Universidad Nacional de Rosario

Tesina de Grado
para la obtención del grado de
Licenciado en Ciencias de la Computación

Invertibilidad de un Generador Entrenado Adversariamente

Autor
Marcos Pividori

Director
Dr. Lucas Uzal

Co-Director
Dr. Guillermo Grinblat

Departamento de Ciencias de la Computación
Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Av. Pellegrini 250, Rosario, Santa Fe, Argentina

16 de noviembre de 2019

Resumen

Las Redes Adversarias Generativas (GAN) han demostrado resultados excepcionales en el modelado de la distribución de imágenes naturales, aprendiendo representaciones latentes que capturan variaciones semánticas sin supervisión. Además de la generación de imágenes nuevas, es de especial interés explotar la capacidad del generador GAN para modelar el manifold de las imágenes naturales y, por lo tanto, generar cambios creíbles al manipular imágenes. Sin embargo, esta línea de trabajo está condicionada por la calidad de las reconstrucciones obtenidas sobre las imágenes reales al proyectarlas al espacio latente. Mientras que trabajos previos solo han considerado la inversión hasta el espacio latente, en este trabajo proponemos explotar la representación en las capas intermedias del generador, y mostramos que esto conduce a una mayor capacidad. En particular, observamos que la representación después de la primera capa densa, presente en todos los modelos GAN del estado del arte, es lo suficientemente expresiva como para representar imágenes naturales con gran fidelidad visual. Es posible interpolar entre estas imágenes obteniendo una secuencia de nuevas imágenes sintéticas de gran calidad que no se pueden generar desde el espacio latente. Finalmente, como ejemplo de aplicaciones potenciales que surgen de este mecanismo de inversión, mostramos que se puede explotar la representación aprendida en el mapa de atención del generador para obtener una segmentación no supervisada de imágenes naturales.

Agradecimientos

Este trabajo realmente no hubiera sido posible sin el acompañamiento de muchas personas a lo largo de estos años.

Quiero agradecer a mis padres, por las inmensas oportunidades que me dieron, las condiciones para estudiar, la libertad de elegir y el acompañamiento constante, día a día. De igual manera a mis hermanos, que siempre estuvieron al lado, escuchando mis charlas monótonas de "Machine Learning".

A la comunidad de los Focolares, que son mi segunda familia, por enseñarme a vivir por un ideal, apostar por la unidad en la diversidad, descubrir el valor único de cada persona en su individualidad.

A mis amigos en general, por tantos momentos extraordinarios, que resultaron motor y motivación para todas las demás cosas.

A mis compañeros de la facultad, la "banda del Pelle", por todos estos años de estudio, los desayunos en el bar de la facu, los debates intelectuales y no tanto, las anécdotas de clase, las reuniones en el DCC.

A los tutores Lucas y Guille, por su gran disponibilidad y acompañamiento, por compartir el entusiasmo en los temas investigados. A todo el CIFASIS, por las posibilidades que me dieron, en recursos y en integración.

Inmensamente agradecido a todos los profesores de la carrera, particularmente Ana, Dante, Eric, Pablo, Mauro, Guido. Me llevo el aprendizaje de apuntar alto, la disciplina y perseverancia como camino a los resultados, la importancia del rigor científico, el compromiso de los profesores y su relación cercana, y sobre todo, los más importante, el seguir una pasión, disfrutar lo que hacemos, realmente apasionarse.

Con muchos de ellos tuve la oportunidad de descubrir el lado docente, Fede, Dante, Pablo, Natalia, Pamela. Gracias por enseñarme el valor de donar los conocimientos a los demás y apostar a construir el ambiente de la facultad.

Índice general

Capítulo 1

Introducción

1.1. Objetivo

Hoy en día tenemos acceso a muchísimos datos, fácilmente disponibles, el principal desafío es desarrollar modelos y algoritmos que puedan analizar y comprender la información contenida. En esta dirección se enfoca el área de *Aprendizaje No Supervisado*, donde se intenta descubrir la estructura subyacente en los datos.

A diferencia del Aprendizaje Supervisado, en este caso no se cuenta con la ayuda de una señal supervisora que determine la respuesta correcta. En general las etiquetas de los datos poseen relativamente poca información, que no resulta útil para determinar los parámetros de modelos complejos como los que se intentan construir. Por lo tanto, es necesario aprender directamente de los datos en concreto.

Los *Modelos Generativos* son unos de los más prometedores en lograr este objetivo. Toman un conjunto de entrenamiento que consiste en una gran cantidad de datos reales (millones de imágenes, oraciones, sonidos, etc) tomados de una distribución dada P_{data} y aprenden a representar una estimación de esa distribución, resultando en una distribución probabilística P_{model}. En algunos casos esta se modela explícitamente, en otros solo es posible generar muestras de acuerdo a P_{model}.

La intuición detrás de este enfoque es que, en la medida en que se aprenda cómo generar los datos, se tendrá una mejor comprensión de las características intrínsecas de los mismos. Los recientes avances en el entrenamiento de redes neuronales como aproximadores de funciones, potenciadas por el algoritmo de Backpropagation, dieron lugar a muchas propuestas en el uso de redes neuronales para construir modelos generativos, los que suelen llamarse *Modelos Generativos Profundos*. Entre los resultados más prometedores, las *Redes Adversarias Generativas* (GAN, por sus siglas en inglés) [1] han sido el principal foco de investigación en los últimos años debido a que, entre otros motivos, han demostrado ser muy efectivas para generar imágenes sintéticas.

Las GAN establecen el problema como un juego entre dos redes: un generador encargado de producir datos sintéticos a partir de un vector z de ruido dado como entrada, y un discriminador que diferencia entre los datos generados sintéticamente por el generador y los datos reales tomados de un conjunto de entrenamiento.

La particularidad de las GAN respecto a otros modelos generativos basados en redes neuronales es que el generador (que es simplemente una red neuronal multicapa), una vez entrenado, transforma determinísticamente un vector z de ruido, que se inyecta en la entrada de la red, en una imagen en la salida de la red. Es decir que la generación no implica ningún proceso iterativo de optimización para cada imagen sino que basta una sola propagación a lo largo de la red neuronal. Toda la variabilidad en la salida de la red es capturada por el vector z. No hay procesos estocásticos en la transformación.

En este contexto, resulta de especial interés estudiar el espacio latente aprendido por el generador: la relación semántica entre imágenes correspondientes a valores de z cercanos, las transformaciones causadas por desplazamientos en una dada dirección en el espacio de entrada, etc. Un primer antecedente de este tipo de análisis es el realizado por Radford y col. [2] quienes muestran que operaciones aritméticas básicas de suma y resta sobre los vectores z tienen un correlato semántico en las imágenes generadas. A pesar de los resultados sorprendentes en datasets limitados a un tipo particular de situación, como ser imágenes de dormitorios, rostros humanos, etc, los modelos GAN actuales no logran producir resultados convincentes cuando son entrenados en datasets de alta variabilidad, aún cuando se trabaja con datasets de imágenes de baja resolución, como CIFAR-10 [3] (de 32×32 píxeles) (si bien trabajos recientes permitieron una mejora muy importante en la formulación condicional [4]).

En esta tesina deseamos analizar la dificultad del generador en aproximar la distribución real de las imágenes, en particular en relación al rol que cumple la primera capa densa (característica presente en todas las arquitecturas del estado del arte en GAN). La propuesta es poder comparar ambas distribuciones (P_{data}, P_{model}) no solo a nivel de píxeles, sino en cada representación a lo largo de la profundidad de la red. Para esto, proponemos partir de las imágenes reales (trabajando con los principales datasets en el estado del arte, como ser CIFAR-10 [3] y ImageNet [5]) y buscar una representación para las mismas en cada capa del generador, desde el espacio de salida hacia atrás, y de esta manera analizar si existe un valor latente en cada punto de la red que permita reconstruir la imagen original. A través de este procedimiento podremos determinar hasta qué punto son invertibles las diferentes capas del generador.

Para invertir el generador, se estudiarán diferentes algoritmos, incluyendo recientes resultados en plantear el problema como una optimización sobre el generador, haciendo descenso por el gradiente respecto a la entrada del mismo.

1.2. Organización

El trabajo está organizado de la siguiente manera:

El Capítulo 2 introduce los conceptos propios del Aprendizaje Profundo necesarios para entender la tesina y el Capítulo 3 revisa las distintas alternativas presentes en el estado del arte para invertir un generador GAN.

En el Capítulo 4 se analiza el problema de invertir el generador en diferentes niveles, detallando el algoritmo utilizado y las dificultades a tener en cuenta.

A continuación, en los Capítulos 5, 6 y 7 se presentan los resultados experimentales sobre tres formulaciones distintas de GAN, que cubren las principales alternativas presentes en el estado del arte: Unconditional GAN sobre la arquitectura DCGAN para el dataset CIFAR-10 (Capítulo 5), Conditional GAN con la arquitectura BigGAN para el dataset ImageNet (Capítulo 6), y un generador ProGAN entrenado con Crecimiento Progresivo sobre el dataset CelebA-HQ de rostros humanos (Capítulo 7).

El Capítulo 8 explora distintas aplicaciones prácticas de los resultados obtenidos, estableciendo posibles trabajos a futuro.

Finalmente, el Capítulo 9 compone un resumen final de lo investigado.

Capítulo 2

Conceptos Generales

2.1. Aprendizaje Automatizado

Hoy en día estamos inmersos en lo que se conoce como la *Era del Big Data*, donde se tiene acceso a volúmenes de datos masivos de un tamaño sin precedentes [6]. Por ejemplo, se estima que existen alrededor de mil millones de páginas web; cada minuto, se incorporan más de 300 horas de video en YouTube, lo que equivale a 50 años de contenido todos los días [7]; cada minuto en Facebook se publican 510.000 comentarios, se actualizan 293.000 estados y se suben 136.000 fotos [8]; Walmart maneja más de 1 millón de transacciones por hora y tiene bases de datos que contienen más de 2,5 petabytes ($2{,}5 \times 10^{15}$) de información, etc.

Este diluvio de datos requiere métodos automatizados de análisis, que es lo que proporciona el *Aprendizaje Automatizado* (*Machine Learning*).

Un algoritmo de Aprendizaje Automatizado es un algoritmo que puede aprender de los datos, es decir, detectar patrones automáticamente, y luego usarlos para predecir datos futuros o realizar otros tipos de toma de decisiones bajo incertidumbre [6]. De manera más precisa, Mitchell [9] propone la siguiente definición:

> *"Se dice que un programa de computadora aprende de la experiencia E con respecto a alguna clase de tareas T y la medida de rendimiento P, si su desempeño en las tareas T, medido por P, mejora con la experiencia E"*

El Aprendizaje Automatizado nos permite abordar tareas que son demasiado difíciles de resolver con programas escritos explícitamente y diseñados por seres humanos.

Algunas tareas no se pueden definir bien excepto a través de ejemplos. Es decir, puede suceder que seamos capaces de especificar pares de entrada/salida pero no una relación concisa entre las entradas y las salidas deseadas. Quisiéramos que las máquinas puedan ajustar su estructura interna para pro-

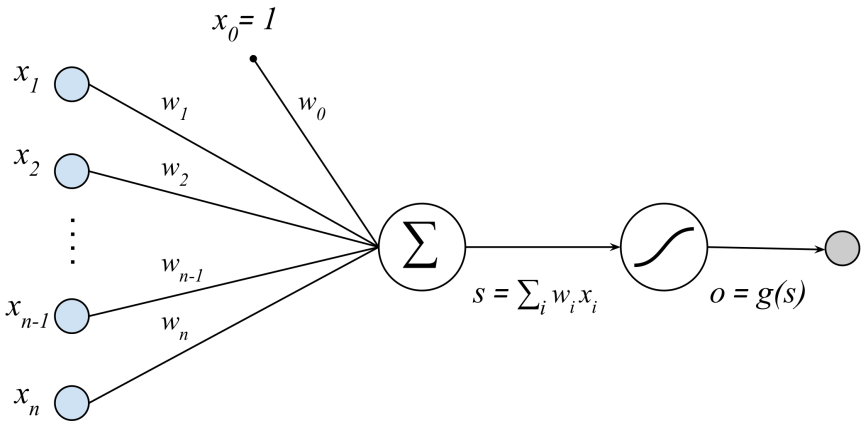


Figura 2.1: Perceptrón. La información fluye de izquierda a derecha, los valores de entrada (x_i) se combinan linealmente de acuerdo a los pesos entrenados (w_i) y luego se aplica una función de activación (g). $x_0 = 1$ representa un valor constante que permite establecer un sesgo w_0 (también notado b en referencia a *bias*).

ducir salidas correctas y aproximar la relación implícita en los ejemplos. Esto comprende el área de *Aprendizaje Supervisado*.

Por otro lado, es posible que dentro de los grandes volúmenes de datos estén ocultas relaciones importantes. Los métodos de Aprendizaje Automatizado se pueden usar para entender la estructura fundamental de los datos y extraer estas relaciones. Esto abarca el área de *Aprendizaje No Supervisado*.

Algunas de las tareas más comunes en las que se aplica Aprendizaje Automatizado [10] incluyen: clasificación, regresión, traducción automática, detección de anomalías, síntesis y muestreo, etc.

2.2. Redes Neuronales Artificiales

Las *Redes Neuronales Artificiales* [9] (RNA) son una familia de modelos que proveen un enfoque robusto para aproximar funciones a valores reales, discretos y vectoriales.

El estudio de las RNA se ha inspirado en parte por la observación de que los sistemas de aprendizaje biológicos se construyen a partir de redes muy complejas de neuronas interconectadas. Sin embargo, la investigación moderna está guiada por disciplinas matemáticas y de ingeniería, y el objetivo de las RNA no es modelar perfectamente el cerebro.

Las RNA están compuestas por unidades llamadas neuronas o perceptrones. Cada neurona toma como entrada un vector a valores reales, computa una combinación lineal de los mismos y luego produce una salida, que es una función de dicha combinación y representa el grado de activación de la neurona

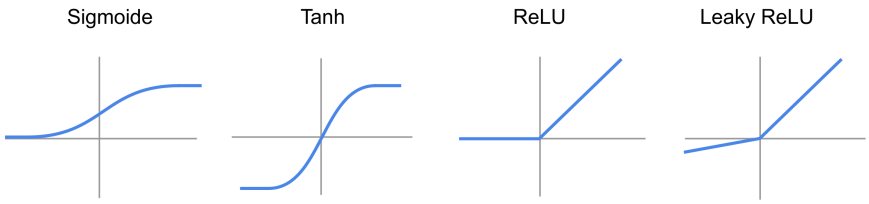


Figura 2.2: Diferentes alternativas para la función de activación de una neurona artificial. Históricamente, las funciones sigmoide y tanh surgieron como primeras alternativas. Recientemente, las ReLUs y sus variantes adquirieron mayor popularidad, mostrando resultados superadores en el entrenamiento de redes profundas.

(Figura 2.1). Formalmente:

$$o(x) = g\left(\sum_i w_i\ x_i + b\right)$$

donde los valores w_i representan el peso que se asigna a cada entrada de la neurona, b es un valor constante llamado sesgo (bias) (también notado w_0) y la función de activación g, que modifica el valor resultado o impone un umbral que se debe sobrepasar antes de propagarse a otra neurona.

2.2.1. Funciones de Activación

La aplicación de la función de activación sobre la combinación lineal de la entrada permite componer neuronas aumentando el poder expresivo hacia funciones no lineales. De otra manera, simplemente componer combinaciones lineales se reduciría a una simple combinación lineal, y modelos de múltiples capas no permitirían tener modelos más expresivos.

Históricamente, una opción común para la función de activación fue la función *sigmoide* σ, que toma una entrada de valor real y la ajusta para que se mantenga en el rango $[0, 1]$:

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$

Un problema importante con este tipo de activaciones al componer múltiples capas de neuronas es que saturan el gradiente, es decir, cuando el valor de entrada se aleja del origen, el gradiente de la función de activación respecto a la entrada se acerca rápidamente a cero, impidiendo que fluya la señal hacia las capas anteriores en el proceso de propagación hacia atrás.

Si bien se han propuesto distintas variantes para la función de activación (Figura 2.2), en las implementaciones modernas la función más comúnmente utilizada es la *Rectified Linear Unit* (ReLU), definida como:

$$\text{ReLU}(x) = \max(0, x)$$

En [11] muestran que las neuronas con ReLUs representan un mejor modelo de las neuronas biológicas y producen un rendimiento tan bueno o aún mejor que las redes con activaciones sigmoide o tanh, aprendiendo representaciones dispersas, lo que presenta varias ventajas desde el punto de vista computacional.

En la práctica, se encontró que las ReLUs aceleran en gran medida la convergencia del descenso por el gradiente estocástico en comparación con las funciones sigmoide o tanh (por ejemplo, 6 veces más rápido en [12]). Se argumenta que esto se debe a su forma lineal, que permite que el gradiente fluya sin problemas por los caminos activos de la red, sin el problema de la saturación del gradiente presente en las otras alternativas.

Además, a diferencia de las funciones tanh y sigmoide, que involucran operaciones costosas (exponenciales), las ReLUs se pueden implementar eficientemente simplemente con un umbral en cero sobre las activaciones.

Una desventaja de las ReLUs es que no permiten aprender por descenso por el gradiente en los casos donde su activación es nula. Es decir, el gradiente no fluye cuando la neurona no está activada. Incluso, puede resultar que por la actualización de los pesos durante el entrenamiento, algunas neuronas no se vuelvan a activar nunca para ninguna entrada, llamadas neuronas muertas. De esta manera, parte de la red se mantiene constantemente apagada independientemente de los datos de la entrada.

Las *Leaky ReLUs* son un intento de solucionar este problema, permitiendo que el gradiente siempre fluya. En lugar de que la función sea cero cuando el valor de entrada es negativo, una Leaky ReLU tendrá una pequeña pendiente negativa (por ejemplo: 0.01). Es decir, la función calcula:

$$\text{LeakyReLU}(x) = \begin{cases} \alpha\ x, & \text{si } x < 0. \\ x, & \text{si no.} \end{cases}$$

donde α es una pequeña constante.

En algunos casos se han logrado mejores resultados con estas funciones de activación, pero los resultados no siempre son consistentes.

2.2.2. Organización de las neuronas

Cada neurona está interconectada con otras a través de enlaces que transmiten señales, formando una red, donde los pesos asociados a estos enlaces pueden incrementar o inhibir el estado de activación de las neuronas adyacentes. La información de entrada atraviesa la red neuronal, siendo sometida a diferentes operaciones y finalmente produciendo valores de salida.

Estos modelos se dicen *feedforward* porque la información fluye desde la entrada hasta la salida sin conexiones de retroalimentación. Por lo tanto, el modelo tiene asociado un grafo acíclico dirigido que describe cómo se componen las diferentes funciones de manera conjunta.

La palabra *arquitectura* se refiere a la estructura general de la red: cuántas unidades debe tener y cómo deben conectarse entre sí.

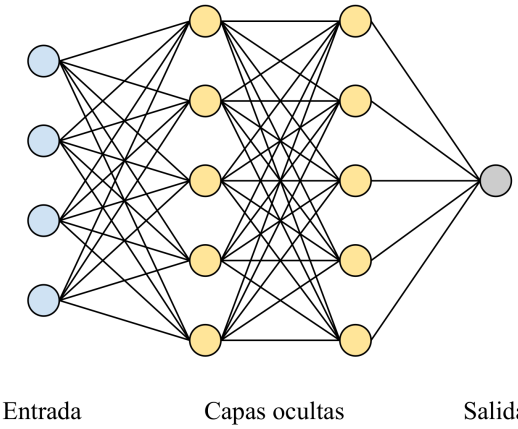


Figura 2.3: Organización en capas de la redes neuronales artificiales.

La mayoría de las redes neuronales feedforward están organizadas en grupos de unidades llamadas *capas*, en una estructura en cadena, donde cada capa es una función de la capa que la precedió (Figura 2.3).

La primera capa está asociada a la entrada de la red, la última a la salida, y el resto son llamadas capas ocultas, la cuales aprenden múltiples representaciones intermedias de los datos (a medida que se propagan a lo largo de la red) y corresponden a diferentes niveles de abstracción.

El término *profundidad* se refiere al número de capas que componen la red.

2.2.3. Aproximador universal

El objetivo de una red neuronal es aproximar una determinada función f. Por ejemplo, para un clasificador, se intenta aproximar: $y = f(x)$ que asigna a cada entrada x una categoría y.

Se puede demostrar que una red neuronal con una única capa intermedia es un aproximador universal, es decir, puede modelar cualquier función continua en un espacio compacto de $\mathbb{R}^n$, a un grado de precisión arbitrario, dado un número suficiente de neuronas intermedias [13].

Sin embargo, la capa oculta necesaria puede ser demasiado grande, requiriendo un número exponencial de unidades en el peor caso.

Además, no existen garantías de que el algoritmo de entrenamiento pueda aprender esa función. Aún si la red es capaz de representar la función, el aprendizaje puede fallar porque el algoritmo de optimización no sea capaz de encontrar el valor de los parámetros que corresponden a la función deseada, o elija la función incorrecta al sobreajustar los datos de entrenamiento. No existe un procedimiento universal para elegir una función que generalice a puntos fuera del conjunto de entrenamiento [14].

En muchas circunstancias, el uso de modelos más profundos permite reducir el número de unidades requeridas para representar la función deseada y también reducir el error de generalización [10], como se ha comprobado empíricamente en diferentes tareas.

2.2.4. Entrenamiento

Consideremos primero el caso de una simple neurona sin función de activación:

$$o(x) = \sum_i w_i \; x_i$$

Entrenar esta neurona implica elegir valores para los pesos w_0 ... w_n. Por lo tanto, el espacio de hipótesis candidatas $\mathcal{H}$ es el conjunto de todos los posibles vectores de pesos en valores reales:

$$\mathcal{H} = \{w \mid w \in \mathbb{R}^{n+1}\}$$

Entonces, entrenar el modelo se reduce a buscar la hipótesis más adecuada del espacio de hipótesis $\mathcal{H}$, de acuerdo a un criterio. Este criterio dependerá del objetivo y el método de entrenamiento. Por ejemplo, en aprendizaje supervisado para regresión sobre un conjunto de datos D, una opción frecuente es el error cuadrático definido como:

$$J(w) = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

donde t_d es la salida esperada para la muestra d y o_d es la salida de la neurona.

Se puede demostrar que minimizar el error cuadrático equivale a buscar la hipótesis de *máxima likelihood*, bajo la asunción de que los datos de entrenamiento fueron independientemente perturbados con ruido gaussiano de media cero.

En cambio, si el objetivo es clasificación, otras opciones frecuentes para el error incluyen *cross entropy* (también se reduce de maximizar la likelihood) o *hinge loss*.

Dada una definición del error, entrenar el modelo consiste en una optimización sobre el espacio de hipótesis, partiendo de un punto aleatorio y reduciendo iterativamente el error de entrenamiento.

Descenso por el gradiente

Si la función J es derivable, podemos computar el gradiente, resultado de obtener las derivadas parciales de J respecto a cada componente de w, como:

$$\nabla J(w) = \left(\frac{\partial J}{\partial w_0}, \dots , \frac{\partial J}{\partial w_n} \right)$$

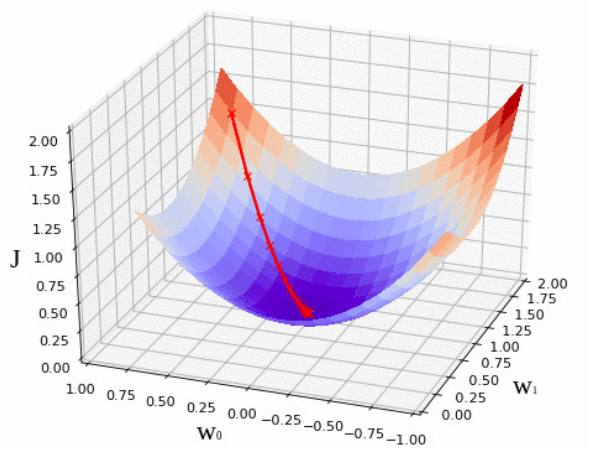


Figura 2.4: Ejemplo de descenso por el gradiente sobre un error J (función convexa) con respecto a los parámetros w_i del modelo.

$\nabla J(w)$ es en efecto un vector que, cuando se lo interpreta en el espacio de pesos, representa la dirección que produce el mayor incremento en J. De manera opuesta, $-\nabla J(w)$ determina la dirección de mayor decremento.

Por lo tanto, la regla de entrenamiento de descenso por el gradiente consiste en actualizar los pesos en cada punto moviéndose en la dirección opuesta al gradiente:

$$w \leftarrow w - \eta \, \nabla J(w)$$

donde η es una constante positiva llamada *learning rate*, que determina cuánto se modifica w en cada iteración del algoritmo.

Por ejemplo, para el caso del error cuadrático, podemos obtener el gradiente calculando las derivadas parciales como:

$$\begin{aligned}
\frac{\partial J}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \\
&= \sum_{d \in D} (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \sum_i w_i x_{di}) \\
&= \sum_{d \in D} (t_d - o_d)(-x_{di})
\end{aligned}$$

El algoritmo converge cuando cada componente del gradiente es cero (o muy cercano a cero), es decir, cuando alcanza un punto crítico.

Optimalidad de la solución

Cuando la función a optimizar es convexa, como la superficie de error parabólica al minimizar el error cuadrático de una neurona lineal, todos los mínimos locales también son mínimos globales, por lo que en este caso el

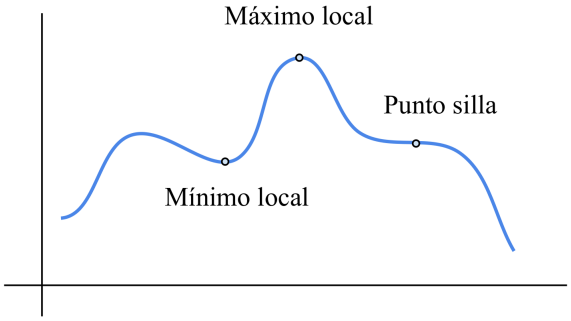


Figura 2.5: Distintos puntos críticos que pueden ocurrir sobre la función de costo. Históricamente se creyó que los mínimos locales eran un problema común que afectaba la optimización de las redes neuronales. Hoy en día esto parece no ser así, en general se estima que para redes neuronales suficientemente grandes la mayoría de los mínimos locales tienen un valor bajo, donde no es importante encontrar un verdadero mínimo global mientras se encuentre un punto en el espacio de parámetros que tenga un bajo costo [10]. Para muchas funciones no convexas de alta dimensionalidad, los mínimos locales de hecho son mucho menos comunes que los puntos silla [15].

algoritmo de descenso por el gradiente puede converger a la solución global. Es decir, sabemos que hemos llegado a una buena solución cuando encontramos un punto crítico.

En cambio, la no linealidad presente en las redes neuronales multicapa hace que las funciones a optimizar se vuelvan no convexas, donde hay múltiples mínimos locales o zonas planas subóptimas (Figura 2.5). Por lo tanto, el descenso por el gradiente no tiene dichas garantías de convergencia y es sensible a la inicialización de los parámetros (punto de partida).

Sin embargo, aún si no se garantiza que el algoritmo de optimización llegue a una solución óptima, a menudo encuentra un valor muy bajo de la función de costo lo suficientemente rápido como para ser útil. En el contexto del Aprendizaje Profundo, generalmente se aceptan tales soluciones aunque no sean realmente óptimas.

