frac{1}{2} D_{KL}(P||M) + \frac{1}{2} D_{KL}(Q||M) \tag{4.5}$$

Esta función de perdida, así como ocurría con el hecho de que z no tiene porqué ser aleatorio, tampoco es un hecho constante. Otras arquitecturas de GAN, como la Wasserstein GAN o Least Squares Generative Adversarial Networks, presentan una función de pérdida totalmente distinta. Sin embargo, el proceso iterativo de minimización se mantiene común a los cambios que se plantean. Esto es para dar robustez a los entrenamientos de estos modelos, ya que JS en algunos casos puede no converger y tener gradientes iguales a 0. En el proyecto introduciremos una función de pérdida que usaremos en capítulos posteriores.

4.3. Retos en el entrenamiento

Los retos en el caso del entrenamiento de las GANs pueden englobarse en los siguientes grandes puntos:

- Mode collapse y mode drop.
- Inestabilidad del entrenamiento.
- Sensibilidad por la inicialización de los hiperparámetros.
- Vanishing gradients.

En el capítulo 5 se presentarán las medidas para resolver los puntos más sensibles en nuestro trabajo.

4.3.1. Mode collapse y mode drop

Este es un problema común de las GANs. Se refiere a la reducida variedad de *samples* producidos por el generador. Esto se puede traducir en los siguientes casos:

- El generador sintetiza muestras con *intra-mode variety*, pero algunas muestras se pierden o no son *sampleadas* suficientemente.
- El generador sintetiza muestras con *inter-mode variety*, pero cada una reduce su variedad.

Nótese que en ambos casos se asume que el modelo tiene una alta precisión pero una baja sensibilidad (*recall*). Esto se debe a que los elementos que más se repiten en el conjunto de datos reales tienen mayor probabilidad de ser elegidos y por lo tanto el generador recibe más información sobre estos, que acaban tomando excesiva importancia. Algunas de las soluciones que se presentan consisten en utilizar diferentes tipos de funciones de perdida a la original. Por ejemplo, Wassertein GAN utiliza la distancia de Wasserstein. Esta es una medida de distancias entre dos medidas de probabilidad μ y σ en $P_p(M)$ que refiere a una colección de medidas de probabilidad de μ sobre un conjunto M con un momento finito p^{th}. Se define como

$$W_p(\mu, \sigma) = \left(\inf_{\gamma \in \Gamma(\mu,\sigma)} \int_{M \times M} d(x,y)^p d_\gamma(x,y) \right)^{\frac{1}{p}} \tag{4.6}$$

donde $\Gamma(\mu, \sigma)$ denota un conjunto de medidas sobre M con marginales μ y σ en el primer y segundo factor respectivamente. Una manera mas sencilla de interpretar es la siguiente

$$W_p(\mu, \sigma) = \left(\inf_{\mathbb{E}} [d(X, Y)^p] \right)^{\frac{1}{p}} \tag{4.7}$$

y refiere al ínfimo de los valores esperados de todas las distribuciones de probabilidad conjunta de las variables aleatorias X e Y con marginales μ y σ.

Alternativamente a esto existe la posibilidad de actualizar los pesos de D durante más iteraciones que los de G. Esta solución se basa en que de esta manera el discriminador tendrá acceso a un mayor *pool* de muestras propiciando que G no tenga *overfit*. También se intuye que realizando múltiples actualizaciones de D, G tiene acceso indirecto a la trayectoria de las futuras actualizaciones del discriminador.

En [17] se presentan más alternativas, donde exponen la posibilidad de escoger las muestras de una normal truncada. Así las muestra con una norma superior a un umbral son remuestreadas con mayor frecuencia permitiéndoles tomar la misma importancia que las muestras más comunes.

4.3.2. Inestabilidad

El entrenamiento de GAN consiste en encontrar un equilibrio de Nash para un juego no cooperativo de dos jugadores. Desafortunadamente, encontrar equilibrios de Nash es un problema muy difícil. Existen algoritmos para casos especializados, pero no conocemos ninguno que sea factible aplicar al juego GAN, donde las funciones de coste no son convexas, los parámetros son continuos y el espacio de parámetros es *high-dimensional* [18]. La sensibilidad en la actualización de los pesos de la GAN también es un punto clave y varios factores pueden contribuir a dicha inestabilidad en la actualización de los pesos:

- Gradientes poco densos (*sparse*).
- Un soporte disjunto entre las imágenes reales y las imágenes falsas.

Las no–linearidades producidas por las funciones de activación como la ReLU, una función lineal definida a trozos de manera que:

$$f(x) = \begin{cases} x \text{ si } x \geq 0 \\ 0 \text{ si } x < 0 \end{cases} = \text{máx}\{0, x\} \tag{4.8}$$

pueden dar lugar a gradientes sparse. Esto se debe a que todo valor negativo se convierte en cero al realizar la actualización de los pesos.

Por otro lado utilizar *Max Pooling* después de las capas convolucionales también produce *sparse gradients* que pueden producir un entrenamiento inestable. En el caso de las funciones de activación ReLU es bastante intuitivo el por qué se generan gradientes poco densos, debido a que todos los valores negativos se convierten en cero. En el caso del *Max Pooling*, esta es una técnica que después de aplicar el *kernel* en la convolución selecciona el máximo de los valores resultantes para contribuir a la capa siguiente. Debido a esto, en el *backpropagation* los valores en las posiciones no máximas serán nulos. Para resolver estos gradientes *sparse* existen diferentes soluciones, como el tipo de funciones de activación que utilizamos (PreLU, LeakyReLU, etc), evitar aplicar *Max Pooling* en cada capa, etc. Ahora veamos el problema con los soportes.

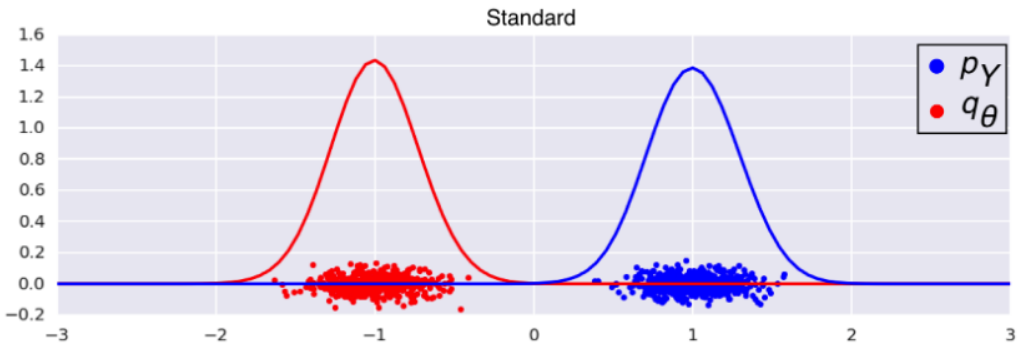

Figura 4.5: Soportes disjuntos.

En la Figura 4.5 observamos dos distribuciones de densidad. La línea roja representa una distribución de datos estimados en función de un parámetro θ mientras que la azul corresponde a la distribución de datos reales, ambas siguiendo una distribución normal. Debido a que tienen una media distante y las varianzas de los datos no es lo suficientemente grande los soportes son disjuntos y ninguno de los conjuntos formados por estas distribuciones tomarán valores de la otra.

En este caso D, el discriminador, puede diferenciar perfectamente entre ambas clases. Es más, no se requiere de una red neuronal para la tarea ya que dos soportes disjuntos en $\mathbb{R}$ son linealmente separables y para $\mathbb{R}^n$ siempre podemos llevarlo mediante un kernel a $\mathbb{R}^{n+1}$ donde es linealmente separable. Para evitar esto se puede aplicar ruido a las etiquetas y/o imágenes de las muestras reales y falsas. Esta técnica recibe el nombre de *Instance Noise*.

4.3.3. Sensibilidad a la inicialización de los hiperparámetros

En [19] se realizó un gran estudio en el cual múltiples funciones de perdidas de las GAN eran evaluadas con diferentes inicializaciones. En este estudio encontraron que muchos modelos podían alcanzar resultados similares con suficiente optimización de hiperparámetros y comienzos aleatorios. Esto sugiere que muchos modelos son muy sensibles a estas modificaciones y pueden mejorar simplemente con más potencia de computación sin necesidad de cambios en los algoritmos. Los conjuntos de datos utilizados en [19] fueron

- Celeba
- Cifar10
- Fashion-Mnist
- Mnist

Estos conjuntos de datos han sido superados ya por muchas arquitecturas alcanzando niveles de precisión muy altos, por lo que no hay que tomar los resultados como un dogma. En *datasets* más complejos o con unas características específicas, como veremos en el capítulo 5, la importancia de la arquitectura puede ser clave y a pesar que los hiperparámetros son un pilar, su optimización no tiene porqué llevar a resultados tan buenos como GANs con arquitecturas diseñadas para el problema específico.

4.3.4. Vanishing gradient

Empecemos recordando que el discriminador de las GAN se entrena con el objetivo de maximizar

$$\max_D V(D,G) = \mathbb{E}_{x\sim p_z}[\log(1 - D(G(z)))]$$

Y que cuando se entrena hasta converger la función *loss* del discriminador será mínima cuando $D(x) = 1$ para los ejemplos reales, y $D(x) = 0$ para los ejemplos falsos. Esto sucede cuando los conjuntos son completamente separables o las distribuciones no son continuas. En el caso de estar llegando al óptimo el gradiente será cero casi en su totalidad, lo que dificulta la actualización de los pesos. Otra razón para la aparición del *vanishing gradient* es la que comentábamos en la subsección 4.2.1, en redes neuronales muy profundas (con una gran cantidad de capas) el gradiente se anula en etapas muy tempranas sin ser capaz de actualizar los pesos de todas las capas de la red y por ello introducíamos técnicas como los *residual blocks* y *skip connections*.

4.3.5. Tricks and treats

Las GAN son un conjunto de modelos relativamente nuevos por lo que su estudio y mejora todavía se encuentra en una primera etapa donde la comunidad de investigadores continua realizando avances. Muchos de estos avances sirven para entender el comportamiento de errores que ocurren durante el entrenamiento y que se buscan evitar. Por ejemplo, cuando la función *loss* de $D \longrightarrow 0$, el proceso produce actualizaciones no significativas en el generador y no tiene sentido continuar con el entrenamiento. A continuación se enumeran algunos de los trucos o ideas a tener en cuenta.

- La varianza del discriminador tiene que reducirse a lo largo del tiempo, es esperado y beneficioso para la obtención de unos resultados buenos. La razón detrás de esto es que si se está realizando un correcto aprendizaje de la tarea la precisión de D debería aumentar de manera cada vez más regular para cualquier tipo de muestra. Si se producen picos espontáneos significaría que puede existir un subconjunto de datos que D no ha aprendido ya que se ha estado centrando en particularidades que no generalizan correctamente.

- Aunque teóricamente, concatenar *inputs* discretos como las máscaras junto con imágenes en la dimensión de los canales es preferible y después aplicar convoluciones o capas densas, en la práctica puede ser muy difícil de optimizar. Por lo tanto, es preferible tratar de añadir o multiplicar los *inputs* discretos con las imágenes directamente.

- Como ya se ha comentado, añadir ruido es un procedimiento que permite evitar soportes disjuntos y que permite eliminar, potencialmente, el efecto de *vanishing gradient* y las inestabilidades en el entrenamiento. En el trabajo se ha introducido una variante inspirada en el *Reinforcement Learning* y los entrenamientos $\varepsilon-$*greedy* con el $\varepsilon-$*decay* que se explica en el capítulo 5. Un ejemplo de las técnicas de introducción de ruido sería el que se muestra en la Figura 4.6.

