Facultad de Ciencias

Generación de imágenes simulando distintas condiciones meteorológicas mediante el uso de redes generativas antagónicas (GANs)

Generation of images simulating different meteorological conditions using generative adversarial networks (GANs)

Trabajo de Fin de Máster
para acceder al

Máster en Ciencia de Datos

Autor: Antonio Cuadrado Cobo (cuadradot@predictia.es)

Directores: Lara Lloret Iglesias y Daniel San Martín Segura

Empresa: Predictia Intelligent Data Solutions SL

Junio 2021

Resumen

Las redes generativas antagónicas, comunmente conocidas por su acrónimo en inglés como GANs, son un tipo de modelos generativos de aprendizaje profundo que están formadas por un sistema de dos redes neuronales que compiten mutuamente en un juego de suma cero.

En este trabajo, un parte fundamental será comprender el funcionamiento de este tipo de modelos y sus diferentes variantes entre los que se encuentran la *cycle* GAN, cuya función es aprender a traducir una imagen de un dominio de origen X a un dominio de destino Y en ausencia de ejemplos emparejados.

Este modelo será utilizado para modificar las condiciones meteorológicas de una imagen dada, por ejemplo, transformar una imagen de un día soleado a un día lluvioso o nublado o viceversa.

Palabras clave: aprendizaje automático, red neuronal, discriminador, generador, red generativa antagónica.

Abstract

The generative adversarial networks, commonly known as GANs, are a type of generative models of deep learning that that consists of two neural networks that compete with each other in a zero-sum game.

In this work, a fundamental part will be to understand how these models work and their different variants among which are the *cycle* GAN, whose function is to learn to translate a image from a source domain X to a target domain Y in the absence of unpaired examples.

This model will be used to modify the meteorological conditions of a given image, by for example, transform an image from a sunny day to a rainy or cloudy day or viceversa.

Key words: machine learning, neural network, discriminator, generator, generative adversarial network.

Índice general

Capítulo 1

Introducción

1.1. Motivación

En la última década se ha visto cómo la Tierra está sufriendo una serie de cambios en los patrones climáticos promedio que han llegado a definir los climas locales, regionales y globales. Este tema ha generado gran preocupación en la sociedad ya que según el *IPCC* (Intergovernmental Panel on Climate Change) en [1] el cambio climático en la actualidad se debe en su mayoría (probabilidad mayor del 95 %) al factor humano desde mediados del siglo XX, y la velocidad con la que se está produciendo este cambio es enorme.

Las evidencias de este proceso de cambio son numerosas y destacan especialmente [2]:

- **Aumento de la temperatura global**: el aumento de las emisiones de dióxido de carbono a la atmósfera ha provocado un aumento de 1.18°C desde finales del siglo XIX.
- **Retroceso glacial y cubierta de nieve reducida**.
- **Aumento del nivel del mar**: el nivel de los mares ha aumentado cerca de 20 cm en el último siglo, y cada año este aumento se ve acelerado considerablemente.
- **Aumento de eventos extremos** como inundaciones, incendios...

Sin embargo, estos son los efectos que ya ha producido este cambio climático en nuestro entorno, pero se sabe que continuará a lo largo de este siglo y posteriormente, por lo que las consecuencias a la larga podrían ser catastróficas. Algunas de ellas son una prolongación del aumento de la temperatura terrestre, huracanes más fuertes e intensos, mayor número de sequías y olas de calor, aumento del nivel del mar de hasta 1 metro de altura e incluso que el ártico se quede sin hielo antes de mediados de siglo. [3]

Además, este cambio climático no solo traerá problemas al ecosistema terrestre, sino a nosotros mismos social y económicamente. Estos problemas pueden ser daños en las cosechas y en la producción alimentaria, riesgos en la salud debido a que las enfermedades se transmitirán con mayor facilidad por el aumento de temperatura, pobreza generalizada debido a fenómenos meteorológicos extremos como incendios, danas, huracanes etc. [4]

Por todas estas razones, el objetivo y motivación de este trabajo es explorar la viabilidad de una aplicación con la cual la sociedad sea capaz de comprobar por sí misma los efectos que tendría este cambio climático en su entorno y concienciarla para que la situación se revierta o que al menos, se mantenga en los niveles actuales. Algunos ejemplos podrían ser ver fenómenos meteorológicos extremos en la zona de residencia como inundaciones, incendios, nevadas... o ver cómo cambiarían las estaciones del año debido a la presencia de veranos más largos.

Este enfoque se enmarca dentro de las buenas prácticas en comunicación de cambio climático [5], en las que se señala como un reto el romper la distancia psicológica que se crea al comunicar fenómenos globales como el cambio climático, que son percibidos por el público general como distantes de forma temporal, geográfica y social. Para ello, es necesario mostrar los impactos que el cambio climático está teniendo (y tendrá) en entornos y situaciones cotidianas. En este caso, un primer paso a la hora de poder incorporar elementos de cambio climático en la modificación de imágenes, sería estudiar la posibilidad de una aplicación capaz de modificar el aspecto de las fotografías en función de diferentes condiciones meteorológicas como lluvia, nieve... y posteriormente extenderlo a situaciones más extremas propias del cambio climático como sequías e inundaciones.

Por otro lado este tipo de aplicación no solo podría utilizarse para concienciar sobre el cambio climático sino también para actividades de ocio como puede ser viajar o hacer rutas. Esta otra función consistiría en, teniendo una foto del lugar de destino, transformar esa imagen con el tipo de tiempo (lluvioso, nevado, nublado, soleado...) que vaya a hacer en las fechas elegidas para ver si merece o no la pena hacer el viaje o la ruta.

1.2. Estado del arte

El objetivo de este trabajo es utilizar imágenes para aplicaciones en meteorología ya que es un campo que hasta ahora no ha sido explotado y actualmente tiene gran interés en el campo del aprendizaje automático.

Los modelos que más se están investigando son aquellos construidos para, a partir de una imagen dada, obtener variables meteorológicas como la temperatura, la humedad, la visibilidad... o el tipo de tiempo (soleado o nublado por ejemplo). En 2017 *Wei-Ta Chu et al.* propusieron un modelo en [6] de *Random forest* que, a partir de variables latentes de la imagen como el histograma de color RGB, el histograma de intensidades o el patrón binario local, era capaz de predecir el tipo de tiempo (soleado, nublado, nevado, lluvioso y neblinoso), la temperatura y la humedad de cuando fue tomada la foto, es decir, construyeron un "sensor meteorológico" a través de una cámara. En 2018 *Wei-Ta Chu et al.* en [7] emplearon una red convolucional para predecir la temperatura ambiente en una imagen y una LSTM (*Long-Short Term Memory* para tener en cuenta la evolución temporal y estimar la temperatura de la última imagen de una secuencia. También ese año *L.Kang et al.* propusieron en [8] un modelo de aprendizaje profundo basado en redes neuronales convolucionales (GoogLeNet) capaz de predecir nuevamente el tipo de tiempo de una imagen (neblinoso, lluvioso, nevado y otro). Este tipo de predicciones en imágenes tiene gran utilidad ya que muchas veces los dispositivos visuales de detección como los sistemas de conducción asistida se ven degradados por condiciones meteorológicas adversas (nieve, niebla o lluvia) y es importante detectar el tipo de tiempo de la imagen para posteriormente, aplicar algún tipo de técnica que elimine esos elementos meteorológicos adversos de la imagen (*de-weathering*).

Introducida la parte de detección de condiciones meteorológicas a través de fotos, toca la parte de modificar la imagen dada en función de las condiciones meteorológicas que uno desea. Aquí es donde entra en juego un nuevo tipo de modelo de aprendizaje automático conocido como red generativa antagónica (GAN) propuesto en [9] por Ian Goodfellow *et al.* en 2014. Este modelo generativo está constituido por dos redes neuronales, un clasificador y un generador, y ambos compiten mutuamente en un juego de suma cero. Las aplicaciones que poseen este tipo de modelos es enorme, desde aplicaciones en imágenes como transferencia de estilos, traducción de texto a imagen, generación de imágenes hiperrealistas para aumentar las muestras en datasets e imágenes de alta resolución..., a aplicaciones en

música y video como generación de nuevas muestras musicales o predicción de $frames$ en un video [10].

La redes generativas antagónicas tienen un amplio abanico de variantes, cada una utilizada para distintos tipos de aplicaciones. Los modelos más simples son los modelos *Deep Convolutional* GAN [11] y *Wassertein* GAN [12] que son utilizados para generar nuevas muestras $fake$ (p.e imágenes) a partir de un vector de ruido que es pasado a través de una serie de capas convolucionales y transformado en una imagen. Sobre esta imagen generada no se tiene ningún tipo de control, es decir, las nuevas muestras se generan de manera aleatoria. Un tipo de GAN más avanzado es la *conditional* GAN [13] en la cual la estructura de la red es similar a las mencionadas anteriormente pero donde el vector de ruido es concatenado con un vector *one-hot* para generar muestras de un determinado tipo de clase como pueden ser imágenes de perros de distinta raza. También destaca la *controllable* GAN [14], la cual mediante el ajuste y variación del vector de ruido es capaz de controlar aspectos, detalles y características más concretos de una imagen como hacen en [15], donde controlan la edad, el sexo, el peinado o incluso la pose de una persona.

Uno de los principales desafíos de este tipo de redes generativas es conseguir la generación de imágenes de alta resolución ya que normalmente la calidad de las imágenes suele ser baja (64×64, 128×128). De esta manera surge la *Progressive* GAN, otro tipo de modelo GAN propuesto en 2017 por *Tero Karras et al.* en [16] donde se hace crecer tanto el generador como el discriminador de forma progresiva, es decir, a partir de una resolución baja, se van agregando nuevas capas que modelan detalles cada vez más pequeños a medida que avanza el entrenamiento. Esto hace que el entrenamiento vaya mucho más rápido y que se estabilice, permitiendo producir imágenes de gran calidad. Además de este tipo de GAN para aumentar la resolución de las imágenes generadas por la propia GAN, también existe otra estructura denominada *Super Resolution* GAN [17] que transforma imágenes de baja resolución a imágenes de alta resolución.

Juntando todas estas variantes surge la que hoy se considera el estado del arte de este tipo de modelos generativos, la *Style* GAN, propuesta por *Teo Karras et al.* en [18] y que destaca en su capacidad de generar caras humanas extremadamente realistas.

Además de los modelos citados para generación de nuevas muestras de manera controlada, la redes generativas antagónicas también son utilizadas para transferencia de estilo entre imágenes [19]. Destaca el modelo $Pix2Pix$ propuesto por *Phillip Isola et al.* en [20] donde las entradas y salidas del generador son similares a una *conditional* GAN, excepto por que ahora el vector de ruido es sustituido por una imagen para ser transformada al estilo de destino. Este modelo es necesario que sea entrenado con datos emparejados, es decir, que la imagen esté presente en los dos estilos, el de partida y el de destino. No obstante, el estado del arte de este tipo de modelos de transferencia de estilo es la *cycle* GAN. Esta red generativa antagónica fue presentada por *Jun-Yan Zhu et al.* en [21] y su principal función es aprender a traducir una imagen de un dominio de origen X a un dominio de destino Y en ausencia de ejemplos emparejados, por ejemplo pasar imágenes de verano (soleado) a invierno (nieve). Aquí se ve la gran ventaja de este tipo de GAN frente a $Pix2Pix$ ya que soluciona el problema que se da en determinadas ocasiones por la falta de muestras emparejadas en el entrenamiento.

Introducida la parte de detección de condiciones meteorológicas a través de fotos y el estado del arte de las redes generativas antagónicas, toca la parte de modificar la imagen dada las condiciones meteorológicas que uno desea utilizando este tipo de modelos.

La mayor parte de estos modelos consisten en una *cycle* GAN. Por ejemplo *Praveer Singh et al.* propusieron este tipo de red en [22] para eliminar nubes de imágenes de satélite.

En [23] *Victor Schmidt et al.* proponen el uso de una *cycle* GAN para simular fenómenos extremos como inundaciones en lugares residenciales con el fin de concienciar también sobre el cambio climático. También en [24] proponen otro modelo para poner niebla en imágenes soleadas.

Sin embargo, aunque en este tipo de aplicaciones predominan los modelos *cycle* GAN también se utilizan otro tipo de redes generativas antagónicas como las *conditional* GAN. Por ejemplo en [25] *He Zhang et al.* proponen este tipo de red la cual es capaz de eliminar la lluvia de imágenes. También *Wei-Ta Chu et al.* propusieron en [26] un modelo similar con el cual generar imágenes sintéticas en diferentes condiciones meteorológicas a través una imagen de referencia y un vector con las condiciones meteorológicas.

Hay que destacar que este tipo de modelos está muy limitado debido a su gran complejidad y que los modelos citados anteriormente se simplifican bastante de manera que se trabaje con un solo tipo de escenario (misma foto en distintas condiciones meteorológicas) y que, por tanto, su capacidad de generalización es muy reducida.

1.3. Objetivos

Una vez establecido el estado del arte y la motivación del proyecto a continuación se detallarán los objetivos del mismo.

El principal objetivo de este trabajo consiste en explorar la viabilidad de una aplicación capaz de transformar las condiciones meteorológicas de una imagen dada. Para ello se han desarrollado en total cuatro modelos *cycle* GAN con los cuales transformar imágenes con un tipo de tiempo nublado, lluvioso, nevado y neblinoso a soleado y viceversa.

No obstante, existen otros objetivos que, aunque no son el principal, son de vital importancia para la consecución del proyecto. Por un lado, adquirir los conocimientos fundamentales de las redes neuronales (funciones de activación, algoritmo *Backpropagation*), los diferentes tipos de capas que estas presentan (capas densas, convolucionales, dropout...), así como optimizadores, funciones de pérdida etc. Todo esto es necesario para posteriormente construir los distintos modelos GAN.

Por otro lado, otra finalidad es entender el funcionamiento de los modelos de redes generativas antagónicas para lo cual se realizará un análisis crítico sobre las limitaciones de los modelos y los resultados obtenidos.

1.4. Organización del trabajo

Este trabajo se ha estructurado en 4 capítulos siendo esta introducción, donde se hablan sobre los objetivos y ambiciones de la memoria y el estado del arte, el primero de ellos.

A continuación, en el segundo capítulo se explican los conceptos esenciales del aprendizaje automático junto con las diferentes estructuras y capas de aprendizaje profundo utilizadas. También se dedica una sección a la explicación teórica del funcionamiento de una red generativa antagónica y se describen las distintas variantes empleadas de este tipo de modelos.

En el capítulo 3 se detallará información relevante sobre los datasets utilizados para el entrenamiento de la red neuronal y se expondrán los resultados obtenidos con los modelos GAN junto con un análisis crítico de los mismos.

Por último, la memoria se concluirá en el cuarto capítulo con las principales conclusiones de este estudio y la dirección que tomará en un futuro.

Capítulo 2

Metodología y técnicas

2.1. Introducción al aprendizaje profundo o *Deep Learning*

El *Deep Learning* o aprendizaje profundo comprende un conjunto de algoritmos de aprendizaje automático basados en el uso de redes neuronales con excelentes resultados en aplicaciones como el reconocimiento de voz, la visión por ordenador y el procesamiento del lenguaje natural.

Las redes neuronales son un sistema de programas y estructuras de datos que se inspiran en el diseño y el funcionamiento del cerebro, que es una colección de neuronas interconectadas.

Al igual que en el cerebro humano, las redes neuronales están compuestas por estructuras más simples, las neuronas o nodos, las cuales a su vez están agrupadas en capas. Cada una de estas neuronas recibe unos datos de entrada con los que realiza una suma ponderada cuyos pesos se ajustan durante el proceso de aprendizaje. Además de esos pesos ajustables la neurona recibe también un sesgo, el cual es ajustable también durante el entrenamiento. Sin embargo, hasta ahora la salida de la neurona se corresponde con una suma de funciones lineales que da lugar a otra función lineal, es decir, sería equivalente a que nuestra red tuviera una única neurona/nodo. Por ello es necesario la utilización de una *función de activación* con la cual captar las características no lineales de los datos.

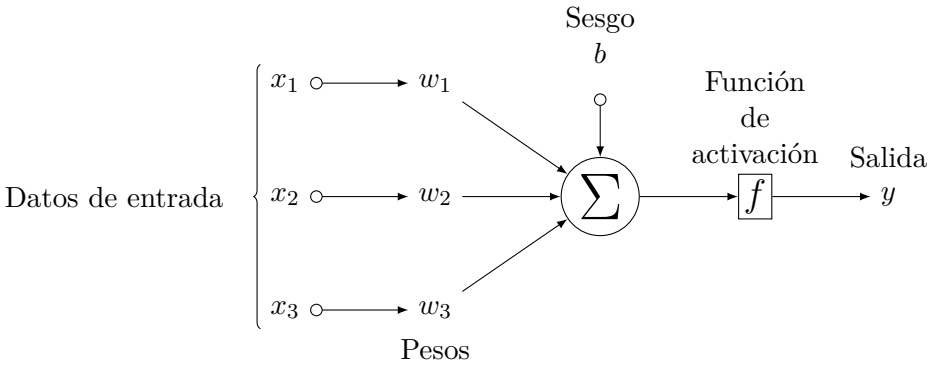


Figura 2.1: Esquema de funcionamiento de una neurona que recibe como input tres valores (x_1, x_2, x_3) los cuales son ponderados mediante los pesos ajustables (w_1, w_2, w_3) a los cuales se añade el término bias (b). Una vez hecha la suma ponderada de los inputs, el resultado se pasa a través de la función de activación (f) obteniendo el output de la neurona (y).