Aproximación Estocástica

En general, para que los modelos generalicen adecuadamente, se necesita entrenar sobre grandes datasets de entrenamiento. Sin embargo, esto implica mayores costos computacionales. Por ejemplo, para computar el gradiente del error cuadrático, debemos visitar cada elemento del dataset en cada iteración de la optimización, lo cual puede resultar muy ineficiente.

Entonces, en vez de computar el gradiente del error sobre todos los ejemplos de entrenamiento, el algoritmo de *descenso por el gradiente estocástico* consiste

en tomar una muestra pequeña aleatoria de m elementos (*minibatch*), y estimar el gradiente original computando el gradiente del error sobre esta muestra.

Por ejemplo, para el caso del error cuadrático, cada paso de la optimización consistirá en tomar un minibatch B del dataset de entrenamiento y realizar la actualización:

$$w \leftarrow w - \eta \, \nabla J_B(w)$$

donde: $J_B(w) = \frac{1}{2} \sum_{d \in B} (t_d - o_d)^2$ es el error sobre el minibatch $B \subseteq D$ con $|B| = m$.

Algoritmo de Backpropagation

El procedimiento de *Backpropagation* [16] permite computar de manera eficiente el gradiente de una función con respecto a sus parámetros en una red neuronal multicapa feedforward.

Cuando evaluamos una red que acepta una entrada x y produce una salida y, la información fluye hacia adelante a través de la red. La entrada x provee la información inicial que se propaga a través de las capas ocultas de la red hasta producir el valor de salida y (*forward propagation*). Durante el entrenamiento, la propagación continúa incluso hasta computar el valor de la función costo a optimizar.

El algoritmo de Backpropagation permite que la información de la función de costo fluya hacia atrás a través de la red, computando el gradiente respecto a cada parámetro de manera eficiente [10].

Backpropagation no es más que una aplicación práctica de la regla de la cadena para las derivadas. El punto clave es que la derivada (o gradiente) de la función objetivo con respecto a la entrada de una capa (x) puede ser computada utilizando el gradiente con respecto a la salida de la misma ($f(x)$):

$$\frac{\partial L(f(x))}{\partial x} = \frac{\partial L(f(x))}{\partial f(x)} \, \frac{\partial f(x)}{\partial x}.$$

De esta manera, se aplica este proceso repetidamente para propagar el gradiente a través de cada capa, empezando de la salida de la red hasta la entrada de la misma.

Una vez que los gradientes respecto a cada capa han sido calculados, se puede computar de manera simple el gradiente respecto a los parámetros de las mismas [17].

Esto constituye la base de los frameworks modernos de aprendizaje automatizado, que permiten expresar un modelo a través de un grafo computacional del cual se calcula de manera automática el gradiente respecto a cada parámetro a través del algoritmo de Backpropagation.

Extensiones

Una vez computado el gradiente, existen diferentes extensiones al algoritmo de descenso por el gradiente que modifican la forma en que los parámetros se actualizan.

Momentum. Consiste en acumular una media móvil de los gradientes en las iteraciones pasadas con decaimiento exponencial y continuar moviéndose en esa dirección. Intuitivamente, se puede pensar al proceso de optimización como el movimiento de una partícula sobre la superficie de error. Esta partícula rueda a una velocidad igual al learning rate por el módulo del gradiente y en cada instante cuenta con una inercia asociada. De esta manera, se logra reducir el riesgo de quedar atrapado en un mínimo local (la inercia permite continuar en movimiento y escapar en muchos casos) y se acelera la convergencia considerablemente.

Learning rate adaptativo. El learning rate resulta uno de los parámetros que es más difícil de establecer porque tiene un impacto significativo en la optimización del modelo. Por lo tanto, algunas extensiones proponen adaptar de manera automática el learning rate a lo largo del entrenamiento. Por ejemplo, en Adam [18] se consideran valores del learning rate independientes para cada parámetro que se adaptan durante el entrenamiento a partir de la estimación del primer y segundo momento del gradiente. Como resultado, a pesar de ser relativamente simple, este optimizador tiene muy buenas propiedades (como por ejemplo, invariancia a la escala del gradiente) que lo hacen apropiado para problemas de grandes datos y números de parámetros, siendo una de las principales opciones en el estado del arte de Aprendizaje Profundo.

2.3. Arquitecturas Profundas

Muchos problemas de Inteligencia Artificial pueden resolverse a través del diseño de un conjunto apropiado de características (*features*) para la tarea que se desea realizar y luego simplemente aplicar un algoritmo de Aprendizaje Automatizado sobre estos features.

Por ejemplo, si deseáramos identificar automóviles en una imagen, los datos de entrada como valores de intensidad en los píxeles no resultan por sí mismos muy informativos. En cambio, el problema se simplificaría si pudiéramos de alguna manera obtener una representación de más alto nivel de la imagen, por ejemplo, que entre otras cosas nos determine si hay ruedas presentes en la misma.

Sin embargo, en general, para tareas complejas resulta difícil saber qué features deben ser extraídos, o cómo expresarlos algorítmicamente. Durante mucho tiempo, la construcción de sistemas de Aprendizaje Automatizado requirió una ingeniería cuidadosa y una considerable experiencia de dominio

para diseñar la extracción de features que transforme los datos sin procesar en una representación interna adecuada, desde la cual el subsistema de aprendizaje, como un clasificador, pudiera detectar o clasificar patrones en la entrada [17].

Como solución a este problema, el área de *Representation Learning* investiga cómo lograr que la máquina, a través del Aprendizaje Automatizado, aprenda no solo la función objetivo, sino también la representación más adecuada de los datos para la tarea a realizar [19].

La experiencia muestra que las representaciones aprendidas de manera automática en general resultan ser mucho más efectivas que representaciones cuidadosamente diseñadas por un experto humano. Al mismo tiempo, el proceso de aprendizaje automatizado, debidamente guiado, resulta mucho más eficiente comparado con el tiempo que puede llevar a un conjunto de investigadores identificar la representación adecuada para una tarea, y, en general, las representaciones aprendidas resultan ser más relevantes y adaptables a múltiples tareas.

El *Aprendizaje Profundo* propone aprender la representación adecuada de los datos a través de la introducción de representaciones jerárquicas que son expresadas en término de otras representaciones más simples, aumentando progresivamente en abstracción a medida que se aumenta la profundidad, construyendo conceptos complejos a partir de conceptos más simples [10].

El punto fundamental es que estas capas de features no son diseñadas por humanos, sino que se aprenden directamente de los datos usando un procedimiento general [17]. El propósito de las capas ocultas en la red es ir transformando la entrada, a través de extracción de patrones con transformaciones no lineales, en diferentes representaciones, donde cada capa aprende a transformar los datos en un nivel de abstracción un poco más alto [6].

Entonces, las redes neuronales profundas se pueden entender como una extensión de los modelos lineales para representar funciones no lineales de la entrada, donde, en lugar de aplicar un modelo lineal directamente sobre los datos x, lo aplicamos sobre la entrada transformada $\phi(x)$, por una transformación no lineal. La estrategia de Aprendizaje Profundo es *aprender* la transformación $\phi(x)$, que se puede interpretar que proporciona un conjunto de características que describen a x, o una nueva representación para x [10].

2.3.1. Historia

Históricamente, ha habido tres grandes períodos de desarrollo del Aprendizaje Profundo [10].

Las primeras investigaciones (40s-60s) se concentraron en modelos simples lineales de las redes neuronales inspirados en los avances de la neurociencia. Cuando las limitaciones de estos modelos se hicieron evidentes (por ejemplo, la incapacidad de representar la función lógica XOR [20]), se produjo una caída importante en la popularidad.

Posteriormente, en los 80s se produjo un interés renovado en esta área, bajo la idea central de que un gran número de pequeñas unidades de cómputo podrían lograr un comportamiento inteligente cuando están conectadas en red.

Esto aportó nuevos conceptos como el de *representación distribuida* (cada entrada al sistema debe ser representada por muchos features, y cada uno de estos features debe estar involucrado en la representación de múltiples entradas).

Otro avance importante fue la aplicación con éxito del algoritmo de Backpropagation para entrenar redes un poco más profundas con capas ocultas, y el progreso en el modelado de secuencias con redes neuronales (*redes recurrentes*).

Frente a estos resultados prometedores, empresas basadas en redes neuronales y otras tecnologías de inteligencia artificial comenzaron a hacer afirmaciones ambiciosas poco realistas con el objetivo de obtener inversiones. Como los resultados de las investigaciones en redes neuronales no cumplieron las expectativas desmedidas de los inversores, se produjo otra caída de popularidad en comparación a otros enfoques de aprendizaje automatizado con mejores resultados que se volvieron el principal foco de la comunidad científica. A este punto, las redes profundas se consideraban muy difíciles de entrenar.

Finalmente, una nueva ola de investigación comenzó cuando en 2006 se mostró que era posible entrenar una red profunda (*Deep Belief* [21]) usando una estrategia de pre-entrenamiento no supervisado sobre los datos. A partir de este punto, diferentes grupos de investigación lo extendieron a nuevos modelos, y empezó a tomar popularidad el término Aprendizaje Profundo (*Deep Learning*) para hacer referencia al hecho de que finalmente estaba siendo posible entrenar redes más profundas y la importancia del concepto de profundidad [10].

Gran parte del éxito de estos nuevos intentos en entrenar redes más profundas se debió a la disponibilidad de datos masivos para el entrenamiento y gigantes incrementos en los recursos de cómputo respecto a décadas anteriores.

De esta manera, las redes profundas fueron superando a los métodos tradicionales en múltiples tareas retomando la atención de la comunidad científica y despertando un gran interés desde el punto de vista comercial.

2.3.2. Redes Convolucionales

Las *Redes Neuronales Convolucionales* [22], una clase de red neuronal que es particularmente adecuada para procesar imágenes, fueron desarrolladas principalmente por LeCun y col. [22] a fines de los ‘80, donde algunos de los principios claves en el diseño fueron tomados de la neurociencia, en particular de estudios previos en cómo funciona la corteza visual primaria.

Las primeras implementaciones resultaron ser exitosas para procesar imágenes en pequeña escala, como reconocimiento de caracteres, pero las limitaciones en la capacidad de cómputo de esa época impidieron que escalen a tareas de mayor complejidad. En 2012, potenciados por el poder de cómputo de las GPUs, un nueva red convolucional más profunda propuesta por Krizhevsky

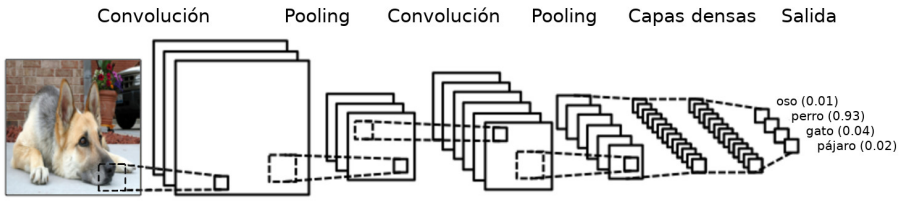


Figura 2.6: Arquitectura típica de una red convolucional para clasificación de imágenes. Sobre la imagen de entrada se aplican un conjunto de capas convolucionales intercaladas con operaciones de pooling. Las últimas capas por lo general son de conectividad total, donde se combinan todos los features presentes en la imagen para determinar el valor final de salida.

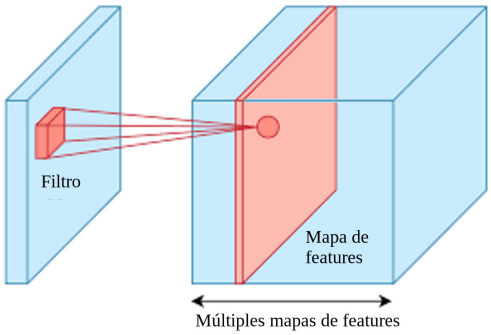


Figura 2.7: Estructura de una capa convolucional. Para cada mapa de features de salida, un mismo filtro se desplaza espacialmente a lo largo y ancho de la entrada, computando los valores de activación.

et al [12] ganó la competencia 2012 ImageNet Challenge por amplio margen (16.4 % de error comparado con el resultado en segundo lugar de 26.1 %), reviviendo el interés de la comunidad científica. Desde ese momento, las redes convolucionales han probado ser muy exitosas en múltiples tareas de visión por computadora [12, 23, 24, 25].

Capas convolucionales

En una red convolucional, las neuronas ocultas tienen campos receptivos locales, es decir, una conectividad limitada a una porción de la entrada, y los pesos están vinculados o compartidos en la imagen, para reducir el número de parámetros. De manera intuitiva, el efecto de dicha vinculación de parámetros espaciales es que cualquier característica útil que se descubra en alguna parte de la imagen se puede reutilizar en cualquier otro lugar sin tener que ser aprendida de forma independiente [6] (Figura 2.6).

De esta manera, las unidades ocultas se agrupan en *mapas de features*, donde todas comparten el mismo vector de pesos $w = (w_0, w_1, ..., w_n)$ (filtro), pero cada una tiene una conectividad local distinta, resultado de desplazar el filtro en diferentes direcciones espaciales de la entrada (Figura 2.7).

En el caso concreto del procesamiento de imágenes, usualmente se organiza a las neuronas de cada capa en 3 dimensiones: (ancho, alto, profundidad), donde *profundidad* representa el número de mapas de features en ese nivel de la red. El filtro asociado a cada mapa está acotado a un tamaño espacial $W \times H$ (por ejemplo 3×3 o 5×5) que se desplaza a un cierto paso horizontal y verticalmente, actuando siempre sobre todos los mapas de entrada.

Típicamente, las primeras capas cuentan con una gran cantidad de mapas que servirán para representar features de bajo nivel, y las capas superiores aprenderán conceptos más refinados y abstractos, producto de la combinación de los mapas de niveles anteriores.

Las conexiones locales limitan el rango de visión de una neurona dada. Por lo tanto, las primeras capas tendrán una visión acotada, mientras que las capas superiores desarrollarán una visión más global del espacio de entrada.

El tamaño del filtro suele ser de varios órdenes de magnitud menor que el de la entrada. Por ejemplo, al procesar una imagen, la imagen de entrada puede tener miles o millones de píxeles, pero podemos detectar características pequeñas y significativas como bordes con filtros que ocupan solo decenas o cientos de píxeles. Esto significa que la conectividad de la red resultante será mucho más dispersa en comparación a una red neuronal tradicional. Entonces se necesitan almacenar muchísimos menos parámetros, reduciendo los requisitos de memoria del modelo y requiriendo mucho menos operaciones para la evaluación. Estas mejoras en la eficiencia suelen ser bastante grandes, permitiendo entrenar modelos a una mayor escala.

Como los parámetros son compartidos por todas las neuronas en el mismo mapa de features, el requerimiento de memoria para los parámetros se reduce aún más. Además, esta forma particular de compartir parámetros resulta en una propiedad llamada *equivarianza a la traslación*, lo que significa que si movemos el objeto en la entrada, su representación se moverá la misma cantidad en la salida [10].

Capas de pooling

Además de las convoluciones, las redes convolucionales suelen incluir otro tipo de capas, llamadas *pooling*, que permiten reemplazar la salida de la red en una ubicación determinada con un resumen estadístico de las salidas cercanas.

Existen distintas formulaciones, por ejemplo, la operación de *max pooling* devuelve la salida máxima dentro de un vecindario rectangular, la operación de *average pooling*, en cambio, devuelve el promedio de las unidades.

Las capas de pooling reducen progresivamente el tamaño espacial de la representación para decrementar la cantidad de unidades de la red y cómpu-

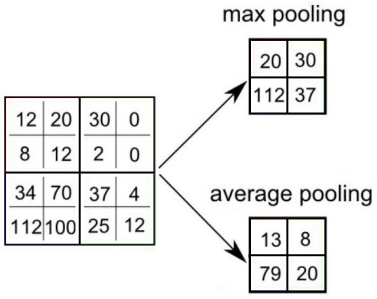


Figura 2.8: Estructura de una capa de pooling. La salida de la red en una ubicación determinada es reemplazada por un resumen estadístico de las salidas cercanas.

tos asociados. Además, permite que la representación sea aproximadamente invariante a pequeñas traslaciones de la entrada.

Si bien las capas de pooling forman parte de la mayoría de las redes convolucionales del estado del arte, otros trabajos cuestionan la necesidad de las mismas, mostrando que se pueden obtener resultados competitivos utilizando únicamente capas convolucionales (*All Convolutional Net* [26]), permitiendo a la red aprender su propio muestreo espacial (*downsampling*) con convoluciones de mayor desplazamiento espacial (*Strided Convolutions*).

2.3.3. Batch Normalization

Batch Normalization [27] propone estabilizar el entrenamiento a través de la normalización de los valores en cada punto de la red para reducir los cambios pronunciados en la distribución interna de los nodos (*internal covariate shift*), permitiendo entrenar con valores más grandes para el learning rate y reduciendo el impacto de la inicialización de los parámetros.

Dada una capa con d dimensiones $x = (x^{(1)}... \ x^{(d)})$, se normalizan los valores de cada dimensión para que tengan media 0 y varianza 1:

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathbb{E}[x^{(k)}]}{\mathrm{Var}[x^{(k)}]}$$

donde la media y varianza se computan sobre todo el conjunto de entrenamiento.

Para asegurarse que la transformación insertada en la red puede representar la transformación identidad, para cada activación $x^{(k)}$ se introduce un par de parámetros extra: $\gamma^{(k)}$ y $\beta^{(k)}$, que escalan y trasladan el valor normalizado:

$$y^{(k)} = \gamma^{(k)}\hat{x}^{(k)} + \beta^{(k)}.$$

Nótese que estableciendo $\gamma^{(k)} = \mathrm{Var}[x^{(k)}]$ y $\beta^{(k)} = \mathbb{E}[x^{(k)}]$, es posible recuperar la activación inicial, en caso de ser necesario.

Sin embargo, como el entrenamiento se realiza a través de descenso por el gradiente estocástico, este método se simplifica estimando la media y varianza de cada activación únicamente sobre los elementos del minibatch. De esta manera, durante el entrenamiento la evaluación de la función en una entrada termina condicionada en las otras entradas presentes en el mismo minibatch.

Al momento de la evaluación (luego del entrenamiento), las capas de Batch Normalization se reducen a una simple transformación afín independiente del lote de valores sobre el que se aplique, donde la media y varianza se mantienen fijos con valores estimados por estadísticas sobre todo el dataset. Esta estimaciones se obtienen durante el entrenamiento manteniendo medias móviles sobre la media y varianza de las activaciones en cada minibatch.

Usualmente, la capas de Batch Normalization se introducen inmediatamente después de las capas lineales y convolucionales, y antes de las capas no lineales de activación.

En el caso particular de las convoluciones, para permitir que la normalización preserve la propiedades de la convolución, se normalizan en conjunto todas las unidades pertenecientes al mismo mapa de features. Es decir, se computa la media y varianza sobre todas las activaciones de un mapa de features y se asocia un único par de parámetros γ y β.

Esta mejora se convirtió en un componente fundamental de la mayoría de los modelos profundos del estado del arte.

2.3.4. Resnets

Al diseñar redes cada vez más profundas, resulta necesario determinar cuándo agregar más capas realmente permite aumentar el poder de representación de la red.

En muchos casos, se observa empíricamente que después de cierta profundidad, simplemente agregar más capas a un modelo profundo de buen desempeño resulta en un error más grande [28, 29]. Esto es inesperado, ya que la representación permitiría un desempeño al menos tan bueno como la red anterior, donde las nuevas capas podrían ser simplemente funciones identidades. Sin embargo, en la práctica, los experimentos muestran que muchas veces los optimizadores tienen dificultades en encontrar estas soluciones al menos tan buenas como las anteriores.

Por lo tanto, para lograr que el proceso de optimización sea más simple, se propone en lugar de aprender capas totalmente independientes, aprender capas residuales [30], que se construyen por encima de la función identidad sobre la salida de las capas anteriores.

En [30] empíricamente muestran que este tipo de redes son más fáciles de optimizar y permiten mejorar el desempeño con mayor profundidad.

La nueva formulación consiste en incluir un conjunto de atajos (*shortcut connections*) que salteen una o más capas de la red, agrupando conjuntos de capas en grupos denominados *bloques residuales* (Figura 2.9).

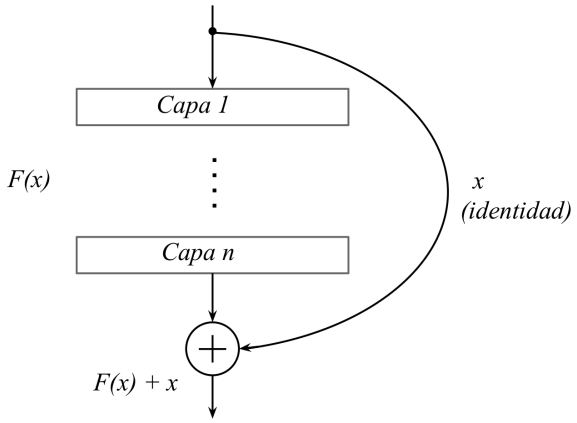


Figura 2.9: Estructura de un bloque residual. La arquitectura original es modificada para introducir atajos que representan la función identidad.

2.3.5. Bloques Non-local

Las redes basadas en capas convolucionales presentan un estructura que restringe el patrón de conectividad donde los features en una capa se obtienen como función de los features ubicados únicamente en zonas espacialmente cercanas en las capas anteriores.

Esta restricción en la estructura de la red puede resultar limitante para modelar dependencias entre diferentes regiones de las imágenes, donde los features de diferentes zonas solamente se pueden procesar en conjunto luego de varias convoluciones.

Si bien incrementar el tamaño de los filtros de las convoluciones permite aumentar la capacidad de representación de la red, esto elimina las ventajas en la eficiencia computacional de las convoluciones.

Frente a esta limitación, se presentan las *operaciones non-local* [31], que calculan la respuesta en una posición como una suma ponderada de los features en todas las posiciones. Dada una entrada x con una estructura espacial enumerada por $i \in I$, la operación non-local se define como:

$$y_i = \frac{1}{C(x_i)} \sum_j f(x_i, x_j)\ g(x_j)$$

donde la función f calcula un escalar entre la posición i y cada posible posición j (que representa una relación como la *afinidad*), la función g calcula una nueva representación de la entrada y $C(x_i)$ representa un factor de normalización.

Si bien existen diferentes opciones para cada uno de estos componentes, una configuración común es la llamada *Embedded Gaussian* [31] (también denominada *self-attention* en [32, 33]).

En este caso, la función de afinidad f consiste en primero proyectar la entrada a dos espacios de features independientes: $\phi(x_i) = W_\phi\ x_i$ y $\psi(x_i) =$

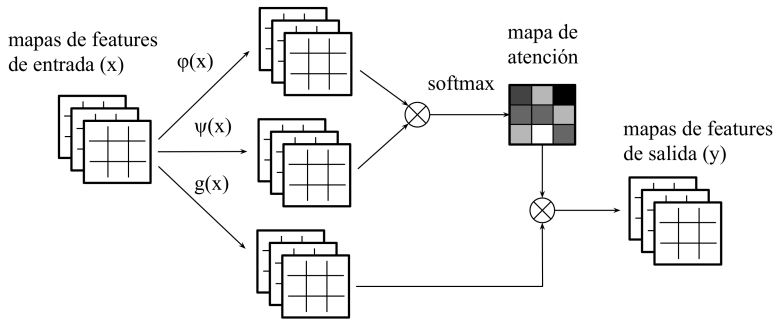


Figura 2.10: Operación de self-attention. Las proyecciones $\psi(x)$, $\phi(x)$ y $g(x)$ se implementan con convoluciones 1×1. Los resultados se combinan a través de *capas multiplicativas* y *softmax*.

$W_\psi \; x_i$ y luego computar la similitud como el exponencial del producto escalar:

$$f(x_i, x_j) = e^{\phi(x_i)^T \psi(x_j)}$$

utilizando el factor de normalización: $C(x_i) = \sum_j f(x_i, x_j)$. La nueva representación g comúnmente se define como una simple proyección lineal: $g(x_i) = W_g \; x_i$ (Figura 2.10).

En general, esta operación se compone con una conexión residual, formando un *bloque non-local*, definido como: $o_i = W_o \; y_i + x_i$ (donde y_i es el resultado de la operación non-local sobre x_i).

Un punto importante a notar es que a diferencia de las convoluciones, que se podían interpretar como un subconjunto de las neuronas conectadas completamente con valores dispersos para los parámetros, las operaciones non-local son sustancialmente distintas por las operaciones de multiplicación. La salida en un punto está basada en la relación entre los valores en diferentes ubicaciones (*attention map*), mientras una neurona completamente conectada relaciona las entradas a través de pesos fijos aprendidos durante el entrenamiento.

Estas capas se volvieron un componente integral en modelos que requieren capturar dependencias globales, como procesamiento de video [31], generación de imágenes [33], traducción automática [32], etc.

2.4. Aprendizaje No Supervisado

Hoy en día tenemos acceso a muchísimos datos, fácilmente disponibles, donde el principal desafío es desarrollar modelos y algoritmos que puedan analizar y comprender la información contenida.

En esta dirección se enfoca el área de *Aprendizaje No Supervisado*, donde se intenta descubrir la estructura subyacente en los datos. Es decir, de manera más formal, dado un conjunto de elementos $\{x^{(1)}, x^{(2)} ... \; x^{(n)}\}$ de cierta

dimensionalidad d, teniendo una función de densidad de probabilidad conjunta $P(\mathcal{X})$, se busca descubrir la estructura relevante en esta distribución.

A diferencia del Aprendizaje Supervisado, en este caso no se cuenta con la ayuda de una señal supervisora que determine la respuesta correcta. En general los datos etiquetados poseen relativamente poca información, que no resulta útil para determinar los parámetros de modelos complejos como los que se intentan construir. Por lo tanto, es necesario aprender directamente de los datos en concreto.

2.4.1. Modelos Generativos

Los *Modelos Generativos* son unos de los más prometedores en lograr este objetivo. Toman un conjunto de entrenamiento, que consiste en una gran cantidad de datos reales (millones de imágenes, oraciones, sonidos, etc) tomados de una distribución dada P_{data} y aprenden a representar una estimación de esa distribución, resultando en una distribución probabilística P_{model}. En algunas casos esta se modela explícitamente, en otros, solo es posible generar muestras de acuerdo a P_{model}.

La intuición detrás de este enfoque es que, en la medida en que se aprenda cómo generar los datos, se tendrá una mejor comprensión de las características intrínsecas de los mismos. Los recientes avances en el entrenamiento de redes neuronales como aproximadores de funciones, potenciadas por el algoritmo de Backpropagation, dieron lugar a muchas propuestas en el uso de redes neuronales para construir modelos generativos, los que suelen llamarse *Modelos Generativos Profundos*.

2.4.2. Redes Adversarias Generativas

Entre los resultados más prometedores, las *Redes Adversarias Generativas* (GAN) [1] han sido el principal foco de investigación en los últimos años debido a que, entre otros motivos, han demostrado ser muy efectivas para generar imágenes sintéticas.

GAN establecen el problema como un juego entre dos redes: una red generadora (G) encargada de producir datos sintéticos a partir de un vector z de ruido dado como entrada, y una red discriminadora (D) que diferencia entre los datos generados sintéticamente por el generador y los datos reales tomados de un conjunto de entrenamiento.