  Por un lado se pueden modificar algunas de las etiquetas reales y/o generadas e identificarlas como verdaderas. Esto elimina el problema de los soportes disjuntos como se observa en la gráfica superior de la Figura 4.6. Por otro lado se puede añadir un ruido ϵ a las muestras generadas y reales de manera que la varianza

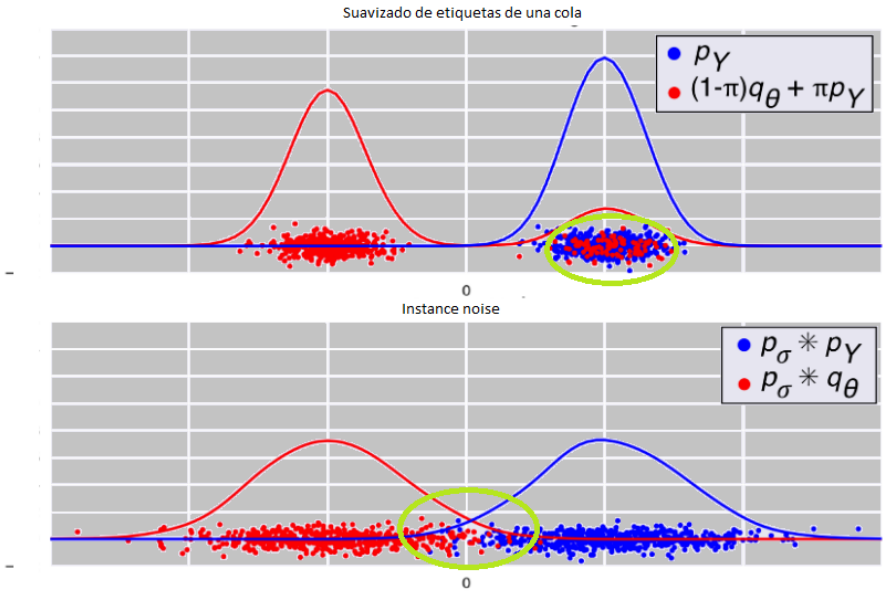

Figura 4.6: Técnicas y efectos de introducción de ruido.

de su distribución aumente lo suficiente provocando que si S_θ es el soporte de los *outputs* de G y S_Y es el soporte de muestras reales entonces se cumple

$$S_\theta \cap S_y \neq \emptyset \tag{4.9}$$

- Habitualmente se hace referencia a una normalización $[0, 1]$ en caso de que la sigmoide esté como la salida de G, o bien $[-1, 1]$ si la salida de G está asociada a una tanh. En este trabajo se ha considerado que ninguna de ambas posibilidades acaba de tener unas propiedades adecuadas debido a posibles *outliers* o picos de brillo en las imágenes, lo que puede provocar una distorsión substancial de dichas normalizaciones. Se ha considerado que la mejor opción es realizar un estandarizado utilizando el *Z-score* o un *Z-score* modificado según las características de las imágenes. Por ejemplo, en caso de encontrar valores extremos en los pixeles sería mas interesante cambiar el *Z-score* de la siguiente manera.

$$\frac{X - \hat{X}}{\frac{\sigma}{\sqrt{n}}} \longrightarrow \frac{X - median(X)}{\frac{\sigma}{\sqrt{n}}} \tag{4.10}$$

  donde $\hat{X}$ es la media y $median(X)$ hace referencia a la mediana de los datos. Incluso se pueden utilizar los logaritmos de los datos para normalizar en caso de encontrarnos con distribuciones lognormales.

- Es preferible utilizar funciones de activación como la LeakyReLU o PreLU que eviten *sparse gradients*. Ambas funciones modifican la función original de la ReLU en la ecuación (4.8) de manera que permiten valores negativos para evitar los *sparse gradients*.

$$LeakyReLU(x) = \text{máx}(x, \alpha \cdot x) \qquad \text{con } \alpha \in (0, 1) \tag{4.11}$$

$$PReLU(x) = \text{máx}(0, x) - \alpha\, \text{máx}(0, -x) \tag{4.12}$$

En la función de activación PReLU el coeficiente α también es un valor que se aprende en la optimización de la red. Además, es preferible utilizar optimizadores como ADAM que empíricamente parecen dar mejores resultados [20, 12].

- Realizar una buena optimización de los hiperparámetros. Como en el resto de arquitecturas deep learning los hiperparámetros son clave. Para un D y G de igual tamaño se proponen *learning rates* de diferentes tamaños siendo más pequeño el de G, o se realizan más actualizaciones de D que de G. Si D y G tienen tamaños diferentes el objetivo es equilibrar los modelos para que las actualizaciones del grande sigan el ritmo de las del pequeño.

  Otra opción es realizar un entrenamiento asíncrono, es decir, no se actualizan los pesos de D en cada iteración si no que cuando este tiene buen *accuracy* no se actualiza y se comienzan a actualizar los pesos cuando su precisión baja.

Capítulo 5

Análisis y resultados

5.1. Arquitectura del proyecto

Entrenar una GAN requiere de una capacidad de cálculo elevada e incluso con el uso de una GPU si se busca realizar un trabajo eficiente. En el caso de este Trabajo de Fin de Máster, no contaba con la tecnología necesaria para realizar los procesos de entrenamiento en local. Inicialmente intenté trabajar con *Google Colab*, una plataforma de *Google* que pone a disposición del público de manera limitada con recursos como GPUs en un entorno de *Jupyter Notebook*. *A priori* parece una buena solución en el caso de no tener los recursos suficientes, por lo que con una cuenta de *Drive* se pueden alojar los datos y posteriormente trabajar en *Google Colab* con ellos. Sin embargo, no es una solución adecuada por las siguientes razones:

- Si se busca realizar un proyecto de ciencia de datos o simplemente un contexto de investigación, es necesario tener un entorno controlado donde poder trabajar con una serie de versiones definidas, el control de los cambios que se realizan y la posibilidad de que colaboradores realicen cambios en todo momento sin afectar al *core* del trabajo.

- Si se necesita realizar largos procesos de entrenamiento no es posible hacerlo con herramientas como *Google Colab* ya que tiene un tiempo límite de espera y para evitar la desconexión se ha de estar realizando alguna operación cada cierto tiempo. Debido a esto, si se desea realizar un proceso de entrenamiento demasiado largo el usuario ha de estar presente durante todo el periodo para evitar la desconexión.

Finalmente se optó por el uso de la computación en la nube. De entre las tres posibilidades presentes, *Microsoft Azure*, *Amazon Web Services* y *Google Cloud* la tercera opción fue la seleccionada. *Google Cloud* tiene muchas funcionalidades de todo tipo, pero debido a las necesidades básicas que tenía, la opción más clara era levantar una instancia de Máquina Virtual (VM) Linux e instalar las herramientas necesarias para el proyecto.

Característica	Valor	Tipo
Zona	Europa	europe-west4-b
Disco	4092	Disco persistente estándar
CPUs	8	104 Gb
GPUs	NVDIA V100	
Sistema Operativo	Debian(9)	

Cuadro 5.1: Tabla de características de la VM.

Paquete	**Versión**
Scipy	1.1.0
Scikit-image	0.14.2
Tensorflow GPU	1.14
Pandas	0.24.2
Numpy	1.16.4
Glob	0.7

Cuadro 5.2: Versiones necesarias para el funcionamiento del proyecto.

Una vez instalada la VM se accede a ella mediante SSH, lo que en un principio puede suponer muchos problemas si desconoces su funcionamiento interno. Al realizar una conexión mediante SSH tienes, así como en *Google Colab*, un tiempo límite de inactividad del cual, por cierto, avisa en la documentación de sus VM. Existen diversas maneras de solucionar el problema, *tmux* o el comando *nohup* que permiten lanzar los procesos fuera del nodo hijo al que te conectas mediante SSH, y de esta manera se pueden recuperar o siguen corriendo respectivamente cuando somos expulsados y volvemos a conectarnos a la VM. Sin embargo, ninguna de las dos opciones ha sido elegida. Por el contrario se ha utilizado *Putty* debido a que tiene la opción de enviar mensajes cada cierto tiempo llamados *Keepalive* que permiten que la sesión a la que te conectas no entre en estado de inactividad. Aprovechando que se estaba trabajando con una SSH private key y debido a los problemas que se pueden tener para conectar por SCP desde local con la VM para descargar los archivos, se utiliza *FileZilla* para conectarse a la instancia y realizar el manejo de los archivos.

El proceso se ha llevado a cabo en la nube y todo el proyecto se puede encontrar en *https://github.com/Oteo95* [21]. El objetivo del repositorio es ofrecer una herramienta para continuar probando las arquitecturas e ideas presentadas en el TFM, ya que la estructura del código esta planteada para que se puedan realizar modificaciones o incluso una GAN totalmente nueva, y si mantienen ciertas funciones base se puede mantener el mismo proceso de entrenamiento y funciones de carga de datos. Además, se proporcionan los archivos Docker necesarios para mantener el entorno de versiones en cualquier ordenador sin necesidad de modificar las versiones locales que tiene el usuario.

5.1.1. Docker

Antiguamente las aplicaciones se desarrollaban directamente sobre el hardware, sobre el host OS. Debido a esto, el tiempo de ejecución es compartido entre las aplicaciones. El desarrollo estaba atado al hardware y esto suponía largos ciclos de mantenimiento que permiten menos flexibilidad a los desarrolladores. En la Figura 5.1 se observa un esquema de un diseño tradicional sobre el cual se desarrollan las aplicaciones.

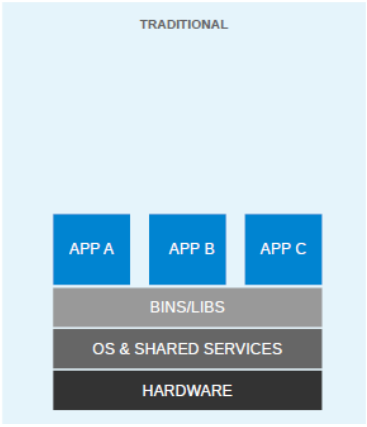

Figura 5.1: Diseño tradicional en el desarrollo de aplicaciones [1]

Para aumentar la flexibilidad con el objetivo de utilizar mejor los recursos del sistema de host se inventó la virtualización. De esta manera se desarrollaron técnicas para la emulación del hardware creando maquinas virtuales (VM). Las máquinas virtuales pueden tener un sistema operativo diferente que su host. Esto significa que somos responsables de administrar parches, seguridad y el rendimiento de esa VM. Con la virtualización, las aplicaciones se aíslan a nivel de VM y se definen por el ciclo de vida de las VM. Esto nos brinda un mejor retorno de nuestra inversión y una mayor flexibilidad a costa de una mayor complejidad y redundancia. La Figura 5.2 muestra un entorno virtualizado típico.

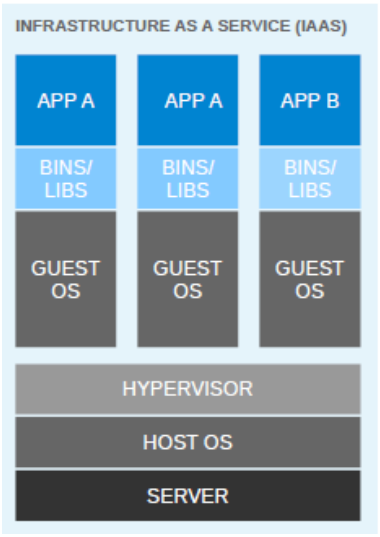

Figura 5.2: Virtualización de los entornos [1]

Desde que se desarrolló la virtualización, nos hemos estado moviendo hacia una tecnología más centrada en las aplicaciones. Hemos eliminado la capa de hipervisor para reducir la emulación y la complejidad del hardware. Las aplicaciones se empaquetan con su entorno de tiempo de ejecución y se implementan mediante contenedores. La Figura 5.3 muestra cómo se implementa una aplicación utilizando contenedores.

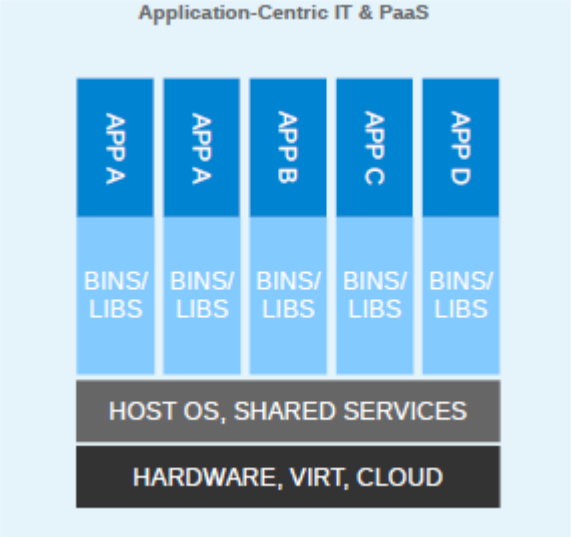

Figura 5.3: Desarrollo de aplicaciones con contenedores [1]

Con Docker, los contenedores se convirtieron de repente en ciudadanos de primera clase. Todas las grandes corporaciones, como Google, Microsoft, Red Hat, IBM y otras, ahora están trabajando para que los contenedores se generalicen.