Las funciones de activación son muy variadas y dependiendo del objetivo marcado en el entrenamiento de la red, se hará uso de unas u otras. En este trabajo se han utilizado:

- **Lineal**: es una función lineal que toma las entradas, multiplicadas por los pesos de cada neurona, y crea una señal de salida proporcional a la entrada. (Ec (2.1))

$$f(z) = cz \tag{2.1}$$

- **Sigmoide** [0,1]: se trata de una función en la que la suma ponderada de la entrada de la neurona es pasada a través de una función sigmoide (Ec (2.2)), es decir:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.2}$$

- **Tangente hiperbólica** [-1,1]: se trata de una función en la que la suma ponderada de la entrada de la neurona es pasada a través de una función tangente hiperbólica (Ec (2.3)), es decir:

$$\tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{2.3}$$

- **Relu** (*Rectified linear unit*): es una función no lineal a trozos que toma los valores de entrada y los transforma de tal forma que anula los valores negativos y deja los positivos tal y como entran (Ec (2.4)). Entre sus principales ventajas destacan su fácil implementación, y que soluciona gran parte del problema del desvanecimiento del gradiente presente en las funciones sigmoide y tangente hiperbólica.

$$f(z) = \text{máx}(0, z) \tag{2.4}$$

- **Leaky Relu**: es una variante de la función de activación Relu cuyo funcionamiento se basa en transformar los datos de entrada de tal manera que los valores negativos se multiplican por un coeficiente rectificativo y los valores positivos se dejan tal y como entran. (Ec (2.5))

$$f(z) = \begin{cases} \alpha z & \text{si} \quad z \leq 0 \\ z & \text{si} \quad z \geq 0 \end{cases} \tag{2.5}$$

Como se explicó previamente, la red neuronal está compuesta por capas que a su vez están compuestas por neuronas como la de la Figura (2.1), de tal manera que se formará un sistema como el de la Figura (2.2). En esta figura se ilustra una red multicapa formada por la colocación secuencial de capas densas. La capa densa, el tipo de capa mas sencilla, consiste en una agrupación de neuronas en el que se conectan todos los datos de entrada. También hay otras capas más complejas cuyo detalle se muestra en las siguientes secciones.

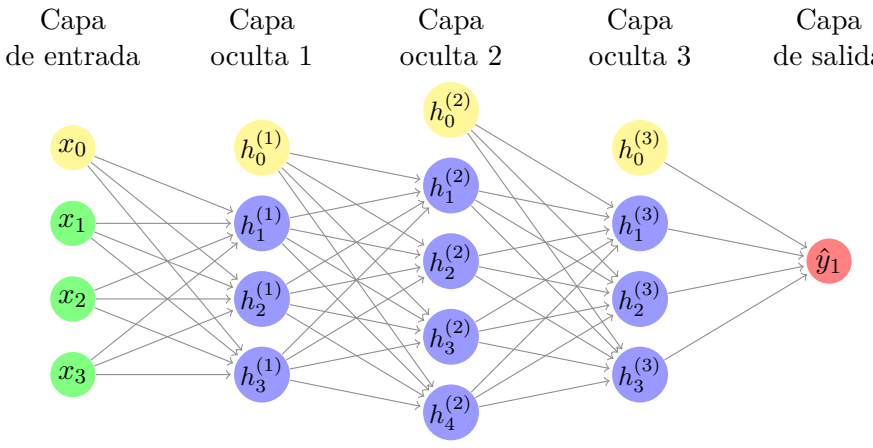


Figura 2.2: Esquema de una red neuronal con tres capas ocultas. Se ha añadido también en cada capa el término bias.

El número de neuronas que contiene una capa es definido al inicio al igual que el método de inicialización de los pesos de la capa. Asimismo, el número de capas que conforma una red neuronal es muy variable y puede verse aumentado enormemente dependiendo de la problemática considerada, es por esto que este tipo de aprendizaje se denomine *profundo*.

El entrenamiento de una red neuronal es un proceso de optimización iterativo donde a cada una de las iteraciones se les denomina época (*epoch*). La red neuronal aprende mediante la optimización de los parámetros ajustables con respecto a la función de pérdida (*"loss function"*). Esta función de pérdida toma las predicciones de la red y el valor real y mide qué tan lejos está la salida de la red del valor real. La red utiliza el algoritmo de propagación hacia atrás o *Backpropagation* para asignar un porción del error anterior a cada parámetro ajustable. A continuación, los pesos de las neuronas se actualizan utilizando un algoritmo de optimización, como puede ser el algoritmo *Adam* (de sus siglas en ingles *Adaptative Moment estimation*) [27].

El algoritmo de *backpropagation* propaga el error de la función de perdida a cada neurona, y después el algoritmo de optimización se ocupa de actualizar los pesos de la red neuronal. Existen una variedad de funciones de pérdida en el aprendizaje automático, pero las que se han utilizado en este trabajo han sido:

- **Entropía cruzada binaria** (BCE): en esta función binaria la pérdida entre la predicción y el valor real se calcula mediante la Ec (2.6).

$$BCE = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log(y_i') + (1 - y_i) \cdot \log(1 - y_i') \tag{2.6}$$

  donde y_i son los valores reales (0 o 1), y_i' los valores predichos por el modelo y N el número total de entradas.

- **Error cuadrático medio** (MSE): mide el promedio de los errores al cuadrado, es decir, la diferencia entre el valor predicho y lel real. (Ec (2.7))

$$MSE = \frac{1}{N}\sum_{i=1}^{N} (y_i' - y_i)^2 \tag{2.7}$$

- **Error absoluto medio** (MAE): mide el promedio de los errores en valor absoluto. (Ec (2.8))

$$MAE = \frac{1}{N}\sum_{i=1}^{N} |y_i' - y_i| \tag{2.8}$$

Una vez calculado el valor de la función de pérdida, este se transmite al optimizador que es el encargado de ajustar los pesos mediante el algoritmo de *Backpropagation*. Este algoritmo es un método de cálculo del descenso del gradiente y como tal, el gradiente obtenido es multiplicado por un escalar conocido como tasa de aprendizaje o *learning rate* para determinar los nuevos pesos de la red. Este hiperparámetro del entrenamiento de la red neuronal es de gran importancia y precisa de bastante tiempo para ajustar ya que la elección de una tasa de aprendizaje muy pequeña, llevará a que el aprendizaje dure demasiado tiempo mientras que una tasa de aprendizaje grande puede hacer que se dé el problema de explosión del gradiente haciendo que el modelo se vuelva inestable.

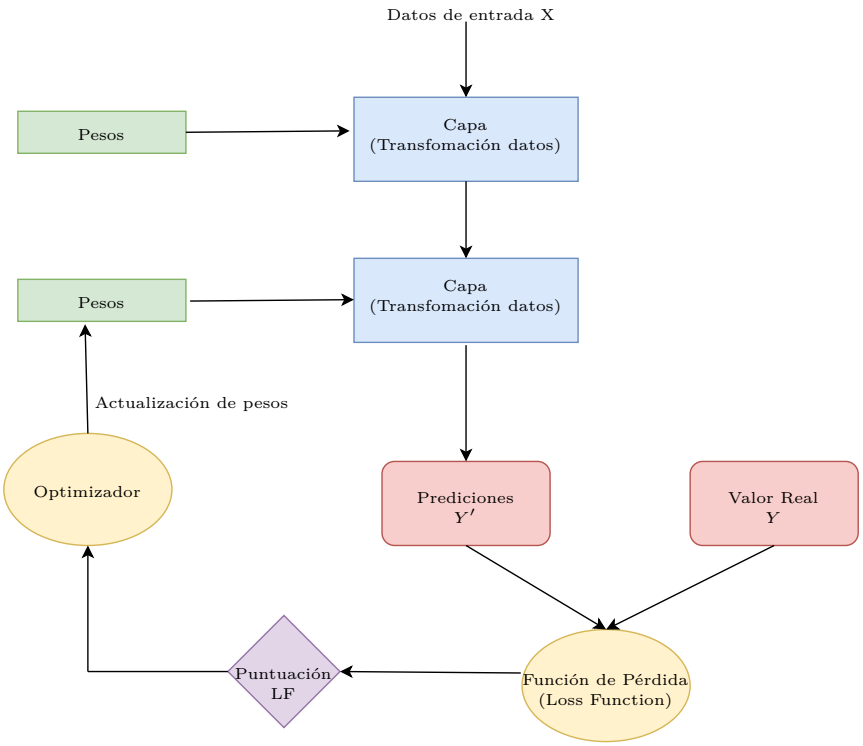


Figura 2.3: Representación esquemática del entrenamiento de una red neuronal.

2.2. Capas

Para la construcción de las redes generativas antagónicas ha sido necesario utilizar distintos tipos de capas. A continuación se hará una breve explicación de cada una de ellas.

2.2.1. Capa convolucional *2D*

Una capa convolucional es un tipo de capa especializada en el procesamiento de imágenes que ha revolucionado el campo de la visión artificial por sus resultados tratando con datos de muy alta dimensionalidad. Estas capas consisten en una serie de M filtros o *kernels* de tamaño $N_w \times N_h$ que recorren la matriz o mapa de características de entrada (ver Figura (2.4)). Su principal ventaja frente a las capas densas o *fully conected* es su capacidad de generalización puesto que estas capas aprenden patrones locales lo que hace que esos patrones aprendidos sean invariantes a translaciones.

Estas capas además del número filtros y su tamaño tienen otros dos hiperparámetros, el relleno o *padding* y el paso o el *stride*.

El *padding* consiste en agregar píxeles de valor cero alrededor del mapa de características de entrada de tal manera que el output del filtro sea una matriz de igual tamaño que la original tras aplicar el filtro. Esta operación se realiza con el objetivo de, por un lado evitar una reducción excesiva del tamaño original de la imagen, y por otro evitar la infrarrepresentación de los píxeles situados en esquinas y bordes en el espacio de características (conocido como efectos de borde).

Por otra parte, el parámetro *stride* indica cuántas posiciones ya sean columnas o filas se moverá el filtro o *kernel* en cada paso.

El tamaño final de una imagen de anchura W, alto H y C canales tras haberle aplicado M filtros de tamaño (N_w, N_h) con un *stride* (S_w, S_h) y *padding* (P_w, P_h) viene dado por

la Ec (2.9):

$$W_n = \frac{W - F_w + 2P}{S_w} + 1; \quad H_n = \frac{H - F_h + 2P}{S_h} + 1; \quad C_n = M \tag{2.9}$$

En la Figura (2.4) se muestra un esquema de cómo funciona un filtro convolucional al aplicarlo sobre una matriz de entrada dado un *padding*, UN *stride* y un tamaño de filtro determinado.

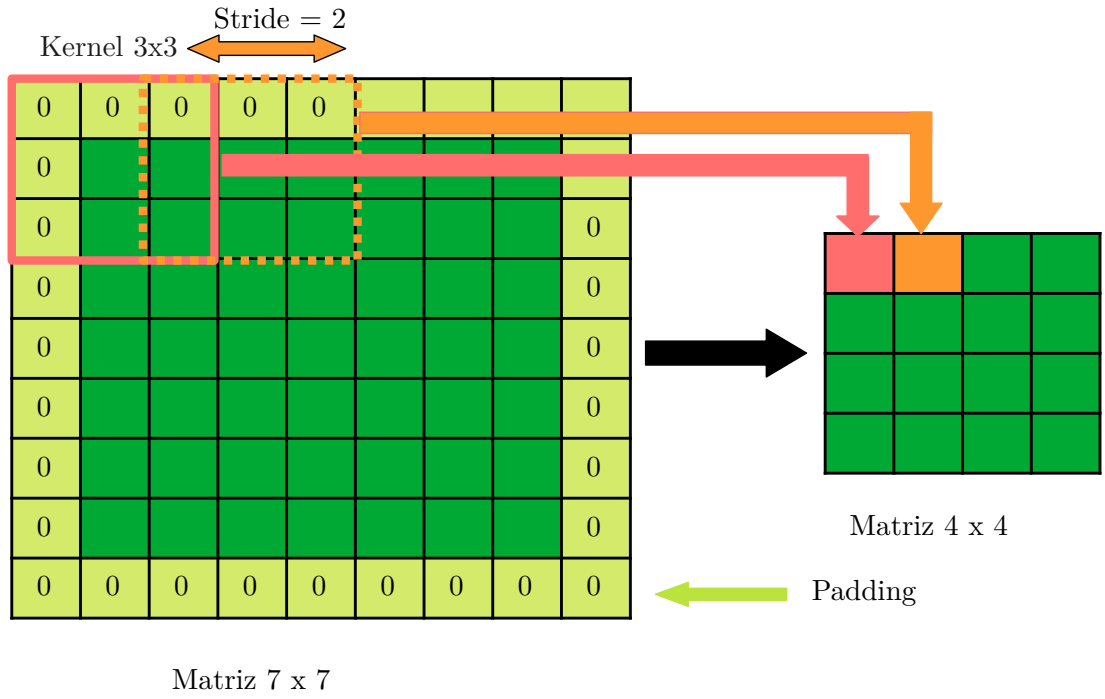


Figura 2.4: Esquema del funcionamiento de un filtro convolucional de tamaño 3×3 al aplicarlo sobre una imagen de tamaño 7×7. Se ha aplicado un *padding* de 1 tanto a lo ancho como a lo alto y un *stride* de 2. Introduciendo estos datos en la Ec (2.4) se obtiene un output de tamaño 4×4.

2.2.2. Capa convolucional transpuesta *2D*

En la siguiente sección se introducirán las redes generativas antagónicas las cuales están formadas por dos redes neuronales, un generador y un discriminador (clasificador). En el caso del generador es necesario introducir un nuevo tipo de capa capaz de incrementar la dimensionalidad de los mapas de características que recibe y así aumentar la resolución.

La manera más sencilla de hacerlo sería a través de una capa de incremento de la dimensionalidad como la utilizada por el paquete *keras* [28], *Upsampling*, la cual utiliza un esquema basado en el vecino más cercano. Sin embargo, este tipo de capa no tiene ningún parámetro ajustable durante el entrenamiento de la red neuronal por lo que se ha utilizado otro tipo de capa de aumento de la dimensionalidad denominada convolucional transpuesta.

El funcionamiento de este tipo de capa es similar a la capa convolucional estándar pero de manera inversa, es decir, genera un mapa de características de salida que tiene una dimensión espacial mayor que la del mapa de características de entrada. Al igual que la capa convolucional estándar, la capa convolucional transpuesta también está definida por

el *padding* y el *stride*, aunque estos hay que entenderlos de una manera diferente. En el ejemplo de la Figura (2.5) se detalla el funcionamiento de este tipo de capa.

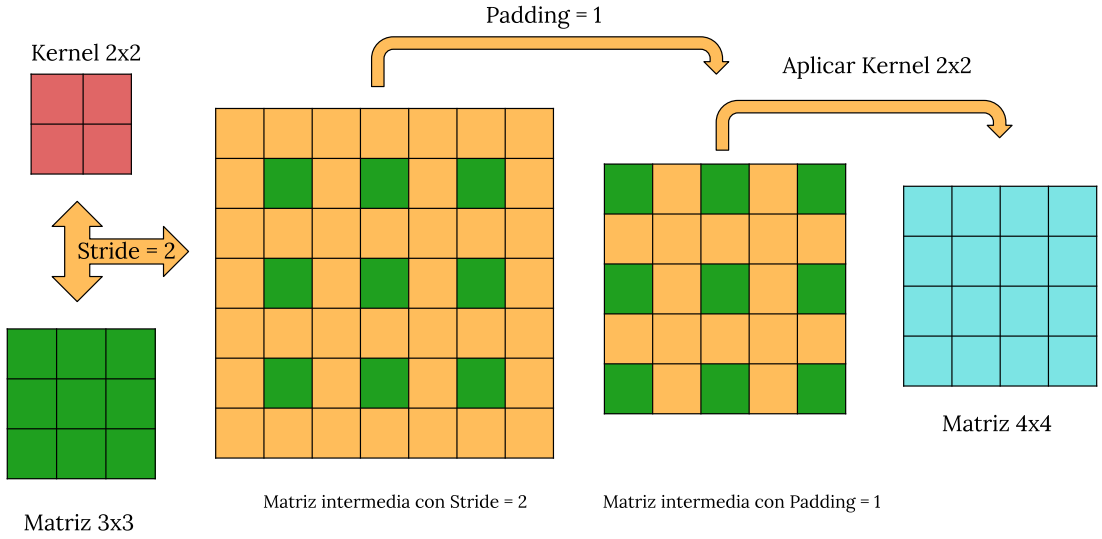


Figura 2.5: Esquema del funcionamiento de un filtro de tamaño 2×2 en una capa convolucional transpuesta al aplicarlo sobre una imagen de tamaño 3×3. Se ha aplicado un *padding* de 1 tanto a lo largo como a lo ancho y un *stride* de 2. Se obtiene un output de tamaño 4×4.

En el ejemplo de la Figura (2.5) donde se aplica un filtro de tamaño 2×2 en una capa convolucional transpuesta a una imagen de tamaño 3×3 con *padding* 1 y *stride* 2 se siguen los siguientes pasos:

1. Se crea una matriz intermedia que contiene las celdas de la entrada original separadas una distancia igual al *stride* utilizado, en este caso 2.

2. A continuación, se añaden celdas adicionales con valor 0 alrededor de las celdas originales hasta que el kernel en la parte superior izquierda cubra una de las celdas originales.

3. Lo siguiente será construir otra matriz intermedia en la que se aplique el *padding*. En este tipo de capas este parámetro no se utiliza para aumentar el tamaño del mapa de características sino para reducirlo, por tanto, un *padding* de 1 eliminará de la anterior matriz sus filas y columnas externas.

4. Por último, una vez obtenida esa matriz intermedia se le aplicará el filtro de tamaño 2x2 con un paso de 1 obteniendo una matriz final de tamaño 4x4 en el ejemplo de la Figura (2.5). Hay que recalcar que en este tipo de capa el *stride* no se utiliza para ver cuántas celdas se va moviendo el kernel en la imagen intermedia obtenida.

Al igual que en la capa convolucional es posible conocer el tamaño S_n del mapa de características final mediante la Ec (2.10) para el caso de una imagen cuadrada ($S \times S$) con *padding* P y *stride* S.

$$S = (W - 1) \cdot S - 2 \cdot P + (N - 1) + 1 \tag{2.10}$$

Uno de los principales problemas que tienen este tipo de capas es la aparición de artefactos en la imagen con forma de tablero de ajedrez (*checkboard pattern*) [29]. Esto se debe a que

con este método algunos píxeles de la imagen son "visitados" por el filtro más veces que otros, como es el caso de los píxeles en el centro frente a los que los rodean.