El discriminador D se entrena en forma estándar para resolver el siguiente problema de clasificación binario: dada una imagen devolver la probabilidad de que esta provenga del dataset (imagen real), en oposición a que sea una imagen generada por el generador G (imagen falsa). Este entrenamiento se consigue actualizando los parámetros de D en descenso por el gradiente estocástico sobre una función de costo estándar J para el problema de clasificación binaria. En simultáneo a cada actualización de los parámetros de D, se actualizan los

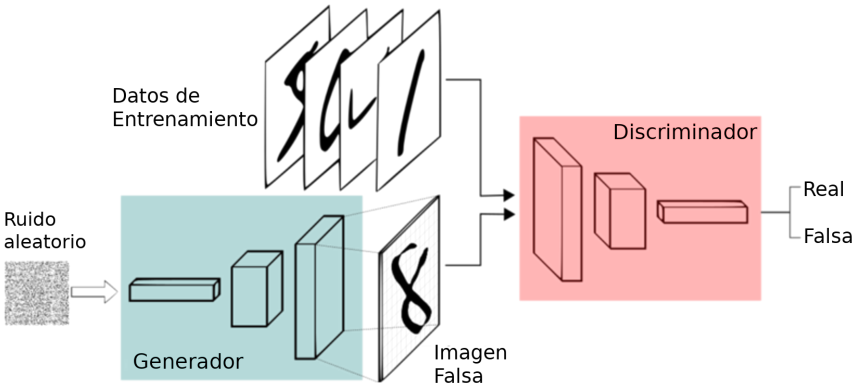


Figura 2.11: Redes GAN.

parámetros de G con el objetivo opuesto: aumentar esta función de costo J con un ascenso por el gradiente, es decir con el objetivo de engañar al discriminador.

Este esquema de entrenamiento adversario ha despertado gran interés en el área de Aprendizaje Profundo dada la calidad de los resultados obtenidos pero por sobre todo por la potencialidad que implica entrenar modelos en forma no supervisada. Las redes adversarias generativas han logrados grandes avances en tareas de generación de imágenes incluyendo la traducción imagen a imagen [34, 35], generación de imágenes de super resolución [36, 37], generación de imágenes a partir de texto [38, 39].

Formulación original

Más formalmente, para aprender una distribución P_{data} sobre los datos, se define una distribución simple P_z sobre el vector latente de entrada, y representamos al generador como una función $G(z;\theta_g)$ que transforma este vector al espacio de datos en una distribución generada P_g, diferenciable con respecto a los parámetros θ_g. De manera análoga, se define el discriminador como una función opuesta $D(x;\theta_d)$ que dado un valor de entrada devuelve un escalar.

Ambos modelos D y G compiten en un juego minimax con la función de valor $V(G,D)$:

$$\min_G \max_D V(G,D)$$

donde: $V(G,D) = \mathbb{E}_{x\sim P_{data}}[\log D(x)] + \mathbb{E}_{z\sim P_z}[\log(1 - D(G(z)))]$

El entrenamiento consiste en optimizar ambos modelos simultáneamente a través de descenso por el gradiente. En cada paso, se toman dos lotes de datos, un lote x de valores reales y un lote z de valores latentes. Luego, se realizan dos pasos de optimización, en los que los parámetros de un modelo se actualizan, mientras que los parámetros del contrincante se mantienen fijos,

primero actualizando θ_d para maximizar $V(G, D)$ y luego actualizando θ_g para minimizar $V(G, D)$.

En [1], muestran que este juego minimax tiene un óptimo global donde $P_g = P_{data}$, que es equivalente a que el discriminador retorne 0.5 para todos los ejemplos de x. Es decir, el generador es óptimo cuando el discriminador realmente no puede distinguir entre ejemplos reales y ejemplos generados.

Dado un generador G, el discriminador óptimo resulta ser:

$$D^*_G(x) = \frac{P_{data}(x)}{P_{data}(x) + P_g(x)}.$$

Si G y D tienen suficiente capacidad, y en cada paso del algoritmo se permite que el discriminador alcance su punto óptimo D^*_G, optimizar G respecto a D^*_G equivale a minimizar la divergencia de *Jensen-Shannon* entre las distribuciones P_g y P_{data}, por lo que se demuestra que el algoritmo converge al óptimo global donde $P_g = P_{data}$.

A pesar de los buenos resultados teóricos, en la práctica se utilizan modelos para G y D de una capacidad limitada, y el discriminador no es entrenado hasta su punto óptimo en cada paso, sino un número fijo de iteraciones.

La investigación actual está principalmente concentrada en obtener nuevas formulaciones de GAN que resulten más estables en el entrenamiento y lograr imágenes de cada vez mayor resolución y calidad. A continuación se mencionan las principales extensiones.

Arquitectura DCGAN

Las primeras versiones de GAN eran difíciles de entrenar debido al delicado equilibrio del entrenamiento adversario. Radford y col. [2] inicialmente propusieron una serie de recetas para entrenar en forma relativamente estable redes profundas convolucionales sobre imágenes (DCGAN). El interés y el volumen de publicaciones sobre GAN ha crecido exponencialmente desde entonces. Las principales características de la arquitectura DCGAN (Figura 2.12) se detallan a continuación.

La primera parte de la red consiste en una capa densa, es decir todas las unidades están conectadas con todos los valores de entrada, que permite proyectar el vector latente de 100 dimensiones en un espacio de mucha mayor dimensionalidad (16384). A este vector se le da una interpretación espacial, agrupándolo en 1024 canales de mapas 4×4.

En los siguientes pasos, se utilizan diferentes capas de *convoluciones transpuestas* [40] (llamadas así porque mantienen los patrones de conectividad de las convoluciones asociadas en la dirección opuesta). Estas capas aprenden su propio *upsampling*, aumentando el componente espacial de la representación progresivamente hasta una representación final de 3 canales RGB (*All Convolutional Nets* [26]). También se utiliza Batch Normalization en todas las capas excepto la de salida del generador y la de entrada del discriminador.

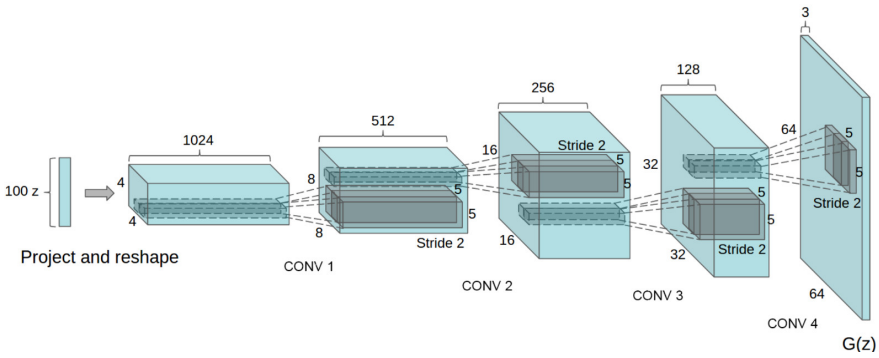


Figura 2.12: Arquitectura DCGAN.

Finalmente, en todas las capas del generador se utilizan ReLUs como funciones de activación, a excepción de la capa final, sobre la que se aplica la función Tanh que restringe la salida al rango $[-1, 1]$. En el discriminador, en cambio, se utilizan Leaky ReLUs.

Conditional GAN

Si se cuenta con datos etiquetados, el modelo generativo puede ser extendido para que tanto el generador como el discriminador estén condicionados en la información de la clase que se provee como entrada, llamado *Conditional GAN* [41]. La información puede ser introducida de diferentes formas, en [42] proponen concatenar la información de las clases al vector latente de entrada y en [43, 44], en cambio, utilizan el valor de las clases para definir los coeficientes de las capas de Batch Normalization. En general, aumentar la información de entrada con las clases provee un salto importante en la calidad de las imágenes generadas, por esto el estado del arte en generación para datasets de alta variabilidad (por ejemplo ImageNet) se basa en esta formulación.

Formulaciones más estables

Dado que los datos reales y los datos generados pueden tener muy poca superposición, la divergencia de Jensen-Shannon (JS) entre la distribución generada y la distribución real puede resultar prácticamente constante, lo que causa el problema del desvanecimiento del gradiente al usar el método de descenso de gradiente para entrenar GAN [45]. Es decir, en la formulación original, a medida que el discriminador se optimiza, este aproxima de mejor manera la divergencia de JS, por lo que el gradiente del discriminador se degrada, lo cual se observa empíricamente y formalmente se demuestra en [45].

Para abordar este problema, en [46] proponen *Wasserstein GAN* (WGAN) utilizando la distancia *Earth Mover* (EM) para reemplazar la divergencia de Jensen-Shannon como medida de la distancia entre las distribuciones de los

datos reales y los datos generados, mostrando que es más adecuada para la optimización sobre redes neuronales.

En esta formulación, la distancia de Wasserstein es aproximada por el discriminador, al que se refiere como *crítico*, y se restringe al conjunto de funciones continuas *k-Lipschitz*. Esta restricción inicialmente se implementó a través de *clipping* en los parámetros del discriminador, es decir, restringir los parámetros a un intervalo fijo en cada actualización. Sin embargo, investigaciones posteriores [40] sugirieron que el método de clipping reduce la capacidad del modelo discriminador, forzándolo a aprender funciones más simples. En cambio, se propuso un método distinto de imponer la restricción k-Lipschitz, penalizando la norma de gradiente del discriminador con respecto a su entrada (WGAN-GP).

En [47] proponen aplicar *Spectral Normalization* (SNGAN), una nueva técnica para normalizar las capas del discriminador y restringirlo a las funciones k-Lipschitz. El método consiste en restringir la norma espectral de la matriz de pesos asociada a cada capa para que sea constante en 1, demostrando ser más robusto que las propuestas anteriores.

En [48] proponen una nueva metodología de entrenamiento para redes Generativas donde el tamaño del generador y discriminador crece progresivamente (*Progressive Growing*) comenzando desde una baja resolución y agregando incrementalmente nuevas capas que permiten modelar más detalles a medida que el entrenamiento progresa hasta un tamaño final de alta resolución. Demostraron acelerar y estabilizar el entrenamiento, con resultados sorprendentes especialmente en datasets de rostros humanos.

En SAGAN [33], agregan un módulo de *Self-Attention*, que permite que la red combine los features de diferentes partes de la imagen a través de una suma ponderada por valores de atención. Estos valores de atención se aprenden en el entrenamiento, y la red puede descubrir partes de las imágenes que están relacionadas y tener una visión global, superando las limitaciones de las convoluciones que tienen un rango de alcance limitado. De esta manera logran que diferentes partes de la imagen sean más consistentes y respeten mejor objetos que tienen gran estructura, como por ejemplo, el número de piernas en un animal, el número de ojos, etc. Además, lo combinan con Spectral Normalization no solo en el discriminador, sino también en el generador. Muestran un importante salto en el *Inception Score*[1] para Conditional GAN en ImageNet.

Posteriormente, en BigGAN [4], dan un salto gigante en el Inception Score para Conditional GAN en ImageNet (de 52.52 a 166.5). Muestran que las GAN se benefician de aumentar el tamaño de la red. Se basan en la arquitectura SAGAN, aumentando el número de mapas de features y el tamaño del lote, entre

[1]Inception Score [49] es una métrica respecto a la calidad de las imágenes generadas, que se computa de manera automática utilizando los resultados de clasificación de una red convolucional y se ha demostrado que se correlaciona muy bien con el criterio humano. Valores mayores implican mayor calidad de imágenes.

otros detalles. Proponen usar una distribución distinta para tomar muestras del espacio latente que la que se utiliza durante el entrenamiento, truncando una distribución normal (*truncation trick*).

Representación latente

La particularidad de GAN respecto a otros modelos generativos basados en redes neuronales es que el generador transforma determinísticamente un vector z de ruido en una imagen de salida. Es decir, el proceso de generación no implica ningún proceso iterativo de optimización para cada imagen sino que basta una sola propagación a lo largo de la red neuronal. Toda la variabilidad en la salida de la red es capturada por el vector z. No hay procesos estocásticos en la transformación.

En este contexto, resulta de especial interés estudiar el espacio latente aprendido por el generador: la relación semántica entre imágenes correspondientes a valores de z cercanos, las transformaciones causadas por desplazamientos en una dada dirección en el espacio de entrada, etc. Un primer antecedente de este tipo de análisis es el realizado por Radford y col. [2] donde muestran que operaciones aritméticas básicas de suma y resta sobre los vectores z tienen un correlato semántico en las imágenes generadas.

Capítulo 3

Inversión del Generador

La capacidad de las redes GAN para aprender modelos generativos que traducen distribuciones latentes simples en distribuciones de datos arbitrariamente complejas se ha demostrado empíricamente, con resultados convincentes que muestran que el espacio latente de tales generadores captura variaciones semánticas en la distribución de datos [2].

Por lo tanto, resulta de interés explotar la representación aprendida de manera no supervisada, proyectando las imágenes reales en el espacio latente y usando esta representación para diferentes tareas. Por ejemplo, para la recuperación y clasificación de imágenes [50, 51]. También se ha despertado un interés reciente en acceder al espacio $\mathcal{Z}$ para manipular imágenes [52, 53, 54, 55], permitiendo desplazarse en una aproximación del manifold de imágenes reales. Además, invertir el generador proporciona información relevante para esclarecer los aspectos que el generador ha aprendido a modelar [56].

Invertir el generador significa encontrar un vector $z \in \mathcal{Z}$ que cuando se provee como entrada resulta en una imagen que es muy similar a la imagen objetivo. Proyectar una imagen desde el espacio de píxeles hacia el espacio latente no es una operación trivial, ya que requiere invertir el generador, que por lo general consiste en un modelo complejo de varias capas no lineales. Una misma imagen podría obtenerse de diferentes valores latentes z o de ninguno.

Desafortunadamente, en su formulación original, las redes GAN no ofrecen un *modelo inverso* que permita encontrar dicho vector z de manera automática. En esta dirección se presentan trabajos previos con motivaciones similares a la presente tesina.

3.1. Modelos paramétricos

En trabajos paralelos, Dumoulin y col. [50] y Donahue y col. [51] proponen aprender un modelo paramétrico, llamado *encoder* (E), que asocia a cada imagen una representación en el espacio $\mathcal{Z}$, y es entrenado en conjunto con el generador y discriminador (Figura 3.1). En este caso, el discriminador recibe

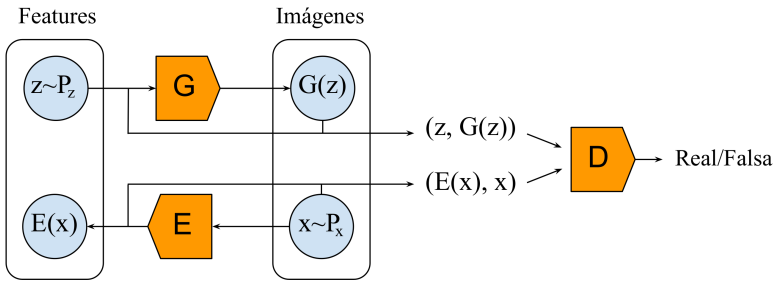


Figura 3.1: Entrenamiento adversario en conjunto del encoder y generador, donde el discriminador recibe pares de la distribución conjunta de features y imágenes [50, 51].

pares *(imagen, representación)* de $\mathcal{X} \times \mathcal{Z}$, y intenta diferenciar entre: pares construidos con imágenes del dataset y la codificación dada por E, y imágenes generadas por G, junto a su representación latente. Es decir, no solo se le pide al discriminador que distinga entre la distribución real y generada en el espacio de datos, sino que distinga entre las dos distribuciones conjuntas en el espacio de datos y valores latentes. Si bien muestran buenos resultados en utilizar el encoder para extraer patrones y utilizarlos en tareas de clasificación, en general las reconstrucciones no son buenas, fallando en preservar el estilo y clase de las imágenes originales. Recientemente, Li y col. [57] propusieron métodos para mejorar las reconstrucciones.

Una limitación adicional de los encoders entrenados en conjunto con los generadores es que esta técnica no se puede aplicar sobre generadores pre-entrenados independientemente. Otras propuestas consisten en entrenar un encoder como un paso posterior al entrenamiento del generador. Existen diferentes métodos para guiar este entrenamiento, por ejemplo hacer una regresión en el espacio $\mathcal{Z}$ [51], o acoplarlos de manera opuesta y entrenar el encoder haciendo regresión en el espacio de imágenes $\mathcal{X}$ [58].

Estos enfoques en general tienen la desventaja de requerir el entrenamiento de un tercer modelo (encoder) incrementando el número de parámetros a aprender y posibilidades de sobreajuste del encoder sobre los ejemplos sobre los que es entrenado, y por lo tanto no sea aplicable a imágenes correspondientes a una distribución distinta. Por ejemplo si se utilizan ejemplos sintéticos en el entrenamiento, aunque se desee aplicar luego sobre imágenes reales. Finalmente, el hecho de introducir un modelo complejo para invertir el generador hace que su valor como herramienta de diagnóstico para evaluar GAN sea cuestionable.

3.2. Optimización sobre el Generador

Creswell y Bharath [56] proponen reconstruir el vector z^* que da origen a una determinada imagen x resolviendo un problema de optimización sobre el

generador. Esencialmente, utilizan el gradiente del generador G con respecto a la entrada del mismo (a diferencia del entrenamiento, donde se calcula el gradiente respecto a los parámetros del modelo), desde un vector z inicializado al azar sobre la distribución del espacio $\mathcal{Z}$, siempre que el grafo computacional de G esté disponible. El proceso de optimización es independiente para cada imagen, lo que permite realizarlo en paralelo para muchas imágenes, de manera eficiente.

Al no requerir el entrenamiento de ningún parámetro específico asociado al generador, a diferencia de otros métodos que implican entrenar un encoder, se puede utilizar el mismo procedimiento para evaluar y comparar la representación aprendida por diferentes generadores. Por ejemplo, los casos que no sea posible encontrar un valor adecuado para z pueden considerarse como un indicio de que el generador no es capaz de modelar toda la imagen, o al menos ciertos atributos de la misma.

Creswell y Bharath [56] muestran resultados obtenidos sobre el dataset MNIST (dígitos escritos a mano) donde la imagen generada por el valor latente recuperado mantiene el estilo y identidad del dígito original, y de igual manera lo realizan con otros datasets específicos a un tipo particular de situación.

Zhu y col. [52] utilizan el mismo enfoque como parte de un sistema para manipular imágenes. Previamente, un algoritmo similar fue propuesto por Mahendran y Vedaldi [59] para estudiar la representación interna de redes profundas.

Lipton y Tripathi [60] proponen una variante llamada *stochastic clipping* para el caso particular de una distribución uniforme sobre el espacio latente y muestran que es posible invertir el 100 % de las imágenes sintéticas a su valor latente original.

La ventaja de utilizar un encoder es que permite obtener inversas de manera muy eficiente, al costo de tener que entrenar un modelo general para todas las imágenes, que termina siendo un problema más complejo que hacerlo independientemente para cada una, y por lo tanto se espera lograr peores resultados de reconstrucción. Por otro lado, planteando el problema como una optimización sobre G respecto a los valores de z, se esperan obtener mucho mejores resultados, pero requerirá de más tiempo para encontrar la solución adecuada, generalmente involucrando muchas iteraciones.

Un posible enfoque híbrido, como se menciona en [52], consiste en utilizar el valor inferido por el encoder como inicialización del proceso de optimización por descenso por el gradiente. De esta manera se puede reducir significativamente el número de iteraciones.

Nuestro trabajo se centra en el estudio de las representaciones intermedias del generador, la invertibilidad de las diferentes capas y el grado de reconstrucción de las imágenes reales. Decidimos optar por el enfoque de optimización sobre el generador, porque no requiere la introducción y entrenamiento de un nuevo conjunto de parámetros para modelar la función inversa, lo cual podría debilitar las conclusiones de nuestros experimentos. A diferencia de los

trabajos previos, nuestro objetivo es trabajar sobre datasets de más amplia variabilidad, que incluyan diferentes clases de objetos en diferentes situaciones, como ser CIFAR-10 [3] o ImageNet [5], los cuales requieren modelos de una complejidad mayor. Los métodos anteriormente mencionados no se desempeñan bien o directamente no muestran resultados sobre estos datasets más complejos. Además, proponemos extender este mecanismo de inversión hacia las diferentes representaciones intermedias del generador.

Capítulo 4

Invirtiendo el Generador en diferentes niveles

En esta tesina deseamos analizar la dificultad del generador en aproximar la distribución real de las imágenes, en particular, en relación al rol que cumple la primera capa densa (característica presente en todas las arquitecturas del estado del arte en GAN). La propuesta es poder comparar ambas distribuciones (P_{data}, P_{model}) no solo a nivel de píxeles, sino en cada representación a lo largo de la profundidad de la red. Para esto, proponemos partir de las imágenes reales e ir buscando una representación para las mismas en cada capa del generador, desde el espacio de salida hacia atrás, y de esta manera analizar si existe un valor latente en cada punto de la red que permita reconstruir la imagen original. A través de este procedimiento podremos determinar hasta qué punto son invertibles las diferentes capas del generador.

4.1. Representaciones intermedias

Consideremos cada imagen representada como un punto en el espacio $\mathcal{X} = \mathbb{R}^{W \times H \times C}$ donde cada canal (RGB) de cada píxel se considera una dimensión independiente (W:ancho de la imagen, H:alto de la imagen, C:número de canales).

Sea G un generador de arquitectura profunda, compuesto de n capas, que transforma un vector latente $z \in \mathbb{R}^{d_z}$ tomado de una distribución $z \sim P_z$ (donde P_z es por lo general una distribución sencilla como ser $\mathcal{N}(0,1)$ o $\mathcal{U}(0,1)$) en una imagen $G(z)$, con una distribución implícita P_{model} que intenta aproximar la distribución P_{data} de datos reales sobre el espacio de imágenes $\mathcal{X}$.

Podemos partir al generador en una capa oculta l (de cierta dimensionalidad d_l), y analizar la representación aprendida en dicho punto (Figura 4.1), reexpresando al generador como la composición de dos generadores:

$$G(z) = G_2^l(G_1^l(z))$$

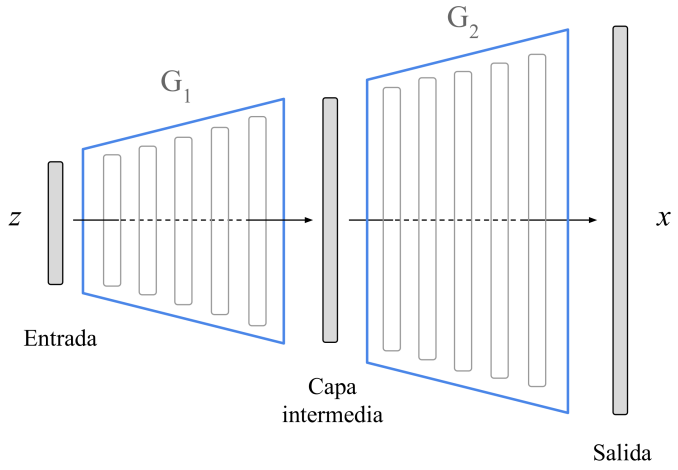


Figura 4.1: Representación en las capas intermedias del generador. Podemos considerar al generador como la composición de dos sub-generadores G_1 y G_2.

donde $G_1^l : \mathcal{Z} \to \mathbb{R}^{d_l}$ representa la transformación desde el espacio latente hasta la capa l, y $G_2^l : \mathbb{R}^{d_l} \to \mathcal{X}$ desde la capa l hacia el espacio de imágenes $\mathcal{X}$.

Sea P_h^l la distribución generada en la capa oculta l ($\mathbb{R}^{d_l}$) de acuerdo a la variable aleatoria: $H_{gen}^l = G_1^l(z)$ con $z \sim P_z$. Luego, podemos considerar a $G_2^l(h)$ como un generador desde la distribución latente aprendida $h \sim P_h^l$, que tiene el mismo desempeño que el generador original $G(z)$ con $z \sim P_z$.

4.2. Invertibilidad del Generador

Al mencionar que una parte de la red sea *invertible* en un conjunto de imágenes estaremos haciendo abuso de notación para denotar que es aproximadamente invertible a derecha, es decir, para cada imagen existe un valor de entrada que permite reconstruir aproximadamente la misma imagen. Más formalmente, dada una función $f : X \to Y$ y un subconjunto del conjunto de llegada $S \subseteq Y$, al decir que f es *invertible* en S, denotaremos que f es *aproximadamente invertible a derecha* en S, es decir, existe una función $g : S \to X$ tal que $f(g(y)) \approx y \ \ \forall y \in S$.

Si podemos invertir el generador desde el espacio de imágenes hasta la capa l, asociando a cada imagen $x \in \mathcal{X}$ una representación $G_2^{l\,-1}(x) \in \mathbb{R}^{d_l}$ tal que $G_2^l(G_2^{l\,-1}(x)) \approx x$, entonces podemos definir una nueva distribución de puntos en $\mathbb{R}^{d_l}$ asociada a la representación de las imágenes reales en este espacio: $H_{real}^l = G_2^{l\,-1}(x)$ con $x \sim P_{data}$. Vale la aclaración que la función $G_2^{l\,-1}$ no necesariamente existe, y, en caso de existir, no necesariamente es única, ya que podríamos tener múltiples representaciones en la capa dada que nos generan

la misma imagen real.

Si el generador fuera óptimo, es decir, capaz de generar la distribución real ($P_{model} = P_{data}$), entonces todas las capas deberían ser invertibles para las imágenes reales, donde la distribución en espacio latente P_z se traduce en distribuciones generadas en distintos puntos de la red hasta la distribución en la capa final a nivel de píxeles P_{data}. La demostración es trivial.

Teorema 1. *Un generador óptimo G^* es invertible a derecha casi en todas partes en $\mathcal{X}$ de acuerdo a P_{data}.*

Demostración. Sea $\mathcal{X}' = \mathcal{X} - G^*(\mathcal{Z})$ el conjunto de imágenes que no pueden ser generadas. Luego, $\mathcal{X}'$ tiene medida nula bajo P_{data}, es decir:

$$P_{data}(\mathcal{X}') = P_{model}(\mathcal{X}') = 0 \qquad \square$$

Es decir, que el generador sea invertible en las imágenes reales es condición necesaria para un generador óptimo. Sin embargo, claramente no es condición suficiente, ya que aún si existiera una representación para las imágenes reales en cada punto de la red, el generador podría no ser capaz de generar la distribución apropiada sobre dichos puntos desde la distribución latente.

En un punto extremo, podemos definir a un generador trivial que toma un punto en el espacio (de suficiente dimensionalidad) y devuelve el mismo punto. Este generador será perfectamente invertible, pero claramente no está aproximando desde la distribución de ruido de entrada a la distribución de salida en las imágenes reales.

Para un generador no óptimo, se espera que las imágenes reales sean aproximadamente invertibles, donde la calidad de la reconstrucción depende de la capa considerada para la representación. En un extremo, en la representación de salida a nivel de píxeles, es posible representar todas las imágenes con un error de reconstrucción nulo. A medida que uno se mueve en la representación de la red hacia atrás (considerando el espacio de entrada de G_2^l para una capa l), se espera que el error de reconstrucción para ciertas imágenes se incremente.