Para ponerlo en el contexto práctico en el que nos encontramos, este proyecto ha sido desarrollado en un sistema operativo Linux con una serie de programas como es Python 3, *drivers* para el uso de GPUs, paquetes de Python con versiones específicas, etc. Diversos factores pueden provocar que la transferencia de la aplicación a otros ordenadores o sistemas genere conflictos. Existe la solución de montar una máquina virtual en nuestro ordenador pero esto supone montar todas las componentes de la maquina, es decir, sistema operativo, interfaz, programas, etc. lo que puede ser poco eficiente. Además, si tenemos varias aplicaciones con características diferentes se han de montar VM diferentes por lo que puede ser exigente a nivel de recursos. En este punto entra Docker. Lo que se ha desarrollado para el TFM es una imagen especifica sobre un sistema operativo Linux con los drivers, Python 3, Tensorflow-gpu y los paquetes en las versiones necesarias para el proyecto. De esta forma cualquier usuario con Docker puede levantar un contenedor en su ordenador únicamente con las especificaciones necesarias para lanzar los entrenamientos a partir de las GANs desarrolladas. Esto es más eficiente a nivel de recursos y agiliza mucho el proceso de obtención de resultados y permite a cualquier usuario desarrollar modificaciones o nuevos modelos y correrlos sobre la estructura del proyecto, ya desarrollada, sin preocuparse.

5.1.2. Github

El proyecto finalmente esta distribuido en un repositorio subdividido en directorios segun el carácter de los elementeos que va a almacenar.

- data: Directorio con la las imágenes del proyecto separadas, en este caso, en dos subdirectorios. El primero para las imágenes de train y el segundo para las imágenes de test.
- outputs: Volumen reservado para guardar todos los modelos y resultados intermedios durante el proceso de entrenamiento.

- resultados: Volumen reservado para el análisis final de las imágenes y modelos generados.

- src: Directorio con las clases de las diferentes GANs que se implementen.

- utils: Directorio con las funciones de análisis, carga de datos y evaluaciones de los resultados.

- scripts: Directorio con los scrips para lanzar los entrenamientos de las GANs.

- notebooks: Directorio para el almacenamiento de los notebooks de analisis y pruebas realizadas.

Las clases para cada GAN heredan de una clase base *abstract* que les obliga a implementar la función que define el generador, la función que define al discriminador y la función gan. Dentro de las clases GAN, salvo por las funciones obligatorias, hay libertad de implementar cualquier paso intermedio necesario por el usuario y se puede utilizar libremente. El uso de la herencia y la obligación de implementar las funciones generador, discriminador y gan vienen dadas por el desarrollo de una clase *Train* genérica. Esta clase *Train* inicializará la clase GAN que le indiquemos y se ejecutarán el generador, discriminador y gan para el entrenamiento con los parámetros de *instance noise*, funciones de perdida, ratios de aprendizaje, optimizadores, etc. que necesitemos.

Los volumenes y directorios también están pensados para que la función train sea generica y pueda acceder a todos ellos por path relativo, evitando que los usuarios que desarrollen tengan que realizar cambios sobre el código.

5.2. Imágenes NIR y RED

Como ya se ha comentado en la introducción, el conjunto de datos son imágenes provenientes del satélite PROVA-V de tipo NIR y RED pero, ¿qué son realmente estas imágenes? Empecemos explicando las imágenes NIR (Near Infrared) o imágenes del infrarrojo cercano.

NIR es un subconjunto de bandas infrarrojas del espectro electromagnético que cubren las longitudes de onda en un rango entre 0.7 y 1.4 microns. Esta longitud de onda está fuera del rango visible de los humanos y puede ofrecer detalles más claros que aquellos visibles mediante luz blanca para determinadas aplicaciones. NIR está muy cerca de entrar en la visión humana. Los árboles y las plantas son altamente reflectantes para las longitudes de onda NIR. Esta diferencia en la capacidad de reflexión de ciertos objetos en combinación con la reducida bruma atmosférica y distorsión de las longitudes de onda NIR significa que la visibilidad y detalle son mejorados en rangos amplios como se muestra en la figura 5.4.

En nuestro caso las imágenes no son tan claras pues se realizan fuera de la atmósfera y por lo tanto identificar los arboles o la vegetación no es una tarea inmediata al ojo humano. Las figuras 5.5 y 5.6 muestran ejemplos de imágenes NIR del PROBA-V (a la izquierda) junto con sus respectivas máscaras (a la derecha).

Figura 5.4: Representación de las capas RGB (izquierda) y NIR (derecha) de una misma escena.

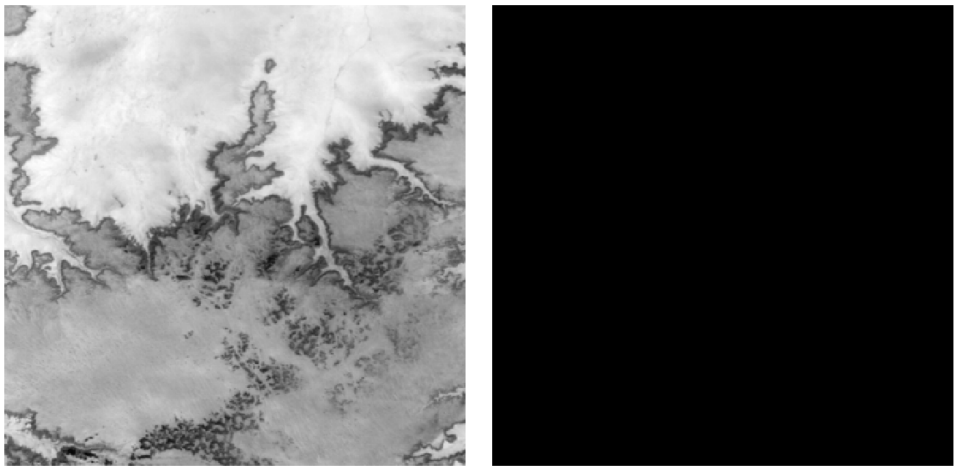

Figura 5.5: Fotografía NIR (izquierda) y su máscara de nubes (derecha).

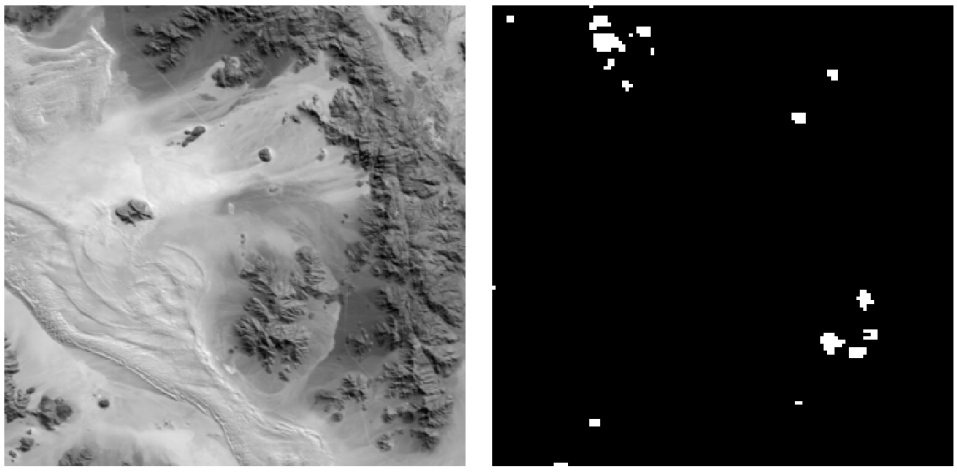

Figura 5.6: Fotografía NIR (izquierda) y su máscara de nubes (derecha).

Nuestro conjunto de datos cuenta con 1159 imágenes divididas en un subconjunto de 593 escenas RED y 566 escenas NIR. Cada una de las escenas cuenta con un conjunto de imágenes LR 128×128 con sus respectivas máscaras QM (quality-map). Cada escena tiene al menos 9 imágenes LR con sus QM. Además, cada escena cuenta con la respectiva imagen HR y su QM de tamaño 384×384 que es nuestro objetivo, es decir, la imagen que buscamos reconstruir a partir de las LR.

Es importante entender que a la hora de trabajar con imágenes la percepción visual que tenemos de ellas puede no corresponder con los valores de sus píxeles, y dos imágenes aparentemente iguales pueden tener valores totalmente distintos. Por ello es importante el estudio estadístico del comportamiento de los píxeles, así como el rango de valores que toman. Para ello vamos a estudiar los máximos y los mínimos de las imágenes y ver si el comportamiento general de las NIR (Figura 5.7) es homogéneo, o si se han podido producir anomalías o efectos no deseados en la adquisición de estas, como se comentaba en la sección 3.1. En la Figura 5.7 observamos que ambas bandas tienen máximos y mínimos similares, sin embargo se puede ver una asimetría en los máximos de las imágenes LR. La aparición e píxeles con un valor máximo más alto que en las HR aunque no es habitual se puede dar y es un detalle a tener en cuenta. Sobre todo al normalizar como comentábamos en 4.3.5.

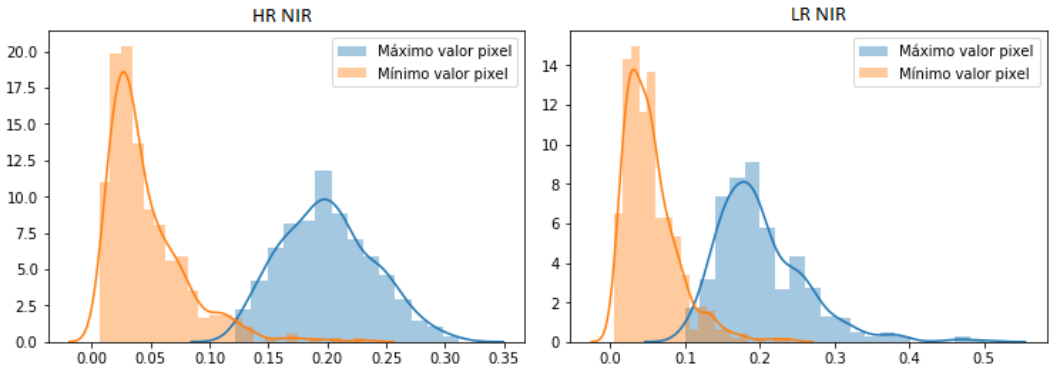

Figura 5.7: Distribuciones de máximos y mínimos en NIR de las imágenes LR y HR.

El caso de las imágenes RED es más conocido pues es la banda R de las imágenes a color compuestas por Red, Green y Blue (RGB). Dichas imágenes son muy similares a las anteriores y su distribución no es muy diferente por lo que no es necesario realizar grandes cambios en el preprocesado habitual de las imágenes para adecuar ambos tipos escenas. La distribución en las de tipo RED es la que se muestra en la Figura 5.8 donde observamos que ocurre el mismo fenómeno que en las LR NIR. Se aprecia una asimetría provocando que los máximos de las imagenes LR tomen valores más altos que los de las HR.

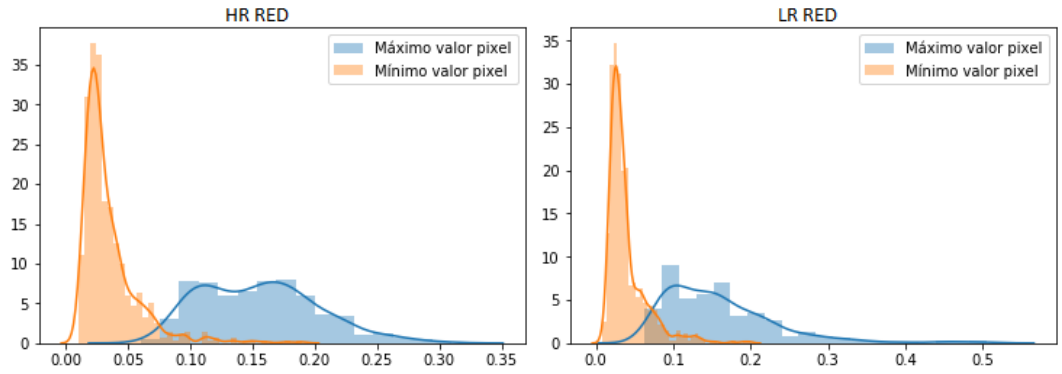

Figura 5.8: Distribuciones de máximos y mínimos en RED de las imágenes LR y HR.