2.2.3. Capas de normalización

Los datos de entrada al modelo siempre es conveniente normalizarlos ya que de esta manera todas las variables predictoras tendrán magnitudes comparables pues, si no, cuando se propaguen los errores puede que la mayoría del error se asocie a las variables de una mayor magnitud.

Cada vez que se actualizan los pesos la distribución de los datos de entrada de cada capa cambia ya que se han actualizado los parámetros de las capas anteriores. Esta diferencia en las distribuciones de los datos de entrada se denomina *Internal Covariance Shift* y fue introducida en Ioffe y Szegedy en [30].

Para solucionar este problema existen distintos tipos de capas de normalización, aunque aquí se han utilizado únicamente la normalización por lotes o *batch normalization* e *instance normalization*. Ambas capas realizan un escalado y una traslación de los datos utilizando la media μ y la desviación estándar σ de los mismos (Ec (2.11)), pero difieren en la manera en que se calculan esos estadísticos.

$$\hat{x} = \frac{x - \mu}{\sigma} \tag{2.11}$$

Para explicar ambos tipos de capa se va a considerar un conjunto de mapas de características con dimensiones (N, C, H, W) donde N es el tamaño del lote, C el número de canales y H y W las dimensiones de ancho y alto del mapa de características.

Normalización por lotes

En este tipo de normalización, la media y la varianza se calculan para cada canal individual en todas las muestras y en ambas dimensiones espaciales como se muestra en la Ec (2.12) y en la Figura (2.6).

$$\mu_c = \frac{1}{NHW}\sum_{i=1}^{N}\sum_{j=1}^{H}\sum_{k=1}^{W} x_{icjk}; \quad \sigma_c^2 = \frac{1}{NHW}\sum_{i=1}^{N}\sum_{j=1}^{H}\sum_{k=1}^{W}(x_{icjk} - \mu_c)^2 \tag{2.12}$$

$$\hat{x} = \frac{x - \mu_c}{\sqrt{\sigma_c^2 + \epsilon}}$$

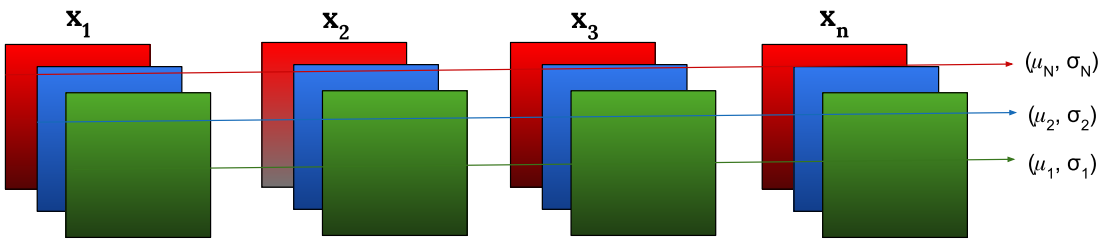


Figura 2.6: Ilustración de cómo calcular la media y varianza en un conjunto de N mapas de características mediante la normalización por lotes.

Instance normalization

En esta capa, la media y la varianza se calculan para cada canal en cada una de las muestras en ambas dimensiones espaciales como se ve en la Ec (2.13) y en la Figura (2.7).

$$\mu_{nc} = \frac{1}{HW}\sum_{j=1}^{H}\sum_{k=1}^{W} x_{ncjk}; \quad \sigma_{nc}^2 = \frac{1}{HW}\sum_{j=1}^{H}\sum_{k=1}^{W}(x_{ncjk} - \mu_{nc})^2 \tag{2.13}$$

$$\hat{x} = \frac{x - \mu_{nc}}{\sqrt{\sigma_{nc}^2 + \epsilon}}$$

Este tipo de normalización podría entenderse como un caso particular de la normalización por lotes donde el tamaño del lote fuese uno.

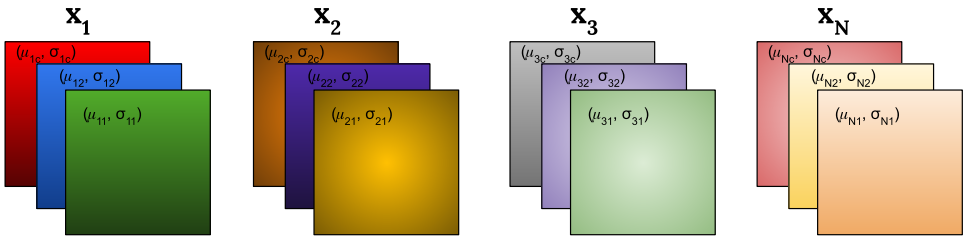


Figura 2.7: Ilustración de cómo calcular la media y varianza en un conjunto de N mapas de características mediante *instance* normalization.

Finalmente tanto en la normalización por lotes como en *instance* normalization se produce un escalado y traslación de los datos obtenidos con dos parámetros γ y β que son aprendidos a lo largo del entrenamiento de la red neuronal. De esta manera el resultado quedaría final de esta capa quedaría según la Ec (2.14).

$$y = \gamma\hat{x} + \beta \tag{2.14}$$

2.2.4. Bloques residuales

Un bloque residual no es una capa en sí misma sino un conjunto de capas configuradas de tal manera que la salida de una capa se toma y se concatena con otra capa más profunda en el bloque mediante una conexión de derivación denominada conexión de salto. De esta manera, se garantiza que las propiedades de entrada de capas anteriores también estén disponibles para capas posteriores, de modo que su salida no se desvíe mucho de la entrada original. Para el caso estudiado, la estructura de un bloque residual es la mostrada en la Figura (2.8), es decir, está formado por dos capas convolucionales y de *Instance* normalization con activaciones Relu.

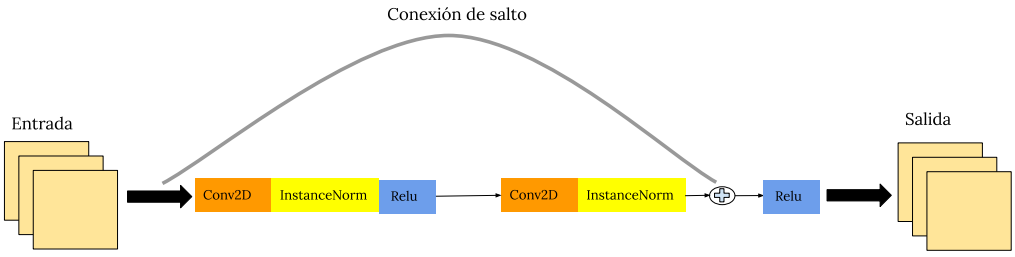


Figura 2.8: Esquema del bloque residual utilizado.

2.2.5. Capas de apagado de neuronas (*Dropout*)

Estas capas fueron propuestas por Srivastava *et al.* en [31] y son un tipo de regularización en la red neuronal ya que es un método que desactiva un número de neuronas determinado de manera aleatoria de la red. En cada iteración esta capa "apagará" diferentes neuronas lo cual hace que las neuronas cercanas no dependan tanto de las neuronas desactivadas. Esto permite la reducción del sobre ajuste al conjunto de datos de entrenamiento. Además, el apagado aleatorio de las neuronas hace que esta capa se puede aproximar a la creación de un modelo en cada iteración dando lugar a un modelo por conjuntos.

2.3. Redes Generativas Antagónicas (GANs)

Las redes generativas antagónicas o *Generative Adversarial Networks* son un modelo de aprendizaje automático presentado por Ian Goodfellow et al. en 2014 [9].

Estas se utilizan en el aprendizaje no supervisado principalmente para la generación de imágenes sintéticas realistas al forzar las imágenes generadas a ser estadísticamente casi indistinguible de los reales. Este modelo está formado por un sistema de dos redes neuronales que compiten mutuamente en un juego de suma cero.

Por tanto, una GAN está formada por dos partes como se muestra en la Figura (2.9).

- **Generador**: toma como entrada un vector aleatorio (un punto aleatorio en el espacio latente), y lo decodifica en una imagen sintética.
- **Discriminador**: toma como entrada una imagen (real o sintética), y predice si la imagen proviene del conjunto de entrenamiento o ha sido creada por el generador.

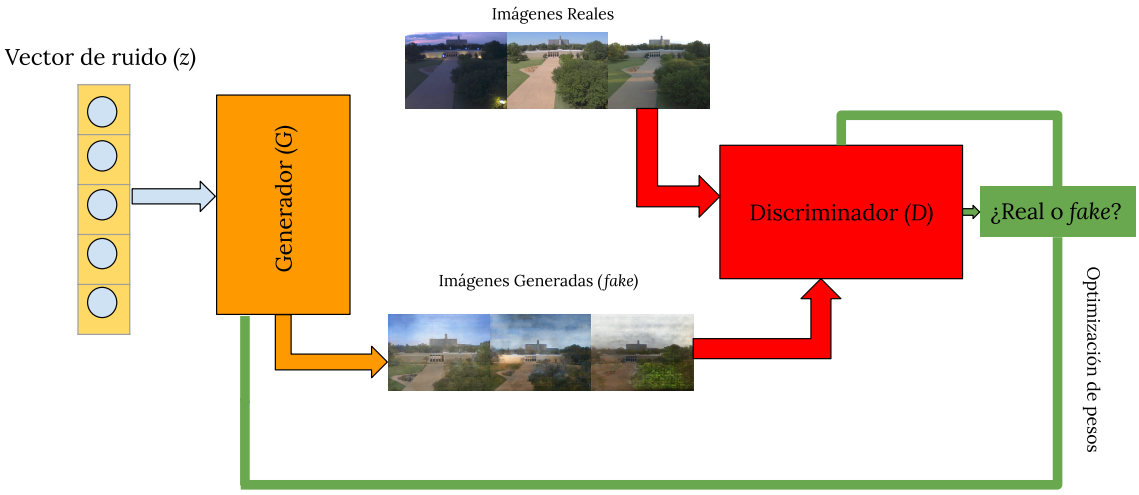


Figura 2.9: Representación esquemática del funcionamiento de una GAN formada por un generador y un discriminador.

Tal y como se dijo anteriormente, el aprendizaje de la GAN es un juego de suma cero de las funciones de pérdida de las dos redes. El generador intenta minimizar la siguiente ecuación que deriva de la *Binary Cross Entropy* mientras que el discriminador intenta maximizarla:

$$L = \min_G \max_D \left(E_x \left[\log(D(x)) \right] + E_z \left[\log(1 - D(G(z))) \right] \right) \tag{2.15}$$

donde $D(x)$ es la probabilidad predicha por el discriminador de que la instancia de datos reales x sea real, $G(z)$ es la salida del generador cuando se le introduce un vector de ruido z, $D(G(z))$ es la predicción del discriminador de la probabilidad de que una instancia falsa sea real, E_x es el valor esperado sobre todas las instancias de datos reales y por último, E_z es el valor esperado sobre todas las instancias falsas generadas $G(z)$.

Teóricamente, tras varias épocas de entrenamiento si el generador y el discriminador tienen suficiente capacidad, alcanzarán un punto en el que ambos no pueden mejorar porque la distribución de los datos reales $p_{data}(x)$ es igual que la distribución de la muestra generada $p_g(x)$. En este caso, el discriminador sería incapaz de diferenciar entre las dos distribuciones, es decir, $D(x) = \frac{1}{2}$.

Se puede resumir la estructura y el funcionamiento de una GAN de la siguiente manera:

- Una red generadora mapea vectores de dimensión determinada a imágenes de tamaño (h, w, c).
- Una red discriminadora que mapea imágenes de forma (h, w, c) a una predicción binaria y así permite estimar la probabilidad de que la imagen sea real.
- Una red GAN que contiene el generador y el discriminador juntos: GAN(x) = $D(G(x))$. Por tanto, esta red mapea los vectores espaciales latentes a una imagen que luego es introducida en el discriminador para dar la probabilidad de que sea real.
- El discriminador se entrena usando ejemplos de imágenes tanto reales como "falsas" junto con sus correspondientes etiquetas, es decir, como cualquier modelo de clasificación de imágenes normal.
- Para el caso del generador, se utilizan los gradientes de los pesos del generador con respecto a la pérdida del modelo GAN (*discriminador + generador*). Esto conlleva que, para cada iteración, los pesos del generador se mueven en una dirección que hace que la probabilidad de que el discriminador clasifique como "reales" las imágenes del generador aumente.

A continuación se van a explicar las diferentes estructuras *GAN* utilizadas a lo largo del trabajo.

2.3.1. *Deep Convolutional* GAN (DCGAN)

Este tipo de red generativa antagónica utiliza un serie de convoluciones y convoluciones transpuestas como las que han sido explicadas en la sección de capas, para mapear vectores de una dimensión determinada a imágenes de un tamaño concreto [11]. Utiliza como función de coste la entropía binaria cruzada (BCE) y la estructura utilizada ha consistido en la que se muestra en la Figura (2.10).

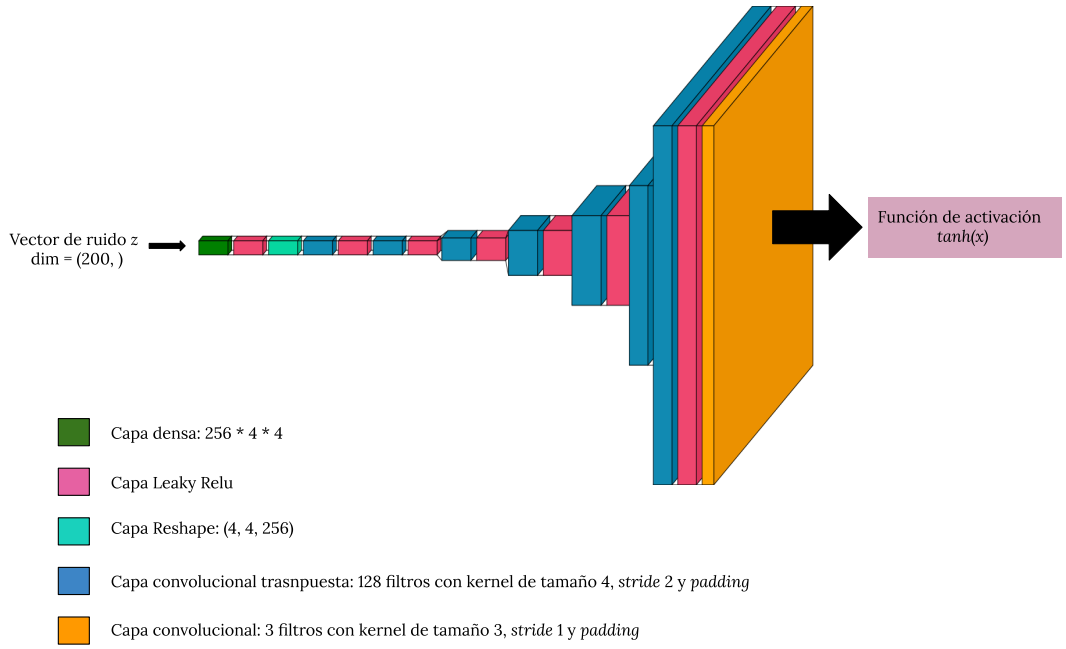


Figura 2.10: Estructura de red del generador utilizada para entrenar una *deep convolutional* GAN con imágenes de tamaño 512×512.

Como se ve en la Figura (2.10) el generador recibe un vector de ruido de una determinada dimensión que es pasado a través de una capa densa con activación *Leaky* Relu y se hace un *reshape* de dicho output. A continuación ese resultado se pasa a través de una serie de bloques de convolución transpuesta con activación *Leaky* Relu. Cada bloque de estos contiene 128 filtros de tamaño 4 con *stride* 2 y *padding*. Finalmente, tras el último bloque de este tipo los mapas de características se pasan a través de una capa convolucional con 3 filtros de tamaño 3, *stride* 1 y *padding* con activación de tangente hiperbólica.

En cuanto al discriminador, se trata de un clasificador de imágenes normal donde estas viajan a lo largo de una serie de capas convolucionales que aumentan el número de mapas de características pero disminuyen su tamaño. Finalmente los últimos mapas de características obtenidos son pasados a una capa de aplanamiento (*Flatten*) y una capa *Dropout* que son conectadas con una capa densa con una sola salida y activación sigmoide (*real* o *fake*).

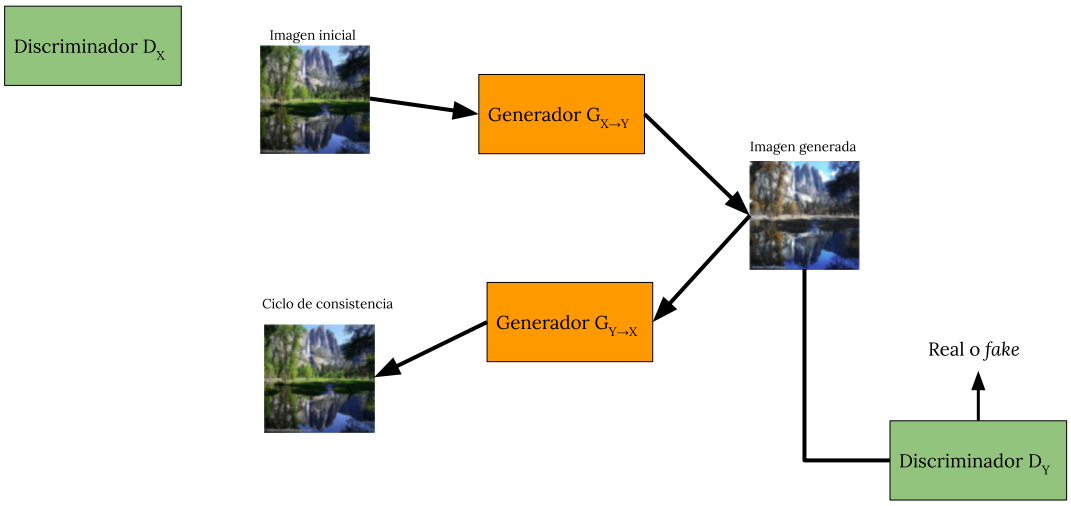
2.3.2. *Least Squares* GAN (LSGAN)

Este otro tipo de GAN es igual que la anterior pero cambiando la función de activación de salida y la función de pérdida del discriminador. Hasta ahora las redes generativas antagónicas GAN plantean el discriminador como un clasificador con la función de pérdida de entropía cruzada binaria. Sin embargo, Xudong Mao *et al.* encontraron en [32] que la función de pérdida puede conducir al problema del desvanecimiento del gradiente durante el proceso de aprendizaje. Para corregir este problema, han propuesto la red generativa de mínimos cuadrados (LSGAN). Esta red utiliza la función de pérdida de mínimos cuadrados para el discriminador y su función de activación a la salida ya no es sigmoide sino lineal. Existen dos beneficios de las LSGAN sobre las GAN normales: generan imágenes de mayor calidad y funcionan de manera más estable durante el proceso de aprendizaje.