Como en el generador la información fluye hacia adelante, toda imagen representada en una capa l dada, tendrá una reconstrucción no mejor en las capas anteriores, y al menos tan buena en las capas siguientes. Si consideramos dos capas $l < m$, el conjunto imagen de G_2^l es un subconjunto del conjunto imagen de G_2^m ($G_2^l(\mathbb{R}^{d_l}) \subseteq G_2^m(\mathbb{R}^{d_m})$). Por lo tanto, cuanto más cercano al espacio latente, más restringido será el conjunto imagen $G_2^l(\mathbb{R}^{d_l})$ a nivel de píxeles.

La *Hipótesis del Manifold* [61, 62] postula que los datos naturales en una representación de alta dimensionalidad (como por ejemplo las imágenes reales, textos de lenguaje natural, secuencias de audio, etc) residen en un manifold de baja dimensionalidad embebido en un espacio de alta dimensionalidad. Es decir, la mayor parte del espacio está compuesto de valores irrelevantes, y la distribución de puntos interesantes se concentra únicamente alrededor de

una colección de manifolds, donde las variaciones relevantes de los datos son capturadas por las direcciones de estos manifolds, o cuando uno se mueve de un manifold a otro. Si bien es una hipótesis que no siempre puede ser correcta, está sustentada en las observaciones experimentales sobre los datos reales, donde, por ejemplo, se puede comprobar que las distribuciones se encuentran altamente concentradas en pequeñas regiones del espacio, (por ejemplo si tomamos puntos de manera uniforme en el espacio de imágenes las posibilidades de obtener una imagen real son nulas), y se pueden identificar, al menos de manera informal, diferentes direcciones de variabilidad en los datos como por ejemplo al rotar los objetos en las imágenes, modificar gradualmente los colores, etc [10].

Bengio y col. [63] presentan la hipótesis de que las representaciones más profundas pueden desentramar de mejor manera los factores que explican la variabilidad en los datos. Y representaciones más desentramadas permiten desplegar los manifolds sobre los que los datos se concentran al mismo tiempo que expanden el volumen relativo ocupado por los puntos de alta probabilidad cercanos a estos manifolds. Es decir, en el problema de un generador entrenado para generar imágenes reales, se esperaría que las representaciones más profundas capturen los factores de variabilidad en las imágenes, y que las imágenes reales ocupen un volumen relativo mayor, y de manera opuesta, las imágenes de baja probabilidad en la distribución real, como el ruido blanco, ocupen menos porción del espacio.

De acuerdo a lo mencionado, se esperaría que en un generador entrenado de manera adecuada, el incremento en el error de reconstrucción se manifieste principalmente en las zonas irrelevantes de la distribución a nivel de píxeles (por ejemplo imágenes de ruido blanco) y en mucho menor medida en las imágenes reales.

4.3. Primera capa densa

En particular, en este trabajo estamos interesados en analizar la representación aprendida en la primera capa densa (presente en todas las arquitecturas GAN del estado del arte). Supongamos que la primera capa del generador consiste en d_1 unidades de conectividad completa, luego partimos al generador en este punto:

$$G_1^1(z) = W_0 z + b_0$$

donde $W_0 \in \mathbb{R}^{d_1 \times d_z}$. $G_1^1(\mathcal{Z})$ representa un subespacio lineal de dimensionalidad a lo sumo d_z (espacio columna de W_0) en $\mathbb{R}^{d_1}$. En general d_1 es uno o dos órdenes de magnitud mayor que d_z.

Tenemos un particular interés en la representación aprendida en esta capa por los siguientes motivos:

- Representa un salto importante en el poder de representación. En el espacio de la primera capa solo el subconjunto $G_1^1(\mathcal{Z})$ se considera al

moverse en el espacio latente, pero la dimensionalidad del espacio es mucho mayor. Como veremos a lo largo de esta tesina, al considerar todo el espacio de la primera capa ($\mathbb{R}^{d_1}$) se puede ampliar en gran medida el rango de imágenes representadas en relación al espacio latente.

- De todas las capas intermedias del generador, es la mas profunda, es decir, la más alejada del espacio de píxeles. Por lo tanto, la que tendrá una representación de más alto nivel.

- Como la primera capa es una transformación afín, preserva la estructura lineal del espacio latente, por ejemplo, las interpolaciones lineales en espacio latente se traducen en interpolaciones lineales en el espacio de la primera capa. Esto significa que es posible realizar el mismo tipo operaciones de aritmética vectorial que comúnmente se realizan sobre el espacio latente.

En adelante, cuando se omita el índice l asumiremos que se hace referencia a la primera capa densa (G_1 denota el primer mapping lineal G_1^1 y G_2 el resto de la red G_2^1). Nos referiremos a $\mathbb{R}^{d_1}$ como *el espacio de la capa densa.*

4.4. Algoritmo de inversión

Para invertir el generador, se decidió optar por el algoritmo que plantea el problema como una optimización, haciendo descenso por el gradiente respecto a la entrada del mismo, ya que como fue mencionado anteriormente, es un método simple que no requiere entrenar un tercer modelo (encoder), y por lo tanto, resulta más adecuado como herramienta de diagnóstico para evaluar el generador.

Dado un generador G para el cual conocemos el grafo computacional que lo define y resulta ser una función diferenciable, buscamos una representación en cada nivel del generador que permita reconstruir una imagen objetivo x, guiados por el gradiente en $\mathcal{L}(G_2^l(h), x)$ con respecto al vector h. Donde $\mathcal{L}$, función de costo a minimizar, representa el error de reconstrucción entre la imagen generada y la imagen objetivo.

En otras palabras, resolvemos la optimización $h^* = \arg\min_{h \in \mathbb{R}^{d_l}} \mathcal{L}(x, G_2^l(h))$ por descenso por el gradiente:

Algoritmo 1: Inversión del generador

$h \sim P_{init}^l$
mientras *no converge* **repetir**
  $L \leftarrow \mathcal{L}(G_2^l(h), x)$
  $h \leftarrow h - \eta \, \nabla_h L$
fin
resultado: h

Donde P^l_{init} determina la inicialización aleatoria en el espacio de la capa l.

Detalles de la implementación

Todo el código fue desarrollado en el entorno *TensorFlow* [64], utilizando una implementación general propia que permite cargar un grafo computacional arbitrario de un generador pre-entrenado e invertirlo hasta la capa deseada.

Es posible invertir lotes de imágenes al mismo tiempo, haciendo uso del poder de cómputo de las GPUs para ejecutar operaciones en paralelo. En cada caso, se eligió el tamaño de lote (número de imágenes a ser invertidas al mismo tiempo) de máximo *throughput* (número de imágenes invertidas por unidad de tiempo) dentro de las restricciones de memoria disponible en la placa gráfica.

Para el proceso iterativo de descenso por el gradiente se utilizó el optimizador Adam [18].

4.5. Definición del error de reconstrucción

Para buscar la proyección de una imagen dada, es necesario definir un error de reconstrucción que determine qué tan parecida es la imagen original (x) a la imagen reconstruida ($\hat{x}$) a partir de la representación obtenida. Para esto se decidió considerar la raíz cuadrada del error cuadrático medio, a nivel de píxeles, entre la imagen original y la imagen generada (RMSE, por *root mean square error*). Es decir:

$$RMSE(x, \hat{x}) = \sqrt{avg((x - \hat{x})^2)} = \frac{1}{\sqrt{n}} \|x - \hat{x}\|_2$$

($n = W \times H \times C$ número de componentes en x).

Otra opción, presente en algunos trabajos de la literatura asociada [54, 65, 66, 67], consiste en utilizar una métrica que compare ambas imágenes en el espacio de features, por ejemplo utilizando las primeras capas del discriminador como feature extractor u otro clasificador entrenado sobre el dataset. Este valor se puede combinar con el error cuadrático medio de manera de buscar una imagen que sea semánticamente similar a la original, aunque no tanto pixel a pixel.

En [68], proponen extender esta definición aún más, sumando una tercera componente, basada en la evaluación del discriminador. Es decir, combinar 3 valores: distancia a nivel de píxeles, a nivel de features extraídos por un clasificador y un valor de qué tan real es la imagen dado por un discriminador.

En adelante, salvo aclaración, al mencionar error de reconstrucción denotaremos RMSE a nivel de píxeles. En los experimentos iniciales, se lograron reconstrucciones de muy bajo valor para el RMSE, por lo que no fue necesario incorporar las posibles variantes antes mencionadas. Sin embargo, al trabajar con redes más complejas, sí fue necesario extender la definición para incorporar patrones extraídos por un clasificador.

Escala del error

El error cuadrático medio como error de reconstrucción tiene el problema de que su valor depende de la escala utilizada para medir la intensidad de los píxeles de la imagen. En nuestros experimentos, al computar el RMSE consideraremos las imágenes normalizadas, donde cada píxel contiene un valor en el rango $[0, 1]$.

En la representación de las imágenes reales, la intensidad de cada píxel se mide a través de un valor discreto entero en el rango $[0, 255]$. Sin embargo, el generador y discriminador trabajan con imágenes en el rango continuo, por lo que para computar el error de reconstrucción se normalizan al rango $[0, 1]$.

El error entre un píxel generado y el valor objetivo deja de ser relevante si ambos valores son discretizados al mismo entero en el rango $[0, 255]$ (escalando la imagen al rango $[0, 255]$ y asociando a cada píxel el entero más cercano). Es decir, si la distancia en el continuo ($[0, 1]$) entre el píxel generado y el píxel objetivo, es menor a $\gamma = \frac{1}{2\times 255}$, serán discretizados al mismo valor. Por lo tanto, un RMSE menor a 0.002 es despreciable. Sin embargo, a partir de lo observado en los experimentos sobre esta escala, se puede concluir que un error de reconstrucción menor a 0.02 ya significa que es difícil distinguir cuál es la imagen objetivo y cuál la reconstrucción.

4.6. Inicialización

El generador G sobre el que estamos optimizando puede ser muy complejo combinando múltiples capas lineales y no lineales. Como consecuencia, la función de costo a optimizar respecto a la entrada de la red puede resultar altamente no convexa, complicando el proceso de optimización.

En este escenario, es crítica una correcta inicialización, pudiendo condicionar si el algoritmo converge, si lo hace rápido y si se estanca en un mínimo local o punto silla. Sin embargo, dada una imagen a invertir, no es trivial cómo determinar el punto h_{init} desde donde inicializar la optimización sobre la capa l.

Una posibilidad es elegir puntos de manera aleatoria desde una distribución P^l_{init} sobre el espacio de la capa que se desea invertir ($\mathbb{R}^d$). Por ejemplo, estableciendo P^l_{init} como una distribución uniforme $\mathcal{U}(0, 1)$ o normal $\mathcal{N}(0, 1)$. Sin embargo, la distribución particular elegida puede resultar muy distinta a la distribución de los datos generados en la capa l durante el entrenamiento del generador (P^l_h).

Si evaluamos al generador en puntos lejanos a las zonas visitadas durante el entrenamiento, es probable que aparezcan deformaciones en la imágenes (*saturation artifacts* [4]) que no fueron corregidas por el generador. En cambio, nos interesa que nuestra búsqueda se mueva por las zonas visitadas durante el entrenamiento, para las cuales el generador se ajustó para producir bue-

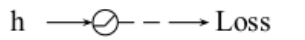
nas imágenes y donde seguramente se encuentre el mínimo deseado (mejor reconstrucción).

Por lo tanto, una mejor opción consiste en inicializar desde la distribución generada (P_h^l), tomando un vector z_{init} desde la distribución latente conocida (P_z) y evaluando la primera parte del generador hasta la capa deseada: $h_{init} = G_1^l(z_{init})$.

4.7. Problema de las neuronas muertas

Monitoreando el proceso de optimización sobre el espacio de distintas capas en un generador que utiliza ReLUs como función de activación, se puede observar que el número de neuronas encendidas va decreciendo a lo largo de las iteraciones.

Esto es resultado de cómo actúan la funciones ReLUs. El gradiente de una ReLU sobre una entrada h, se calcula de la siguiente manera[1]:



$$\begin{aligned}
\text{grad}(h) &= \frac{\partial Loss(\text{relu}(h))}{\partial h} \\
&= \frac{\partial Loss(\text{relu}(h))}{\partial\, \text{relu}(h)} \; \frac{\partial\, \text{relu}(h)}{\partial h} \\
&= \begin{cases} 0, & \text{si } h \leq 0. \\ \frac{\partial Loss(\text{relu}(h))}{\partial\, \text{relu}(h)}, & \text{si no.} \end{cases}
\end{aligned}$$

Si la neurona está encendida (el valor h es positivo), el gradiente fluirá, siendo señal para aumentar o disminuir el valor h. En cambio, si la neurona está apagada (valor h no positivo), el gradiente será cero, por lo que la única manera en que se vuelva a activar es por acción indirecta de otras neuronas, cuyos gradientes modifiquen la evaluación de h de tal manera que indirectamente resulte en la activación de la neurona dada.

Es decir, el proceso de optimización es guiado por el gradiente, y el gradiente solo fluye por una ReLU si está activada. Por lo tanto, la única manera en que se encienda una neurona inicialmente apagada es por acción indirecta del gradiente que fluye por otras neuronas encendidas en la misma capa.

Si la *conectividad* entre las neuronas es mayor, donde comparten gran cantidad de entradas, entonces una neurona inicialmente desactivada tiene más posibilidades de volver a ser activada por acción indirecta de otras neuronas. Si en cambio, la conectividad es menor, una neurona que inicialmente está muerta, es probable que no se active, aún si es necesaria para una adecuada reconstrucción.

[1]La derivada de la función ReLU no está definida en cero, pero en modos prácticos se suele considerar la subderivada cero.

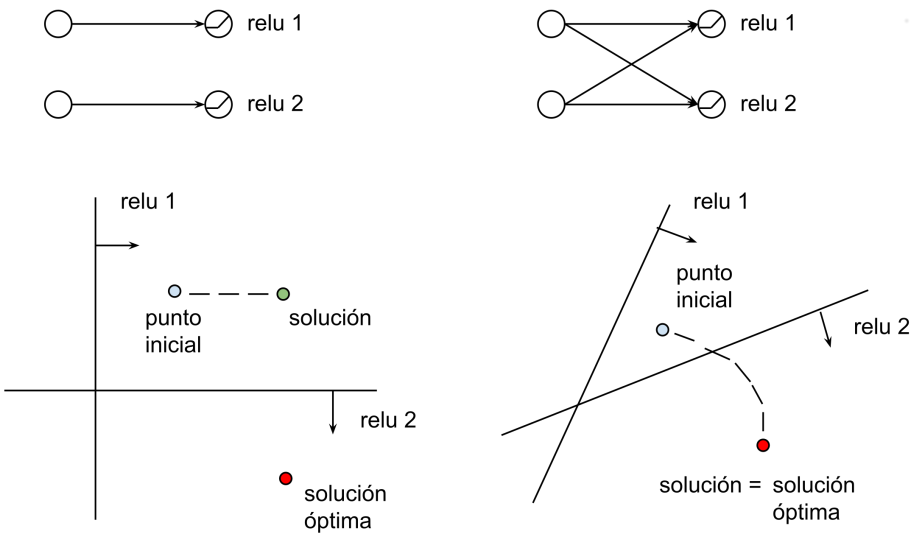


Figura 4.2: Ejemplo ilustrativo en dos dimensiones del problema de las neuronas muertas cuando la conectividad de las mismas es baja. A la izquierda, en el punto inicial, solo una neurona está encendida. Como ambas neuronas no comparten entradas, la segunda neurona no se enciende durante la optimización, impidiendo alcanzar la solución óptima. En cambio, en el ejemplo a la derecha, al compartir entradas, el gradiente que fluye por la primer neurona termina provocando la activación de la segunda neurona, permitiendo alcanzar la solución óptima.

Por lo tanto, al optimizar sobre el espacio latente, el problema no está acentuado porque la primera capa densa establece una conectividad total, donde todas las unidades están conectadas a todos los valores de entradas. En cambio, cuando optimizamos sobre las capas siguientes, al ser la conectividad mucho menor, en muchas neuronas que se inician apagadas no fluye el gradiente en el proceso de propagación hacia atrás, y, por lo tanto, el vector a optimizar no recibe una señal para moverse en la dirección adecuada. En la Figura 4.2 se muestra un ejemplo ilustrativo en dos dimensiones.

Esto se puede comprobar empíricamente analizando el porcentaje de neuronas encendidas en la capa de ReLUs más próxima a la que se está optimizando, a medida que progresa la optimización desde un punto inicial aleatorio, como se puede ver en la Figura 4.3.

Entonces, frente a esta situación, el valor de inicialización termina siendo crítico para los resultados de la optimización, ya que determina cuáles unidades estarán inicialmente apagadas y cuáles prendidas y esto tendrá un gran impacto en el resto de la optimización. Sin embargo, como mencionamos anteriormente, determinar el punto más adecuado no es trivial.

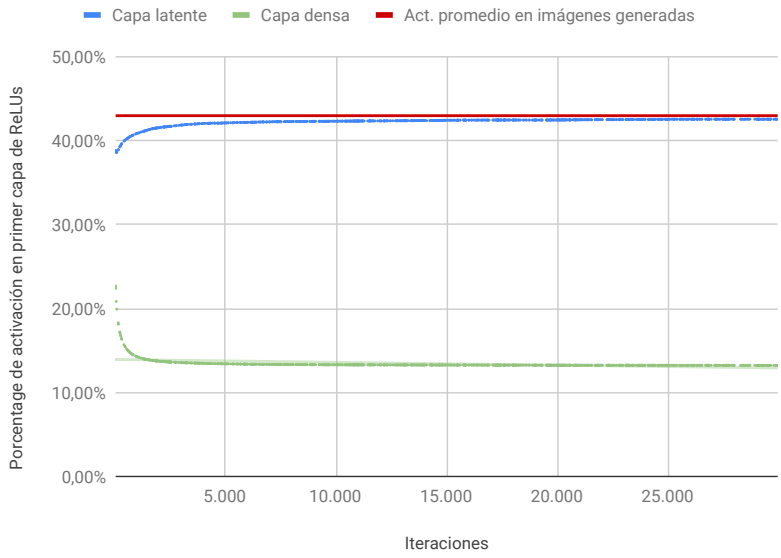


Figura 4.3: Porcentaje de activación en la primera capa de ReLUs al invertir un conjunto de imágenes generadas hasta el espacio latente y hasta el espacio de la primera capa densa. Al optimizar sobre el espacio latente, el porcentaje de neuronas encendidas en la primera capa de ReLUs se mantiene cercano al promedio de activación para las imágenes generadas. En cambio, al optimizar sobre el espacio de la primera capa, el porcentaje de neuronas encendidas decrece a lo largo de la optimización a valores mucho más bajos.

Activación completa

Una posible opción, en el caso de que deseemos invertir las capas que se encuentran inmediatamente antes a la aplicación de la función de activación (ReLUs), consiste en inicializar la optimización desde un punto h_{init} con todas sus componentes positivas (en el ortante positivo), por ejemplo, desde una distribución uniforme: $h_{init} \sim \mathcal{U}(0, 1)$. De esta manera, todas las unidades en el siguiente punto de la red se iniciarán prendidas, fluyendo el gradiente por todas ellas. Sin embargo, si bien vimos que empíricamente este enfoque funciona razonablemente bien, solo puede ser aplicado en las capas anteriores a las ReLUs y además, como se mencionó anteriormente, tiene el defecto de considerar una distribución particular que puede ser muy distinta de la distribución generada en ese nivel de la red.

Inicialización pre-entrenada

Otra posible opción consiste en optimizar primero hasta el espacio latente: $z^* = \arg\min_{z \in \mathcal{Z}} G_2^l(G_1^l(z))$, donde vimos que las neuronas muertas no son un

problema debido a la gran conectividad entre las mismas, y luego inicializar la optimización en el espacio de la capa l desde el punto $h_{init} = G_1^l(z^*)$. Esto permite mitigar el problema, partiendo desde una mejor aproximación en la que el patrón inicial de neuronas apagadas y prendidas está asociado a la imagen que se quiere reconstruir.

Gradiente modificado para las ReLUs

Otra alternativa, independiente de la inicialización, consiste en modificar la computación del gradiente de las ReLUs, de manera que cuando el valor de entrada es negativo ($h < 0$) y el gradiente de la función de costo respecto a la salida de la ReLU es negativo (si la neurona se activara disminuiría el error de reconstrucción) se deje fluir el gradiente multiplicando por un coeficiente que va decayendo exponencialmente a lo largo del proceso de optimización. Es decir, cuando el gradiente es negativo, en el proceso de propagación hacia atrás, la función de activación se comporta como una Leaky ReLU, y de esta manera permite que la neurona se vuelva activar de ser necesario.

Normalmente, el gradiente se calcula analíticamente aplicando la regla de la cadena:

$$\text{grad}(h) = \frac{\partial Loss(\text{relu}(h))}{\partial h} = \frac{\partial Loss(\text{relu}(h))}{\partial \,\text{relu}(h)} \; \frac{\partial \,\text{relu}(h)}{\partial h}$$

En cambio, proponemos modificar el gradiente:

$$\text{grad}_{custom}(h) = \begin{cases} \frac{\partial Loss(\text{relu}(h))}{\partial \,\text{relu}(h)} \; \alpha, & \text{si } h \leq 0 \text{ y } \frac{\partial Loss(\text{relu}(h))}{\partial \,\text{relu}(h)} < 0. \\ \frac{\partial Loss(\text{relu}(h))}{\partial \,\text{relu}(h)} \; \frac{\partial \,\text{relu}(h)}{\partial h}, & \text{si no.} \end{cases}$$

Donde α se inicializa en 0.1 y va decreciendo exponencialmente con el correr del entrenamiento. De esta manera, con la versión modificada del gradiente, muchas neuronas que están inicialmente apagadas se activan, pero con el correr de las iteraciones, al ir disminuyendo el coeficiente α, la fórmula converge al gradiente real, y de esta manera, estabiliza la optimización hacia la solución adecuada.

En la Figura 4.4 se puede observar el porcentaje de unidades activas a lo largo de la optimización para cada una de las propuestas mencionadas, invirtiendo un conjunto de imágenes generadas sobre la primera capa densa de un generador. Partiendo desde un punto aleatorio, la inicialización no está relacionada con la imagen objetivo, por lo que en las primeras iteraciones de la optimización muchas neuronas se apagan para reducir el error de reconstrucción, produciendo una caída en el porcentaje de neuronas activas muy por debajo del promedio de activación en esta capa para las imágenes generadas. En cambio, cuando se utilizan los distintos enfoques mencionados, el porcentaje de activación aumenta considerablemente.

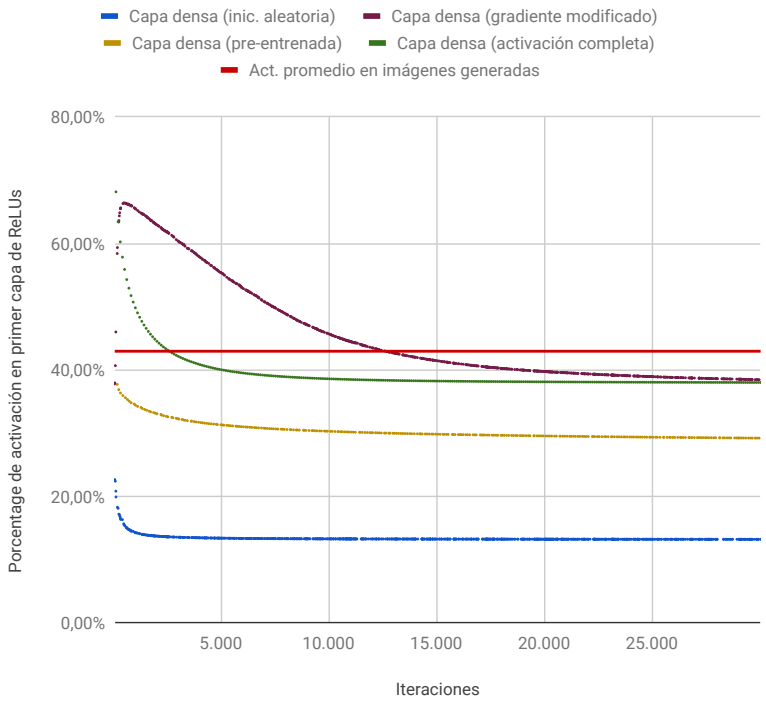


Figura 4.4: Porcentaje de activación en la primera capa de ReLUs al invertir un conjunto de 1000 imágenes generadas hasta el espacio de la primera capa densa, con diferentes métodos.

Como se puede ver en la Figura 4.5, al permitir una mayor activación, estas modificaciones logran reducir el error de reconstrucción considerablemente, obteniendo los mejores resultados con la alternativa del gradiente modificado.

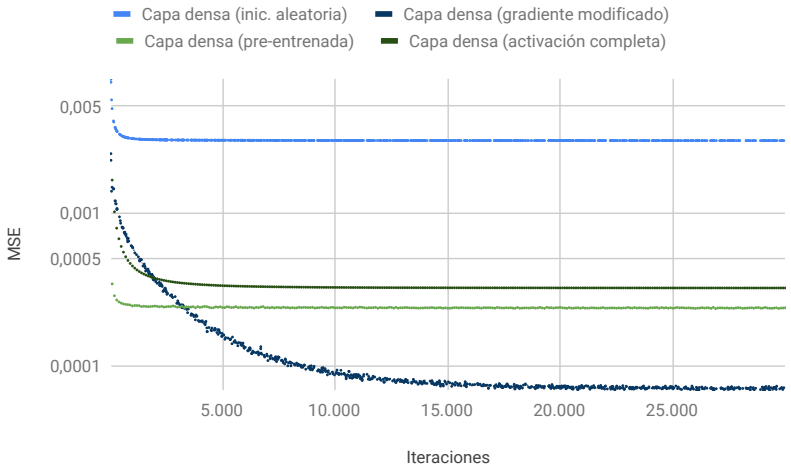


Figura 4.5: Error de reconstrucción al invertir 1000 imágenes generadas hasta el espacio de la primera capa densa, con diferentes métodos.

Capítulo 5

Invirtiendo el Generador DCGAN

En este capítulo analizaremos las representaciones intermedias del generador DCGAN.

5.1. Descripción del Generador y método de entrenamiento

El primer paso consiste en entrenar el generador siguiendo el procedimiento de entrenamiento *Improved WGAN* [69], ya que es una versión que ha mostrado ser mucho más estable que formulaciones previas y constituyó, hasta recientemente, el estado del arte en generar imágenes de manera no supervisada para CIFAR-10 [3]. Para esto, se parte de la versión oficial publicada, trabajando inicialmente con la arquitectura más simple disponible, que sigue el diseño DCGAN [2] para imágenes de 32×32, con la única diferencia de que parte de un ruido Gaussiano en vez de Uniforme. Además, de la formulación original de [69], se modifica la cantidad de mapas de features en la primera capa, de 512 a 128, ya que se obtiene un Inception Score similar (6.68), pero con una dimensionalidad a nivel de features mucho menor.