Por otro lado, hay que tener ciertas consideraciones, ya que a pesar de que las imágenes LR representan fuertemente su escena HR, existen múltiples factores a ser considerados:

- Las imágenes LR no están registradas al mismo tiempo con el resto de su escena.
- Las imágenes LR y las HR no están registradas en el mismo instante.
- El brillo de las imágenes HR puede ser diferente de cualquiera de las imágenes LR.
- Las escenas cambian debido a las múltiples adquisiciones.
- Las escenas LR y HR pueden estar cubiertas por diferentes nubes y sus sombras, produciendo patrones o artefactos que pueden corromper el valor de los píxeles.

Para lidiar con este problema propongo emplear una aproximación utilizando un entrenamiento *adversarial* que pueda aprender, dadas la imágenes LR *raw* o alguna transformación de estas, la distribución real de los píxeles en las imágenes HR.

5.2.1. Mejoras en el muestreo de las imágenes

Como hemos presentado anteriormente las imágenes del conjunto de datos no han sido registradas ni temporalemente ni espacialmente, por lo que es imposible conectar el canal RED ni el canal NIR, ya que no tenemos información sobre qué conjuntos pares de imágenes (RED_i, NIR_i) son coincidentes. En el caso de las imágenes del satélite PROBA-V, estas se puede descargar desde la sección de datos gratuitos de *PROBA-V product distribution portal* [22]. Haciendo esto somos capaces de registrar espacial y temporalmente las imágenes, permitiéndonos calcular un indicador de vegetación como es el *Normalized Difference Vegetation Index* (NDVI) que es frecuentemente utilizado en la práctica para medir biomasa, indicios de sequías o la densidad de las hojas en los bosques [23].

Para calcular el NDVI se necesita la información de los canales NIR y RED y se realiza mediante

$$NDVI = \frac{(NIR - RED)}{(NIR + RED)} \tag{5.1}$$

Mediante este indicador tendríamos unas imágenes capaces de captar mejor la vegetación global y con conocimiento de la localización geográfica y temporal de las imágenes

LR podríamos construir las imágenes NDVI SR que permitirían un mejor seguimiento de la vegetación a nivel global.

5.3. Evaluación de las imágenes

Una imagen SR ha de tener los siguientes requisitos:

- Los valores en píxeles en SR deben estar lo más cerca posible de la imagen HR de destino (después de eliminar el sesgo innecesario).
- La calidad de la imagen debe ser independiente de los valores de píxeles marcados como ocultos en la imagen de destino.
- La imagen SR no debe reconstruir características volátiles (como nubes) ni introducir artefactos.

Dados estos requisitos, tomamos la relación de ruido de señal máxima (PSNR) como nuestro punto de partida y modificamos algunos aspectos. El PSNR se basa en el error cuadrático medio (MSE) en píxeles que podemos restringir fácilmente a píxeles claros (es decir, no ocultos), ya que la reconstrucción de las nubes no tiene sentido.

Un inconveniente potencial del PSNR es una alta sensibilidad a los sesgos en el brillo. Un cambio constante en la intensidad de cada píxel aumenta rápidamente y puede ser aún más perjudicial que enviar imágenes con artefactos y nubes. Por lo tanto, ecualizamos las intensidades de las imágenes enviadas de modo que el brillo de píxeles promedio de ambas imágenes coincida. Llamamos al PSNR modificado para brillo y nubes cPSNR .

Por último, observamos que el valor absoluto del cPSNR depende del conjunto de imágenes. Así, algunos conjuntos de imágenes dan (en promedio) valores de cPSNR más altos que otros. Para que cada conjunto de imágenes contribuya igualmente en [2] se establece una solución de referencia, calculamos su cPSNR y la usamos para la normalización.

El cálculo formal del cPSNR se realizará de la siguiente forma. Suponemos que las intensidades de los píxeles se representan como números reales $\in [0, 1]$

$$|clear(HR)| = \sum_{x,y} mask(HR)_{x,y} \tag{5.2}$$

$$bias = \frac{1}{|clear(HR)|} \left(\sum_{x,y \in clear(HR_{x,y})} HR_{x,y} - SR_{x,y} \right) \tag{5.3}$$

A continuación calculamos el cMSE de las imágenes SR y las HR

$$cMSE(HR, SR) = \frac{1}{|clear(HR)|} \left(\sum_{x,y \in clear(HR_{x,y})} HR_{x,y} - (SR_{x,y} + b) \right)^2 \tag{5.4}$$

De esta manera podemos obtener el *Peak signal to noise ratio* de las imágenes

$$cPSNR(HR, SR) = -10 \cdot \log_{10}(cMSE(HR, SR)) \tag{5.5}$$

En caso de tener un baseline como se proporciona en [2] lo definiremos como $N(HR)$. Cada valor resultado individual de las SR responderá a

$$z(SR) = \frac{N(HR)}{cPSNR(HR, SR)} = NcPSNR \tag{5.6}$$

Nota sobre los resultados del trabajo

Los resultados utilizando la normalización han de superar un *baseline* de 1 teniendo en cuenta el NcPSNR. Es decir, todos los resultados que presenten un NcPSNR medio por debajo de 1 serán válidos. Esto se debe a que utilizando el modelo benchmark, en una interpolación bicúbica sobre la mejor imagen LR se obtiene este resultado como se indica en [2].

El mejor método presentado en este trabajo, la SRGAN modificada, presenta un $NcPSNR = 0,9987$ sobre un conjunto de test de 200 imágenes, 100 NIR y 100 RED. La Figura 5.9 muestra un ejemplo utilizando el cPSNR normalizado. Esta figura está dividida en dos filas. La primera corresponde a la comparación con el modelo GAN desarrollado. La segunda fila corresponde a la comparación con el modelo de interpolación bicúbica. Así pues, en la primera columna encontramos las imágenes SR, en la columna central la imagen HR y en la columna de la derecha encontramos el error cometido por los modelos al utilizar el cMSE eliminando el sesgo del error.

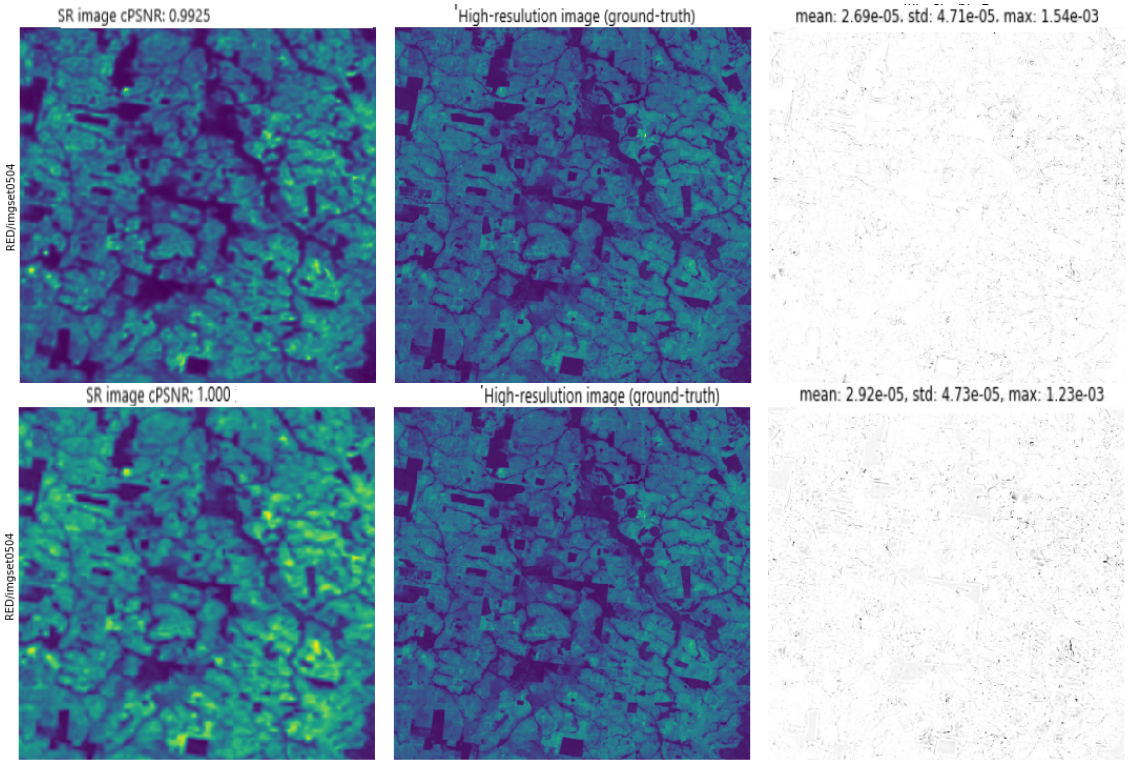

Figura 5.9: NcPSNR. Imágenes SR e interpolada (izquierda), HR (centro) y error cometido (derecha).

En [2] se puede obtener el archivo para realizar la normalización. Aunque contamos con el archivo para la normalización de los resultados, en este trabajo los resultados se presentan sin normalizar. La intención principal de esto es presentar un marco genérico a cualquier conjunto de datos seleccionado de [22], permitiendo a la comunidad desarrollar sus propios algoritmos y medirlos de una manera rápida contra los resultados presentes y futuros.

5.4. Métodos

5.4.1. Interpolación bicúbica

La interpolación bicúbica es el método base elegido por la competición para marcar el *benchmark* al que batir. La interpolación bicúbica es un método clásico que cae en el dominio espacial como se comentaba en la subsección 2.2. El concepto de estos métodos es estimar el valor de los píxeles desconocidos utilizando el valor de los píxeles conocidos de alrededor. La Figura 5.10 presenta un ejemplo visual del funcionamiento de un método de interpolación para la super-resolución de imágenes.

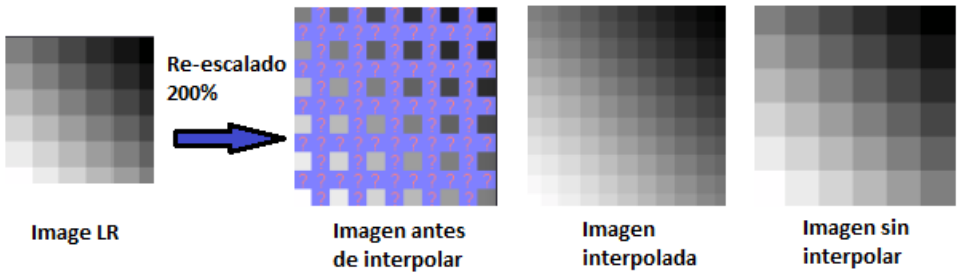

Figura 5.10: Ejemplo visual del funcionamiento de la interpolación

Existen diversos tipos de interpolación: vecinos más cercanos, bilineal, bicúbica, spline, sinc, Lanczos, etc. Posiblemente la interpolación bicúbica sea la más utilizada debido a dos factores combinados. El equilibrio entre eficiencia computacional y resultados hace que sea tan común ver este método en la práctica. Sus resultados son mejores que métodos más sencillos como los vecinos más cercanos o la interpolación bilineal y computacionalmente no conlleva un tiempo mayor. Aunque haya métodos más complejos como los splines o sinc, que son interpolaciones de mayor orden, estas son computacionalmente más exigentes. Todo esto deja a la interpolación bicúbica como una buena elección.

La interpolación bicúbica considera las vecindades 4×4 de píxeles conocidos, por lo que se consideran 16 píxeles para cada píxel no conocido. Además, como estos están a distancias diferentes los pesos van acordes a la lejanía respecto del píxel desconocido. Como comentábamos, es un método computacionalmente eficiente y que ofrece mejores resultados que otros métodos poco costosos convirtiéndose en la aplicación para muchos programas, como *Adobe Photoshop*. Matemáticamente podemos definir la interpolación bicúbica de la siguiente manera.