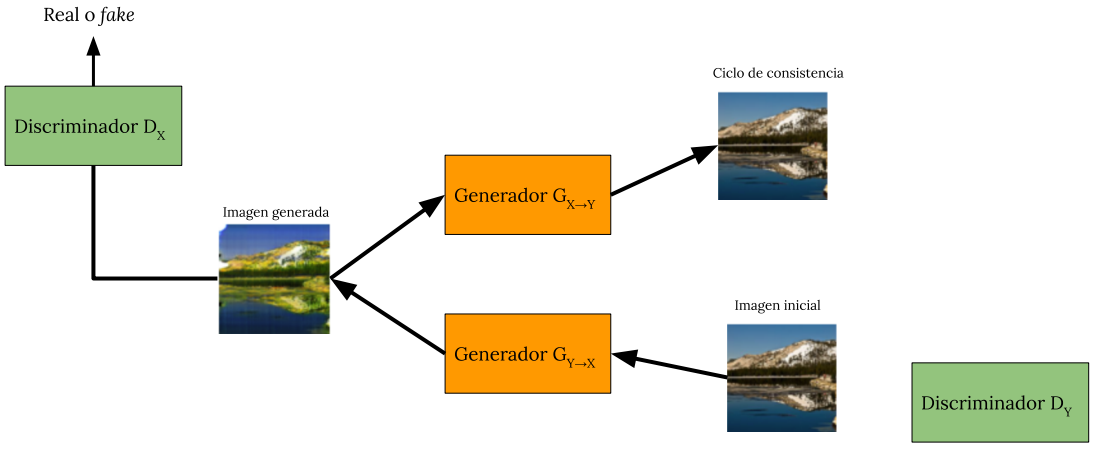
2.3.3. *Cycle* GAN

La Cycle GAN fue presentada por Jun-Yan Zhu *et al.* en [21] y su función es aprender a traducir una imagen de un dominio de origen X a un dominio de destino Y en ausencia de ejemplos emparejados. En este caso tanto el generador como el discriminador sufren una serie de variaciones/cambios que serán explicados en las siguientes líneas.

Una definición bastante esquemática y resumida de la Cycle GAN sería una red generativa antagónica formada a su vez por dos sistemas GAN, cada uno con su respectivo generador y discriminador. Cada uno de estos sistemas GAN se corresponde con uno de los dominios con los que se quiere trabajar, es decir, uno se ocupa de transformar una imagen del dominio X al Y y discriminar si la imagen generada es real o falsa en el dominio Y y viceversa. En la Figura (2.11) se ha representado dicho esquema.



Ciclo $X \to Y$



Ciclo $Y \to X$

Figura 2.11: Esquema e funcionamiento de la Cycle GAN.

En este tipo de GAN el generador que traduce una imagen del dominio X al Y, ya no recibe como input un vector de ruido sino una imagen del dominio X la cual es pasada a través de un estructura $encoder - decoder$ [33] como se muestra en la Figura (2.12).

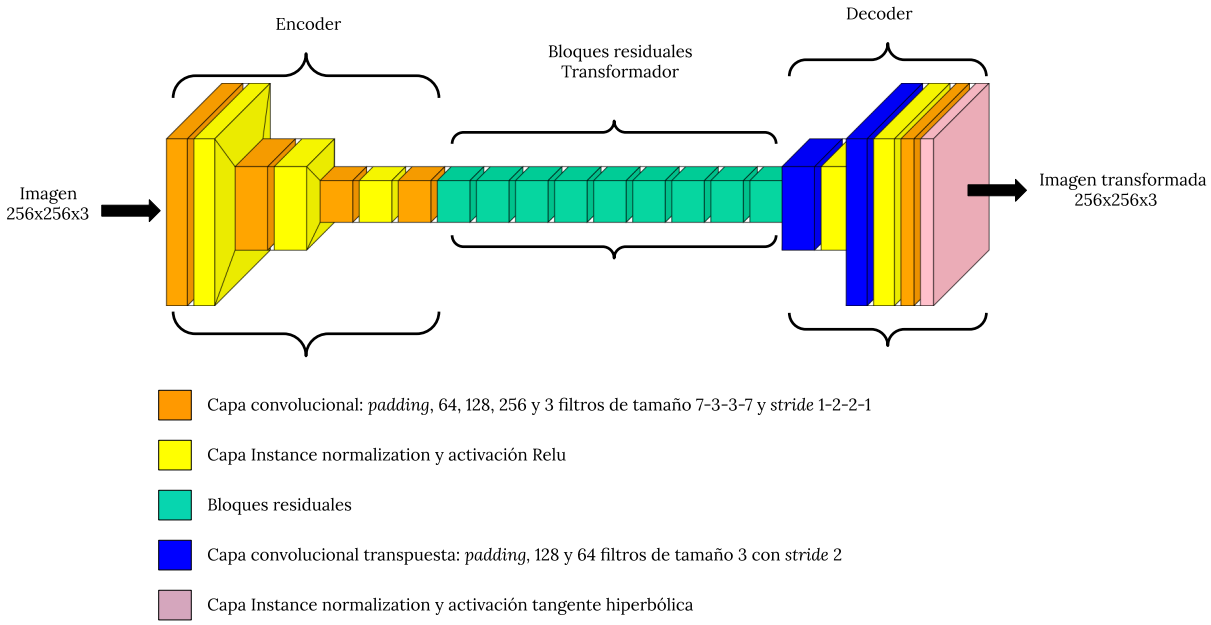


Figura 2.12: Estructura de red del generador utilizada para entrenar una *Cycle GAN* con imágenes de tamaño 256x256.

En la Figura (2.12) se muestra la estructura *encoder* − *decoder* del generador donde el primer elemento se basa en una estructura de capas que produce una reducción de la dimensionalidad espacial de los datos de entrada mediante una serie de convoluciones seguidas de una capa de *Instance* normalization con activación Relu. A continuación, los mapas de características obtenidos en el *encoder* son pasados a través de una serie de bloques residuales (9 en total) que realiza una serie de transformaciones sobre estos. Por último, tras las transformaciones realizadas, el *decoder* realiza la restitución a la dimensionalidad de entrada mediante una serie de capas convolucionales transpuestas seguidas de una capa de *Instance* normalization con activación Relu.

En cuanto al discriminador, se utiliza la estructura *PatchGAN* propuesta por Phillip Isola *et al.* en [20]. *PatchGAN* es un tipo de discriminador de redes generativas antagónicas que genera una matriz de valores entre 0 y 1 ($fake$; real) en lugar de uno solo. Por tanto, este discriminador clasificará si cada parche en una imagen es real o falso en lugar de calificar toda la imagen. Puede entenderse como un tipo de pérdida de textura / estilo. Una vez se obtienen las predicciones se utiliza el error cuadrático medio como función de pérdida al igual que en la LSGAN.

Hasta aquí el funcionamiento de la *cycle* GAN salvo la estructura del generador y el discriminador es similar a lo visto previamente. No obstante, lo que realmente caracteriza a este tipo de GAN, es la manera de actualizar los pesos del generador, es decir, la manera en la que está construida la función de pérdida. En este modelo a parte del término de la función de pérdida adversaria existen otros tres términos que hay que añadir y son de especial importancia: ciclo de consistencia $X \rightarrow Y \rightarrow X$, ciclo de consistencia $Y \rightarrow X \rightarrow Y$ y término de identidad.

A continuación se van a explicar esos tres términos para el caso de uno de los dos generadores ya que para el otro se aplica igual pero de manera inversa.

Término ciclo de consistencia $X \to Y \to X$ y $Y \to X \to Y$

Sea $G_{X \to Y}$ el generador que transforma imágenes del dominio X al Y. Tras introducir una imagen del dominio X en él y obtener la imagen mapeada al dominio Y, si dicha imagen generada se introdujese en el generador $G_{Y \to X}$ que transforma imágenes del dominio Y al X, la imagen resultante tendría que ser igual o muy parecida a la imagen inicial del dominio X, es decir, $G_{Y \to X}[G_{X \to Y}(X)] \approx X$. Esto se denomina ciclo hacia delante y puede verse en la ver Figura (2.13).

Por otro lado, si se pasa una imagen del dominio Y al generador que mapea imágenes de ese dominio al de X y luego esa imagen se introduce en el otro generador, de nuevo la imagen generada tendría que ser muy parecida a la inicial, es decir, , es decir, $G_{X \to Y}[G_{Y \to X}(Y)] \approx Y$. Esto se denomina ciclo hacia atrás y puede verse en ver Figura (2.13).

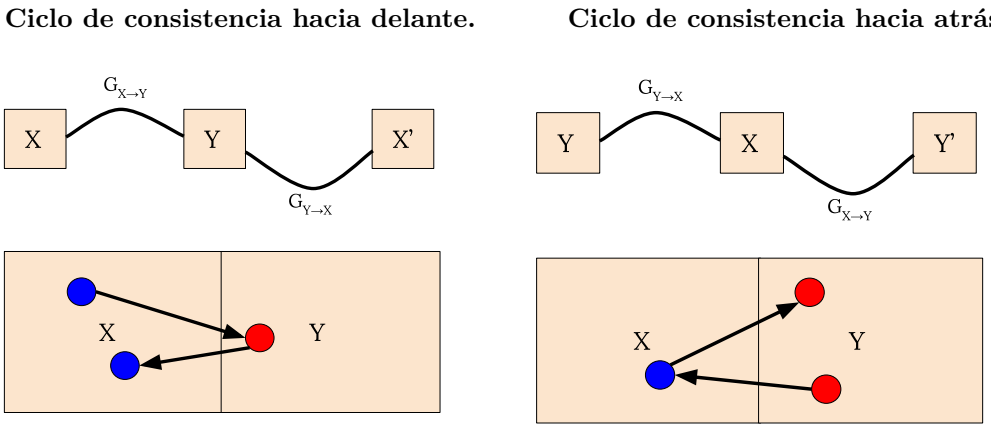


Figura 2.13: Ciclo de consistencia hacia delante y hacia atrás.

Este término en la función de pérdida es imprescindible para entrenar con datos no emparejados y así conseguir que la traducción de una imagen del dominio X al Y sea indistinguible de las imágenes reales de Y. En caso de no forzar esto, las imágenes generadas puede que no tuviesen que ver nada con la imagen original llegando incluso a producirse el colapso de modo, en el cual todas las imágenes de entrada se asignan a la misma imagen de salida.

Término de identidad

Este término de la función de pérdida es bastante sencillo y se propuso principalmente para ayudar con la preservación del color en las salidas. Sin embargo es un término opcional ya que se ha visto que en algunas áreas no supone un gran cambio en el resultado final. Su función consiste en que, dado el generador que mapea del dominio X al Y, si a dicho generador se le pasase una imagen del dominio final (en este caso Y) la imagen de salida tendría que ser exactamente la misma, es decir, el generador no tendría que realizar ninguna transformación en la imagen de entrada. Matemáticamente se traduciría en $G_{X \to Y}(Y) \approx Y$.

Función de pérdida total

La función de pérdida que se aplica a los tres términos explicados es el error medio absoluto, es decir, la distancia entre píxeles de la imagen de entrada y de salida.

De esta manera la función de pérdida final estaría formada por cuatro términos, el término adversario L_{adv}, el término del ciclo de consistencia hacia delante L_{ccf} y hacia atrás L_{ccb} y por último el término de identidad L_{id} como se ve en la Ec (2.16).

$$Loss = \lambda_1 L_{adv} + \lambda_2 L_{ccf} + \lambda_3 L_{ccb} + \lambda_4 L_{id} \tag{2.16}$$

Se puede observar que en la Ec (2.16) hay 4 hiperparámetros $\lambda_1 ... \lambda_4$ los cuales será necesario tunear durante el entrenamiento. Aquí se ve la gran dificultad de entrenar estos modelos ya que además de ser un modelo formado por dos GANs y por ende 4 tasas de aprendizaje, también tiene 4 hiperparámetros correspondientes a la función de pérdida.

Capítulo 3

Entrenamiento, resultados y análisis

Para realizar el entrenamiento de la red neuronal se ha implementado un paquete en *Python* utilizando las APIs de *keras* y *tensorflow* [34]. Este paquete está compuesto por cuatro módulos, diferentes:

- **Módulo GAN**: implementa la *deep convolutional* GAN y la *Least Squares* GAN y su entrenamiento.
- **Módulo *cycle* GAN**: implementa la estructura y el entrenamiento de la *cycle* GAN.
- **Módulo test**: se utiliza para testar los modelos entrenados en los distintos tipos de GAN con un conjunto de vectores de ruido (DCGAN y LSGAN) o de imágenes (*cycle* GAN).

Para mantener un histórico de todo el desarrollo del proyecto y añadir trazabilidad al desarrollo del paquete, se ha utilizado *GitLab*, un servicio web de control de versiones y desarrollo de software colaborativo basado en Git. Todo el código de este proyecto está subido en el *GitLab* de la empresa Predictia Intelligent Data Solutions.

Por último, para ejecutar y correr los entrenamientos de esta estructura de red neuronal se ha utilizado una GPU (*Graphics Processing Unit*), la cual ha sido prestada por el IFCA (*Instituto de Física de Cantabria*). El modelo de esta unidad de procesamiento gráfico es una GPU *NVIDIA TESLA V100* de 32 gb de memoria *RAM*, y ha sido esencial para correr los entrenamientos de las redes generativas antagónicas debido a su gran coste computacional, como se discutirá más adelante.

3.1. Conjunto de datos utilizados en el entrenamiento

En este trabajo se han utilizado en total tres datasets de imágenes: AMOS, *summer2winter yosemite* e $Image2Weather$.

La construcción del dataset AMOS (*Archive of Many Outdoor Scenes*) comenzó en marzo de 2006 y continúa hasta el día de hoy. El conjunto de datos AMOS contiene miles de millones de imágenes que se centran en diferentes entornos tomados de webcams ubicadas en todo el mundo. Esta base de datos es única porque contiene muchas escenas naturales al aire libre durante largos períodos de tiempo. [35] [36]

Su construcción fue debido a la gran utilidad que tiene la recopilación y almacenamiento todos los datos para estudiar los cambios en el medio ambiente, el clima, la temperatura

y el comportamiento de los objetos en la vista de las webcams. El objetivo es observar los cambios en el medio ambiente y analizar qué pudo haber causado esos cambios.

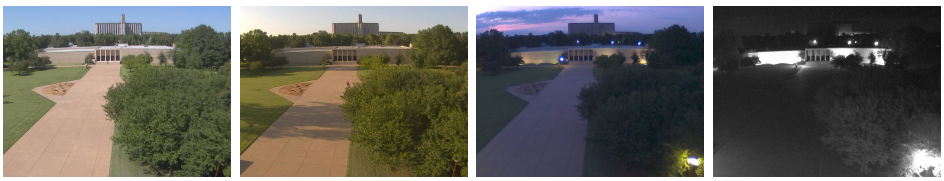


Figura 3.1: Imágenes del dataset AMOS captadas por una webcam a lo largo de un día. Dwight D. Eisenhower Presidential Library and Museum (*lat*: 38.91° *lon*: -97.21°)

Para el estudio propuesto, se han descargado las imágenes correspondientes a una sola escena/webcam a lo largo de los últimos años. Dicha webcam ha sido la que ha captado las imágenes de la Figura (3.1).

El dataset *summer2winter yosemite* consta de 1540 fotografías de verano y 1200 de invierno y fue obtenido del conjunto de datasets *cycle* GAN oficial de la Universidad de Berkeley. Las imágenes se descargaron usando la API de *Flickr* con la etiqueta *yosemite* y el campo $date - taken$. Las imágenes se escalaron a 256 × 256 píxeles. [21]

Figura 3.2: Imágenes del dataset *summer2winter yosemite* de verano e invierno.

El último dataset de imágenes utilizado es el $Image2Weather$. Este dataset fue construido por *Wei-Ta Chu et al.* en [37] con el objetivo de facilitar la estimación de las propiedades meteorológicas a partir de imágenes como harían posteriormente en [6]. Este dataset consiste en 183,798 imágenes de las cuales 70,501 son soleadas, 45,662 nubladas, 1,252 con nieve, 1,369 con lluvia y 357 con niebla.

Figura 3.3: Imágenes del dataset $Image2Weather$. (Soleado, nublado, niebla, lluvia y nieve)

En los tres datasets ha sido necesario realizar un cambio de tamaño ya que no todas las imágenes tenían la misma resolución. Para ello se ha utilizado la función "resize" del paquete de *Pyhton Pillow* con el método *Lanczos*, un filtro de alta calidad basado en convoluciones.

El primer dataset, AMOS, ha sido utilizado para entrenar una *deep convolutional* GAN mientras que los otros dos, *summer2winter* e $Image2Weather$, para entrenar modelos *cycle* GAN.

3.2. Entrenamiento *deep convolutional* GAN

Como se dijo, para este tipo de estructura se ha utilizado el dataset AMOS, concretamente las imágenes captadas por la webcam de la Figura (3.1).

En todos los modelos entrenados se utilizó como optimizador *Adam*. Las tasas de aprendizaje utilizadas para el discriminador y generador son 0.00008 y 0.0005 respectivamente.