Arquitectura

Siguiendo la arquitectura DCGAN, la primera parte de la red consiste en una capa densa (Figura 5.1), que permite proyectar el vector latente de 128 dimensiones en un espacio de mucha mayor dimensionalidad (2048). A este vector se le da una interpretación espacial, interpretándolo como 128 mapas de features 4×4. En los siguientes pasos, se utilizan diferentes capas de convoluciones transpuestas. Estas capas aprenden su propio *upsampling*, aumentando el componente espacial de la representación progresivamente hasta una representación final de 3 canales RGB de 32×32. También se utiliza Batch

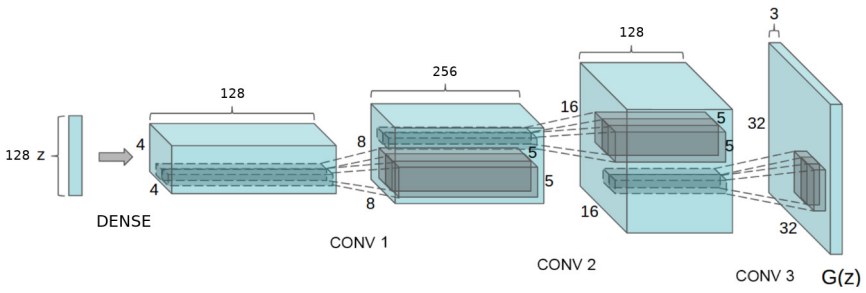


Figura 5.1: Arquitectura DCGAN considerada.

Normalization en todas las capas excepto la de salida. Finalmente, en todas las capas se utiliza ReLUs como funciones de activación, a excepción de la capa final, sobre la que se aplica la función Tanh que restringe la salida al rango $[-1, 1]$.

Diferentes capas

Podemos volver a definir al generador como la composición de operaciones lineales y no lineales:

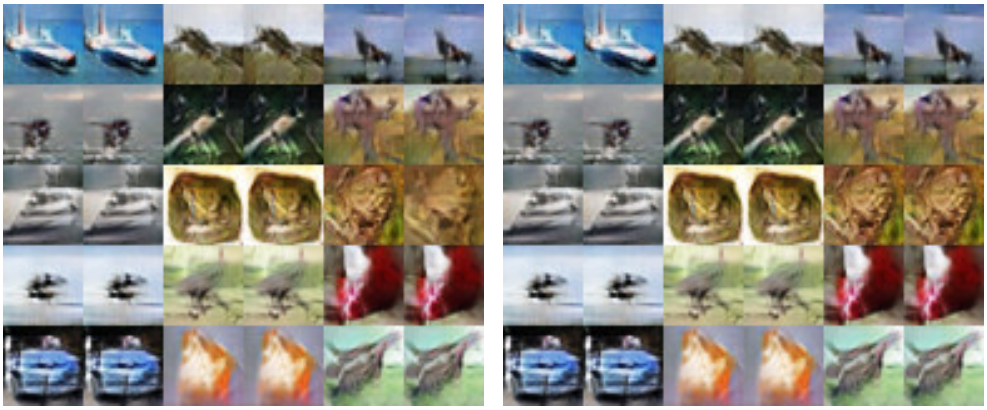
$$G(z) = \tanh(W_3 \ \text{relu}(W_2 \ \text{relu}(W_1 \ \text{relu}(W_0 \ z + b_0) + b_1) + b_2) + b_3)$$

donde las matrices W_i y bias b_i están asociados a los pesos de los parámetros de las capas densas y de manera implícita para las convoluciones transpuestas. Las capas de Batch Normalization se omiten pues, al momento de hacer inferencia, no son más que una transformación afín que puede ser combinada con la matriz y bias asociados a la capa sobre la que se aplica la normalización. De esta manera, si partimos al generador en cada punto de la red, obtenemos 3 sub generadores que representan las operaciones a partir de cada representación intermedia:

$$\begin{aligned} G_2^1(h) &= \tanh(W_3 \, \text{relu}(W_2 \, \text{relu}(W_1 \, \text{relu}(h) + b_1) + b_2) + b_3) \\ G_2^2(h) &= \tanh(W_3 \, \text{relu}(W_2 \, \text{relu}(h) + b_2) + b_3) \\ G_2^3(h) &= \tanh(W_3 \, \text{relu}(h) + b_3) \end{aligned}$$

G_2^1 representa las operaciones a partir de la salida de la primera capa densa, G_2^2 representa las operaciones a partir de la salida de la primer convolución transpuesta, y G_2^3 representa las operaciones a partir de la salida de la segunda convolución transpuesta.

Dada una imagen x, nuestro objetivo es determinar si existe una representación en cada nivel del generador que permita reconstruir dicha imagen. Es decir, encontrar un valor h, tal que la imagen generada a partir de h: $\hat{x} = G_2^l(h)$ sea similar a la imagen original x.



(a) (b)

Figura 5.2: Resultado de invertir imágenes generadas hasta el espacio latente (a) y espacio de la capa densa (b). Se puede observar que las reconstrucciones en ambos casos son casi indistinguibles de la imagen original.

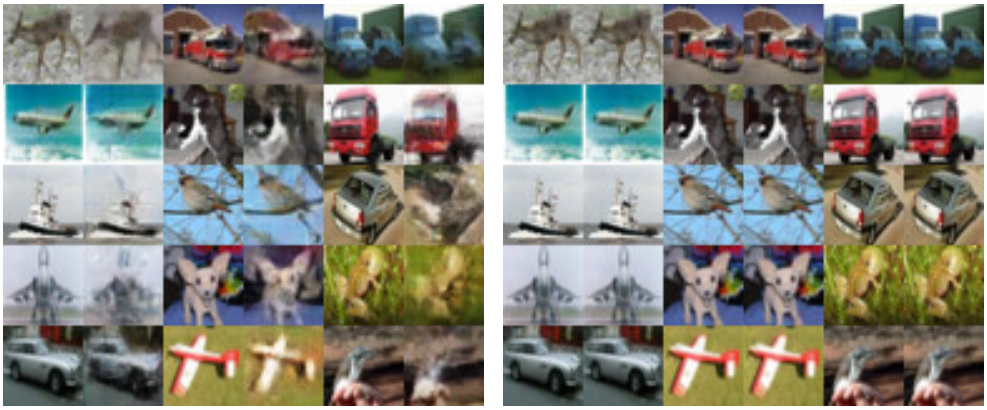
Para G_2^1, G_2^2 y G_2^3, partimos la optimización desde un punto aleatorio h_{init} de acuerdo a la distribución generada en cada punto de la red, y utilizamos la técnica de modificación del gradiente de las ReLUs para evitar el problema de las neuronas muertas.

Analizando las gráficas del error, se decidió por un límite de 30000 iteraciones por imagen, ya que en este punto el error de reconstrucción se estanca en un valor final. Se consideraron diferentes configuraciones para el *learning rate*, incluyendo un posible decremento exponencial a lo largo de la optimización. Se presentan los resultados obtenidos para la mejor configuración encontrada luego de ajustar diferentes parámetros.

5.2. Resultados sobre Imágenes Generadas

Para determinar que el algoritmo funciona correctamente, inicialmente se intenta invertir imágenes generadas, para las cuales efectivamente existe una representación en cada punto de la red. Estos valores se pueden tomar como referencia para después analizar los resultados sobre otros datasets, capturando el error base propio del proceso de inversión.

Como se puede puede ver en la Figura 5.2, se recuperan las imágenes generadas hasta un punto donde resulta muy difícil distinguir entre las reconstrucciones y las imágenes originales.



(a) (b)

Figura 5.3: Resultado de invertir imágenes reales del dataset CIFAR-10 hasta el espacio latente (a) y de la capa densa (b). Se puede observar diferencias significativas en las reconstrucciones en el espacio latente, mientras que las reconstrucciones en el espacio de la capa densa son casi indistinguibles de la imagen original.

5.3. Resultados en CIFAR-10

Se repitió el proceso de inversión para 1000 imágenes aleatorias del dataset CIFAR-10 [3], para analizar hasta qué punto la red puede representar las imágenes reales del dataset sobre el que fue entrenado. Como se puede ver en la Figura 5.4, la red presenta valores similares para el error de reconstrucción a los obtenidos sobre las imágenes generadas en todos los niveles excepto en el espacio latente. Es decir, las imágenes reales se pueden invertir con gran precisión hasta el espacio de la primera capa densa, pero hay un salto importante en la capacidad de representación al moverse al espacio latente.

Este salto en la representación se evidencia en la calidad de las reconstrucciones en el espacio latente en comparación a las obtenidas en el espacio de la primera capa densa, como se puede ver en la Figura 5.3. Este hallazgo constituye la motivación inicial para los métodos desarrollados en esta tesina.

Al restringir nuestro análisis al subconjunto de test, los resultados son idénticos a los obtenidos sobre el dataset de entrenamiento. De esto podemos concluir que el generador no sobre ajusta a las imágenes de entrenamiento, es decir, puede representarlas con el mismo nivel de precisión, no memoriza simplemente las imágenes de entrenamiento.

5.4. Resultados en ImageNet

Como paso siguiente, consideramos invertir imágenes reales de otros datasets que no fueron utilizados para el entrenamiento, por ejemplo, ImageNet en el

Tabla 5.1: RMSE promedio al invertir el generador hasta el espacio de diferentes capas, para 1000 imágenes aleatorias de distintos datasets.

	Capa 0 (latente)	Capa 1 (densa)	Capa 2 (conv1)	Capa 3 (conv2)
Img. Generadas	0.014 (0.017SD)	0.008 (0.003SD)	0.002 (0.0004SD)	0.0003 (0.0002SD)
CIFAR-10	0.061 (0.017SD)	0.013 (0.005SD)	0.003 (0.001SD)	0.0004 (0.0005SD)
ImageNet	0.061 (0.019SD)	0.014 (0.005SD)	0.003 (0.001SD)	0.0005 (0.0003SD)
Ruido Blanco	0.280 (0.002SD)	0.222 (0.003SD)	0.062 (0.003SD)	0.022 (0.003SD)

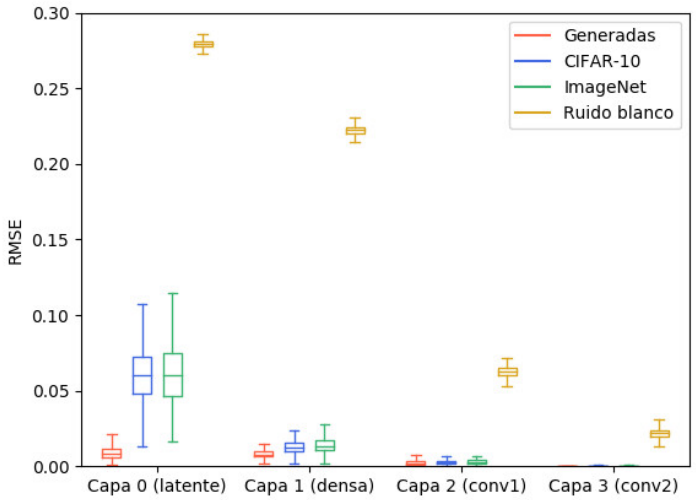
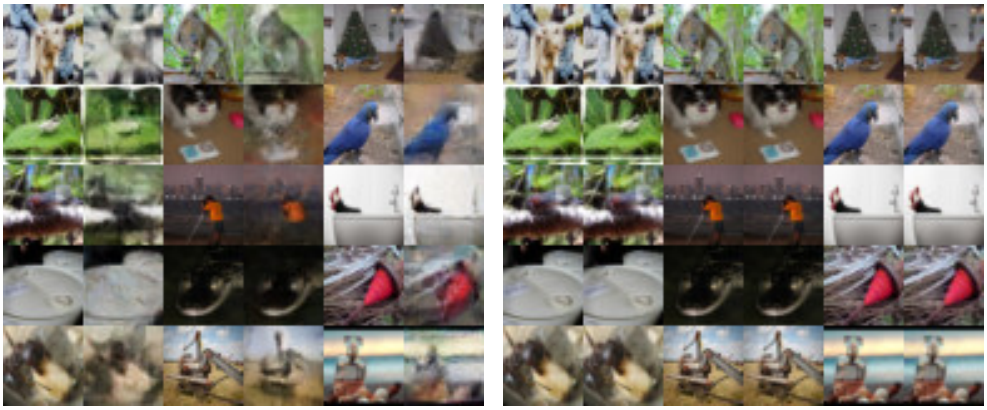


Figura 5.4: RMSE promedio al invertir el generador hasta el espacio de diferentes capas, para 1000 imágenes aleatorias de distintos datasets.

generador entrenado sobre CIFAR-10 (Figura 5.5).

CIFAR-10 [3] consiste de 60000 imágenes color, 32×32 píxeles, de 10 clases de objetos, con 6000 imágenes por clase (50000 de entrenamiento y 10000 de test). En cambio, ImageNet [5] consiste de 1,2 millones de imágenes capturando 1000 clases de objetos. Es decir, ImageNet es un dataset masivo, a otra escala que CIFAR-10, representando mucha más variabilidad en las clases de objetos y situaciones. Para esta aplicación, se realiza un preprocesamiento sobre las imágenes de ImageNet donde son escaladas al tamaño 32×32.

En los experimentos (Figura 5.4) podemos ver valores muy similares a los observados de invertir CIFAR-10. Por lo tanto, podemos concluir que las representaciones intermedias aprendidas por el generador son lo suficientemente expresivas y generales como para representar las imágenes de ImageNet y al mismo tiempo no son específicas para las 10 clases de objetos observados durante el entrenamiento en CIFAR-10.



(a) (b)

Figura 5.5: Resultado de invertir imágenes reales del dataset ImageNet hasta el espacio latente (a) y de la capa densa (b). Se puede observar diferencias significativas en las reconstrucciones en el espacio latente, mientras que las reconstrucciones en el espacio de la capa densa son casi indistinguibles de la imagen original. En este caso, las imágenes son de un dataset distinto al utilizado en el entrenamiento del generador (CIFAR-10).

5.5. Resultados en Ruido Blanco

Para intentar comprender qué tipo de imágenes están filtrando las diferentes capas de la red, y asegurarnos que simplemente no se puede proyectar cualquier imagen, repetimos el proceso de búsqueda de la inversa para imágenes generadas a partir de ruido blanco, es decir, asignándole a cada píxel de la imagen un valor uniformemente aleatorio de manera independiente.

Como se puede ver en las imágenes y sus reconstrucciones (Figura 5.6), a medida que la representación se acerca hacia el espacio latente, las imágenes de ruido blanco no pueden ser representadas con gran precisión.

A diferencia de lo observado sobre las imágenes reales (CIFAR-10 y ImageNet), en las que el error de reconstrucción se mantenía en un rango similar a las imágenes generadas en todas las capas excepto la latente (menor a 0.02), el error de las imágenes de ruido crece de manera considerablemente mayor en cada nivel de la red (Figura 5.4).

Si bien la primera capa (densa) no puede aproximar adecuadamente la imagen de ruido blanco (RMSE de 0.23), la reconstrucción termina siendo bastante aleatoria (Figura 5.6b) donde no se pueden distinguir patrones asociados a los objetos del dataset de entrenamiento (CIFAR-10). Por lo tanto, de esto podemos concluir que, si bien el espacio de representación de la primera capa es capaz de representar las imágenes reales con un error de reconstrucción muy bajo, también permite representar imágenes que no tienen ninguna relación con las imágenes naturales.

(a) (b) (c)

Figura 5.6: Resultado de invertir imágenes de ruido blanco hasta el espacio latente (a), espacio de la capa densa (b), y espacio de la segunda capa (c).

En la representación latente, se puede ver que las reconstrucciones ya no pueden representar píxeles completamente no relacionados (Figura 5.6a). En cambio, el proceso de optimización converge a una imagen con ciertos patrones particulares a los generados en las imágenes de ruido dada, donde en la mayor parte de la imagen tendrá un valor promedio para los píxeles (gris), y en las áreas donde haya una mayor concentración de valores oscuros o claros intentará aproximarlas.

5.6. Análisis de las distribuciones

Una vez obtenidas representaciones en el espacio de la primera capa para todas las imágenes de CIFAR-10, podemos analizar la distribución de las codificaciones de las imágenes invertidas en relación a la distribución de las codificaciones generadas en ese mismo espacio por el generador.

Partiendo al generador en la capa densa, los puntos generados por $G_1(z) = W_0\ z + b_0$ forman parte de un subespacio lineal ($G_1(\mathcal{Z})$) en el espacio de la primera capa ($\mathbb{R}^{2048}$) de dimensión a lo sumo 128, determinado por los 128 vectores columnas de W_0.

De los experimentos por descenso del gradiente en $\mathcal{Z}$, vimos que no es posible conseguir un valor latente que permita reconstruir la imagen real con gran fidelidad. Esto quiere decir que, independientemente de la distribución, no hay punto en el subespacio $G_1(\mathcal{Z}) \subset \mathbb{R}^{2048}$ que genere la imagen objetivo.

Por lo tanto, nos interesa analizar las componentes principales de los puntos obtenidos al invertir G_2 para las imágenes reales, y de esta manera poder compararlos a los puntos generados.

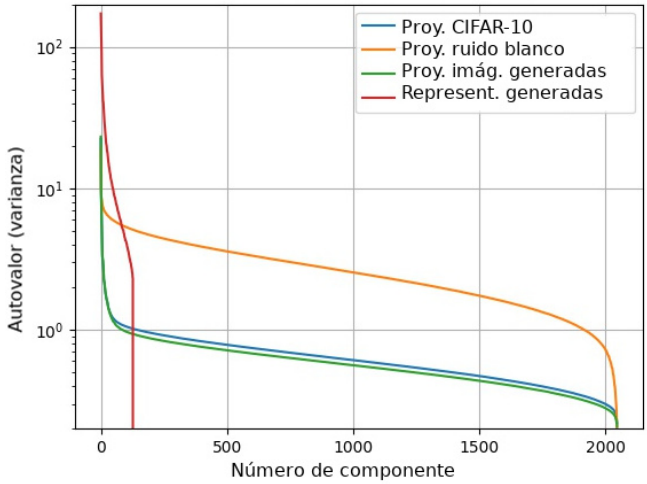


Figura 5.7: Componentes principales para las proyecciones en la primera capa densa de diferentes datasets (CIFAR-10, ruido blanco y imágenes generadas), en comparación a los puntos generados por G_1 en este espacio de la red.

5.6.1. Estudio de las componentes principales

En la Figura 5.7 se puede visualizar los componentes principales para las codificaciones de diferentes datasets en el espacio de la primera capa densa. Como era de esperar, el número de componentes para las codificaciones generadas en este espacio no supera el número de dimensiones en espacio latente (128). Es decir, como las codificaciones generadas son resultado de una transformación afín sobre el vector latente ($G_1(z) = W_0\ z + b_0$) el resultado son puntos en un subespacio lineal (determinado por los 128 vectores columnas de W_0) de dimensionalidad a los sumo 128.

Sin embargo, al invertir las imágenes generadas, es decir, para las que sabemos que hay una representación en este subespacio lineal, obtenemos codificaciones con más de 128 componentes (Figura 5.7). Entonces, podríamos modificar el algoritmo de inversión de manera de incluir una componente en la función de costo que permita buscar, dentro de las posibles codificaciones de la imagen objetivo, aquella más cercana al subespacio lineal de llegada de la capa densa ($G_1(\mathcal{Z})$). Es decir, que anule de ser posible, la mayor cantidad de componentes fuera de las 128 incluidas en este subespacio lineal.

Por lo tanto, proponemos hacer descenso por el gradiente en h, agregando un componente a la función de costo que penalice la distancia de la codificación al subespacio determinado por G_1. Para esto:

- Primero obtenemos una base ortonormal B del espacio columna de W_0.
- Luego, calculamos la proyección de una codificación h en este subespacio

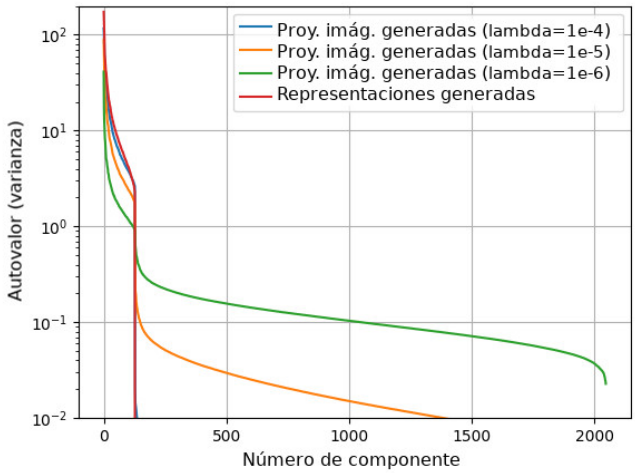


Figura 5.8: Componentes principales para las proyecciones de imágenes generadas en la primera capa densa, para diferentes valores del coeficiente λ_{proy}. Se puede disminuir la distancia a la proyección sin incrementar al error de reconstrucción que se mantiene en 0.01.

lineal como:

$$proy(h) = B\ B^T(h - b_0) + b_0$$

- Y definimos la función de costo:

$$\mathcal{L}(x, h) = \|x - G_2(h)\|_2 + \lambda_{proy} \|h - proy(h)\|_2$$

Invirtiendo imágenes generadas, al incrementar λ_{proy} las principales componentes de las codificaciones se reducen a 128 (Figura 5.8), sin que se reduzca el error de reconstrucción (RMSE promedio se mantiene en el valor 0.01). Esto es razonable ya que para las imágenes generadas hay una representación en el subespacio lineal $G_1(\mathcal{Z})$ que permite reconstruir exactamente la misma imagen.

En cambio, cuando realizamos el mismo experimento con imágenes reales de CIFAR-10 (Figura 5.9), al incrementar λ_{proy} la variabilidad de los codificaciones se concentra en las 128 componentes principales, pero esto viene acompañado de un incremento del error de reconstrucción (RMSE). Cuando el valor de λ_{proy} es muy grande, es decir, la codificación que obtenemos es prácticamente la misma que la proyección en el subespacio $G_1(\mathcal{Z})$, el error de reconstrucción termina siendo de 0.061 en promedio, el cual, como era de esperar, coincide con el error de reconstrucción observado al hacer directamente descenso por el gradiente sobre el vector latente z.

Por lo tanto, podemos concluir que la variabilidad de las imágenes reales de CIFAR-10 no puede ser capturada únicamente por las 128 componentes de la representación aprendida en el primera capa lineal G_1. Si bien podemos obtener una representación de las imágenes reales con un error ínfimo en el espacio de la

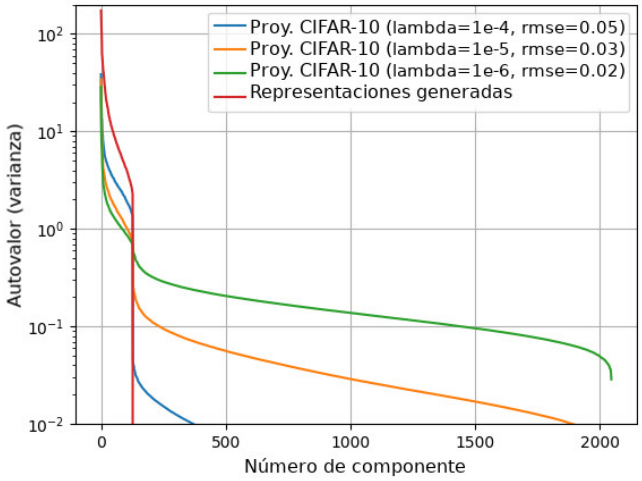


Figura 5.9: Componentes principales para las proyecciones del dataset CIFAR-10 en la primera capa densa, para diferentes valores del coeficiente λ_{proy}. En este caso, al disminuir la distancia a la proyección, se produce un incremento en el error de reconstrucción.

primera capa de la red, cuando intentamos reducir el número de componentes se observa un incremento considerable en el error de reconstrucción.

5.6.2. Calidad de la representación

Si bien las imágenes reales pueden ser representadas en el espacio de la primera capa, estas representaciones serán de utilidad si permiten capturar los factores de variabilidad en los datos, es decir, si son representaciones significativas y no una simple combinación aleatoria de features que permite reconstruir la imagen objetivo. Una forma de analizar la calidad de las representación consiste en visualizar las interpolaciones con otras imágenes representadas en el mismo nivel, donde se espera que *buenas* representaciones resulten en transiciones suaves entre las imágenes, manteniéndose en el conjunto de imágenes reales.

En la Figura 5.10 se puede visualizar interpolaciones entre imágenes reconstruidas en ambos niveles de la red. Si bien se obtienen imágenes intermedias de una calidad similar al espacio latente, en muchos casos las imágenes se degradan perdiendo calidad. Como veremos en las siguientes secciones, es posible mejorar la calidad de las representaciones al regularizar la búsqueda para mantenerse cerca de la distribución de puntos generados. Como la capacidad del generador DCGAN es limitada, resulta más adecuado analizar las interpolaciones en detalle sobre modelos más complejos como los que veremos en las siguientes secciones.

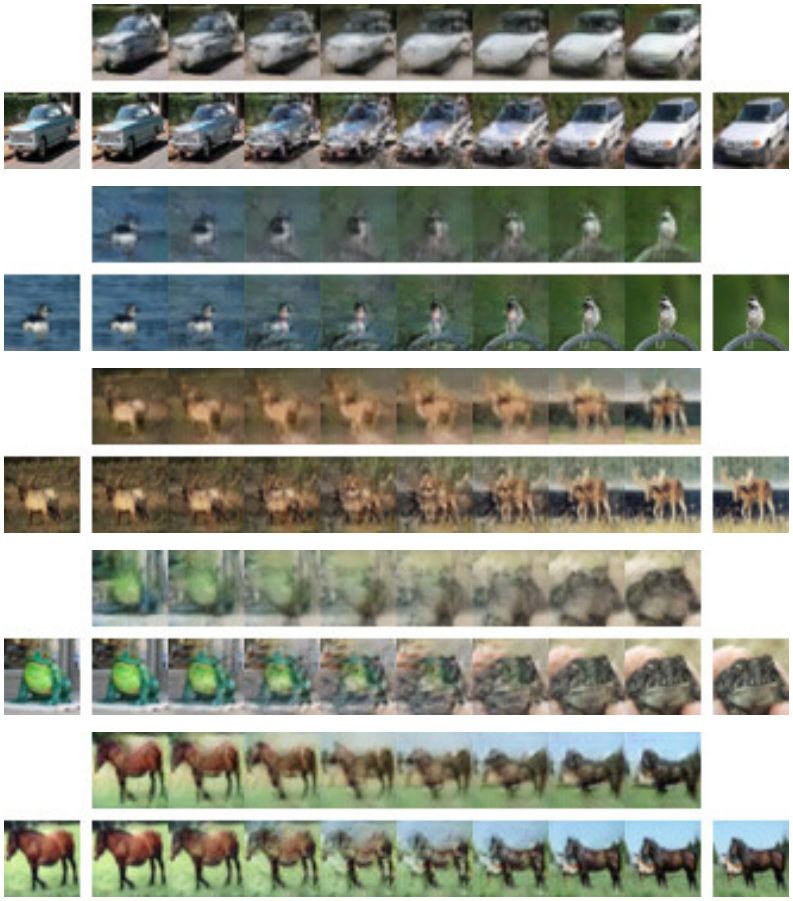


Figura 5.10: Interpolación lineal entre la reconstrucción de dos imágenes reales en el espacio latente (primera fila) y el espacio de la primera capa (segunda fila) del generador DCGAN entrenado sobre CIFAR-10.

5.7. Conclusiones

Las imágenes reales admiten una representación con un error de reconstrucción muy bajo en todos los puntos de la red, a excepción del espacio latente donde se puede ver un salto importante en la capacidad de la representación.

Es decir, empíricamente, vemos que el espacio de salida de la primera capa densa es lo suficientemente expresivo para describir cómo deben ser construidas todas las imágenes reales. Al mismo tiempo, podemos ver que la representación termina siendo una descripción general de las imágenes, pero no particular de las características de las imágenes de CIFAR-10, basado en el hecho de que pudimos invertir también ImageNet-32 utilizando el generador entrenado en CIFAR-10.