Supongamos que la función f y sus derivadas parciales f_x, f_y y f_{xy} son conocidas en los vértices (0,0), (0,1), (1,0) y (1,1). La superficie interpolada puede ser descrita como

$$p(x,y) = \sum_{i=0}^{3} \sum_{j=0}^{3} a_{ij} x^i y^j \tag{5.7}$$

Entonces el problema de interpolación consiste en determinar los 16 coeficientes a_{ij}. Luego mapeando los valores de la función f conocidos con $p(x,y)$ obtenemos cuatro ecuaciones

$$f(0,0) = p(0,0) = a_{00}$$
$$f(1,0) = p(1,0) = a_{00} + a_{10} + a_{20} + a_{30} \tag{5.8}$$

$$f(0,1) = p(0,1) = a_{00} + a_{01} + a_{02} + a_{03}$$
$$f(1,1) = p(1,1) = \sum_{i=0}^{3}\sum_{j=0}^{3} a_{ij}$$

Como las derivadas primeras también son conocidas en el sistema en el que nos encontramos entonces tenemos ocho ecuaciones más definidas por el mapeo de estas como se muestra en las ecuaciones de (5.9).

$$\begin{aligned}
f_x(0,0) &= p_x(0,0) = a_{10} \\
f_x(1,0) &= p_x(1,0) = a_{10} + 2a_{20} + 3a_{30} \\
f_x(0,1) &= p_x(0,1) = a_{10} + a_{11} + a_{12} + a_{13} \\
f_x(1,1) &= p_x(1,1) = \sum_{i=0}^{3}\sum_{j=0}^{3} a_{ij} i \\
f_y(0,0) &= p_y(0,0) = a_{01} \\
f_y(0,1) &= p_y(0,1) = a_{01} + 2a_{02} + 3a_{03} \\
f_y(1,0) &= p_y(1,0) = a_{01} + a_{11} + a_{21} + a_{31} \\
f_y(1,1) &= p_y(1,1) = \sum_{i=0}^{3}\sum_{j=0}^{3} a_{ij} j
\end{aligned} \tag{5.9}$$

Por último tenemos las cuatro ecuaciones de la derivada cruzada f_{xy}

$$\begin{aligned}
f_x y(0,0) &= p_y(0,0) = a_{11} \\
f_x y(0,1) &= p_y(0,1) = a_{11} + 2a_{12} + 3a_{13} \\
f_x y(1,0) &= p_y(1,0) = a_{11} + 2a_{21} + 3a_{31} \\
f_x y(1,1) &= p_y(1,1) = \sum_{i=0}^{3}\sum_{j=0}^{3} a_{ij} ji
\end{aligned} \tag{5.10}$$

Este procedimiento produce una superficie $p(x, y)$ en el cuadrado unidad $[0, 1] \times [0, 1]$ que es continua y con derivadas continuas. La interpolación bicúbica en una cuadrícula regular de tamaño arbitrario se puede lograr combinando las superficies bicúbicas y asegurando que las derivadas coincidan en los límites del borde. En este punto el sistema de ecuaciones presentado puede formularse a nivel matricial tal que $A\alpha = x$ siendo α el vector columna de valores a_{ij} y x el vector columna de valores de la función f y sus derivadas. La matriz A puede ser fácilmente invertida

$$\boldsymbol{A^{-1}} = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-3 & 3 & 0 & 0 & -2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & -2 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -3 & 3 & 0 & 0 & -2 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & -2 & 0 & 0 & 1 & 1 & 0 & 0 \\
-3 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -3 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & -1 & 0 \\
9 & -9 & -9 & 9 & 6 & 3 & -6 & -3 & 6 & -6 & 3 & -3 & 4 & 2 & 2 & 1 \\
-6 & 6 & 6 & -6 & -3 & -3 & 3 & 3 & -4 & 4 & -2 & 2 & -2 & -2 & -1 & -1 \\
2 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 2 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
-6 & 6 & 6 & -6 & -4 & -2 & 4 & 2 & -3 & 3 & -3 & 3 & -2 & -1 & -2 & -1 \\
4 & -4 & -4 & 4 & 2 & 2 & -2 & -2 & 2 & -2 & 2 & -2 & 1 & 1 & 1 & 1
\end{bmatrix}.$$

Se puede construir otra matriz

$$\begin{bmatrix} f(0,0) & f(0,1) & f_y(0,0) & f_y(0,1) \\ f(1,0) & f(1,1) & f_y(1,0) & f_y(2,1) \\ f_x(0,0) & f_x(0,1) & f_{xy}(0,0) & f_{xy}(0,1) \\ f_x(1,0) & f_x(2,1) & f_{xy}(1,0) & f_{xy}(2,1) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 3 \end{bmatrix} \tag{5.11}$$

o bien

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & -1 \\ 2 & -2 & 1 & 1 \end{bmatrix} \begin{bmatrix} f(0,0) & f(0,1) & f_y(0,0) & f_y(0,1) \\ f(1,0) & f(1,1) & f_y(1,0) & f_y(1,1) \\ f_x(0,0) & f_x(0,1) & f_{xy}(0,0) & f_{xy}(0,1) \\ f_x(1,0) & f_x(1,1) & f_{xy}(1,0) & f_{xy}(1,1) \end{bmatrix} \begin{bmatrix} 1 & 0 & -3 & 2 \\ 0 & 0 & 3 & -2 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & -1 & 1 \end{bmatrix} \tag{5.12}$$

donde tenemos que

$$p(x,y) = \begin{bmatrix} 1 & x & x^2 & x^3 \end{bmatrix} \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} 1 \\ y \\ y^2 \\ y^3 \end{bmatrix} \tag{5.13}$$

5.4.2. SR-GAN original

El primer modelo que se ha implementado es la arquitectura presentada en [24], un método de SR para imágenes RGB que utiliza una arquitectura GAN e internamente utiliza una red VGG19 pre-entrenada con el dataset *ImageNet*. Dicha red se utiliza para complementar el *MSE-based content loss* utilizando una *loss* calculada sobre el mapeado de características de la VGG el cual es invariante a los cambios en el espacio de píxeles. Estas son funciones de pérdida *perceptual loss*, por lo que en nuestro caso podrían provocar artefactos no visibles en la imagen, que en el caso de la monitorización de la vegetación terrestre pueden ser notables. La arquitectura a construir que se presenta en [24] se muestra en la Figura 5.11.

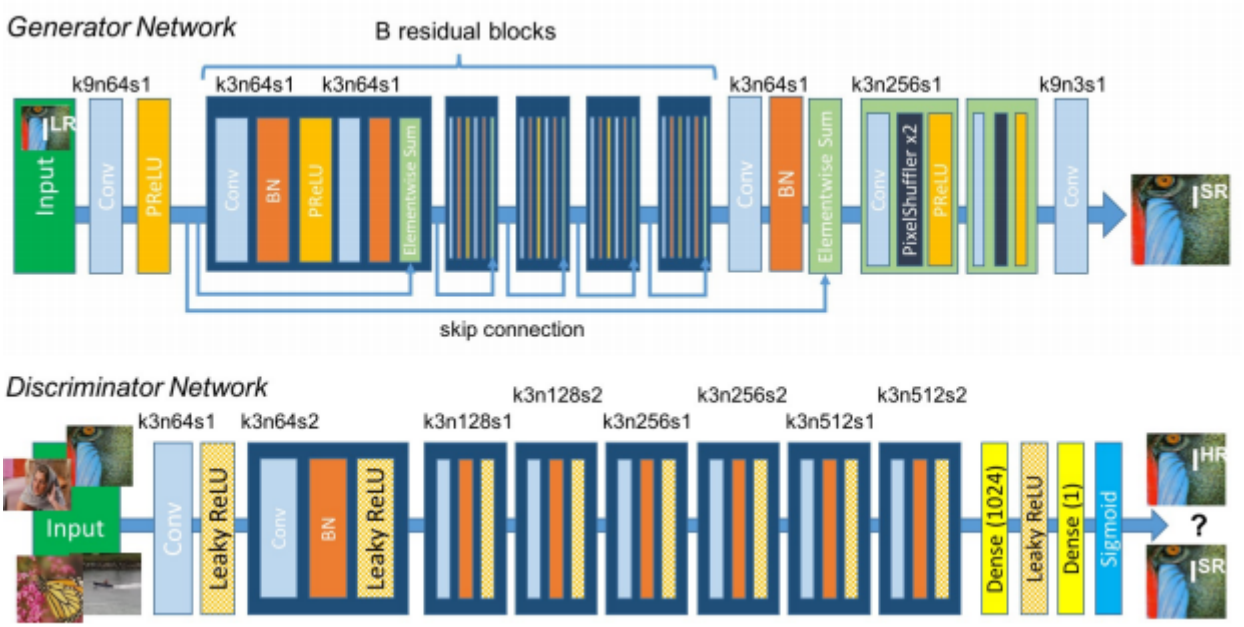

Figura 5.11: Arquitectura original del modelo SRGAN.

En dicha arquitectura se ha realizado un cambio siguiendo el estudio en [25] donde presentan como alternativa a la capa de *SubpixelConvolution* el uso de una capa *UpSampling2D* como un simple *Upsampling* de vecinos más cercanos. Habitualmente una red neuronal genera primero los rasgos mayores de la imagen y después se fija en los detalles. Normalmente para esto utilizamos la operación deconvolución. Desafortunadamente esta operación puede producir fácilmente un *overlap* desigual de las imágenes LR o sus características prestando un mayor esfuerzo en reconstruir o pintar unas zonas más que otras [26]. Dicho *overlap* desigual tiende a ser más extremo en dos dimensiones. Los resultados presentados en [25] ofrecen una alternativa valida a la deconvolución mediante *nearest-neighbor interpolation* para evitar posibles artefactos en las imágenes.

Aunque ya desde un principio dicha arquitectura tenía poca probabilidad de ser exitosa he decidido implementarla para observar la importancia de adecuar el método de resolución de un problema a los datos con los que se trabaja. El principal inconveniente junto con la función *loss* que se utiliza se puede encontrar en el hecho de que la SR-GAN original utiliza la VGG19 entrenada con el conjunto de datos *ImageNet*. Este es un conjunto de datos de imágenes RGB, es decir, imágenes con 3 canales y en ningún caso imágenes de satélite. Esto hace necesario transformar las imágenes NIR y RED

originales a un conjunto de imágenes con 3 canales para adecuarlas a las capas de la VGG19. Aprovechando esta necesidad de preprocesar las imágenes para construir las imágenes de 3 canales se han realizado combinaciones de las imágenes LR combinaciones de N elementos en grupos de tres de manera que cada imagen HR no aparece una única vez si no que aparece tantas veces como combinaciones de las imágenes LR se hayan realizado. El número de veces que aparece la imagen HR asociada a una combinación de 3 imágenes LR es

$$N^{o}img = \frac{N!}{3!(N-3)!} \tag{5.14}$$

En el caso de querer un mayor número de imágenes, se puede realizar un *flip* de estas como técnica de *data augmentation*. Para el caso de las imágenes HR, es la misma imagen la que compone los 3 canales necesarios para coincidir con la arquitectura VGG19 pre-entrenada. Al final queda una red con las características de la Tabla 5.3.

Parámetros	1,452,611
Parámetros entrenables	1,448,387
Parámetros no entrenables	4,224
Número de bloques residuales	16
Optimizador	Adam, $\alpha = 0{,}002$
Número de filtros en G y D	64
Loss	MSE y Binary crossentropy

Cuadro 5.3: Resumen de la arquitectura de SRGAN.

Resultados del modelo original

Varios factores son clave para abandonar el modelo original de [24] y centrarse en las versiones modificadas posteriores. La principal razón para abandonar esta aproximación es muy sencilla, los resultados. En la Figura 5.12 observamos las imágenes SR obtenidas mediante G en la SRGAN original (a la izquierda) y las imágenes HR (a la derecha)

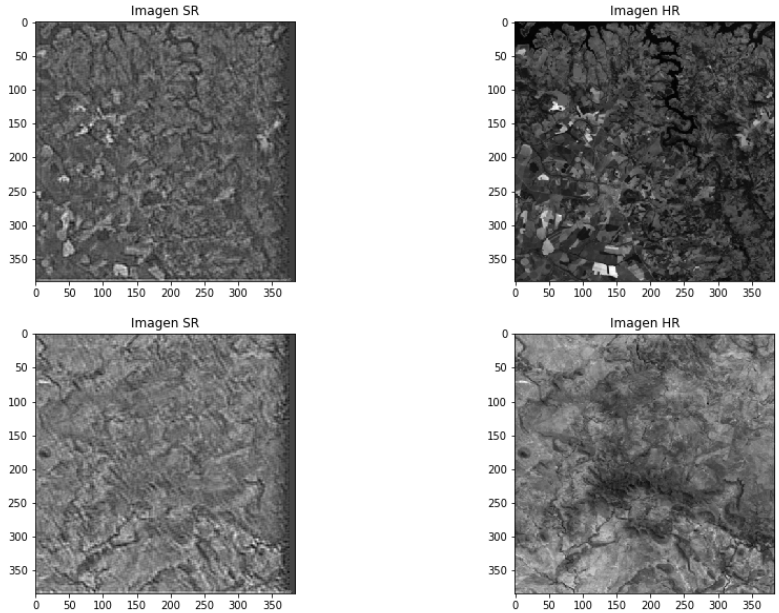

Figura 5.12: Imagen SR vs HR con arquitectura del artículo original en época 29750.