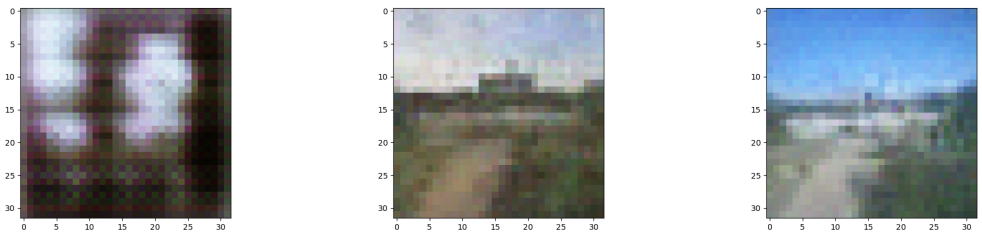
En este tipo de red la estructura del modelo (concretamente el generador) se fue aumentando de manera progresiva para cada vez obtener imágenes de mayor resolución, desde imágenes 32×32 hasta llegar a un modelo de 512×512 como el que se ve en la Figura (2.10).

La principal consecuencia de aumentar el tamaño del modelo y, por consiguiente, el número de parámetros, es que el tamaño del lote o *batch* tuvo que ir reduciéndose poco a poco. Se empezó con un tamaño de 64 en las imágenes de 32×32 y se redujo hasta 8 en las de 512×512. El número de épocas en las que se entrenaron los diferentes modelos variaba entre las 10 y las 20.

3.3. Resultados obtenidos *deep convolutional* GAN

A continuación se van a mostrar los resultados obtenidos para el modelo que genera imágenes de tamaño 32×32, 64×64, 128×128 y 512×512 y ver como ha ido mejorando la calidad de las imágenes generadas en cada uno de ellos. Para el caso de imágenes de tamaño 128×128 y 512×512 se mostrará la evolución de las funciones de pérdida para el discriminador y el generador así como la *accuracy* del discriminador para las imágenes reales y generadas. (ver Figura (3.9)).

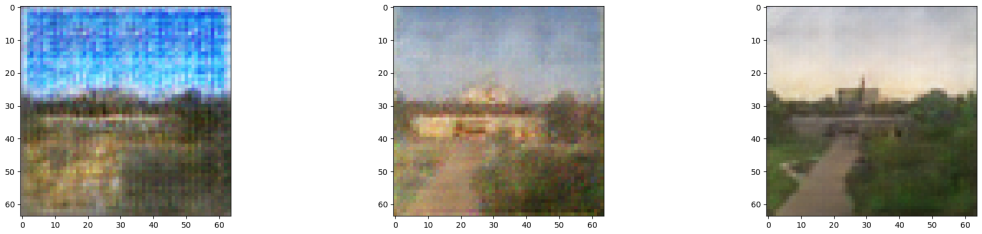
En la Figura (3.4) se muestran las imágenes creadas por el generador a lo largo del aprendizaje del modelo *GAN* para una resolución 32×32.



Época 1/10 ***Época 3/10*** ***Época 10/10***

Figura 3.4: Imágenes creadas por el generador de tamaño 32×32.

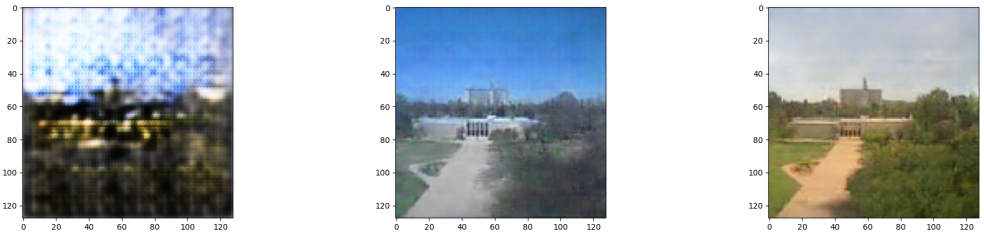
En la Figura (3.5) se muestran las imágenes creadas por el generador a lo largo del aprendizaje del modelo *GAN* para imágenes 64x64.



Época 1/10 ***Época 3/10*** ***Época 10/10***

Figura 3.5: Imágenes creadas por el generador de tamaño 64×64.

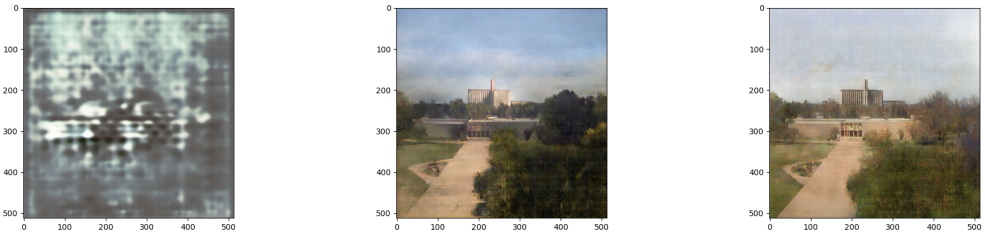
En la Figura (3.6) se muestran las imágenes creadas por el generador a lo largo del aprendizaje del modelo *GAN* para imágenes 128x128.



Época 1/20 ***Época 10/20*** ***Época 20/20***

Figura 3.6: Imágenes creadas por el generador de tamaño 128×128.

Por último, en la Figura (3.7) se muestran las imágenes creadas por el generador a lo largo del aprendizaje del modelo *GAN* para imágenes 128×128.



Época 1/20 ***Época 10/20*** ***Época 20/20***

Figura 3.7: Imágenes creadas por el generador de tamaño 512×512.

Además en la Figura (3.9) se muestra la evolución de las funciones de pérdida para el discriminador y el generador así como la *accuracy* del discriminador para las imágenes reales y generadas de tamaño 128×128 y 512×512.

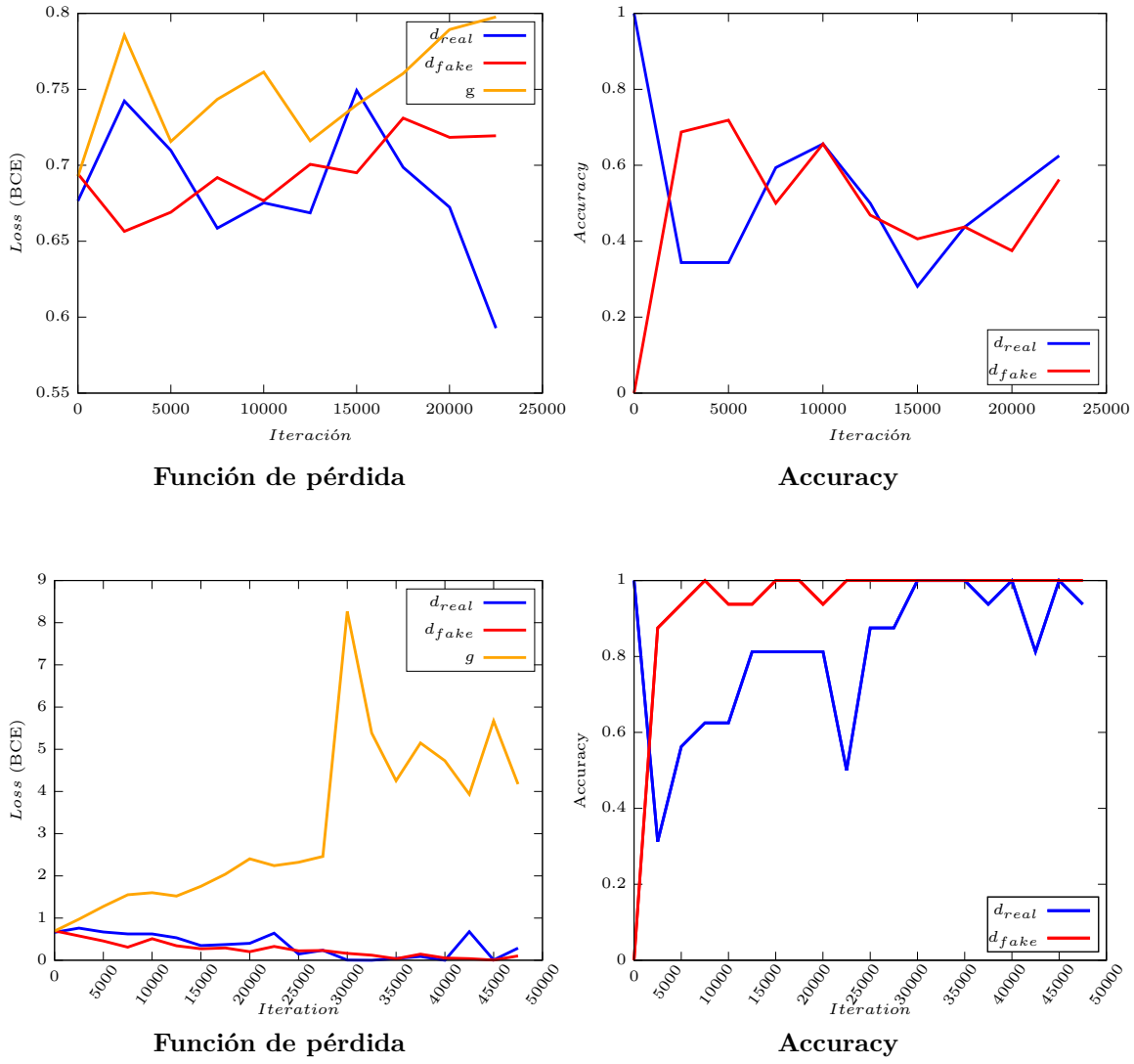


Figura 3.9: Evolución de la función de pérdida del discriminador y del generador así como de la métrica *accuracy* para el discriminador con muestras reales d_{real} y $fake$ d_{fake}. Imágenes de tamaño 128×128 y 512×512.

3.4. Entrenamiento *least squares* GAN

Se entrenó un modelo de imágenes con resolución 512×512, y al igual que la *deep convolutional* GAN se utilizó el dataset AMOS. También se ha empleado como optimizador *Adam* y las tasas de aprendizaje utilizadas para el discriminador y generador son 0.00008 y 0.0005 respectivamente.

El tamaño del lote fue de 8 y la principal diferencia frente al modelo anterior es que la función de activación final del discriminador ya no es sigmoide sino lineal, y la función de pérdida el error cuadrático medio.

3.5. Resultados obtenidos *least squares* GAN

En este apartado se mostrarán los resultados obtenidos para el modelo LSGAN entrenado con imágenes de resolución 512×512. En la Figura (3.10) se observan las imágenes creadas por el generador durante el entrenamiento de la red generativa antagónica y en la Figura

(3.11) la evolución de las funciones de pérdida para el discriminador y el generador así como la *accuracy* del discriminador para las imágenes reales y generadas.

Época 1/20 ***Época 10/20*** ***Época 20/20***

Figura 3.10: Imágenes creadas por el generador de tamaño 512×512.

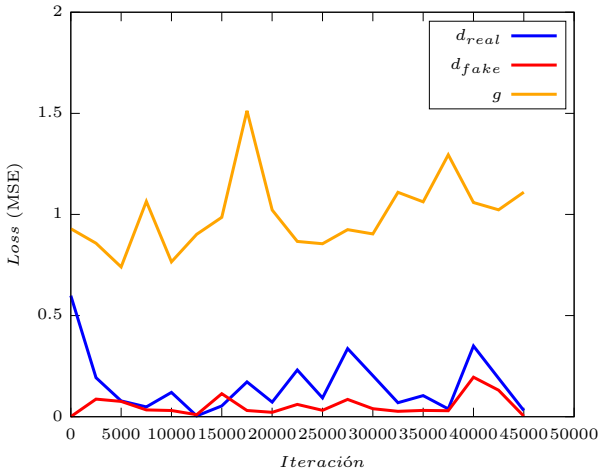


Figura 3.11: Evolución de la función de pérdida del discriminador y del generador. Imágenes de tamaño 512×512.

3.6. Entrenamiento *cycle* GAN

Para el entrenamiento de la *cycle* GAN se han utilizado tanto el dataset *summer2winter yosemite* como el $Image2Weather$. El primero de ellos se ha utilizado para realizar diferentes pruebas sobre el modelo y ver qué hiperparámetros son los que llevan a mejores resultados. Por otra parte, el dataset $Image2Weather$ es el elegido para trabajar y obtener los modelos finales ya que tiene una variedad de tipos de tiempo (soleado, nublado, lluvioso...).

3.6.1. Entrenamiento dataset *summer2winter yosemite*

Como se dijo, este dataset ha sido utilizado para testar los distintos hiperparámetros en el modelo *cycle* GAN, como son las tasas de aprendizaje de los generadores y los discriminadores, y los parámetros de la función de pérdida de la Ec (2.16).

En total se han entrenado 5 modelos *cycleGAN* donde principalmente se han variado los hiperparámetros λ_1, λ_2, λ_3 y λ_4 de la función de pérdida de la Ec (2.16). Las tasas de aprendizaje han sido las mismas tanto en los discriminadores como generadores en 4 de los 5 modelos, y han tomado el valor que proponen en [21] de 0.0002. El número de épocas que se han entrenado los modelos ha sido 100, exceptuando uno en el que se ha entrenado un total de 200 y, además, se ha implementado un decaimiento de la tasa de

aprendizaje para las últimas 100 épocas del entrenamiento. Este decaimiento consistía en una disminución del 25 % de la tasa actual de aprendizaje cada 10 épocas, aunque ambos hiperparámetros podían ser elegidos antes de comenzar el entrenamiento. En la Figura (3.12) puede observarse la evolución de la tasa de aprendizaje durante el entrenamiento del modelo GAN.

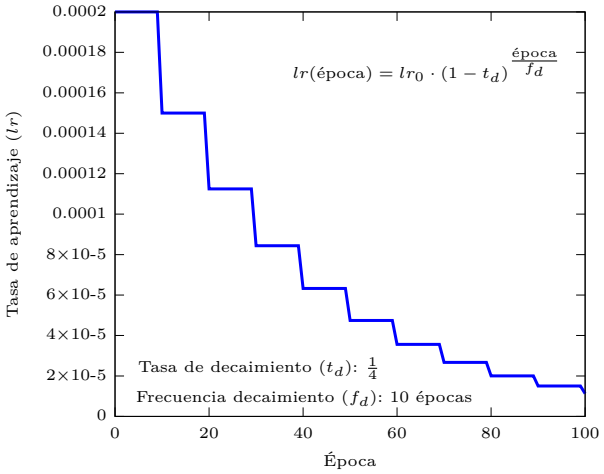


Figura 3.12: Decaimiento de la tasa de aprendizaje del 25 % cada 10 épocas para un entrenamiento de 100 iteraciones completas.

En cuanto al tamaño del lote o *batch* se ha utilizado tamaño 1. Esto es consecuencia de que el modelo construido consta de 4 submodelos (dos discriminadores y dos generadores), en los que cada uno está formado por millones de parámetros. Hay que tener en cuenta además de la desmesurada cantidad de parámetros, que un lote de tamaño 1 significa trabajar con un total de 6 imágenes 256×256, dos correspondientes una imagen real de cada dominio, otras dos correspondientes a las imágenes reales transformadas al otro dominio, y la vuelta a las dos imágenes originales para completar el ciclo de consistencia. Por tanto, estos modelos están muy limitados por el poder computacional del que se dispone. No obstante, se pudo llegar a hacer entrenamientos con tamaños de lote de hasta 4, pero se vio que los resultados que iban obteniendo empeoraban bastante con respecto a entrenamientos que utilizaban tamaño de lote 1.

En la Tabla (3.1) se van a resumir los hiperparámetros utilizados para cada término de la función de pérdida de los generadores en cada uno de los modelos entrenados, así como el tamaño del lote, la tasa de aprendizaje y el número de épocas que se entreno el modelo.

Modelo	Tamaño lote	lr	Épocas	λ_1	λ_2	λ_3	λ_4
I	1	0.0002	100	7	5	10	10
II	1	0.0006	100	7	5	10	10
III	4	0.0002	100	7	5	10	10
IV	1	0.0002	100	1	1	5	5
V	1	0.0002	200	1	1	10	10

Tabla 3.1: Resumen de los hiperparámetros utilizados para cada término de la función de pérdida de los generadores en cada uno de los modelos entrenados, λ_1, λ_2, λ_3 y λ_4. También aparece el tamaño del lote, la tasa de aprendizaje l_r y el número de épocas que se entreno el modelo.

3.6.2. Entrenamiento dataset *Image2Weather*

Este dataset es el empleado para entrenar los modelos finales que posteriormente serán utilizados en la aplicación. La principal razón de su elección es que posee una enorme variedad de imágenes de la vida cotidiana y en gran cantidad de monumentos conocidos, pero sobre todo, porque están catalogadas en 5 tipos de tiempo que permitirán utilizar una *cycle* GAN para realizar transferencia del estilo de los distintos tiempos.

En total cuenta con imágenes de tiempo soleado, nublado, lluvioso, nevado y con niebla. Esto ha hecho que la manera en la que se ha abordado el problema haya consistido en crear 4 modelos *cycle* GAN en los que siempre este presente el tiempo soleado, es decir, soleado-nublado, soleado-lluvioso;, soleado-nevado y soleado-niebla. La razón de esta elección ha sido que salvo soleado (cielo azul, tonos cálidos, gran iluminación...) los demás tiempos son bastante parecidos en cuanto a características como la iluminación, el color del cielo (grisáceo) ... Por tanto, si se quisiera pasar de nublado a nevado por ejemplo, habría que hacer un paso intermedio transformando la imagen real a soleado, y a continuación al tipo de tiempo soleado, es decir, nublado→soleado→ nevado.

Se han entrenado en total 8 modelos, cuatro correspondientes al modelo 4 utilizado con el dataset *summer2winter yosemite* y otros cuatro utilizando nuevos valores de los distintos hiperparámetros. En ambos casos se ha utilizado decaimiento de la tasa de aprendizaje del 25 % cada 10 épocas.

En el segundo tipo de entrenamiento además de los términos de la función de pérdida de la Ec (2.16), se ha añadido un quinto término a la función. Este quinto término fue propuesto en [38] por *Deniz Engine et al.* para eliminar niebla de imágenes y es definido como una función de percepción o *perceptual loss*. La idea principal de esta término es comparar imágenes en un espacio de características en lugar de en un espacio de píxeles con el fin de preservar la estructura de la imagen original y evitar la aparición de artefactos aleatorios. Para ello se utilizan redes previamente entrenadas para extraer características de alto nivel [39]. En este caso se han seguido los pasos de *Ayush Baid et al.* en [40] donde utilizan el primer bloque convolucional de la red preentrenada $VGG-19$, el cual está formado por una capa convolucional, una capa *Instance normalization* y una función de activación *Leaky* Relu. Aplican el error cuadrático medio (MSE). En la Tabla (3.2) se van a resumir los hiperparámetros utilizados en los dos entrenamientos realizados.