Capítulo 6

Invirtiendo el Generador BigGAN

El estado del arte actual trata los datasets de alta variabilidad con Conditional GAN, por ejemplo, en SAGAN [33], donde introducen los bloques Self-Attention y más recientemente en BigGAN [4] donde realmente se presenta un gran salto en la calidad de imágenes generadas para ImageNet, entrenando redes GAN en la escala más grande que se haya intentado.

Por lo tanto, resulta más relevante analizar las representaciones en mayor detalle sobre modelos de Conditional GAN, donde las redes aprenden aparentemente una mejor representación que lo observado en Unconditional GAN.

Muchas veces, al trabajar con modelos más grandes y complejos, resulta difícil poder replicar los experimentos, porque entrenar el modelo involucra mucho tiempo de cómputo, a veces no se encuentra el código disponible, o se encuentra implementado en un entorno distinto al que se acostumbra trabajar, los hiperparámetros deben ser ajustados, etc.

Afortunadamente, el modelo entrenado en BigGAN está disponible, liberado como un módulo en TensorFlow Hub [1], un repositorio donde se pueden publicar modelos previamente entrenados para ser utilizados por otros usuarios de la comunidad. Por lo tanto, es posible cargar el generador de BigGAN y utilizarlo sin necesidad de entrenarlo desde el inicio, lo cual llevaría muchas semanas sobre un cluster con gran capacidad de cómputo, claramente fuera del alcance con los recursos disponibles para esta tesina.

Se encuentran publicadas diferentes versiones del generador para diferentes resoluciones de imágenes. Se decidió trabajar sobre el modelo de más baja resolución (128×128 píxeles) por ser el que requiere menor capacidad de cómputo, pero se espera que los resultados se puedan replicar en cualquiera de las versiones ya que todas comparten una arquitectura similar.

[1] `https://www.tensorflow.org/hub`

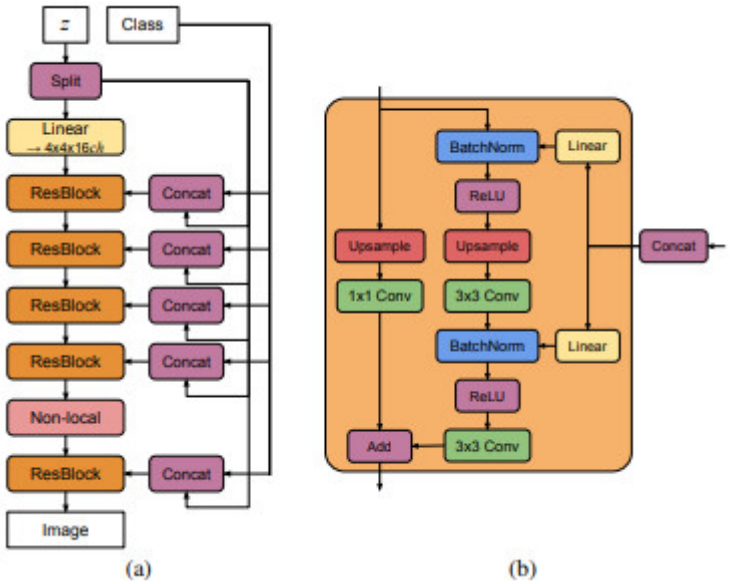


Figura 6.1: Arquitectura general del generador BigGAN para la resolución de 128×128 (a). Estructura interna de cada bloque residual (b). Imagen tomada de [4].

6.1. Arquitectura de la red

El generador BigGan-128 para ImageNet está basado en la arquitectura SAGAN como se muestra en la Figura 6.1.

La red mantiene la primera capa densa inicial en la que parte del vector latente es proyectado en un espacio de alta dimensionalidad y luego se aplica una secuencia de bloques residuales [30] que aumentan la resolución espacial progresivamente, a través de convoluciones y atajos (*shortcut connections*). También se incluye un bloque Self-Attention (Non-local) de acuerdo a lo propuesto en SAGAN.

De manera similar a los modelos anteriores, se utilizan capas de Batch Normalization y ReLUs como funciones de activación.

6.1.1. Bloque Self-Attention (Non-Local)

SAGAN [33] explora la incorporación de los bloques Self-Attention [31][32] en el contexto de GANs para mejorar la capacidad del generador y discriminador para modelar la estructura global en las imágenes.

Las redes GAN tradicionales, basadas en capas convolucionales, presentan un estructura que restringe el patrón de conectividad donde los features en una capa se obtienen como función de los features ubicados únicamente en zonas espacialmente cercanas en las capas anteriores. Esta restricción en

la estructura de la red puede resultar limitante para modelar dependencias entre diferentes regiones de las imágenes, tanto en el discriminador, donde los features de diferentes zonas solamente se pueden procesar en conjunto luego de varias convoluciones, como en el generador, donde se generan detalles de alta resolución considerando únicamente los features de zonas cercanas en los mapas de menor resolución.

Frente a esta limitación, se presentan los bloques Self-Attention, que calculan la respuesta en una posición como una suma ponderada de los features en todas las posiciones. Los pesos, que se interpretan como un mapa de atención a diferentes zonas del espacio, se calculan con un costo computacional bajo.

De esta manera, a través de la incorporación de estos bloques en redes entrenadas adversariamente, en SAGAN logran un salto importante en el estado del arte en ImageNet (IS de 36.8 a 52.52), demostrando que los bloque Self-Attention son complementarios a las convoluciones para la generación de imágenes.

Al visualizar los mapas de atención en las imágenes generadas, se observa que la red aprende a poner atención en sectores de similar color y textura, permitiendo que estas zonas sean coherentes.

La red BigGAN se basa en el modelo presentado en SAGAN, aumentando en gran medida el tamaño (un 50 % el número de canales en cada capa, lo cual aproximadamente duplica el número de parámetros), y entrenando sobre lotes de imágenes 8 veces más grandes. De esta manera logran un salto muy importante en el estado del arte sobre ImageNet (IS de 52.52 a 166.5).

6.1.2. Espacios latentes jerárquicos

El generador BigGAN toma como entrada un vector z de dimensión 120 y un valor y (en el rango $[0, 999]$) para la clase que se desea generar de todas las incluidas en ImageNet.

Se utiliza *espacios latentes jerárquicos*, lo que significa que el vector latente de 120 dimensiones se divide en 6 secciones de 20 dimensiones cada una y estos valores se incorporan en diferentes niveles de la red.

La primera sección se utiliza como entrada de la primera capa densa (quiere decir que el subespacio lineal de llegada de la primera capa tiene a lo sumo 20 dimensiones, aunque está embebido en un espacio de dimensionalidad $4 \times 4 \times 1536$) y define la estructura principal de la imagen.

Las 5 secciones restantes del vector latente se inyectan una a una en cada bloque residual junto con la información de la clase, utilizando *Conditional Batch Normalization* [43, 44], de la siguiente manera:

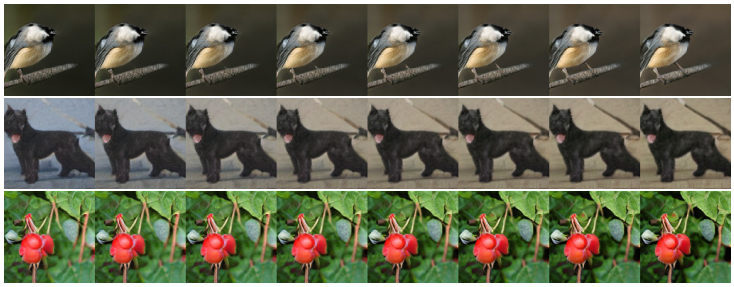
El vector one-hot de la clase (representación dispersa de 1000 dimensiones con un único valor distinto de cero en la dimensión asociada a la clase) se mapea hacia un *class embedding* de dimensión 128. Luego, para cada bloque residual, el *class embedding* se concatena con 20 valores de z, para formar un vector de 148 dimensiones. Este vector se utiliza como entrada para dos capas

densas, una que genera los coeficientes γ, y otra que genera los valores β que se utilizarán para Batch Normalization dentro del bloque residual.

Es decir, la información de las clases y un poco de ruido (z) definen los coeficientes que se utilizan en las capas de Batch Normalization. Si consideramos que la clase se mantiene fija, los componentes de ruido permiten cierta variabilidad en los coeficientes γ y β dentro de la misma clase.

Esta estructura permite comprender lo que luego se observa en los experimentos interpolando sobre el espacio latente:

- Si se interpola linealmente entre dos imágenes generadas, con el mismo valor para las primeras 20 dimensiones de z, y valores aleatorios para las restantes 100 dimensiones (es decir, el mismo valor para la sección de z que se utiliza en la primera capa densa), no se modifica la estructura general de la imagen, sino que únicamente se observan diferencias en algunos detalles:



- En cambio, si de manera opuesta se interpola entre dos imágenes generadas, con valores aleatorios para las primeras 20 dimensiones de z, y el mismo valor para las restantes 100, la estructura de la imagen cambia completamente:

Es decir, a grandes rasgos, los primeros 20 valores de z definen la estructura de la imagen, y los restantes 100 valores simplemente ajustan detalles a través de la variación de los coeficientes de Batch Normalization en cada bloque residual.

El valor de la clase no se incorpora en la primera capa densa. Esto explica que la representación en la primera capa tenga una interpretación general, con

cierta invariancia entre las clases, lo cual se puede comprobar empíricamente interpolando distintas clases para el mismo valor z y observando que la estructura principal de la imagen se mantiene constante (por ejemplo la posición del objeto):

6.2. Invirtiendo al espacio latente

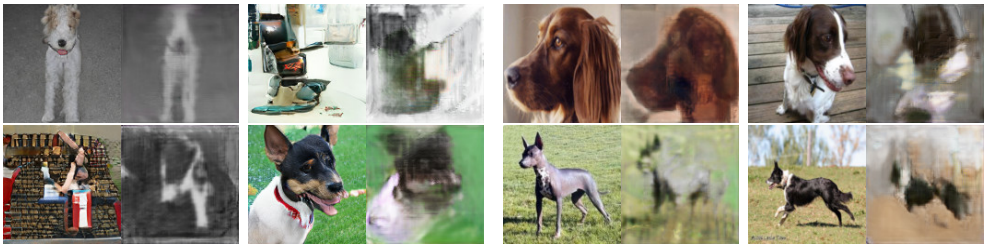
Trabajos previos en aplicar descenso por el gradiente sobre generadores GAN solo trabajan con redes muy simples DCGAN y para datasets de baja variabilidad. Como se detalla a continuación, al intentar invertir redes más complejas como BigGAN, el problema es más difícil, porque la función a optimizar parece ser mucho menos convexa y por lo tanto más susceptible a estancarse en puntos críticos no óptimos.

6.2.1. Dificultad en invertir el espacio latente

Simplemente realizando descenso por el gradiente sobre el MSE a nivel de píxeles no es posible recuperar la representación en espacio latente (z). Esta limitación puede tener dos posibles explicaciones: que no exista tal representación para la imagen objetivo, o que sí exista, pero nuestro mecanismo de optimización fracase en encontrarla.

La red BigGAN es bastante compleja, por lo que la función de costo (MSE) con respecto al vector latente termina siendo altamente no convexa, susceptible a estancarse en mínimos locales o puntos silla, y por lo tanto dependiente, entre otras cosas, de una correcta inicialización.

Esto lo podemos comprobar empíricamente al intentar invertir imágenes generadas, para las cuales sabemos que sí existe un valor de z que permite reconstruir exactamente la misma imagen, donde en muchos casos las optimización no logra buenos resultados (Figura 6.2).



(a) (b)

Figura 6.2: Resultado de invertir imágenes generadas (a) y reales (b) hasta el espacio latente de BigGAN, por descenso por el gradiente sobre el MSE.

6.2.2. Regularización

La red BigGAN es entrenada para generar imágenes reales a partir de valores $z \in \mathcal{Z}$, tomados de un distribución P_z (gaussiana multivariante $\mathcal{N}(0, I)$).

Por lo tanto, deseamos que nuestro proceso de optimización, buscando la inversa de una imagen, se mueva dentro de las regiones probables de $\mathcal{Z}$. Esto podemos lograrlo incorporando términos de regularización a la función de costo que penalicen a los puntos que tienen estadísticas que no son consistentes con la distribución sobre $\mathcal{Z}$.

En [56] proponen agregar un término de regularización que represente la *log likelihood* del vector z en una distribución $\mathcal{N}(0, I)$:

$$\text{log-likelihood}_Z(z) = \log P(z) = \log P(z_1, ..., z_{d_z}) = \sum_{i=1}^{d_z} \log f_{gauss}(z_i)$$

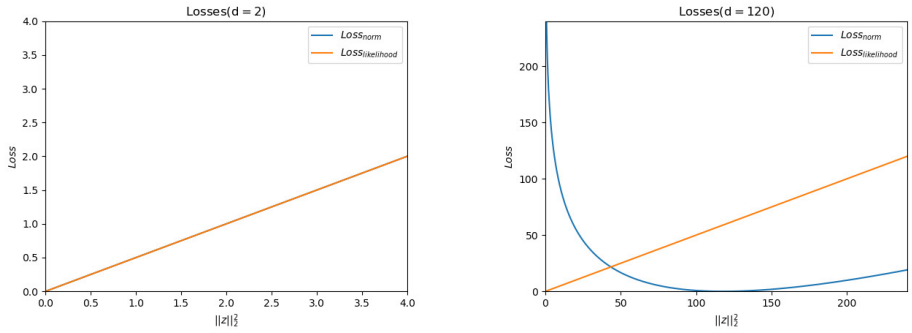
Donde $f_{gauss}(z_i)$ es la función de densidad probabilística de una Gaussiana $(\mu = 0, \sigma = 1)$, y se puede calcular analíticamente, como:

$$\begin{aligned}
\text{log-likelihood}_Z(z) &= \sum_{i=1}^{d_z} \log(\frac{1}{\sqrt{2\pi}} e^{\frac{-z_i^2}{2}}) \\
&= d_z \log(\frac{1}{\sqrt{2\pi}}) - \frac{1}{2} \sum_{i=1}^{d_z} z_i^2 \\
&= -d_z \log(\sqrt{(2\pi)}) - \frac{1}{2} \|z\|^2
\end{aligned}$$

Y se traduce en una regularización sobre la norma cuadrada de z, la cual podemos reformular como:

$$\mathcal{L}_{likelihood}(z) = -\text{log-likelihood}_Z(z) + \text{log-likelihood}_Z(0) = \frac{1}{2} \|z\|^2$$

(sumando el valor mínimo en 0, para normalizar la penalización al rango $[0, \infty]$).



(a) $d_z = 2$ (b) $d_z = 120$

Figura 6.3: Comparación entre los términos de regularización en la norma ($\mathcal{L}_{norm}$) y la log-likelihood ($\mathcal{L}_{likelihood}$) de z, para espacios latentes de diferente dimensionalidad.

Por otro lado, en lugar de la log-likelihood de z, podemos considerar la log-likelihood de la norma cuadrada de z, definiendo la variable aleatoria: $Q = \|z\|^2$, que tiene una distribución *Chi-Squared* (χ^2):

$$\text{log-likelihood}_Q(\|z\|^2) = \log f_{chi-squared(d_z)}(\|z\|^2)$$

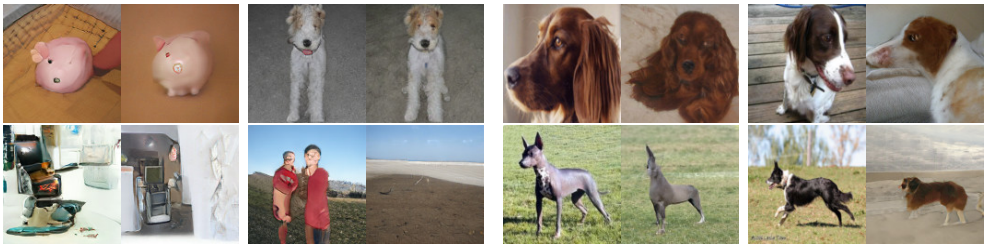
donde $f_{chi-squared(d_z)}$ es la función de densidad probabilística de una distribución *Chi-Squared* con d_z grados de libertad. La podemos reformular como:

$$\mathcal{L}_{norm}(z) = -\text{log-likelihood}_Q(\|z\|^2) + \text{log-likelihood}_Q(d_z - 2)$$

(sumando el valor mínimo en $d_z - 2$, para normalizar la penalización al rango $[0, \infty]$).

La diferencia entre estas dos formulaciones se hace más visible cuando uno trabaja con datos de alta dimensionalidad, ya que por la maldición de la dimensionalidad, la mayoría de los puntos generados no se encuentran en las zonas de máximo valor para la función de densidad probabilística (cerca del origen), sino que se encuentran aproximadamente a la misma distancia del origen. Por lo tanto, la regularización basada en la likelihood de z orienta la búsqueda hacia zonas que, si bien maximizan la función de densidad probabilística, no son las más visitadas durante el entrenamiento.

Sin embargo, en el modelo BigGAN no notamos grandes diferencias entre ambos enfoques. Esto se puede explicar por varias razones. Primero, de acuerdo a lo mencionado en [4], los modelos BigGAN fueron entrenados con regularización ortogonal, que empíricamente se comprobó que resulta en modelos que generan mejores imágenes cuanto más se trunca la distribución (*truncation trick*). Es decir, modelos que generan buenas imágenes incluso para los vectores z de menor norma que los puntos de la distribución utilizada durante el entrenamiento. Seguramente se evidenciaría mayores diferencias entre las



(a) (b)

Figura 6.4: Resultado de invertir imágenes generadas (a) y reales (b) hasta el espacio latente de BigGAN, por descenso por el gradiente sobre $\mathcal{L}_{mse} + \mathcal{L}_{likelihood}$.

diferentes formas de regularizar la búsqueda si se trabajara con modelos que no cumplen esta condición, y generan imágenes con deformaciones de saturación cuando se eligen valores z que no se corresponden con la distribución original.

Segundo, empíricamente vemos que el principal rol de este término de regularización es evitar que la norma de z se dispare hacia valores muy grandes, y produzca las imágenes borrosas que se visualizan al minimizar únicamente el error cuadrático. Y en este sentido, para valores grandes de la norma, ambas alternativas se comportan de manera similar, gradualmente aumenta el costo cuando se aumenta la norma de z.

Otra alternativa para regularizar consiste en simplemente restringir los componentes del vector latente a un cierto intervalo $[-\gamma, \gamma]$ (*clipping*), reemplazando cada valor z_i por $\text{mín}(\text{máx}(z_i, -\gamma), \gamma)$ después de cada iteración de la optimización. Esta regularización restringe la búsqueda a un subconjunto del espacio $\mathcal{Z}$, por lo tanto, no es posible reconstruir exactamente las imágenes generadas (para las cuales el vector latente original podría tener valores por encima del umbral γ), pero puede ser útil para buscar vectores latentes que generen imágenes similares. De acuerdo a lo mencionado en [4], cuanto más se trunca la distribución (*truncation trick*), en nuestro caso, cuanto más pequeño sea el umbral γ, las redes BigGAN generan imágenes que se ven mejor, pero esto viene asociado a una disminución en la variabilidad de las mismas.

En la práctica, no se evidenciaron diferencias significativas entre estos enfoques, más allá de los detalles mencionados. Se decidió utilizar log-likelihood en este caso, porque se comporta mejor para la inicialización elegida en valores pequeños de z, donde la norma inicial del vector latente es muy pequeña.

Al incorporar un término de regularización, las imágenes se ven más reales, debido a que la búsqueda se concentra en las zonas visitadas durante el entrenamiento (Figura 6.4). Sin embargo, por lo general la optimización se estanca en una imagen considerablemente distinta (mínimo local).

6.2.3. Features Inception

Las métricas que comparan imágenes píxel a píxel, como el error cuadrático medio (MSE), no capturan muy bien la similitud de las mismas de acuerdo a nuestra percepción de las imágenes naturales [68, 67, 70]. Por ejemplo, si se traslada una imagen un píxel a la derecha, a pesar de ser casi idénticas, el error cuadrático medio puede ser alto.

Como se menciona en [68], en lugar de la ubicación exacta de las variaciones, es más relevante la distribución de las mismas en la imagen. Por lo tanto, se puede lograr cierta invariancia a transformaciones irrelevantes y al mismo tiempo sensibilidad a propiedades importantes de la imagen, como bordes y texturas, a través del uso de métricas en un espacio de features de más alto nivel, como los extraídos por las redes convolucionales.

Debido a la estructura de la redes entrenadas para clasificar imágenes, con diferentes capas convolucionales que se van componiendo una encima de la otra, se aprenden features de una naturaleza jerárquica. Las primeras capas convolucionales, de poca complejidad y bajo alcance visual, están asociadas a patrones simples de más bajo nivel y, a medida que se incrementa la profundidad hacia las últimas capas de la red, la representación se asocia a patrones más complejos con cierta invariancia a la traslación y escala [71]. Hasta una representación final (penúltima capa) donde las imágenes de una misma clase se ubicarán en un sector del espacio que permita separarlos linealmente del resto en la capa final de clasificación (basado en estudios realizados en la visualización de la representación aprendida en [72, 73, 26]).

Sea C_l la activación en la capa l de una red convolucional C entrenada para clasificar ImageNet (por ejemplo InceptionV3 [74]), entonces se propone comparar dos imágenes a través de la distancia euclídea en el espacio de features en la capa l: $\|C_l(x) - C_l(\hat{x})\|_2$.

Si consideramos la representación de las primeras capas de la red InceptionV3, obtenemos resultados similares a los observados con el MSE a nivel de píxeles (que equivale a trabajar sobre el espacio de la capa cero de la red), ya que termina siendo una representación de bajo nivel (Figura 6.5a).

Por otro lado, si se consideran las últimas capas de la red, se obtiene una representación demasiado general de las imágenes, orientadas a la clasificación y no a similitud más visual de las mismas (Figura 6.5b). Es decir, dos imágenes de la misma clase, aún si son muy distintas visualmente (por ejemplo, perros en posiciones totalmente distintas), se espera que tengan una representación cercana de manera que puedan ser linealmente separables de otras clases.

Por lo tanto, luego de experimentar con diferentes capas de la red InceptionV3, se obtienen los mejores resultados al trabajar en el espacio de la capa 7 (InceptionV3/Mixed_7a). En este nivel de representación se logra un compromiso entre la generalización y la preservación de la estructura visual de las imágenes, como se puede observar en las reconstrucciones de la siguiente sección. Cabe aclarar que la red discriminadora es otra posible alternativa para

(a) (b)

Figura 6.5: Resultado de invertir BigGAN para imágenes reales por descenso por el gradiente hasta el espacio latente, utilizando la tercer capa de InceptionV3: Conv2d_3b_1x1) (a) y la última capa antes de la clasificación final: InceptionV3/global_pool (b).

la extracción de patrones. En el caso particular de BigGAN, no se consideró esta alternativa porque el discriminador asociado al generador no se encontraba disponible para ser utilizado (únicamente el modelo entrenado del generador).

6.2.4. Formulación final

Por lo tanto, redefinimos el error de reconstrucción como una combinación lineal de la distancia a nivel de píxeles y la distancia en la capa de features extraídos por la red Inception:

$$\mathcal{L}_{mse\text{-}feat}(x, \hat{x}) = \|x - \hat{x}\|_2^2 + \lambda_{feat} \|C_{inception}(x) - C_{inception}(\hat{x})\|_2^2$$

Combinando el error de reconstrucción con el término de regularización, proponemos la siguiente formulación final para la función de costo a optimizar:

$$\mathcal{L}(x, z) = \mathcal{L}_{mse\text{-}feat}(x, G(z)) + \lambda_1 \mathcal{L}_{likelihood}(z)$$

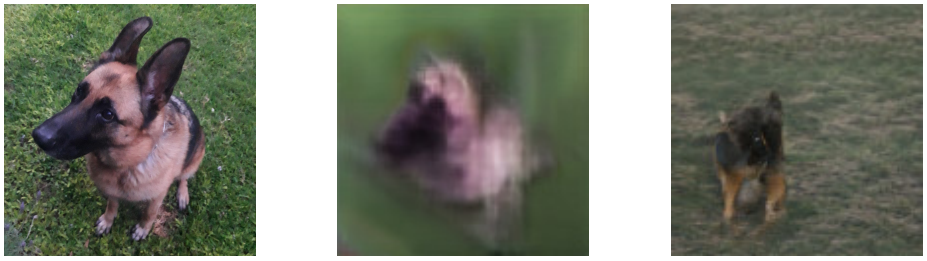
Esta función se puede considerar una variante de las métricas de similitud perceptiva DeePSiM [68], donde en lugar de utilizar un discriminador para computar una función de costo adversaria, esta componente es reemplazada por la regularización en el espacio latente del generador ($\mathcal{L}_{likelihood}$), que guía la búsqueda hacia el espacio de imágenes reales.

Bajo esta formulación se pueden reconstruir sin problemas las imágenes generadas (Figura 6.7a), y al mismo tiempo se obtienen reconstrucciones significativas para las imágenes reales, aunque claramente no exactamente la misma imagen (Figura 6.7b).

En la Figura 6.6 se puede comprobar que todas las componentes son importantes.

6.2.5. Interpretación de lo observado

En la representación a nivel de píxeles, el conjunto de imágenes obtenidas por el generador se encuentra en un manifold sobre el que es posible moverse en



(a) Original (b) $\mathcal{L}_{mse}$ (c) $\mathcal{L}_{mse} + \mathcal{L}_{likelihood}$

(d) $\mathcal{L}_{feat}$ (e) $\mathcal{L}_{feat} + \mathcal{L}_{likelihood}$ (f) $\mathcal{L}_{mse\text{-}feat} + \mathcal{L}_{likelihood}$

Figura 6.6: Resultado de invertir una imagen real hasta el espacio latente para diferentes combinaciones de la función de costo a optimizar.

(a) Imágenes Generadas. (b) Imágenes Reales.

Figura 6.7: Resultado de invertir BigGAN hasta el espacio latente para una muestra aleatoria de imágenes. Las imágenes generadas pueden ser reconstruidas con gran fidelidad (a). Para las imágenes reales, las reconstrucciones en el espacio latente (b) están relacionadas semánticamente a la imagen objetivo, aunque no representan exactamente la misma imagen.

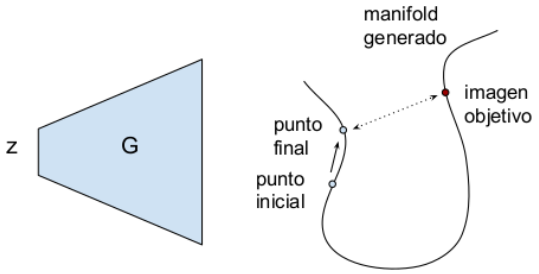


Figura 6.8: Interpretación en el espacio de píxeles de la optimización sobre el espacio latente del generador guiada por el MSE a nivel de píxeles.

diferentes direcciones capturadas por el vector latente. Cuanto más compleja sea la red, más compleja será la forma de este manifold, que, si el algoritmo de entrenamiento funciona correctamente, intentará aproximar el manifold de las imágenes reales, cubriendo la mayor parte de las modas y evitando las zonas de baja densidad (imágenes sin sentido).