Como se observa, el generador es capaz intuir y separas las diferentes zonas de la imagen pero no de construir correctamente los valores reales de los píxeles. Además, produce fuertes artefactos en el borde de la imagen que son zonas propensas a este tipo de sucesos. Estos resultados se obtienen después de 2 horas de entrenamiento y 29750 épocas. Los pobres resultados, junto con que se está utilizando una arquitectura y un modelo VGG19 pensado para otro tipo de imágenes, son una muestra clara de que es necesario plantear el problema de una forma particular para el conjunto de datos con el que tratamos.

5.4.3. SR-GAN modificada

La primera de las modificaciones consiste en eliminar el factor VGG19 de la GAN, debido a que se utiliza para minimizar el error en el espacio de características de las que se construyen en G respecto de las pre-entrenadas con el dataset *ImageNet*. La diferencia entre ambos conjuntos de características no tiene porqué ser similar, ya que el origen de los conjuntos de imágenes es totalmente distinto. Por tanto, el problema de minimización del error en las características se transforma de nuevo a un problema de reconstrucción de imagen puro.

El segundo factor refiere a los *inputs* que se pasan a G. Como se explicaba en el capítulo 4, aunque teóricamente es mejor tratar las imágenes de manera independiente e introducirlas en G para que se realicen las combinaciones óptimas, esta tarea puede ser muy compleja y requerir una optimización de hiperparámetros muy costosa. Por ello, en vez de introducir las imágenes como bandas de una imagen única, se realiza un preprocesado previo de estas y se introducen en G. Dicho preprocesado es una transformación lineal

$$f : X \in \mathbb{M}_N \longrightarrow F(X) \in \mathbb{M} \tag{5.15}$$

donde $\mathbb{M}_N$ es el espacio de las N matrices LR que hacen referencia a cada imagen HR y $\mathbb{M}$ es un espacio matricial de las imágenes únicas $\hat{LR}$ formadas por $F(X)$ que se resume en aplicar la media o la mediana a cada píxel de las imágenes LR para reducir las N matrices a una sola. En la Figura 5.13 se observa una muestra de imágenes LR bastante limpia de artefactos, a partir de la cual se obtendría una imagen SR. A pesar de que las imágenes LR están bastante claras, la imagen superior izquierda no lo es y esto puede ocurrir con relativa frecuencia. En la parte inferior izquierda de la imagen podemos observar artefactos producidos por las nubes. También se observa un tono general mucho más oscuro que en el resto de las imágenes. Este tipo de defectos en las imágenes no tienen por que ser tan exagerados, pueden ocurrir de maneras más locales como se observa en el borde derecho de la imagen superior derecha de la Figura 5.13.

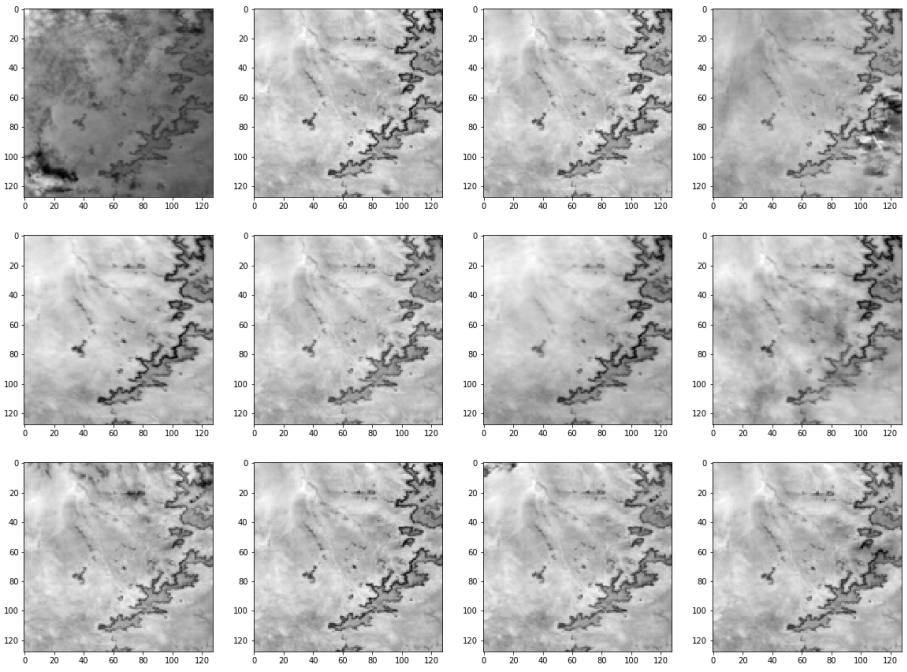

Figura 5.13: Conjunto de imágenes LR

Cabe decir que no todas las imágenes se introducen a la red, previamente a la construcción de $\hat{LR}$, se realiza una selección de las imágenes suficientemente nítidas, es decir, todas aquellas imágenes cuya máscara no tenga más de un 50 % de píxeles claros no se utilizará para el computo de la imagen final. Como hemos visto pueden existir imágenes defectuosas que no aportan información ninguna de la escena HR, e incluso pueden empeorar los resultados. En la Figura 5.14 se muestran los ejemplos de posibles transformaciones.

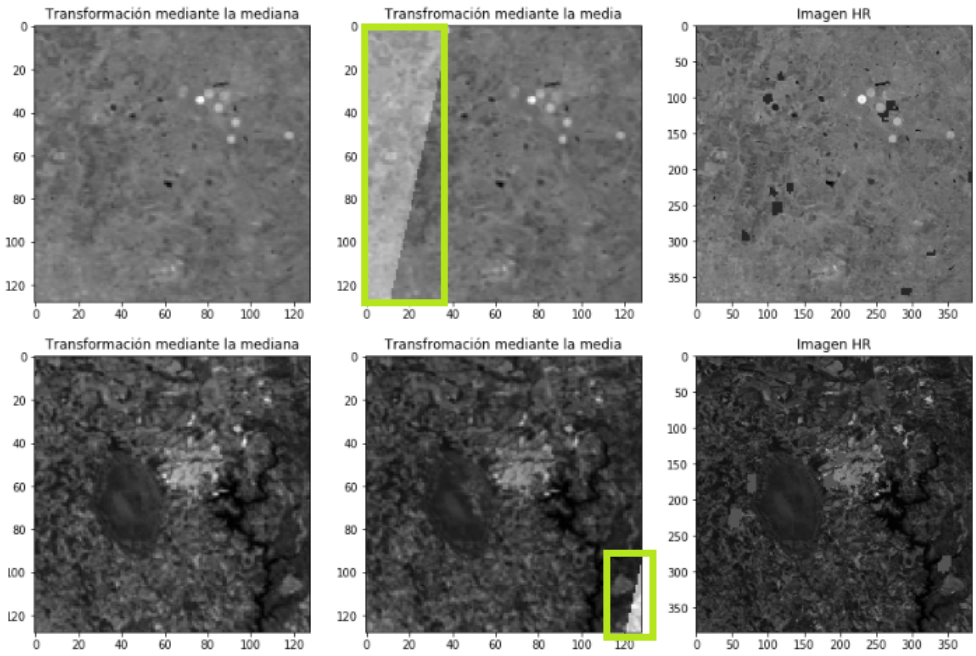

Figura 5.14: Ejemplos de transformaciones de imágenes LR. La fila superior e inferior son dos ejemplos diferentes. La columna de la izquierda es la transformación mediante la mediana. La columna central es la transformación mediante la media y la columna izquierda es la imagen HR.

Así pues, la Figura 5.14 ilustra la razón por la que se ha seleccionado la mediana en vez de la media. Debido a valores extremos en los píxeles, la utilización de la media puede reproducir los artefactos provocados por las nubes, mientras que la mediana evita la reconstrucción de estos valores. Esta es una de las principales necesidades del problema ya que que la reconstrucción de las nubes o efectos climáticos no es algo necesario para la monitorización de la vegetación terrestre. Es apropiado indicar que a pesar de la mejora en el entrenamiento esta aproximación no es correcta en todos sus sentidos. Esto se debe a que la toma de cada una de las escenas no se realiza en el mismo instante temporal por lo que existe un desplazamiento entre nuestras imágenes provocando que los píxeles coincidentes no hagan referencia exactamente a la misma localización espacial. Sin embargo, esperamos que G sea capaz de aprender esto y reconstruir correctamente la única HR a partir de $\hat{LR}$ y solucione el posible ruido o la asimetría que se mostraba en las Figuras 5.7 y 5.8.

La tercera modificación realizada se encuentra en el entrenamiento. Para evitar soportes disjuntos al principio del proceso se ha introducido el mismo planteamiento que se utiliza para entrenar los modelos de *reinforcement learning* como los *Q-learning*. Se introduce un ruido ε sobre las imágenes reales y falsas de manera que $\varepsilon \to 0$ cuando $epoch \to \infty$. De está manera podemos asegurar que tenemos la segunda situación que se muestra en la Figura 4.6.

La cuarta modificación se encuentra en la función de pérdida de G. Debido a que en el entrenamiento buscamos realizar una minimización del error, no desarrollo el cPSNR como función de pérdida, aunque cambiando el signo de esta podría utilizarla. Se ha creado como función de pérdida el cMSE que es el paso previo al cPSNR como se muestra en la ecuación (5.4). Esta modificación nos alinea más con el objetivo de la super-resolución de manera que la red tiene que centrarse principalmente en las zonas

sin efectos climáticos como, por ejemplo, las nubes.

Resultados del modelo modificado

Implementadas las modificaciones mencionadas, los resultados del modelo de SR mejoran significativamente y aunque el modelo bate ligeramente los resultados obtenidos por la interpolación bicúbica en el conjunto de test formado por 200 imágenes HR no se puede afirmar que esta mejora sobre el *benchmark* sea robusta debido a que es mínima. En la Figura 5.15 se muestran los resultados utilizando la métrica cPSNR para el conjunto de test. Observamos cómo los resultados de las predicciones dadas por el generador de la SRGAN modificada, se distribuyen como una curva normal con una media ligeramente superior a la interpolación bicúbica con medias de 45,37 y 44,26 respectivamente. Además, cuando graficamos los resultados de la interpolación frente a los resultados de las predicciones de G, observamos cómo la mayor parte de las imágenes super-resueltas obtienen mejores resultados mediante el uso de G lo que es un resultado muy alentador.

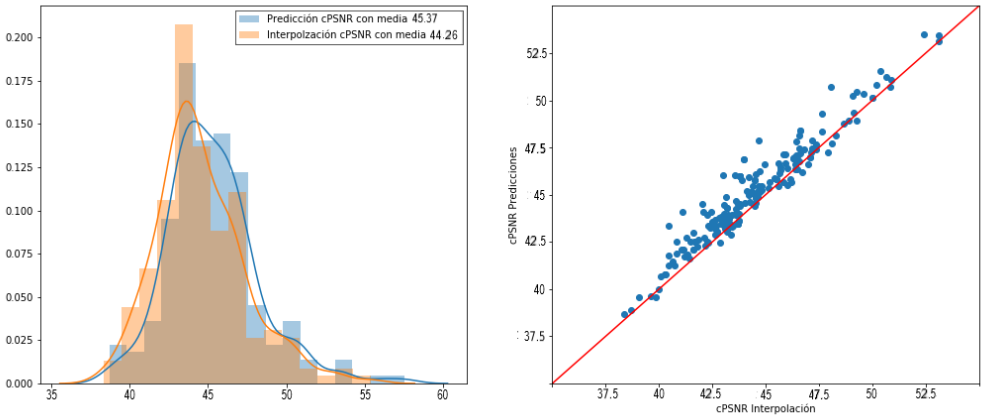

Figura 5.15: Resultados sobre el conjunto de test SR-GAN modificada vs interpolación bicúbica.

En las Figuras 5.15 y 5.17 se muestran algunos resultados finales utilizando la interpolación bicúbica y el G resultante después del proceso de entrenamiento. Observamos en la columna de la izquierda la interpolación bicúbica y su medida de error, en la columna central las predicciones realizadas por el generador desarrollado y entrenado hasta la época 548 y por último en la columna de la derecha se presentan las imágenes HR junto a su máscara. La idea de presentar una HR junto a su máscara es que el lector observe las zonas donde se han producido artefactos debido a las nubes o efectos climáticos. Estos artefactos no han de ser reconstruidos en las imágenes SR y no cuentan para el cálculo de la métrica como se explica en la sección 5.3.