Entrenamiento	Tamaño lote	lr	Épocas	λ_1	λ_2	λ_3	λ_4	*Perceptual* loss
1	1	0.0002	100	1	1	5	5	0
2	1	0.0002	100	3	1	4	4	3

Tabla 3.2: Resumen de los hiperparámetros utilizados para los dos entrenamientos realizados con el dataset $Image2Weather$.

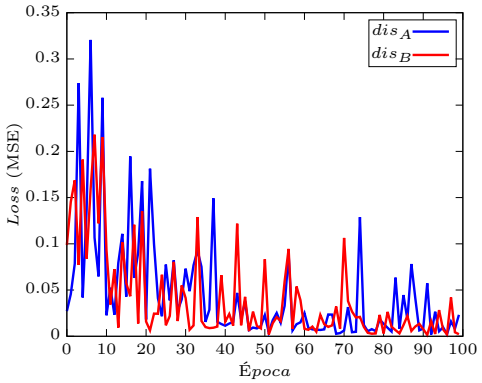
3.7. Resultados obtenidos *cycle* GAN

En esta sección se van a mostrar los resultados obtenidos para los distintos modelos *cycle* GAN entrenados con los datasets *summer2winter yosemite* e $Image2Weather$.

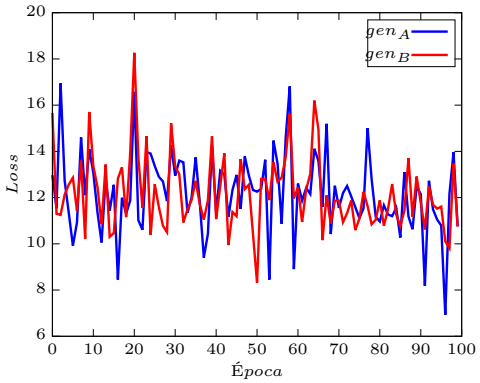
3.7.1. Resultados dataset *summer2winter yosemite*

Lo primero que se va a ver, es la evolución del valor de la función de pérdida tanto en los generadores como en los discriminadores de los 5 modelos entrenados.

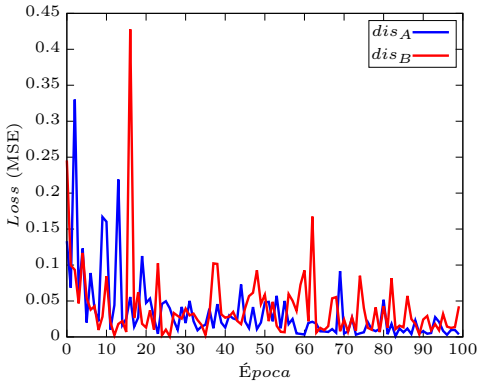
En la Figura (3.13) se muestran dichas representaciones.



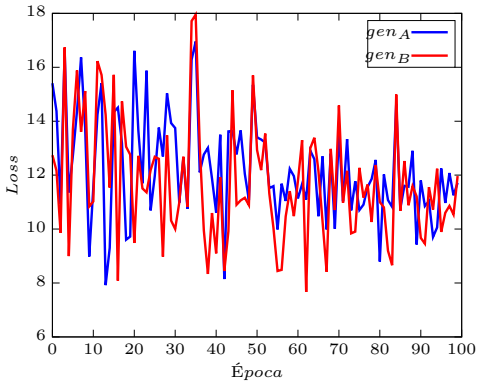
a) **Discriminadores modelo I**



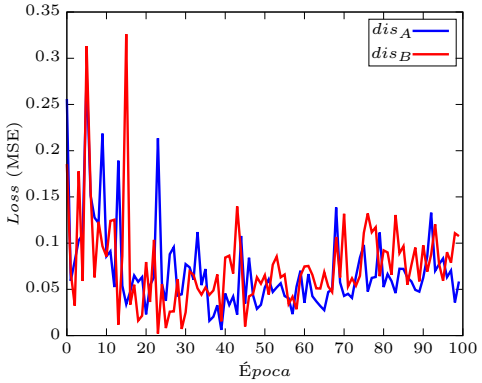
b) **Generadores modelo I**



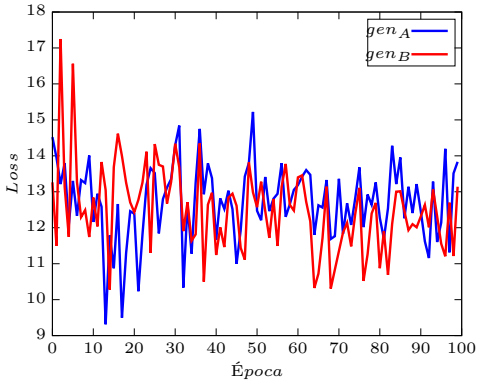
c) **Discriminadores modelo II**



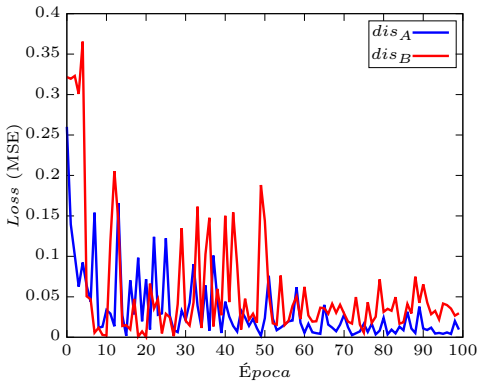
d) **Generadores modelo II**



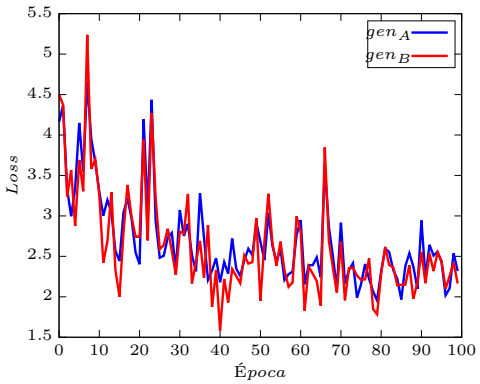
e) **Discriminadores modelo III**



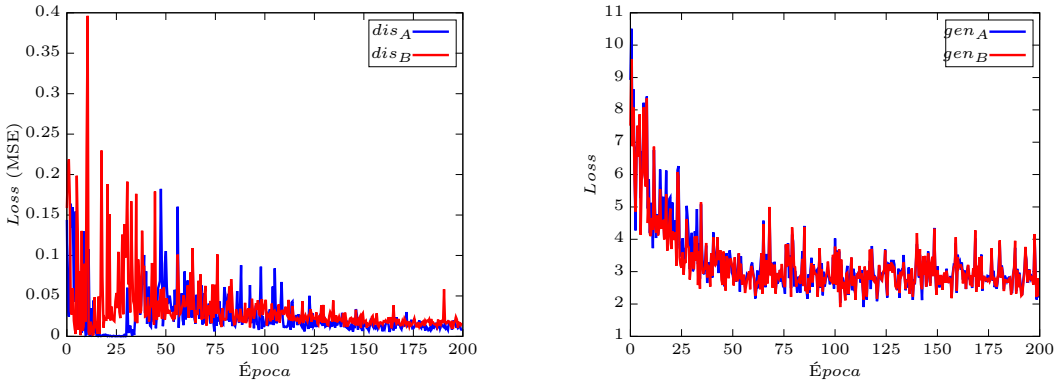
f) **Generadores modelo III**



g) **Discriminadores modelo IV**



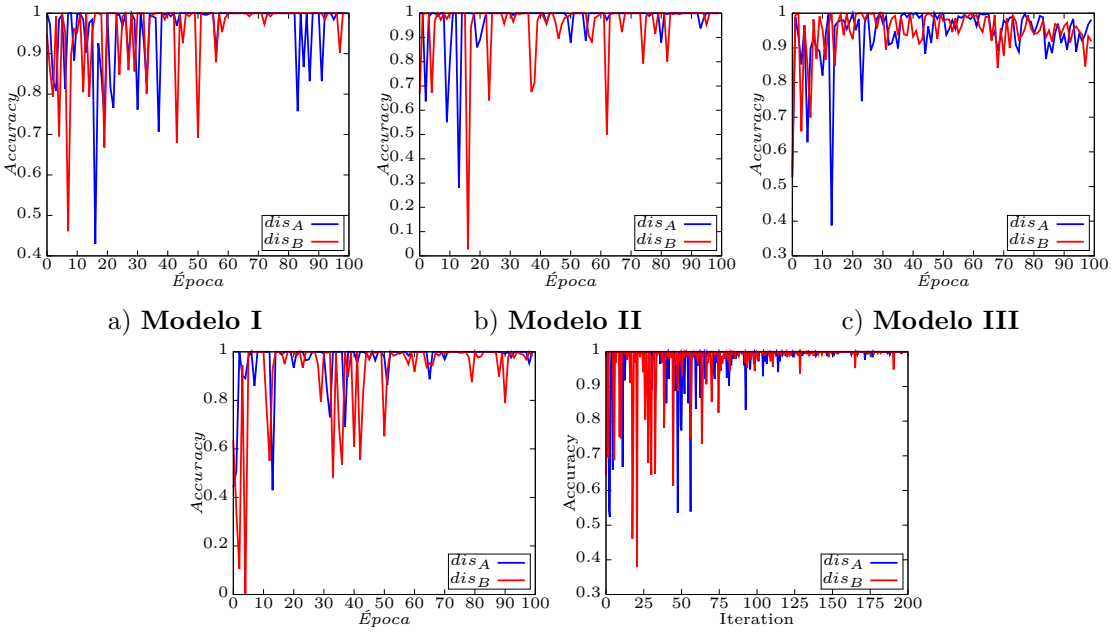
h) **Generadores modelo IV**



i) **Discriminadores modelo V** j) **Generadores modelo V**

Figura 3.13: Evolución de la función de pérdida de los discriminadores y generadores que componen una *cycle* GAN en los modelos entrenados con el dataset *summer2winter yosemite*.

A continuación se va a mostrar la evolución de la métrica *accuracy* de los discriminadores de cada uno de los modelos en la Figura (3.14).



a) **Modelo I** b) **Modelo II** c) **Modelo III**

d) **Modelo IV** e) **Modelo V**

Figura 3.14: Evolución de la métrica *accuracy* en los discriminadores de ambos dominios para los modelos entrenados con el dataset *summer2winter yosemite*.

Una vez presentada la evolución tanto de la función de pérdida como de la métrica *accuracy* para los cinco modelos, se van a enseñar los resultados que se obtienen al aplicar dichos modelos sobre 5 imágenes de verano y otras 5 de invierno. La mayor parte de los modelos entrenados se ha visto que funcionan mejor en torno a las 50-70 épocas de entrenamiento, por lo que se utilizarán esos modelos.

En la Figura (3.15) se muestran las 5 imágenes reales de invierno transformadas a verano mediante los modelos entrenados con la *cycle* GAN.

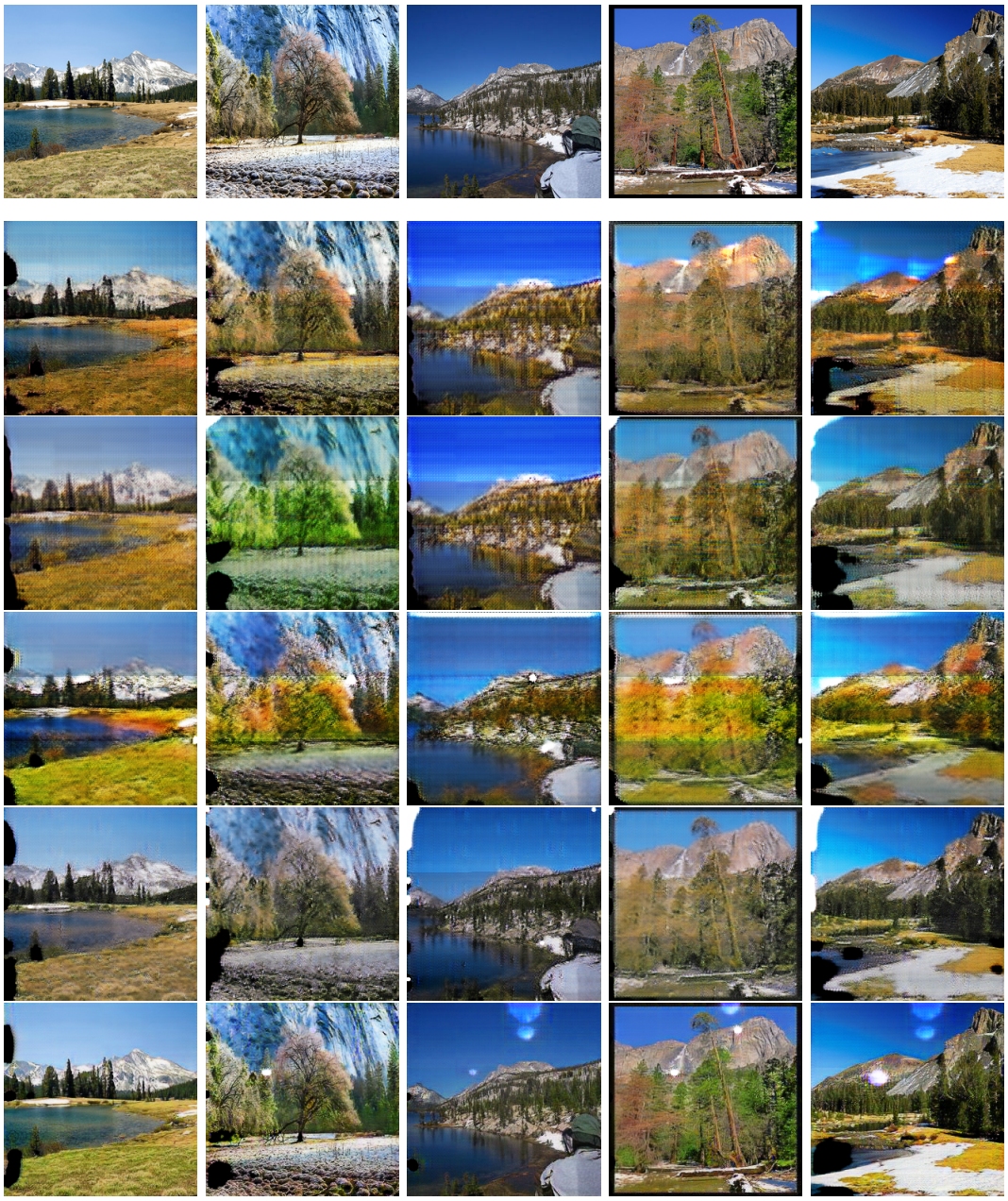


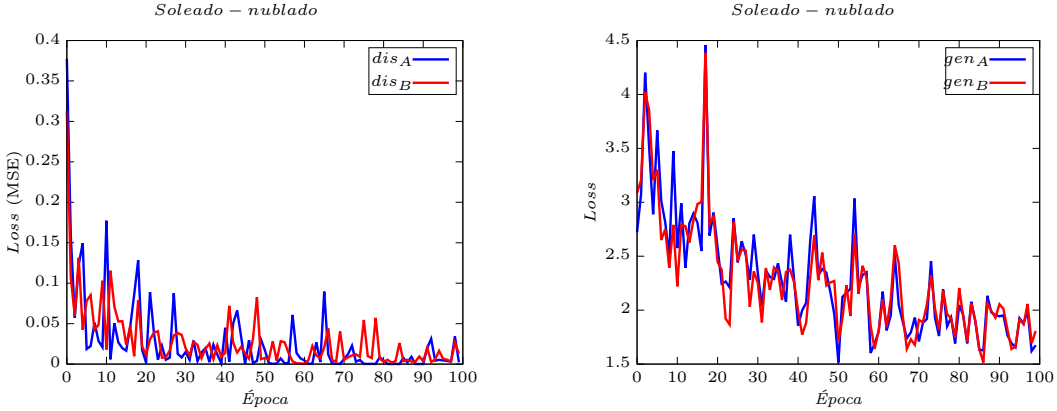
Figura 3.15: Imágenes de invierno transformadas a verano para los 5 modelos entrenados con el dataset *summer2winter yosemite*. La primera fila se corresponde con las imágenes reales de invierno, mientras que las siguientes con las imágenes transformadas con los modelos I, II, ...

En la Figura (3.16) se muestran las 5 imágenes reales de verano transformadas a invierno mediante los modelos entrenados con la *cycle* GAN.

Figura 3.16: Imágenes de verano transformadas a invierno para los 5 modelos entrenados con el dataset *summer2winter yosemite*. La primera fila se corresponde con las imágenes reales de verano, mientras que las siguientes con las imágenes transformadas con los modelos I, II, ...

3.7.2. Resultados dataset *Image2Weather*

Al igual que en el anterior dataset, en las Figuras (3.17), (3.18), (3.19) y (3.20) se van a mostrar la evolución de las funciones de pérdida de los generadores y discriminadores de lo modelos *soleado – nublado*, *soleado – lluvioso*, *soleado – niebla* y *soleado – nevado* para los dos entrenamientos de la Tabla (3.2).