Reducir el error cuadrático medio (MSE) equivale a reducir la distancia euclídea en el nivel de píxeles (Figura 6.8). Por lo tanto, desde un punto inicial aleatorio sobre el manifold generado, la distancia euclídea va a guiar la búsqueda en línea recta hacia el punto objetivo, y, debido a la forma compleja del manifold que representa el espacio de búsqueda, es muy probable que el proceso se estanque en un punto no óptimo, donde no es posible reducir la distancia en cualquiera de las direcciones del manifold. Es decir, el gradiente para reducir la distancia euclídea a nivel de píxeles es ortogonal a la superficie del manifold generado, lo se traduce en un gradiente nulo con respecto a los valores latentes, y por lo tanto, la optimización se estanca en este punto.

Siguiendo la hipótesis [63] de que las representaciones más profundas permiten desplegar los manifolds sobre los que los datos se concentran, se podría hipotetizar que la representación aprendida por una red profunda entrenada sobre el conjunto de imágenes reales (por ejemplo InceptionV3) logra desentramar el manifold de las imágenes reales y, por lo tanto, también el manifold generado que intenta ser una aproximación del mismo (Figura 6.9). Esto coincide con lo que luego se observa empíricamente: disminuir la distancia euclídea a nivel de features extraídos por una red profunda provee mejores direcciones en el gradiente que nos permiten movernos en el manifold generado hasta alcanzar la solución óptima.

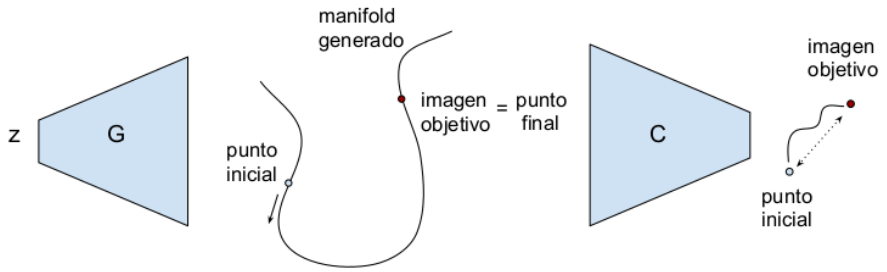


Figura 6.9: Interpretación en el espacio de píxeles de la optimización sobre el espacio latente del generador guiada por la distancia euclídea a nivel de features extraídos por una red clasificadora.

6.3. Invirtiendo al espacio de la primera capa densa

Como paso siguiente, invertimos el generador hasta el espacio de llegada de la primera capa densa, que traduce los primeros 20 valores del vector z hacia puntos en el espacio $\mathbb{R}^{4\times4\times1536}$ (en un subespacio lineal de dimensión a lo sumo 20). Como se mencionó anteriormente, esta primera capa de la red es independiente de la clase sobre la que se trabaja.

Como la red utiliza espacios latentes jerárquicos, aún si se optimiza sobre la primera capa es necesario incluir en la entrada los sectores del vector latente que se inyectan en las siguientes secciones de la red. Por lo tanto, estos valores se optimizan junto con la representación de la primera capa (h).

En coincidencia con lo visualizado para DCGAN en las secciones anteriores, es posible obtener una representación de gran precisión para las imágenes reales en la primera capa, simplemente guiando la búsqueda por el error cuadrático medio (MSE) a nivel de píxeles (Figura 6.10a). Debido a la alta dimensionalidad de este espacio, se tiene mucha flexibilidad para combinar los diferentes features y reconstruir exactamente la misma imagen.

Además, se observa que es posible invertir las imágenes con la misma precisión independientemente del valor de entrada para la clase, es decir, utilizando la clase adecuada o cualquier otro valor aleatorio.

Nuevamente, al replicar los experimentos modificando el gradiente sobre las ReLUs , se reduce el error de reconstrucción considerablemente (Tabla 6.1). Como se puede ver en la Figura 6.10b, al incorporar esta modificación, el proceso de optimización puede encontrar una mejor configuración de neuronas prendidas y apagadas que permiten reconstruir de mejor manera diferentes características de las imágenes.

Al moverse libremente en esta capa del generador, es posible tener una representación general de las imágenes, pero que parece no estar muy asociada a aspectos más semánticos de alto nivel, ya que es posible invertir imágenes

(a) (b)

Figura 6.10: Reconstrucciones al invertir imágenes hasta el espacio de la primera capa densa respecto al MSE (a), incorporando la modificación del gradiente de las ReLUs en (b).

Tabla 6.1: RMSE promedio al invertir el generador hasta el espacio de la primera capa densa, en una muestra de 1000 imágenes aleatorias de ImageNet.

Base	Clase y aleatoria	Gradiente modificado
0.0347 (0.01SD)	0.0382 (0.01SD)	0.0179 (0.006SD)

ignorando la clase sobre la que se trabaja.

A pesar de que se obtiene una muy buena reconstrucción de la imagen, si se modifica en alguna dirección aleatoria, la imagen se degrada rápidamente y no se obtienen interpolaciones interesantes, incluso considerando únicamente las direcciones del subespacio lineal determinado por el vector latente en esta capa (Figura 6.11).

En conclusión, se presentan características similares a la representación de la primera capa del generador DCGAN analizado previamente.

6.3.1. Regularizando la búsqueda en la primera capa

Como se vio en la sección anterior, al moverse libremente en el espacio de la primera capa sin regularizar la búsqueda el algoritmo obtiene representaciones que si bien permiten reconstruir la imagen original, no están relacionadas a la distribución de puntos generados en este espacio de la red durante el entrenamiento. Como resultado, la representación no parece capturar los factores de variabilidad de los datos, ya que al desplazar el vector en cualquier dirección la imagen se degrada.

En cambio, deseamos elegir, de todas las representaciones posibles para una imagen, aquellas más cercanas a la distribución de los puntos generados (P_h), donde el generador fue entrenado para producir imágenes reales. Por

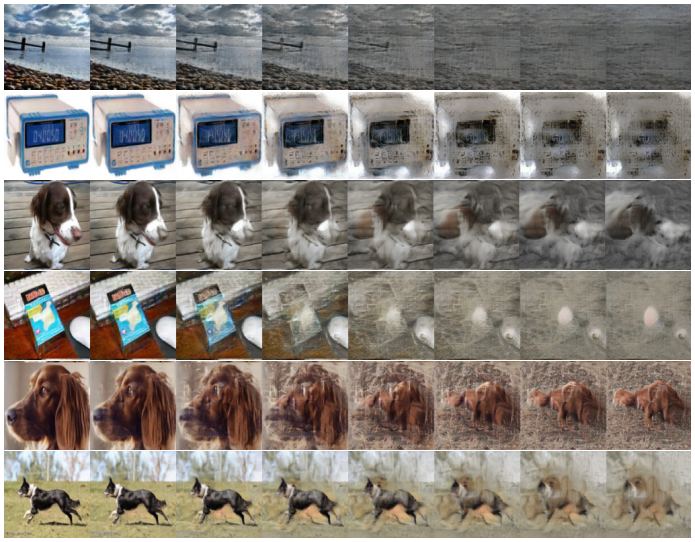


Figura 6.11: Desplazamiento en direcciones aleatorias sobre la primera capa luego de invertir un conjunto de imágenes reales.

lo tanto, proponemos optimizar sobre $G_2(G_1(z) + \delta)$ donde δ representa un desplazamiento en la primera capa con respecto al punto $G_1(z)$. La optimización se divide en dos pasos, primero se optimiza el error de reconstrucción sobre el vector latente:

$$z^* = \arg\min_{z \in \mathcal{Z}} \mathcal{L}_{mse\text{-}feat}(x, G_2(G_1(z))) + \lambda_1 \ \mathcal{L}_{likelihood}(z)$$

Luego, se optimiza sobre un desplazamiento en el espacio completo de la primera capa densa:

$$\delta^* = \arg\min_{\delta \in \mathbb{R}^d} \mathcal{L}_{mse\text{-}feat}(x, G_2(G_1(z^*) + \delta)) + \lambda_2 \ \|\delta\|_1$$

Obteniendo la representación final: $h^* = G_1(z^*) + \delta^*$.

En un espacio de tan alta dimensionalidad como la primera capa, elegimos la norma l_1 para regularizar δ porque promueve soluciones dispersas (*sparse*), incluyendo la menor cantidad de componentes no-nulos necesarios para aproximar adecuadamente la imagen objetivo.

Podemos analizar esta optimización en dos pasos en el espacio de la primera capa densa. Primero, cuando optimizamos hasta el espacio latente, solamente se consideran puntos del subespacio lineal $G_1(\mathcal{Z})$. Incluso, si G_1 es inyectiva (lo cual es razonable de asumir para la primera capa densa), optimizar $G_2(G_1(z))$ con un término de regularización en la log-likelihood de z es equivalente a optimizar $G_2(h)$ sobre todo el espacio $\mathbb{R}^d$ de la primera capa densa, con una regularización en la log-likelihood de h con respecto a la distribución generada P_h.

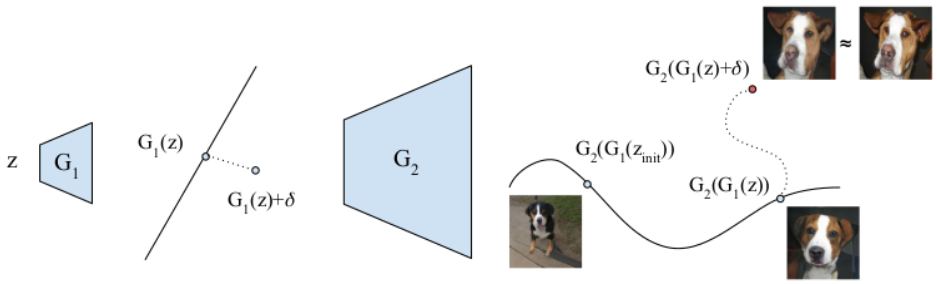


Figura 6.12: Interpretación de la optimización en dos pasos a diferentes niveles de representación. En el espacio de la primera capa densa, primero se considera el subespacio lineal $G_1(\mathcal{Z})$, luego un desplazamiento δ permite explorar el espacio completo de esa capa. A nivel de píxeles, primero buscamos la mejor aproximación en el manifold generado $G(\mathcal{Z})$ y luego δ permite considerar las direcciones capturadas por el manifold $G_2(\mathbb{R}^d)$.

Teorema 2. *Si G_1 es inyectiva:*

$$\arg\min_{h\in\mathbb{R}^d} \mathcal{L}(x, G_2(h)) - \lambda_1 log\ P_h(h) = G_1(\arg\min_{z\in\mathcal{Z}} \mathcal{L}(x, G_2(G_1(z))) - \lambda_1 log\ P_z(z))$$

Demostración.

$$\begin{aligned}
&\arg\min_{h\in\mathbb{R}^d} \mathcal{L}(x, G_2(h)) - \lambda_1 log\ P_h(h) \\
&= \arg\min_{h\in G_1(\mathcal{Z})} \mathcal{L}(x, G_2(h)) - \lambda_1 log\ P_h(h) && (P_h(h) = 0\ \forall h \in \mathbb{R}^d - G_1(\mathcal{Z})) \\
&= \arg\min_{h\in G_1(\mathcal{Z})} \mathcal{L}(x, G_2(G_1(G_1^{-1}(h)))) - \lambda_1 log\ P_z(G_1^{-1}(h)) && (G_1 \text{ inyectiva}) \\
&= G_1(\arg\min_{z\in\mathcal{Z}} \mathcal{L}(x, G_2(G_1(z))) - \lambda_1 log\ P_z(z)) && \square
\end{aligned}$$

Luego, en el segundo paso de la optimización, el desplazamiento δ explota la capacidad interna del generador, considerando todo el espacio de la primera capa ($\mathbb{R}^d$), permitiendo combinar los features de manera libre, fuera de la restricción lineal impuesta por el primer mapping G_1. Notar que esto involucra generar imágenes que no pueden ser generadas desde el espacio latente.

El coeficiente λ_2 representa un compromiso entre la calidad de la reconstrucción y la calidad de la representación obtenida. Valores grandes para λ_2 implicarán que la representación obtenida se mantenga cerca de la distribución generada, pero la reconstrucción podría no recuperar con gran fidelidad la imagen original. Por otro lado, cuando el valor de λ_2 decrece, la optimización se desplaza en mayor medida sobre todo el espacio de la primera capa y por lo tanto, aún si la reconstrucción ($\hat{x}$) se viera más parecida a la imagen original (x), la representación obtenida (h) posiblemente no sea tan significativa.

Esta optimización de dos pasos también puede ser interpretada en el espacio de salida del generador (Figure 6.12). Primero buscamos la mejor aproximación

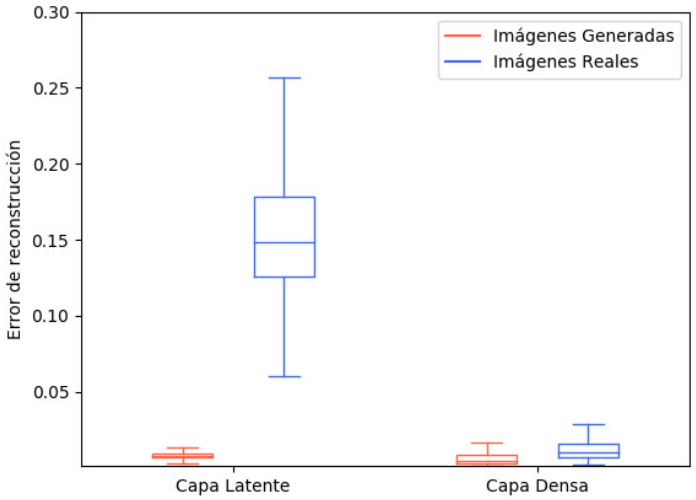


Figura 6.13: Error de reconstrucción ($\mathcal{L}_{mse\text{-}feat}$) para una muestra de 1000 imágenes generadas y 1000 imágenes reales, en diferentes niveles de representación.

de la imagen objetivo considerando d_z direcciones en el manifold generado $G(\mathcal{Z})$. Luego, partiendo de este punto, extendemos la búsqueda a todas la direcciones del manifold $G_2(\mathbb{R}^d)$, buscando la forma más simple de llegar a la imagen objetivo, es decir, la que incluya desplazarse en la menor cantidad de las posibles direcciones.

De esta manera se obtienen representaciones mucho más significativas, lo cual se puede comprobar al interpolar con otras imágenes, como se presenta en la siguiente sección.

6.3.2. Comparación de las representaciones

Para comparar la representación de las diferentes capas de la red, se toma una muestra de 1000 imágenes generadas ($x \sim P_{model}$) y 1000 imágenes aleatorias de ImageNet ($x \sim P_{data}$) y se invierte el generador siguiendo la optimización de dos pasos, primero hasta el espacio latente y luego sobre la primera capa densa. Como se puede observar en la Figura 6.13, hay un salto significativo en el error de reconstrucción sobre imágenes reales en los dos niveles de la red, demostrando la diferencia en el poder de representación de las imágenes naturales.

Al mismo tiempo, las representaciones obtenidas son significativas, como se puede ver al interpolar linealmente en el espacio de la capa densa, resultando en una transición suave entre distintas imágenes. Notar que todas las imágenes intermedias resultantes de estas interpolaciones no pueden ser generadas desde

el espacio lantente, ya que se encuentran fuera del subespacio lineal $G_1(\mathcal{Z})$. Esto demuestra que el generador tiene mayor capacidad de generación de imágenes que la capturada desde el espacio latente.

Salto en la representación. La Figura 6.14 muestra ejemplos de interpolación lineal entre la mejor reconstrucción en el espacio latente y la mejor reconstrucción en el espacio de la capa densa:

$$G_2(G_1(z^*) + \alpha\ \delta^*) \quad \text{con } \alpha \in [0, 1].$$

Interpolación entre imágenes reales. La Figura 6.15 muestra ejemplos de interpolación lineal entre dos reconstrucciones de diferentes imágenes reales (x_1 y x_2) en la misma clase:

$$G_2(\alpha\ h_1 + (1 - \alpha)\ h_2) \quad \text{con } \alpha \in [0, 1]$$

donde $h_1 = G_1(z_1^*) + \delta_1^*$ y $h_2 = G_1(z_2^*) + \delta_2^*$ son las representaciones obtenidas para x_1 y x_2 en el espacio de la capa densa.

Interpolación con imágenes generadas. La Figura 6.16 muestra ejemplos de interpolación lineal entre una imagen real y imágenes generadas aleatorias en la misma clase:

$$G_2(\alpha\ h_1 + (1 - \alpha)\ h_2) \quad \text{con } \alpha \in [0, 1]$$

donde $h_1 = G_1(z^*) + \delta^*$ y $h_2 = G_1(z)$, $z \sim \mathcal{N}(0, 1)$.

Interpolación en la clase. La Figura 6.17 muestra ejemplos de interpolación lineal entre la clase original y una clase distinta para imágenes reales. Es decir, sea $G_2(h, y)$ el generador parametrizado en el vector *one-hot* de la clase y y la representación en la primera capa densa h, se generan las imágenes:

$$G_2(h, \alpha\ y + (1 - \alpha)\ y_{rand}) \quad \text{con } \alpha \in [0, 1]$$

donde h es la representación obtenida para la imagen real, y es la clase de la imagen y y_{rand} es una clase aleatoria (clases representadas como vectores *one-hot*).

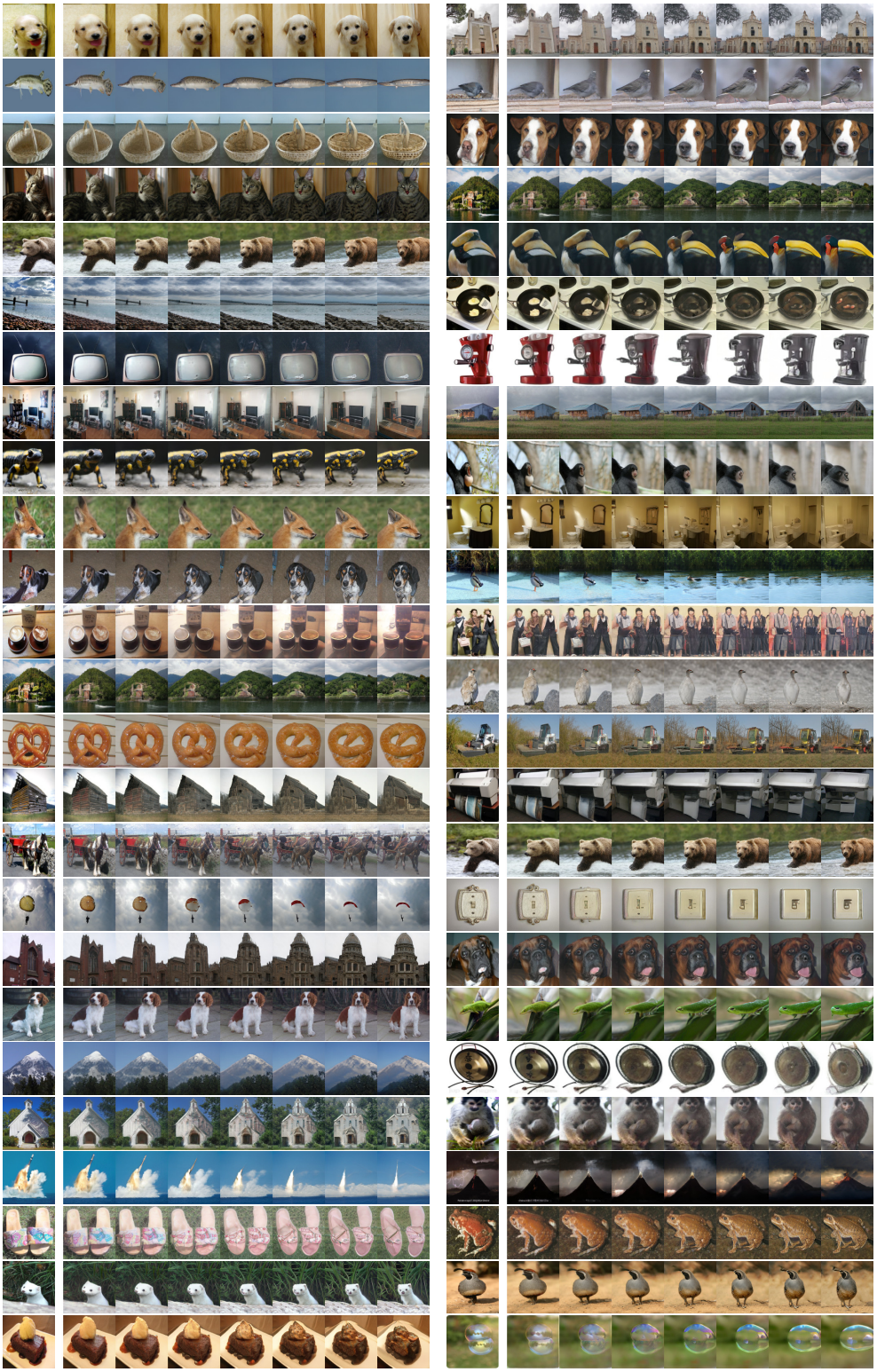


Figura 6.14: Izquierda: Imagen real. Derecha: Interpolación lineal entre la mejor reconstrucción en el primer paso de la optimización ($G_2(G_1(z^*))$, derecha) y la mejor reconstrucción en el segundo paso ($G_2(G_1(z^*) + \delta^*)$, izquierda). Notar que excepto las imágenes en la columna a la derecha, el resto de las imágenes intermedias no pueden ser generadas desde el espacio latente.

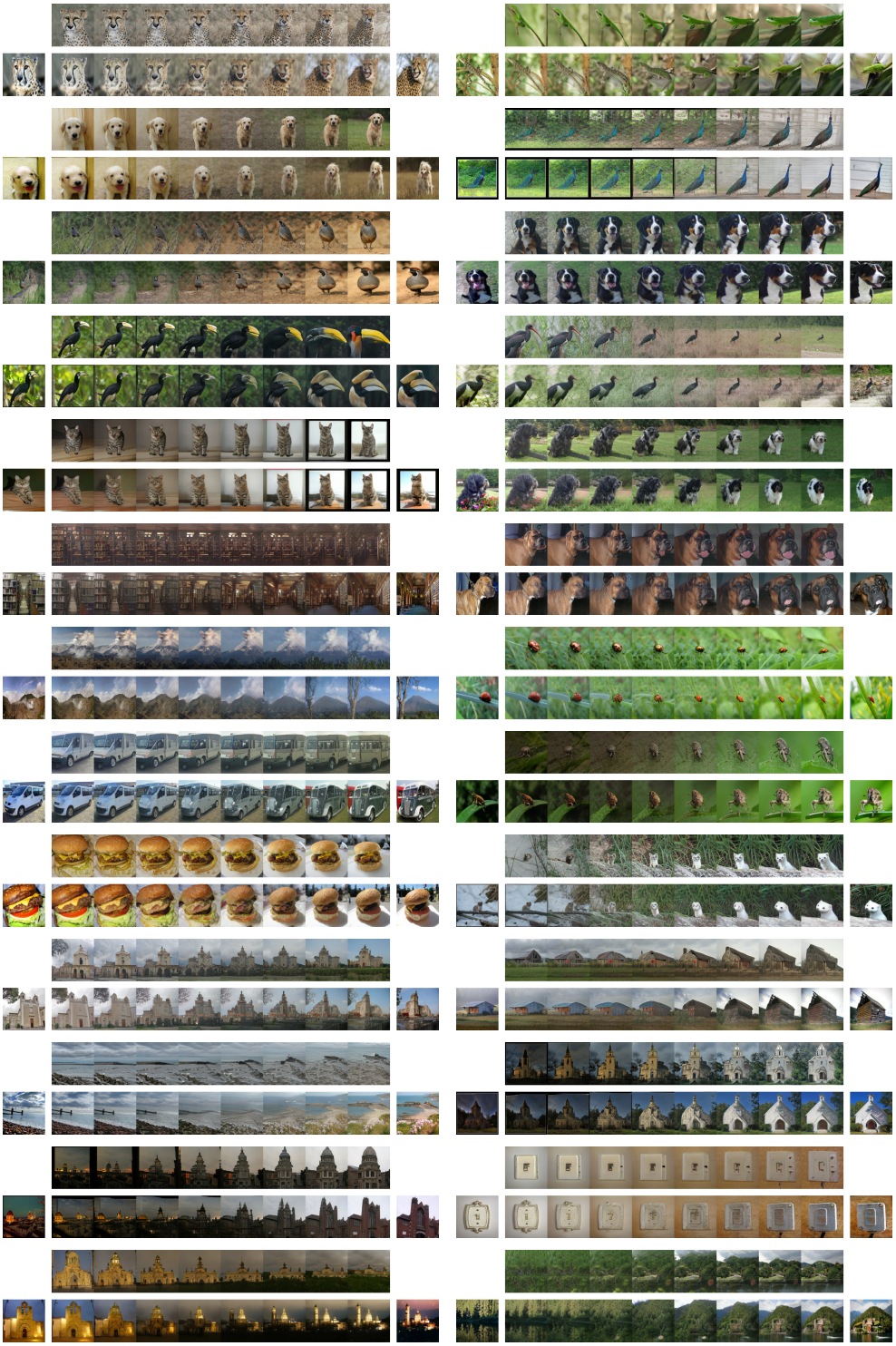


Figura 6.15: Interpolación entre la reconstrucción de dos imágenes reales en la misma clase. Primera fila: reconstrucción en el espacio latente. Segunda fila: reconstrucción en el espacio de la capa densa.

Figura 6.16: Interpolación lineal en el espacio de la capa densa, entre la representación de una imagen real (izquierda) y una imagen aleatoria generada en la misma clase (derecha). Notar que excepto las imágenes en la columna a la derecha, el resto de las imágenes intermedias no pueden ser generadas desde el espacio latente.

Figura 6.17: Interpolación lineal entre la clase original y una clase aleatoria para imágenes reales, luego de invertirlas hasta el espacio de la primera capa densa (reconstrucción de la imagen original en la columna izquierda).

Capítulo 7

Invirtiendo el Generador ProGAN

En esta sección consideraremos un generador entrenado con *Crecimiento Progresivo*, modalidad de entrenamiento que recientemente ha demostrando resultados sorprendentes especialmente para datasets de rostros humanos. Mostraremos que se replican los resultados respecto a la capacidad de la primera capa densa.

7.1. Crecimiento progresivo

Crecimiento progresivo (*Progressive Growing* [48]) consiste en una metodología de entrenamiento para redes generativas donde el tamaño del Generador y Discriminador crecen progresivamente comenzando desde una baja resolución y agregando nuevas capas que permiten modelar incrementalmente más detalles a medida que el entrenamiento progresa hasta un tamaño final de alta resolución.

Esto ha demostrado acelerar el entrenamiento, ya que al ser las redes más pequeñas al inicio, la evaluación es más rápida y por lo tanto es posible iterar sobre más imágenes por unidad de tiempo. También permite estabilizar el entrenamiento, porque el generador aprende de manera incremental, primero modelando la estructura a gran escala presente en la distribución de las imágenes y luego poniendo más atención en los detalles más finos, en lugar de aprender todo junto.

Si bien se ha aplicado en varios casos con muy buenos resultados, implementaciones recientes como BigGAN han encontrado que este proceso no es estrictamente necesario, pudiendo obtener buenos resultados en imágenes de alta resolución, incluso en datasets de alta variabilidad como ImageNet, sin necesidad de incrementar el tamaño de la red progresivamente.