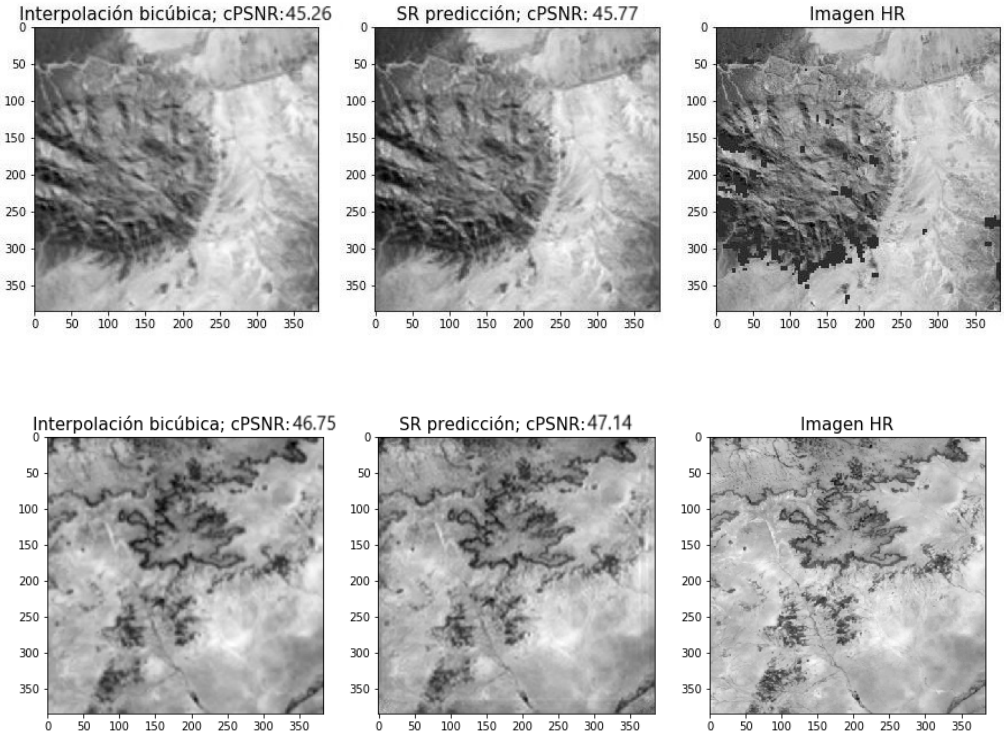

Figura 5.16: Ejemplo de interpolación bicúbica vs SR vs HR.

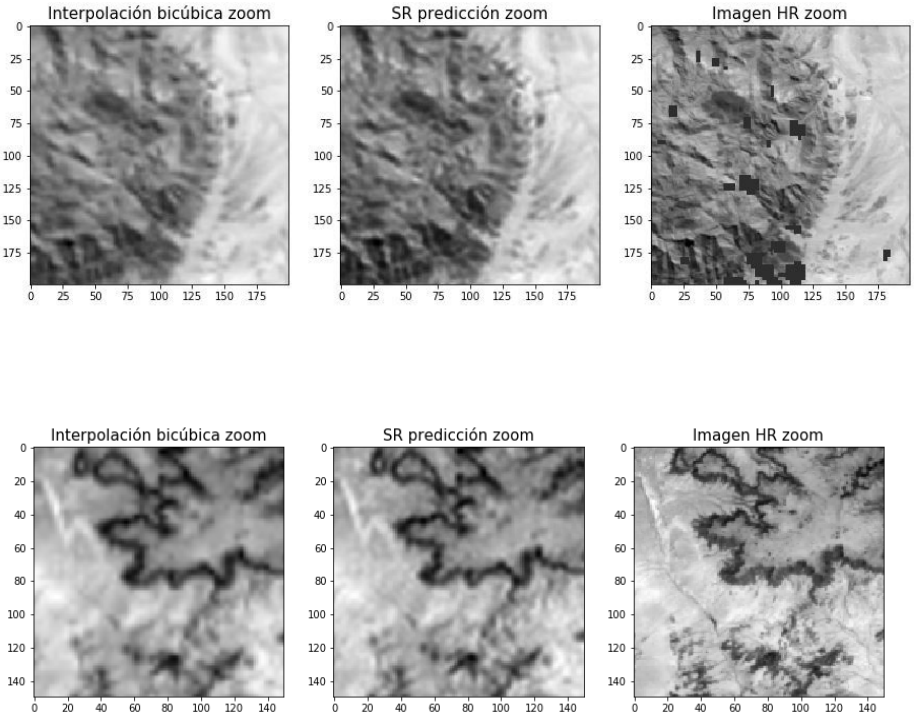

Figura 5.17: Ejemplo zoom de interpolación bicúbica vs SR vs HR.

A pesar de que la mejora no sea grande respecto al *benchmark*, el planteamiento de resolución del problema sí que presenta muy buenas propiedades. Sobre todo en lo que respecta a la evolución en el aprendizaje de la red como muestra la Figura 5.18.

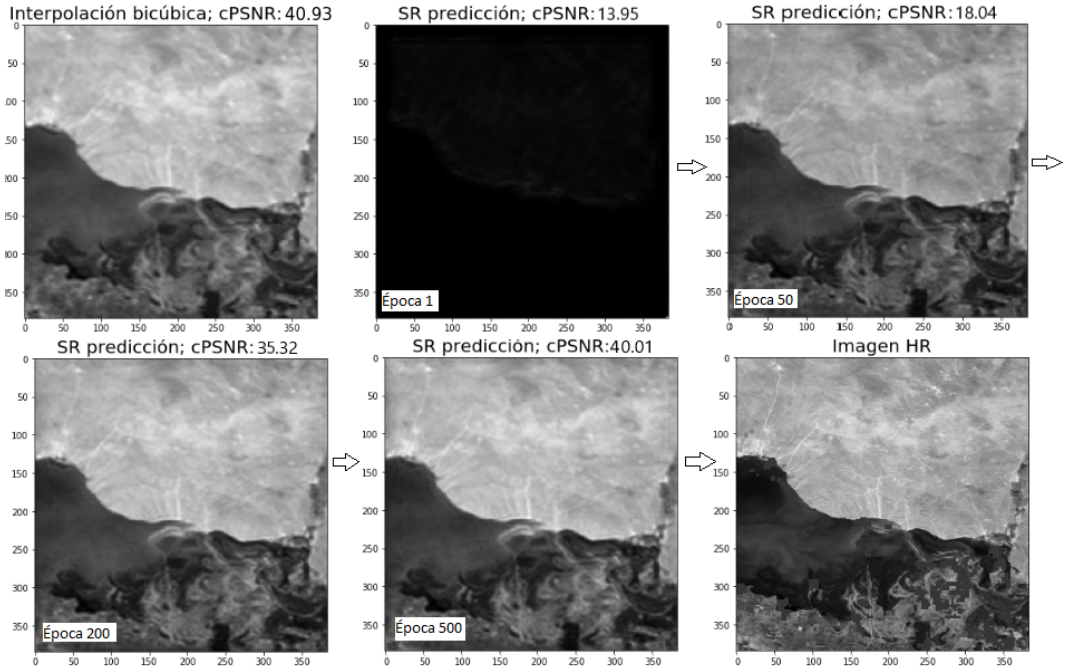

Figura 5.18: Evolución de los resultados de la red con el entrenamiento.

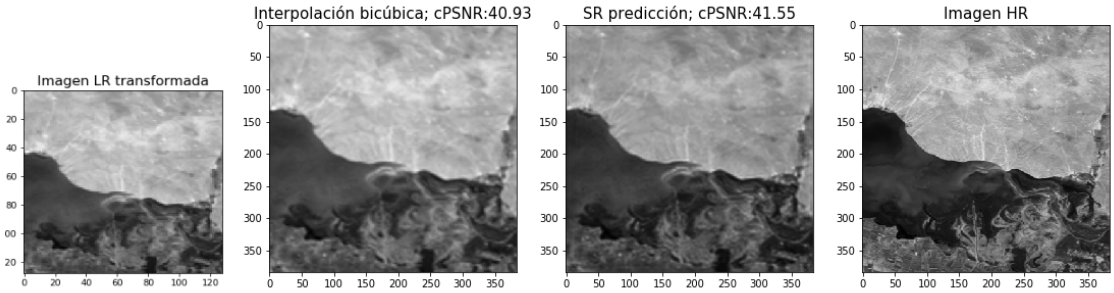

Figura 5.19: Resultados finales de la Figura 5.18 junto con la imagen *input* LR

Esta mejora constante a medida que avanza el entrenamiento nos permite intuir que con más tiempo de entrenamiento los resultados podrían ser significativamente mejores a los resultados de la interpolación bicúbica. En la Tabla 5.4 podemos ver el *summary* de la GAN construida.

Características	Generador	Discriminador
Parámetros entrenables	1,394,305	76,672,033
Parámetros no entrenables	2,176	1,856
Número de bloques residuales	16	7
Optimizador	Adam	Adam
Optimizador learning rate	1e-4	1e-3
Número de filtros	64	64 y 128
Loss	cMSE	Binary crossentropy

Cuadro 5.4: Summary SRGAN modificada

En el caso de este proyecto, el entrenamiento de la red no ha continuado por un factor meramente económico. Los tiempos de entrenamiento se muestran en la Tabla 5.5, los cuales no parecen demasiado grandes. Sin embargo, dichos tiempos corresponden a cada iteración del proceso. Dependiendo del *batch* fijado puede transformarse en horas, días o incluso meses si el *batch* es muy pequeño o la GPU muy poco potente.

Operación	Tiempo
Carga de datos	0.41s
Genera imagen SR	0.23s
Entrenamiento D	0.14s
Entrenamiento GAN	0.85s
Guardado de resultados	1.02s
Total x iteración	1.63s + 1.02s

Cuadro 5.5: Tabla de tiempos por segmento del entrenamiento.

Teniendo en cuenta las características de nuestra VM el modelo está entrenado con un $batch_{size} = 10$ que es el *batch* más grande que la VM podía alojar en memoria durante el proceso. Esto da como resultado un tiempo de alrededor de 3 minutos por época lo que equivale a unas 5 horas cada 100 épocas y teniendo en cuenta que modelo ha sido entrenado durante 554 épocas se tiene alrededor de 32 horas de entrenamiento. En la Figura 5.20 se muestra un ejemplo del proceso en funcionamiento. La ventana superior izquierda es el proceso corriendo en el terminal SSH sacando por pantalla los tiempos y resultados iteración a iteración. La imagen de la derecha es el volumen generado para que el proceso escriba los resultados que va obteniendo en cada época junto con imágenes ejemplo de los resultados de la época. Finalmente en la parte inferior derecha una imagen de la facturación del uso de la VM.

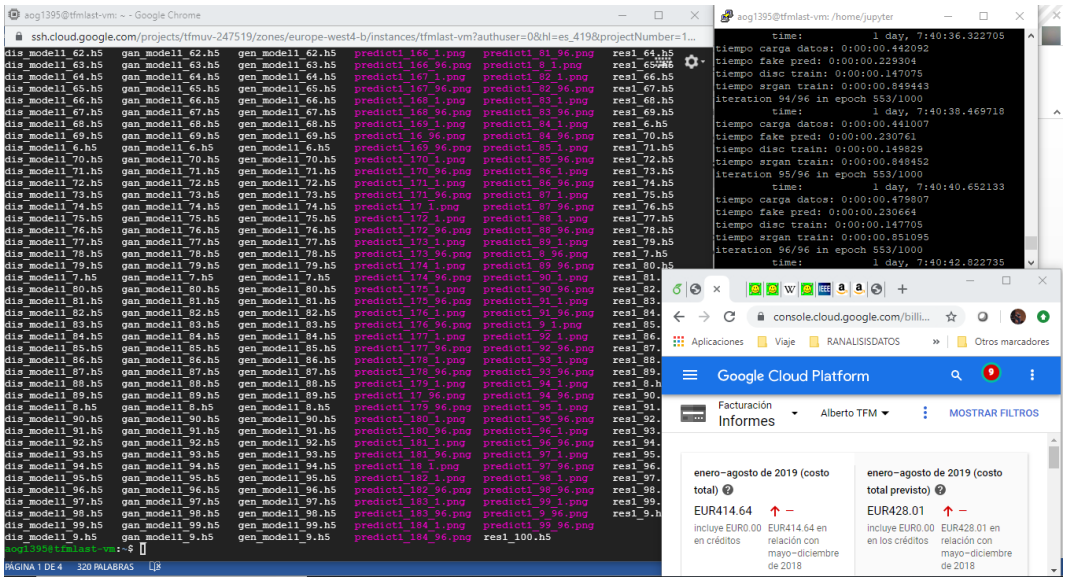

Figura 5.20: Proceso, resultados intermedios y facturación de la VM.