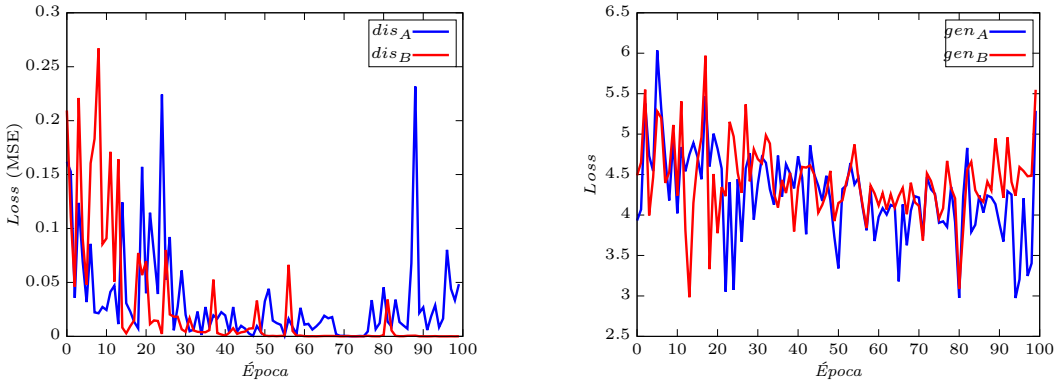


Figura 3.17: Evolución de la función de pérdida de los discriminadores y generadores que componen una *cycle* GAN *soleado – nublado*. Arriba el modelo I y abajo el II.

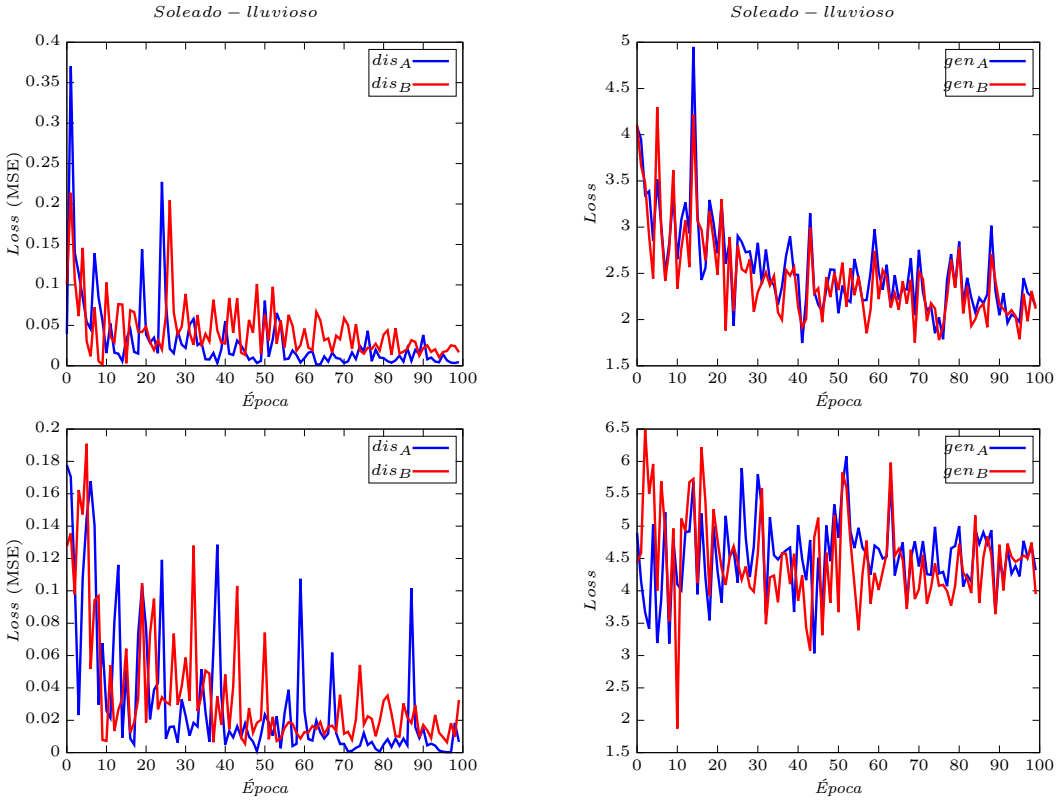
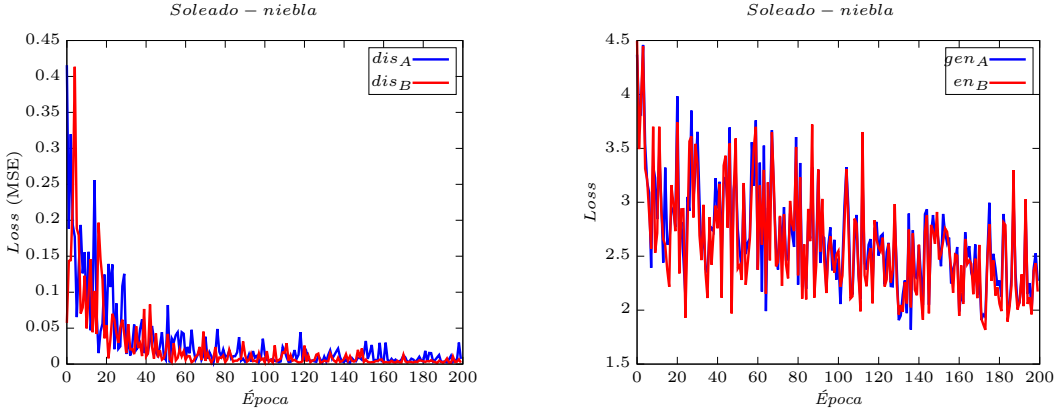


Figura 3.18: Evolución de la función de pérdida de los discriminadores y generadores que componen una *cycle* GAN *soleado – lluvioso*. Arriba el modelo I y abajo el II.



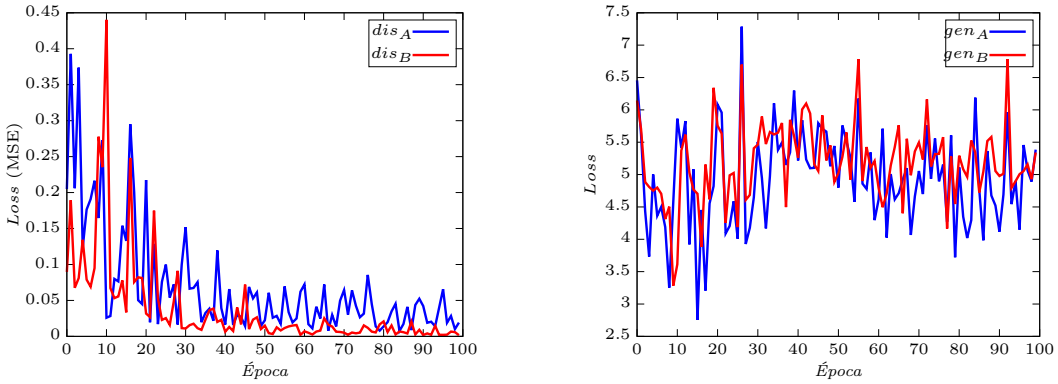


Figura 3.19: Evolución de la función de pérdida de los discriminadores y generadores que componen una *cycle* GAN *soleado – niebla*. Arriba el modelo I y abajo el II.

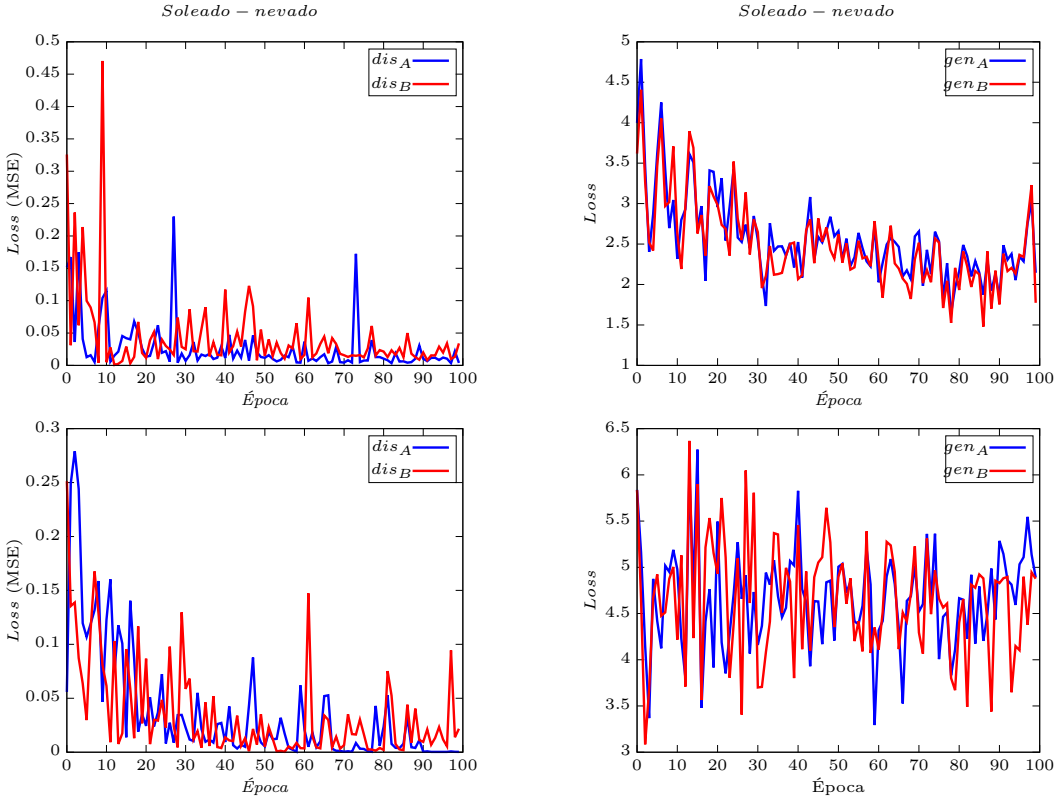


Figura 3.20: Evolución de la función de pérdida de los discriminadores y generadores que componen una *cycle* GAN *soleado – nevado*. Arriba el modelo I y abajo el II.

En cuanto a la evolución de la métrica *accuracy* en los 8 modelos entrenados, el comportamiento es bastante similar al visto en la Figura (3.14) por lo que no se mostrará su visualización y se pasará directamente a mostrar los distintos modelos aplicados a una serie de imágenes.

En la Figura (3.21) se muestran las imágenes con tiempo nublado transformadas a imágenes con tiempo soleado con el modelo I y II *cycle* GAN de la Tabla (3.2).

Figura 3.21: Imágenes de tiempo nublado transformadas a tiempo soleado mediante un modelo *cycle* GAN. Arriba modelo I y abajo modelo II de la Tabla(3.2)

En la Figura (3.22) se muestran las imágenes con tiempo nevado transformadas a imágenes con tiempo soleado con el modelo I y II *cycle* GAN de la Tabla (3.2).

Figura 3.22: Imágenes de tiempo nevado transformadas a tiempo soleado mediante un modelo *cycle* GAN. Arriba modelo I y abajo modelo II de la Tabla (3.2)

En la Figura (3.23) se muestran las imágenes con niebla transformadas a imágenes con tiempo soleado con el modelo I y II *cycle* GAN de la Tabla (3.2).

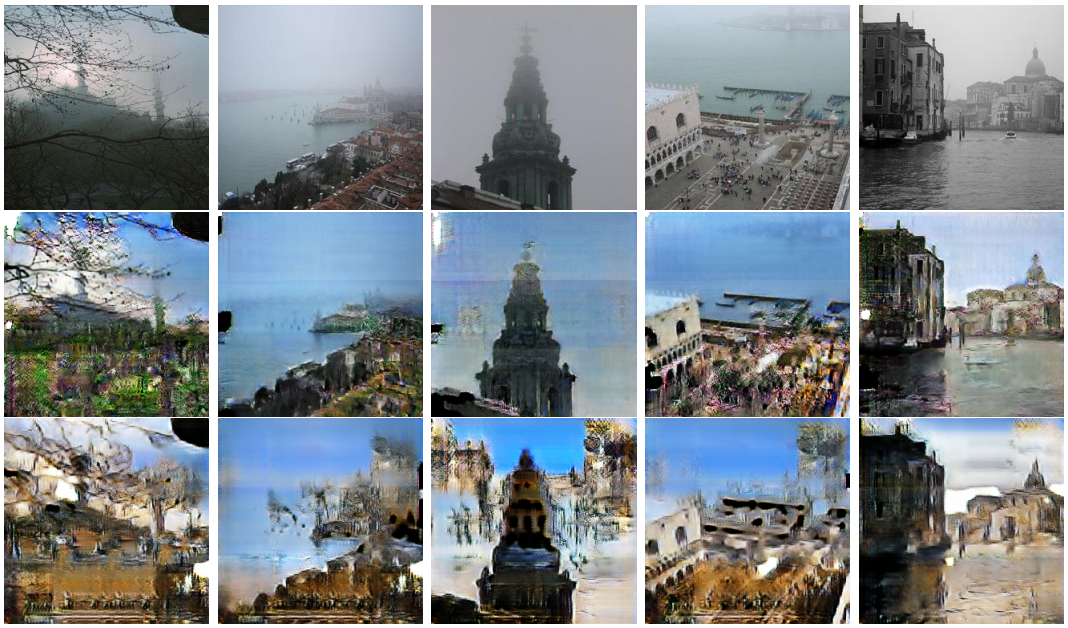


Figura 3.23: Imágenes de niebla transformadas a tiempo soleado mediante un modelo *cycle* GAN. Arriba modelo I y abajo modelo II de la Tabla (3.2)

En la Figura (3.24) se muestran las imágenes con niebla transformadas a imágenes con tiempo soleado con el modelo I y II *cycle* GAN de la Tabla (3.2).

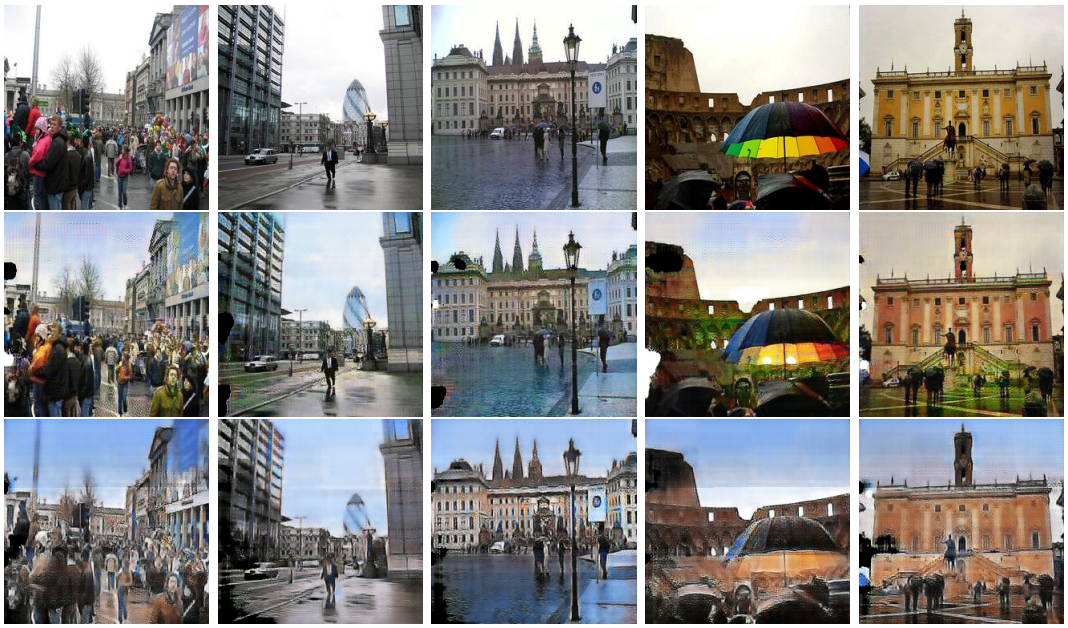


Figura 3.24: Imágenes de lluvia transformadas a tiempo soleado mediante un modelo *cycle* GAN. Arriba modelo I y abajo modelo II de la Tabla (3.2)

Por ultimo, en la Figura (3.25) se va a mostrar la transformación de imágenes soleadas a los otros tipos de tiempo (nublado, nevado, niebla y lluvia) mediante los modelos I y II *cycle* GAN de la Tabla (3.2).

Figura 3.25: Imágenes soleadas transformadas a nublado, nieve, niebla y lluvia (en ese orden) mediante los modelos *cycle* GAN de la Tabla (3.2)

Capítulo 4

Conclusiones y futuro trabajo

En este trabajo se han construido y entrenado un conjunto de modelos de redes neuronales del tipo red generativa antagónica siendo el principal objetivo evidenciar el efecto del cambio climático en la Tierra. Esto se ha conseguido mediante la modificación de las condiciones meteorológicas en imágenes haciendo uso de tres clases de GANs estado del arte, la *deep convolutional*, la *least squares* GAN y la *cycle* GAN.

A continuación se discutirán las principales conclusiones obtenidas para cada uno de estos tipos de modelos.

4.1. Modelos *deep convolutional* y *least squares* GAN

Estos dos tipos de redes generativas antagónicas son muy similares ya que su estructura es idéntica salvo que una utiliza la entropía cruzada binaria como función de pérdida, y la otra el error cuadrático medio. Ambos modelos reciben un vector de ruido de una determinada dimensión como datos de entrada, y mediante una serie de operaciones convolucionales dicho vector es transformado en una imagen. Para estas dos estructuras se ha utilizado el dataset AMOS.

En el caso de la DCGAN puede observarse en la Figura (3.4, 3.5, 3.6, 3.7) que a medida que se aumenta la resolución de las imágenes el generador es capaz de generar imágenes de mayor calidad como consecuencia del aumento de capas de la red neuronal del generador, es decir, del aumento de la complejidad del modelo.

En la Figura (3.9) del entrenamiento de la DCGAN se ve como varían las métricas y funciones de pérdida de este tipo de modelos, algo bastante variable entre los modelos de diferentes resoluciones. En el caso del modelo de imágenes de 128×128 se puede llegar a ver como a partir de la iteración aproximadamente 7000 el discriminador es incapaz de distinguir las imágenes reales de las imágenes $fake$ ($accuracy \approx 50\,\%$) como explican en [9].

Por otro lado, en el modelo entrenado con la LSGAN para imágenes 512×512 se ha comprobado las dos ventajas de utilizar el error cuadrático medio como función de pérdida, la estabilidad del entrenamiento y la obtención de imágenes de mayor calidad. Esto se ve comparando las imágenes de la Figura (3.10) y la Figura (3.7) donde las imágenes generadas en la primera época son mucho más realistas. También en la Figura (3.11) se observa que las funciones de pérdida tanto del generador como el discriminador, siguen una tendencia en torno a un valor constante.