5.4.4. SRGAN Modificado Multi-Imágenes no Registradas

Teniendo en cuenta los resultados obtenidos en la subsección 5.4.3 se pueden tomar dos caminos para batirlos. El primero de ellos es bastante sencillo, continuar entrenando la red hasta que alcance el óptimo o comience a realizar *overfitting*. En tal caso, simplemente nos quedaríamos con el óptimo alcanzado por la red. El segundo de los caminos es un poco más interesante. Hasta aquí he introducido y aplicado algunas de las modificaciones y mejoras desarrolladas e implementadas en la bibliografía del trabajo. Esto ha permitido a la GAN alcanzar unos resultados batiendo el umbral fijado. Sin embargo, todavía queda un punto clave a desarrollar, el input de la red. En 5.4.3 se realiza una transformación linear de los *inputs* de manera que el problema queda simplificado pero también puede perderse una cantidad de información considerable. Por ello, el segundo camino para mejorar los resultados pasa por introducir al generador las imágenes LR de manera independiente.

Así pues, la arquitectura que se presenta como modificación final en este trabajo es la que se muestra en la Figura 5.21

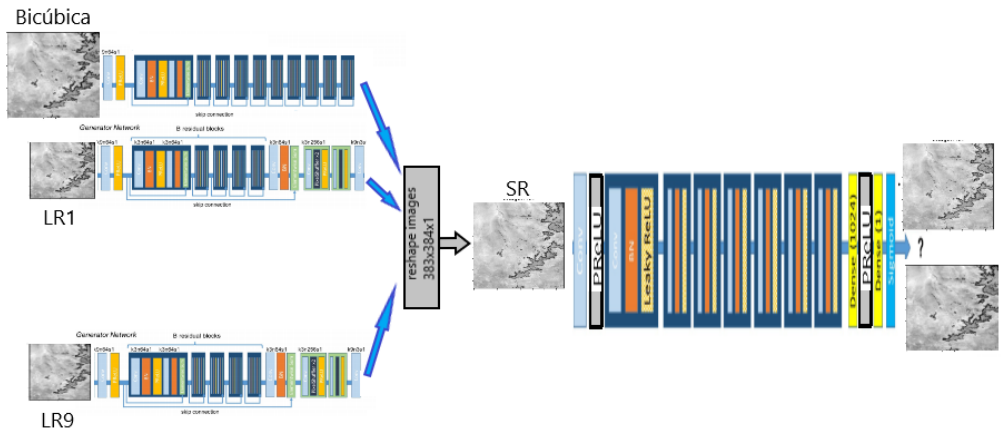

Figura 5.21: SRGAN modificada para tratar las imágenes de manera independiente

La idea de esta arquitectura es que al realizar la tarea de super-resolución también realice la tarea de preprocesado que ya he llevado a cabo en la sección 5.4.3. Es decir, en caso de que la unión de las imágenes LR sea la mediana como se hacía anteriormente se espera que la red replique esa transformación. Sin embargo, si la combinación óptima de las imágenes LR no es esa, cabría esperar que la red combinara las LR de la mejor forma posible. Además, se introduce la imagen SR bicúbica de las LR combinadas por mediana como guía para el generador, G. Nótese, que los *inputs* LR mantienen la arquitectura de bloques residuales y *skip conections* mientras que el *input* SR bicúbico se introduce a la red ya sea directamente o por una rama con una arquitectura diferente, ya que no es necesario, en su caso, realizar un re–escalamiento a 384×384.

Dicha arquitectura cuenta con el precedente ganador de [2]. Aunque dicho precedente tenga una arquitectura diferente y no se trate de una GAN sino de una CNN, la manera de tratar los ha sido la inspiración para el desarrollo del generador de la SRGAN modificado multi-imágenes no registradas (SRGAN-MMnR). Existen múltiples cambios obviamente de G respecto a DeepSUM [27] donde introducen a la red cada una de las imágenes LR interpoladas bicúbicamente. Esta técnica se presenta en [28] con el objetivo de mejorar la velocidad y los resultados de las redes CNN.

Estas mismas consideraciones son aplicables a G. Sin embargo, la construcción base de G utiliza directamente las imágenes LR con el objetivo de que la red se encargue de realizar las operaciones necesarias. En el caso de 5.4.3 no tenía ningún sentido aplicar [28] ya que solo se tiene una imagen. La pérdida de información al aplicar la interpolación bicúbica sin tener el resto de imágenes LR guardando la información complementaria era muy grande. En la Tabla 5.6 se muestra el summary básico del modelo SRGAN-MMnR.

Características	Generador	Discriminador
Parámetros entrenables	12.688.175	76.672.033
Parámetros no entrenables	19.584	1.856
Número de bloques residuales	16×9	7
Optimizador	Adam	Adam
Optimizador learning rate	1e-4	1e-3
Número de filtros	64×9	6 y 128
Loss	cMSE	Binary crossentropy

Cuadro 5.6: Summary SRGAN-MMnR

El SRGAN-MMnR no ha sido entrenado y no se han evaluado sus resultados, como se comentaba en 5.4.3 debido a temas de presupuesto ya que esta arquitectura tiene como mínimo 9 veces más parámetros que la SRGAN modificada y el coste de entrenarla en *Google Cloud* es demasiado. Sin embargo, se han presentado precedentes que pueden indicar que esta arquitectura podría mejorar incluso los resultados de 5.4.3. Mientras que en 5.4.3 se presenta una arquitectura que permite superar a la interpolación bicúbica ligeramente, ahora se presenta una arquitectura aplicando nuevas mejoras. Como se comentaba teóricamente en 4.3.5, tratar las imágenes de manera independiente permite alcanzar mejores resultados y como se presenta en [28] introducir filtros de interpolación permite aumentar la velocidad y puede, incluso, aumentar la precisión. Así pues, SRGAN-MMnR trata de combinar ambos precedentes para mejorar los resultados.

5.5. Conclusiones

Este trabajo se ha centrado en el campo de la super-resolución de imágenes motivado por la competición de [2] que ofrecía un contexto real para la aplicación de este tipo de técnicas. Los métodos del trabajo vienen motivados por el auge en la investigación acerca de modelos generativos adversarios en diversos campos. Estos métodos han obtenido cada vez mejores resultados a partir del descubrimiento de mejoras en el entrenamiento y las arquitecturas.

El punto de partida en el desarrollo de los algoritmos de este trabajo es [24] que se presentaba como el estado del arte en la super-resolución de imágenes RGB. Aquí lo hemos aplicado a imágenes hiper-espectrales y su comparación con el método más frecuentemente utilizado, la interpolación bicúbica. Como se ha explicado en este trabajo, la traducción de los modelos entrenados en un tipo de imágenes a otro tipo distinto no tiene porqué ser directa y esto se observa en los resultados presentados en 5.4.2. Como respuesta a estos resultados he presentado un método que, junto con algunas de las mejoras que se desarrollan en la bibliografía:

- *Instance Noise*
- Combinación de los *inputs* previa al entrenamiento
- Diferentes ratios de aprendizaje para G y D dependiendo de su tamaño
- Uso de funciones de activación que reduzcan el problema de los *vanishing gradients*,

reduce la complejidad del problema para, así, alcanzar unos resultados aceptables y mejorando el benchmark ligeramente, la interpolación bicúbica, sin necesidad de un trabajo de optimización de arquitectura e hiperparámetros costoso 5.4.3.

Los resultados obtenidos presentan a las GAN como una herramienta lo suficientemente potente como para aplicarse a los problemas de super-resolución. Además, de los desafíos presentados en el capítulo 3 (los que son responsabilidad del algoritmo), la SRGAN modificada ha obtenido una robustez considerable distribuyéndose de manera similar a una normal, sin *outliers*. Por otro lado, no parece que la SRGAN modificada haya alcanzado su límite de eficiencia en el entrenamiento y las modificaciones presentadas podrían mejorar los resultados todavía más. Por otro lado, aunque en [29] indican que las GANs pueden no ser la mejor herramienta para el desarrollo de este tipo de aplicaciones porque pueden generar artefactos que nunca han estado antes en la imagen, en este trabajo he mostrado que con una correcta arquitectura y entrenamiento se podría llegar a evitar la generación de artefactos generados por G y que las imágenes SR sean fieles a su imagen HR.

Por tanto, se concluye que el desarrollo de GANs para la super-resolución de imágenes hiper-espectrales parece un camino viable y que puede llevar a la obtención de resultados que compitan o superen a los métodos clásicos. Además con un trabajo de optimización y depuración de los métodos presentados puede superar a arquitecturas más habituales en el *deep learning* como son las redes convolucionales.

Bibliografía

[1] Ken Cochrane, Jeeva S. Chelladhurai, and Neependra K. Khare. *Docker Cookbook - Second Edition*. Packt, 2019.

[2] ESA (European Space Agency). PROBA–V challenge: https://kelvins.esa.int/.

[3] R. Y. Tsai and T. S. Huang. Multiple frame image restoration and registration. *In Advances in Computer Vision and Image Processing*, pages 317–339, 1984.

[4] S. P. Kim, N. K. Bose, and H. M. Recursive reconstruction of high resolution image from noisy undersampled multiframes. *Transactions on Acoustics, Speech and Signal Processing,*, 38:1013–1027, 1990.

[5] Sean Borman and Robert L. Stevenson. Super-resolution from image sequences. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:374–378, 1998.

[6] S. Baker and T. Kanade. Limits on super-resolution and how to break them. *In Proceedings of the 1998 Midwest Symposium on Circuits and Systems*, pages 374–378, 2002.

[7] Jianchao Yang and Thomas Huang. *Image super-resolution: Historical overview and future challenges*. University of Illinois.

[8] S. Farsiu, D. Robinson, M. Elad, and P. Milanfar. Fast and robust multi-frame super-resolution. *IEEE Transaction on Image Processing*, 13:1327–1344, 2004.

[9] Jianchao Yang and Thomas Huang. Image super-resolution: Historical overview and future challenges. *University of Illinois at Urbana-Champaign*, 2010.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[11] Ian Goodfellow *et al. Generative Adversarial Nets.* NIPS, *2014.*

[12] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. ICLR, *2016.*

[13] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. International Conference on Learning Representations (ICLR), *2015.*

[14] C. Szegedy et al. Going deeper with convolutions. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), *pages 1–9, 2015.*

[15] *K. He, X. Zhang, S. Ren, , and J. Sun. Deep residual learning for image recognition.* IEEE Conference on Computer Vision and Pattern Recognition (CVPR), *pages 630–645 (Springer), 2016.*

[16] *K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks.* European Conference on Computer Vision (ECCV), *pages 770–778, 2016.*

[17] *Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis.* ICLR 2019, *2019.*

[18] *Tim Salimans et al. Improved techniques for training gans.*

[19] *Mario Lucic et al. Are GANs Created Equal? A Large-Scale Study.* NIPS, *2018.*

[20] *Rafael Valle.* Hands-On Generative Adversarial Networks with Keras. *Packt, 2019.*

[21] *Alberto Oteo García (Github). https://github.com/oteo95, 2019.*

[22] *PROBA-V product webpage. https://www.vito-eodata.be/pdf/portal/ application.html, 2016.*

[23] *N. Pettorelli, J. O. Vik, A. Mysterud, J. M. Gaillard, C. J. Tucker, and N. C. Stenseth. Using the satellite-derived NDVI to assess ecological responses to environmental change.* Trends in Ecology & Evolution, *pages 503–510, 2005.*

[24] *Christian Ledig et al. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network.* arXiv, *2017.*

[25] *August Odena, Vincent Dumoulin, and Chirs Olah. Deconvolution and checkerboard artifacts, 2016.*

[26] *J. Gauthier. Class Project for Stanford CS231N: Convolutional Neural Networks for Visual Recognition. 2014.*

[27] *Andrea Bordone MoliniDiego Valsesiav Giulia Fracastoroand Enrico Magli. Deepsum: Deep neural network for super-resolution of unregistered multitemporal images.*

[28] *C. DongC. C. Loy K. He and X. Tang. Image super-resolution using deep convolutional networks.*

[29] *D.Krzic A.et al. Märtens, M. Izzo. Super-resolution of proba-v images using convolutional neural networks.*