Un aspecto que hay que tener en cuenta, es que las imágenes con las que se ha trabajado (en total unas 40.000) han sido imágenes de la misma escena, es decir, el discriminador se

ha alimentado con imágenes siempre en el mismo escenario lo cual hace que el generador solo aprenda a generar este tipo de imágenes. Esto hace que el modelo creado sea bastante limitado. Sin embargo, si se hubiese alimentado el modelo con una gran variedad de escenarios, el generador puede que no hubiese sido capaz de aprender a generar ninguna imagen realista o con sentido, pues no es lo mismo representar una cara de una persona (campo de mayor explotación de las GANs como se comentó en el estado del arte) donde más o menos todas muestran las mismas propiedades, que un paisaje aleatorio donde la variabilidad entre escenarios es muchísimo mayor.

4.2. Modelos *cycle* GAN

Para estos modelos se han empleado los datasets *summer2winter yosemite* e *image2Weather*.

El primero de ellos era un conjunto de imágenes de dos dominios, verano e invierno, y ha servido para testar un total de cinco modelos donde se han variado los hiperparámetros (tasas de aprendizaje, pesos términos de la función de pérdida...) de este tipo de red generativa con el fin de ver cuáles de estos son los que dan mejores resultados. En la Figura (3.13) se puede ver el comportamiento de la función de pérdida de cada uno de los discriminadores y generadores que conforman el modelo *cycle* GAN. La función de pérdida de los discriminadores presenta una tendencia decreciente bastante similar en los 5 modelos a pesar de utilizar distintos hiperparámetros. En el caso de los generadores también ocurre algo bastante parecido ya que la tendencia es bastante oscilatoria en torno a un valor medio, el cual varía según los pesos que se hayan dado a los distintos términos de la función de pérdida. En cuanto a la métrica *accuracy* de la Figura (3.14) los 5 modelos muestran una gran oscilación al comienzo del entrenamiento, pero en torno a la época 40-50 comienzan a estabilizarse.

No obstante, donde realmente se entiende cómo afecta la modificación de los distintos hiperparámetros de la *cycle* GAN en los resultados obtenidos, es a través de las imágenes de la Figura (3.16) y Figura (3.15).

Por un lado se puede ver que aumentar el tamaño del lote a 4 (modelo III Tabla (3.1)) o subir la tasa de aprendizaje (modelo II Tabla (3.1)) no ayuda mucho ya que los resultados obtenidos con un lote de tamaño uno o la otra tasa de aprendizaje son similares.

Por tanto, se puede concluir que donde realmente los hiperparámetros juegan un papel fundamental es en los distintos pesos de la función de pérdida de la Ec (2.16). Por ejemplo el modelo II y V de la Tabla (3.1) dan distintos pesos a los términos adversarios y del ciclo de consistencia. El primero de ellos da bastante importancia al término adversario (7) por lo que el modelo GAN busca la generación de imágenes que sean capaces de burlar al discriminador del otro dominio más que generar imágenes de calidad. El caso del segundo modelo es diferente, pues el peso del término adversario es mínimo (1) frente a los términos del ciclo de consistencia (10). Como consecuencia, aquí la red neuronal prioriza mucho más volver a la imagen original que generar imágenes del otro dominio que sean capaces de burlar al discriminador, obteniendo imágenes prácticamente casi inalteradas.

Esto ocurre de igual manera en el dataset $image2Weather$, compuesto por imágenes de tiempo *soleado*, *nublado*, *niebla*, *nieve* y *lluvia*. El modelo I de la Tabla (3.2) da más importancia a los términos del ciclo de consistencia (5) que al adversario (1), en contraposición con el modelo II donde ocurre lo contrario. En las Figuras (3.17-3.20) se muestra la evolución de las funciones de pérdida de los generadores y discriminadores de lo modelos cuya tendencia es similar a la del otro dataset.

El peso que tiene el término adversario en el aprendizaje de la red generativa se hace especialmente notable en el caso del modelo *soleado – niebla*. Se observa cómo al dar gran importancia a este término, la modificación en las imágenes es altísima llegando incluso a sobre ajustar en la tercera foto de la Figura (3.19), ya que se ven edificios alrededor del campanario. Esto es lógico, ya que en la mayor parte de las imágenes con niebla a penas se distingue los elementos de la imagen como pueden ser edificios. Sin embargo, en las imágenes soleadas si que se distinguen perfectamente todos esos elementos, por lo que la red no solo podría estar aprendiendo a poner neblina sino también edificios, personas...

En la Figura (4.1) se muestra un desglose de la importancia de los términos adversario, de ciclo de consistencia y de identidad de la *cycle* GAN para uno de los generadores del modelo II *soleado – nublado*.

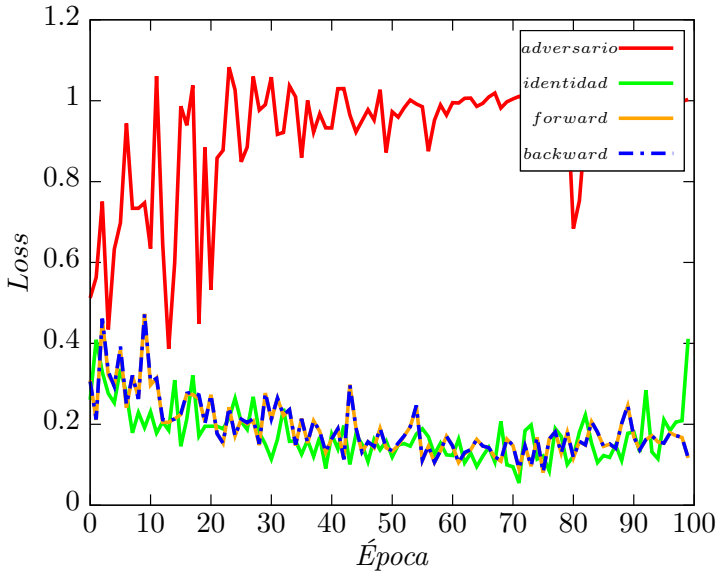


Figura 4.1: Desglose de los distintos términos de la función de pérdida total de uno de los generadores del modelo II *soleado – nublado*. Aparecen el término adversario, el de identidad y los del ciclo hacia delante y hacia atrás.

En esta figura se ve que el término que más peso tiene sin hacer una ponderación es el término adversario (aproximadamente 1) frente a los otros tres términos que oscilan en torno a 0.2 - 0.4, por lo que en algunos modelos el término adversario ha sido de hasta 7 veces más grande que los otros.

También decir que observando las imágenes transformadas por este tipo de modelos se ve que estos se centran principalmente en variar la iluminación de las imágenes o cambiar colores como pueden ser el cielo o el suelo. Por ejemplo, al transformar fotos a tiempo soleado la iluminación de la imagen aumenta considerablemente y el cielo aparece de un tono mucho más azulado. En cambio, al transformar cualquier tipo de imagen soleada a otro tipo de tiempo está adquiere unos tonos mucho más grisáceos y la iluminación de la imagen es mucho menor que en la foto original.

Por último, puede que los resultados de este tipo de modelos tengan cierta dependencia con los conjunto de datos utilizados, ya que en el caso del dataset *summer2winter yosemite* los resultados obtenidos son mejores que con el dataset *image2Weather* a pesar de que se utilizó la misma estructura de red en el entrenamiento. Esto puede ser normal ya que mientras que el primer dataset se centra únicamente en imágenes de paisajes naturales (sin coches, personas, edificios...), el segundo contiene escenarios muy variados desde paisajes

hasta monumentos o edificios, por lo que es mucho más complicado de generalizar.

4.3. Conclusión final

Este tipo de redes neuronales son sistemas de gran complejidad, pues el modelo completo está a su vez compuesto por otros 2 o incluso 4 submodelos que dan lugar a una gran cantidad de parámetros que ajustar a lo largo del entrenamiento. Además, la manera de aprender de estos tipos de redes es mediante un juego de suma cero lo que hace que la estabilidad del entrenamiento sea un problema ya que puede que alguno de los modelos aprenda más rápido que otro y el aprendizaje global no continúe.

A todo eso hay que sumar que el gasto computacional es enorme ya que requiere el uso de GPU de gran potencia y el tiempo que tarda en ejecutarse un modelo de estas características es de aproximadamente 3 días, lo que complica más aún tunear hiperparámetros como pesos en funciones de pérdida, tasas de aprendizaje, tamaños de lote etc.

Por todo ello, aunque estos modelos se correspondan con el estado del arte en este tipo de aplicaciones, los resultados están muy limitados por el poder computacional del que se dispone. Esto complica el desarrollo de una aplicación como la propuesta debido a la gran capacidad de cómputo que debe llevar asociada. Es por esto, que muchas aplicaciones comerciales de este tipo precisan de enormes infraestructuras, como puede ser un clúster de GPUs.

En cuanto al futuro de este proyecto, se continuará realizando nuevas simulaciones de este tipo de modelos con nuevos valores en los distintos hiperparámetros, se utilizarán otro tipo de decaimientos de la tasas de aprendizaje como lineales o polinómicos, y también se usará un decaimiento en los pesos de los términos de la función de pérdida del generador como hicieron en [41] para, a medida que avance el entrenamiento, disminuir la contribución del término adversario y dar prioridad a los términos del ciclo de consistencia. Por otro lado se investigará la viabilidad de construir modelos de menor resolución como 64×64 o 128×128 y luego utilizar una *Super Resolution* GAN para aumentar la resolución de las imágenes generadas por la *cycle* GAN.

Bibliografía

[1] C. W. Team, R. Pachauri y L. M. (eds.), *IPCC, Climate Change 2014: Synthesis Report. Contribution of Working Groups I, II and III to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change*, IPCC, Geneva, Switzerland, 151 pp, 2014.

[2] H. Shaftel, S. Callery, R. Jackson y D. Bailey, *Climate Change: How Do We Know?*, https://climate.nasa.gov/evidence/, Earth Science Communications Team at NASA's Jet Propulsion Laboratory | California Institute of Technology, visitado 29/04/2021.

[3] H. Shaftel, S. Callery, R. Jackson y D. Bailey, *The Effects of Climate Change*, https://climate.nasa.gov/effects/, Earth Science Communications Team at NASA's Jet Propulsion Laboratory | California Institute of Technology, visitado 29/04/2021.

[4] I.P.C.C., J. Shukla, E. Buendia, V. Masson-Delmotte, H.-O. Pörtner, D. Roberts, P. Zhai, R. Slade, S. Connors, R. Diemen, M. Ferrat, E. Haughey, S. Luz, S. Neogi, M. Pathak, J. Petzold, J. Pereira, P. Vyas, E. Huntley, K. Kissick, M. Belkacemi y J. Malley, *Summary for Policymakers. In: Climate Change and Land: an IPCC special report on climate change, desertification, land degradation, sustainable land management, food security, and greenhouse gas fluxes in terrestrial ecosystems P.R*, en, 2019.

[5] S. Scienseed, *Communicating climate change and biodiversity to policy makers*, https://rm.coe.int/168064e897, 2016.

[6] W.-T. Chu, X.-Y. Zheng y D.-S. Ding, *Camera as weather sensor: Estimating weather information from single images*, https://www.sciencedirect.com/science/article/abs/pii/S1047320317300901, Journal of Visual Communication and Image Representation, 46, 233-249, 2017.

[7] W.-T. Chu, K.-C. Ho y A. Borji, *Visual Weather Temperature Prediction*, https://ieeexplore.ieee.org/document/8354136, 2018 IEEE Winter Conference on Applications of Computer Vision (WACV), 2018.

[8] L. Kang, K. Chou y R. Fu, *Deep Learning-Based Weather Image Recognition*, https://ieeexplore.ieee.org/document/8644946, 2018 International Symposium on Computer, Consumer and Control (IS3C), 384-387, 2018.

[9] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville e Y. Bengio, *Generative Adversarial Networks*, https://arxiv.org/pdf/1406.2661.pdf, 2014.

[10] A. Jabbar, X. Li y B. Omar, *A Survey on Generative Adversarial Networks: Variants, Applications, and Training*, https://arxiv.org/pdf/2006.05132.pdf, 2020.

[11] A. Radford, L. Metz y S. Chintala, *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*, https://arxiv.org/pdf/1511.06434.pdf, 2016.

[12] M. Arjovsky, S. Chintala y L. Bottou, *Wasserstein GAN*, https://arxiv.org/pdf/1701.07875.pdf, 2017.

[13] M. Mirza y S. Osindero, *Conditional Generative Adversarial Nets*, https://arxiv.org/pdf/1411.1784.pdf, 2014.

[14] M. Lee y J. Seok, *Controllable Generative Adversarial Network*, https://arxiv.org/pdf/1708.00598.pdf, 2019.

[15] A. Shoshan, N. Bhonker, I. Kviatkovsky y G. Medioni, *GAN-Control: Explicitly Controllable GANs*, https://arxiv.org/pdf/2101.02477.pdf, 2021.

[16] T. Karras, T. Aila, S. Laine y J. Lehtinen, *Progressive Growing of GANs for Improved Quality, Stability, and Variation*, https://arxiv.org/pdf/1710.10196.pdf, 2018.

[17] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang y W. Shi, *Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network*, https://arxiv.org/pdf/1609.04802.pdf, 2017.

[18] T. Karras, S. Laine y T. Aila, *A Style-Based Generator Architecture for Generative Adversarial Networks*, https://arxiv.org/pdf/1812.04948.pdf, 2019.

[19] L. A. Gatys, A. S. Ecker y M. Bethge, *Image Style Transfer Using Convolutional Neural Networks*, https://doi.org/10.1109/CVPR.2016.265, 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[20] P. Isola, J.-Y. Zhu, T. Zhou y A. A. Efros, *Image-to-Image Translation with Conditional Adversarial Networks*, https://arxiv.org/pdf/1611.07004.pdf, 2018.

[21] J.-Y. Zhu, T. Park, P. Isola y A. A. Efros, *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks*, https://arxiv.org/pdf/1703.10593.pdf, 2017.

[22] P. Singh y N. Komodakis, *Cloud-Gan: Cloud Removal for Sentinel-2 Imagery Using a Cyclic Consistent Generative Adversarial Networks*, https://ieeexplore.ieee.org/document/8519033, 2018.

[23] V. Schmidt, A. Luccioni, S. K. Mukkavilli, N. Balasooriya, K. Sankaran, J. Chayes e Y. Bengio, *Visualizing the Consequences of Climate Change Using Cycle-Consistent Adversarial Networks*, https://arxiv.org/pdf/1905.03709.pdf, 2019.

[24] G. Zaher y A. Hadju, *Simulating weather conditions on digital images*, http://ghaiszaher.me/Foggy-CycleGAN/dissertation/Simulating%20Weather%20Conditions%20on%20Digital%20Images%20-%20Final.pdf, 2020.

[25] H. Zhang, V. Sindagi y V. M. Patel, *Image De-raining Using a Conditional Generative Adversarial Network*, https://arxiv.org/pdf/1701.05957.pdf, 2019.

[26] W. T. Chu y L. W. Huang, *Weather Attribute-Aware Multi-Scale Image Generation with Residual Learning*, https://ieeexplore.ieee.org/abstract/document/9181357, 2020 Indo – Taiwan 2nd International Conference on Computing, Analytics and Networks (Indo-Taiwan ICAN), 238-243, 2020.

[27] D. P. Kingma y J. Ba, *Adam: A Method for Stochastic Optimization*, https://arxiv.org/pdf/1412.6980.pdf, 2017.

[28] *Keras: the Python deep learning API*, https://keras.io/.

[29] A. Odena, V. Dumoulin y C. Olah, *Deconvolution and Checkerboard Artifacts*, http://distill.pub/2016/deconv-checkerboard, 2016.

[30] S. Ioffe y C. Szegedy, *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*, https://arxiv.org/pdf/1502.03167.pdf, 2015.

[31] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever y R. Salakhutdinov, *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*, https://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf, Journal of Machine Learning Research, 15, 56, 1929-1958, 2014.

[32] X. Mao, Q. Li, H. Xie, R. Y. K. Lau, Z. Wang y S. P. Smolley, *Least Squares Generative Adversarial Networks*, https://arxiv.org/pdf/1611.04076.pdf, 2017.

[33] K. Cho, B. van Merrienboer, D. Bahdanau e Y. Bengio, *On the Properties of Neural Machine Translation: Encoder-Decoder Approaches*, https://arxiv.org/pdf/1409.1259.pdf, 2014.

[34] *TensorFlow: Una plataforma de extremo a extremo de código abierto para el aprendizaje automático*, https://www.tensorflow.org/?hl=es-419.

[35] N. Jacobs, W. Burgin, N. Fridrich, A. Abrams, K. Miskell, B. H. Braswell, A. D. Richardson y R. Pless, *The Global Network of Outdoor Webcams: Properties and Applications*, https://doi.org/10.1145/1653771.1653789, ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM SIGSPATIAL), 2009.

[36] N. Jacobs, N. Roman y R. Pless, *Consistent Temporal Variations in Many Outdoor Scenes*, https://doi.org/10.1109/CVPR.2007.383258, IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2007.

[37] W.-T. Chu, X.-Y. Zheng y D.-S. Ding, *Image2Weather: A Large-Scale Image Dataset for Weather Property Estimation*, https://ieeexplore.ieee.org/document/7545010, 2016 IEEE Second International Conference on Multimedia Big Data (BigMM), 137-144, 2016.

[38] D. Engin, A. Genç y H. K. Ekenel, *Cycle-Dehaze: Enhanced CycleGAN for Single Image Dehazing*, https://arxiv.org/pdf/1805.05308.pdf, 2018.

[39] J. Johnson, A. Alahi y L. Fei-Fei, *Perceptual Losses for Real-Time Style Transfer and Super-Resolution*, https://arxiv.org/pdf/1603.08155.pdf, 2016.

[40] A. Baid, S. Varshini y V. Upadhya, *Cycle-GANs for Cloudy-Sunny Image Translations*, https://ayushbaid.github.io/files/cycle-gans.pdf, 2019.

[41] T. Wang e Y. Lin, *CycleGAN with Better Cycles*, https://ssnl.github.io/better_cycles/report.pdf, 2017.