Generator	Act.	Output shape	Params
Latent vector	–	512 × 1 × 1	–
Conv 4 × 4	LReLU	512 × 4 × 4	4.2M
Conv 3 × 3	LReLU	512 × 4 × 4	2.4M
Upsample	–	512 × 8 × 8	–
Conv 3 × 3	LReLU	512 × 8 × 8	2.4M
Conv 3 × 3	LReLU	512 × 8 × 8	2.4M
Upsample	–	512 × 16 × 16	–
Conv 3 × 3	LReLU	512 × 16 × 16	2.4M
Conv 3 × 3	LReLU	512 × 16 × 16	2.4M
Upsample	–	512 × 32 × 32	–
Conv 3 × 3	LReLU	512 × 32 × 32	2.4M
Conv 3 × 3	LReLU	512 × 32 × 32	2.4M
Upsample	–	512 × 64 × 64	–
Conv 3 × 3	LReLU	256 × 64 × 64	1.2M
Conv 3 × 3	LReLU	256 × 64 × 64	590k
Upsample	–	256 × 128 × 128	–
Conv 3 × 3	LReLU	128 × 128 × 128	295k
Conv 3 × 3	LReLU	128 × 128 × 128	148k
Upsample	–	128 × 256 × 256	–
Conv 3 × 3	LReLU	64 × 256 × 256	74k
Conv 3 × 3	LReLU	64 × 256 × 256	37k
Upsample	–	64 × 512 × 512	–
Conv 3 × 3	LReLU	32 × 512 × 512	18k
Conv 3 × 3	LReLU	32 × 512 × 512	9.2k
Upsample	–	32 × 1024 × 1024	–
Conv 3 × 3	LReLU	16 × 1024 × 1024	4.6k
Conv 3 × 3	LReLU	16 × 1024 × 1024	2.3k
Conv 1 × 1	linear	3 × 1024 × 1024	51
Total trainable parameters			**23.1M**

Discriminator	Act.	Output shape	Params
Input image	–	3 × 1024 × 1024	–
Conv 1 × 1	LReLU	16 × 1024 × 1024	64
Conv 3 × 3	LReLU	16 × 1024 × 1024	2.3k
Conv 3 × 3	LReLU	32 × 1024 × 1024	4.6k
Downsample	–	32 × 512 × 512	–
Conv 3 × 3	LReLU	32 × 512 × 512	9.2k
Conv 3 × 3	LReLU	64 × 512 × 512	18k
Downsample	–	64 × 256 × 256	–
Conv 3 × 3	LReLU	64 × 256 × 256	37k
Conv 3 × 3	LReLU	128 × 256 × 256	74k
Downsample	–	128 × 128 × 128	–
Conv 3 × 3	LReLU	128 × 128 × 128	148k
Conv 3 × 3	LReLU	256 × 128 × 128	295k
Downsample	–	256 × 64 × 64	–
Conv 3 × 3	LReLU	256 × 64 × 64	590k
Conv 3 × 3	LReLU	512 × 64 × 64	1.2M
Downsample	–	512 × 32 × 32	–
Conv 3 × 3	LReLU	512 × 32 × 32	2.4M
Conv 3 × 3	LReLU	512 × 32 × 32	2.4M
Downsample	–	512 × 16 × 16	–
Conv 3 × 3	LReLU	512 × 16 × 16	2.4M
Conv 3 × 3	LReLU	512 × 16 × 16	2.4M
Downsample	–	512 × 8 × 8	–
Conv 3 × 3	LReLU	512 × 8 × 8	2.4M
Conv 3 × 3	LReLU	512 × 8 × 8	2.4M
Downsample	–	512 × 4 × 4	–
Minibatch stddev	–	513 × 4 × 4	–
Conv 3 × 3	LReLU	512 × 4 × 4	2.4M
Conv 4 × 4	LReLU	512 × 1 × 1	4.2M
Fully-connected	linear	1 × 1 × 1	513
Total trainable parameters			**23.1M**

Figura 7.1: Arquitectura del modelo entrenado sobre CelebA-HQ para la resolución de 1024×1024 [48].

7.2. Generador entrenado sobre CelebA-HQ

Los resultados de aplicar crecimiento progresivo se han destacado particularmente en la generación de imágenes de rostros, donde se ha demostrado que es posible generar imágenes de personas inexistentes con alta resolución.

La red consiste de una primera capa densa desde el vector latente (512 dimensiones) y una larga secuencia de capas convolucionales que incrementan la resolución progresivamente hasta una resolución final de 1024×1024 píxeles (Figura 7.1).

No se utilizan capas de Batch Normalization, y se utilizan funciones Leaky ReLU como activación, lo que elimina el problema de las neuronas muertas presente en los generadores que utilizan ReLUs. Es decir, el gradiente fluirá siempre por todas las neuronas, aunque claramente en menor medida cuando estén apagadas.

El generador es entrenado en una versión de alta resolución del dataset CelebA (CelebA-HQ), que incluye 30000 imágenes de rostros de celebridades. Estas imágenes están normalizadas para que todos los rostros estén centrados.

7.3. Invirtiendo el Generador

El dataset CelebA-HQ tiene mucho menor variabilidad que los analizados previamente (son siempre rostros de personas centrados, con una leve variación en la orientación), como resultado la red es bastante simple de invertir, pudiendo reconstruir las imágenes generadas simplemente guiando la optimización por el error cuadrático medio a nivel de píxeles (Figura 7.2).

Para las imágenes del dataset real (CelebA-HQ), las reconstrucciones al espacio latente mejoran cuando incorporamos la distancia en el nivel de features

Figura 7.2: Resultado de invertir el generador ProGAN hasta el espacio latente para imágenes generadas.

extraídos por el discriminador al error de reconstrucción, obteniendo imágenes más nítidas y preservando mejor las características del rostro (Figura 7.3):

$$\mathcal{L}_{mse-feat}(x, \hat{x}) = \|x - \hat{x}\|_2^2 + \lambda_{feat} \|D_{feat}(x) - D_{feat}(\hat{x})\|_2^2$$

donde D_{feat} representa la primer parte del discriminador, transformando las imágenes a nivel de píxeles hacia la representación en el espacio de features de una de las últimas capas, en la resolución 8×8 (D_paper/8x8/Conv1/LeakyRelu).

Nuevamente, se observa un salto importante en la representación al invertir las imágenes hasta la primera capa densa, con reconstrucciones considerablemente mejores (Figura 7.3).

Se repitió el proceso de inversión hasta el espacio latente y la primera capa, pero para imágenes nuevas de rostros que no fueron incluidos en el entrenamiento (normalizadas de manera similar a las imágenes creadas para el dataset CelebA-HQ). Se puede notar un salto importante entre la reconstrucciones en ambos niveles de representación (Figura 7.4). En la Figura 7.5 se puede observar que las interpolaciones lineales en el espacio de la primera capa capturan modificaciones relevantes en la distribución, prueba de que las representaciones obtenidas son significativas.

Figura 7.3: Resultado de invertir el generador ProGAN para imágenes reales. De izquierda a derecha: imagen original, reconstrucción en el espacio latente optimizando sobre $\mathcal{L}_{mse}$, reconstrucción en el espacio latente optimizando sobre $\mathcal{L}_{mse\text{-}feat}$, reconstrucción en el espacio de la primera capa densa optimizando sobre $\mathcal{L}_{mse\text{-}feat}$.

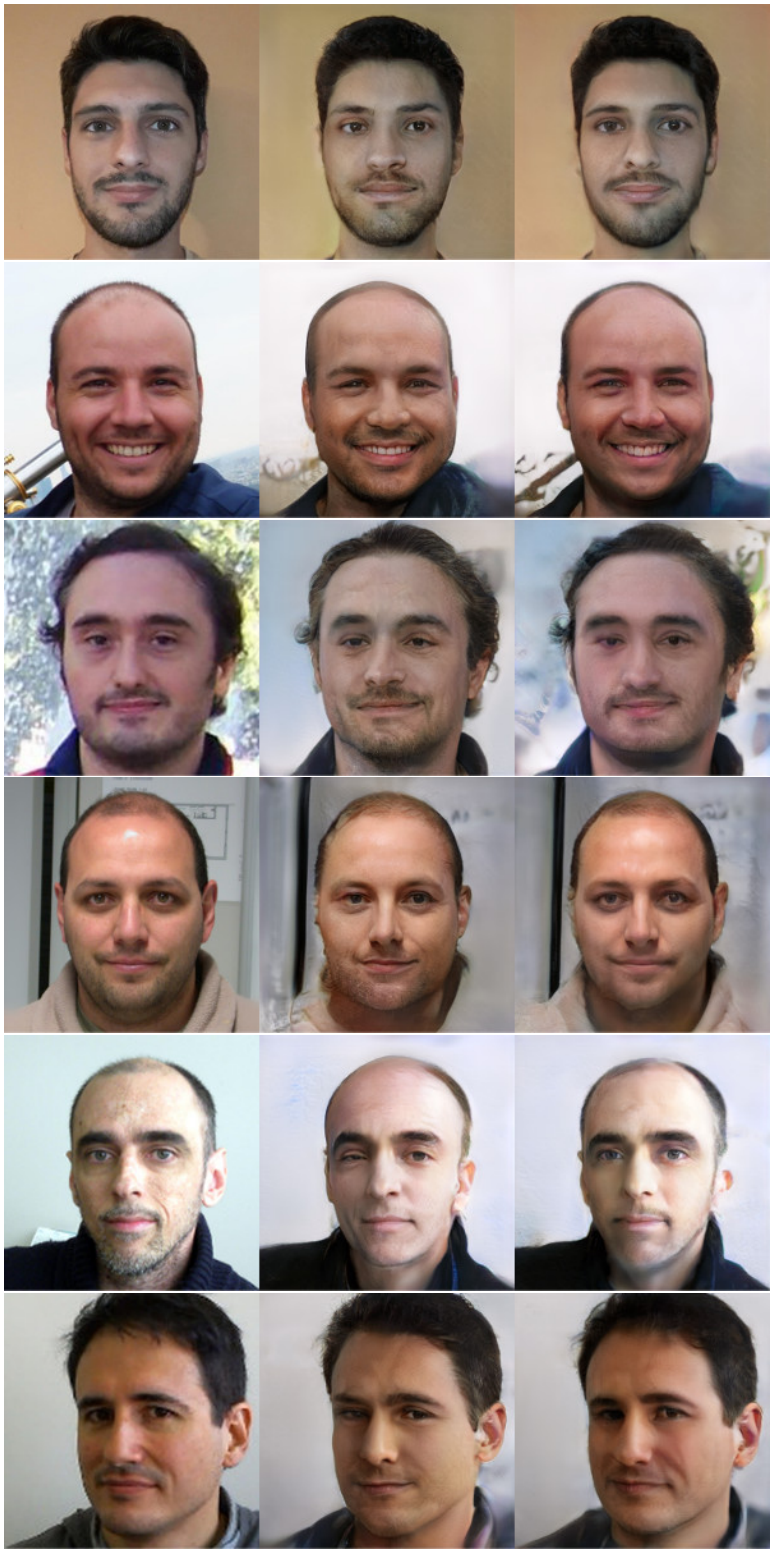


Figura 7.4: Resultado de invertir el generador ProGAN para rostros de personas. De izquierda a derecha: imagen original, reconstrucción en el espacio latente optimizando sobre $\mathcal{L}_{mse\text{-}feat}$, reconstrucción en el espacio de la primera capa densa optimizando sobre $\mathcal{L}_{mse\text{-}feat}$.

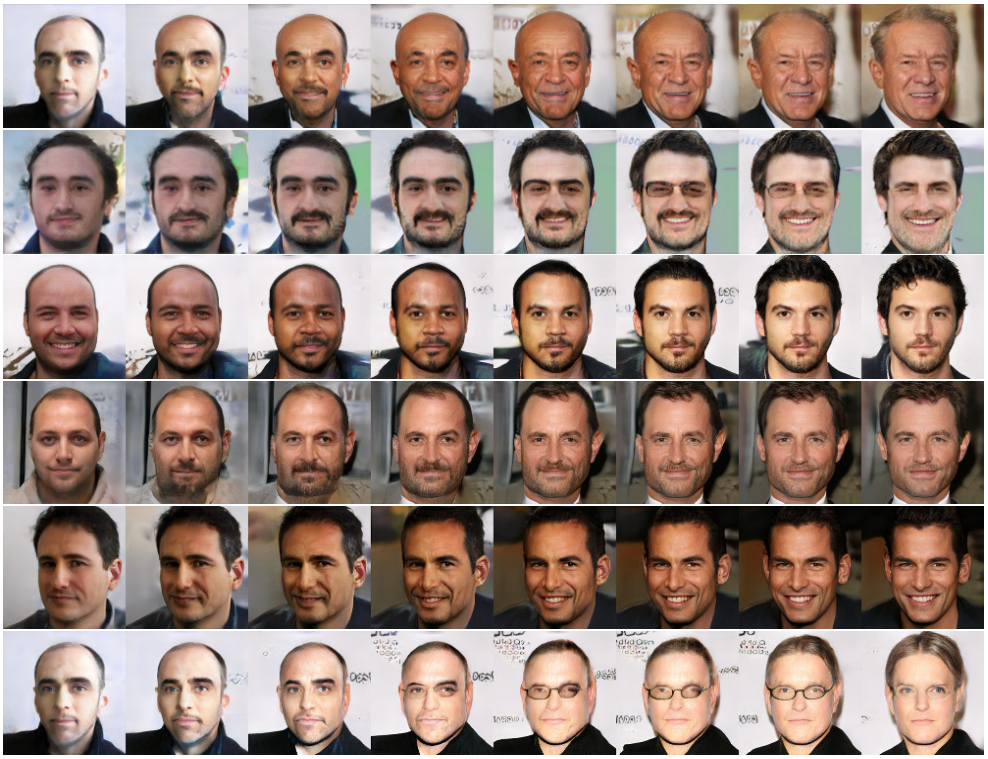


Figura 7.5: Interpolación lineal en la primera capa densa del generador ProGAN, entre la reconstrucción de rostros de personas (izquierda) y imágenes generadas (derecha).

Figura 7.6: Interpolación lineal entre la reconstrucción de dos rostros reales en el espacio latente (primer fila) y en el espacio de la primera capa (segunda fila).

Capítulo 8

Aplicaciones

En la siguiente sección exploramos diferentes aplicaciones como resultado de lograr invertir el generador en diferentes niveles.

8.1. Evaluación del modelo

De acuerdo a lo propuesto por Creswell y Bharath [56], se puede utilizar el error de reconstrucción MSE sobre las imágenes de test para comparar cuantitativamente diferentes modelos generativos. Sin embargo, como vimos para BigGAN, esto no es posible si los modelos son complejos, y por lo tanto difíciles de invertir, ya que un error de reconstrucción alto puede estar asociado a un error en el mecanismo de inversión y no a la capacidad de representación de la red. Es necesario primero comprobar que el mecanismo de inversión pueda reconstruir las imágenes generadas hacia un error de reconstrucción bajo, antes de pasar a comparar las reconstrucciones sobre las imágenes de test.

Por ejemplo, para BigGAN se podría utilizar la formulación final (redefiniendo el error de reconstrucción para incorporar los features de la red Inception y sumando una componente de regularización sobre z), que permite recuperar el vector latente de las imágenes generadas, para analizar el error de reconstrucción sobre imágenes del conjunto de entrenamiento en comparación al conjunto de test. Si se visualizara una gran diferencia, podría suceder que la red haya sobreajustado al conjunto de entrenamiento, memorizando las imágenes y no generalizando adecuadamente para las imágenes nuevas.

Este análisis podría hacerse por clase, y de esta manera determinar qué clases son más difíciles de aproximar por el generador, y si para algunas clases se sobreajusta más que en otras.

8.2. Segmentación no supervisada

Una vez que se adquiere una representación de las imágenes en las primeras capas, se puede obtener información de cómo la red está generando la imagen.

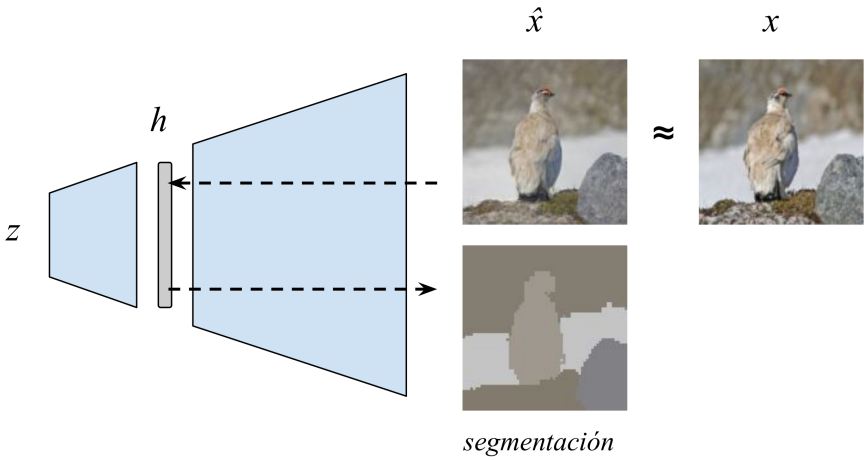


Figura 8.1: Mecanismo de segmentación no supervisada utilizando la representación aprendida por el generador. Primero, la imagen original es reconstruida hasta el espacio de la primera capa densa. Luego, se extrae el mapa de atención para segmentar la imagen.

Es decir, invertir una imagen real, significa aprender a generarla, y la estructura de la red puede brindar información de cuáles son los componentes de la imagen y cómo se relacionan.

Por ejemplo, luego de invertir una imagen real sobre la red BigGAN, se puede analizar el mapa de atención generado en el bloque Self-Attention. El mapa de atención se obtiene de transformar el mapa de features de entrada x_i (i índice sobre N posiciones espaciales) en dos espacios de features ϕ y ψ, y computar la similitud como el producto escalar en estos espacios de embeddings y luego aplicar una capa softmax. Es decir, de acuerdo a la variante Embedded Gaussian [31], la matriz de atención $A \in \mathbb{R}^{N \times N}$ se define como:

$$A_{ij} = \frac{e^{\psi(x_j)^T \ \phi(x_i)}}{\sum_j e^{\psi(x_j)^T \ \phi(x_i)}}$$

A_{ij} indica la atención en la posición j cuando se computa la salida en la posición i. Usando este mapa de atención aprendido, podemos definir una matriz de disimilitud $D \in [0,1]^{N \times N}$ entre los diferentes puntos de la imagen:

$$D = \left(1 - \frac{A + A^T}{2}\right) \odot (1 - I).$$

De esta forma, la matriz es simétrica y la disimilitud de un punto con si mismo es mínima.

En particular, para BigGAN-128, el bloque Self-Attention se aplica al nivel de resolución de 64×64, y incluye una capa intermedia de max pooling, por lo

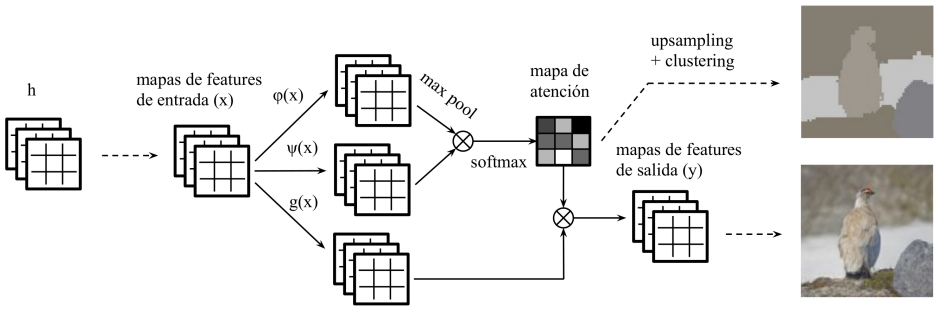


Figura 8.2: Estructura del bloque Self-Attention en la red BigGAN. Luego de invertir una imagen real hasta el espacio de la primera capa densa, se extrae el mapa de atención para segmentar la imagen.

que se obtiene un mapa de atención de 32×32 para cada punto de la resolución 64×64 (modificación para reducir la carga computacional, llamada *subsampling trick* [31]). Es decir, para cada punto en la resolución 64×64, se determina cuánta atención poner en cada sección de la resolución 32×32 (Figura 8.2). Por lo tanto, dada la matriz resultante A, la expandimos espacialmente a un mapa de 64×64 (*upsampling*) A', y luego computamos la matriz de disimilitud D.

Luego, se puede aplicar Hierarchical Clustering sobre la matriz D, utilizando la disimilitud promedio como criterio de enlace entre diferentes clusters. Como resultado, estos clusters se pueden interpretar como una segmentación no supervisada de la imagen a la resolución del mapa de atención (64×64 en nuestro caso).

De esta manera, dada una imagen generada, o una imagen real invertida hasta la primera capa, es posible segmentar la imagen utilizando la estructura de la red. Como se puede ver en la 8.3, secciones de la imagen asociadas a un mismo concepto terminan siendo parte del mismo cluster. En el caso de imágenes reales, la calidad de la segmentación estará relacionada con la calidad de la reconstrucción.

8.3. Editor gráfico

De acuerdo con la *Hipótesis del Manifold*, las imágenes naturales se estima que se encuentran aproximadamente en un manifold a nivel de píxeles [75, 76, 61]. Este manifold es claramente no lineal, como se puede comprobar al realizar simples interpolaciones lineales entre las imágenes, donde las imágenes se degradan rápidamente y se obtienen resultados lejanos al espacio de imágenes reales. Como consecuencia, si simplemente se edita la imagen al nivel de píxeles, es muy fácil moverse fuera del manifold de las imágenes naturales.

Por otro lado, como mencionamos anteriormente, un generador entrenado

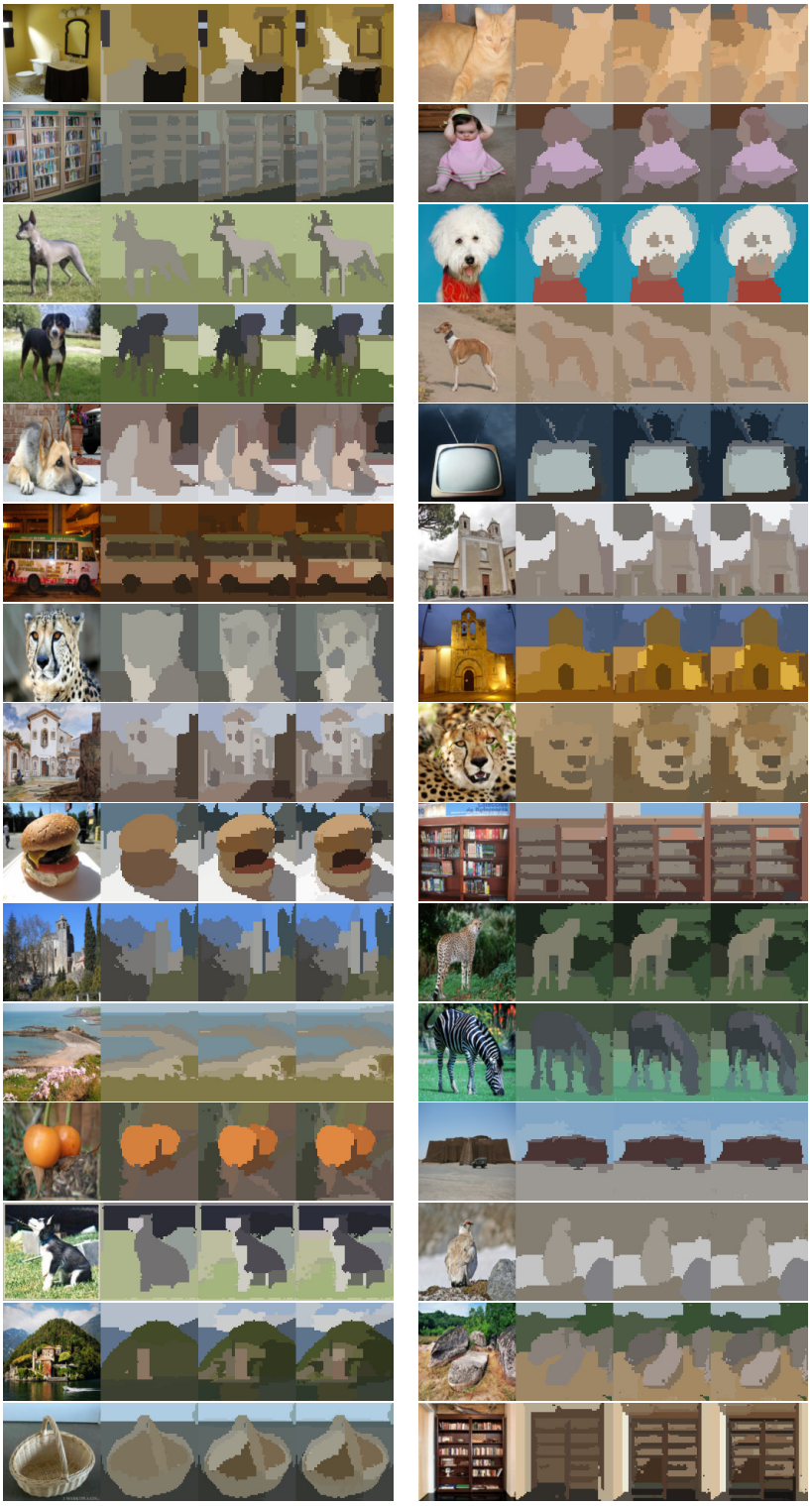


Figura 8.3: Ejemplos de segmentación no supervisada (64×64) para imágenes reales, luego de invertir el generador hasta la primera capa (diferentes números de clusters: 8, 20, 40). Solo para simplificar la visualización, cada cluster se asocia al color promedio de todos sus píxeles miembros.

de manera adversaria aprende a modelar el manifold de las imágenes naturales. Por lo tanto, una vez que se obtiene una representación latente de la imagen objetivo, es posible realizar diferentes operaciones en este nivel de la red, asegurándose que las modificaciones en la imagen de salida se mantienen dentro del conjunto de imágenes reales.

En modelos entrenados de manera no supervisada, los vectores latentes aprendidos son difíciles de interpretar, y las modificaciones esperadas sobre la imagen de salida en general son controladas por un grupo de valores latentes en conjunto. En diferentes trabajos relacionados a esta tesina estudian cómo determinar las direcciones en el espacio latente que generan modificaciones deseadas a nivel de píxeles.

En [52, 53], muestran el desarrollo de editores que permiten modificar imágenes reales, primero invirtiendo el generador y luego traduciendo operaciones al nivel de píxeles (fáciles de expresar para un usuario) como restricciones a optimizar sobre el vector latente, y de esta manera ir moviéndose en el espacio latente buscando generar imágenes reales que satisfagan las modificaciones pedidas, guiados por el gradiente.

En [54, 55] exploran cómo explotar datos etiquetados para encontrar direcciones en el espacio latente correspondientes a características visuales específicas de las imágenes en el espacio de píxeles, por ejemplo, en qué dirección se aumenta la sonrisa de una persona. Computando, por un lado, la media de la representación latente de las imágenes con un atributo, y por otro lado, la media de las imágenes sin el atributo. Entonces, definen el *vector de atributo* como la diferencia entre estos dos vectores, y muestran que moviendo el valor latente en la dirección de este vector (y de manera opuesta) permite modificar el atributo en la imagen de salida. También describen posibles problemas con atributos que están altamente correlacionados en las imágenes (por ejemplo el atributo "hombre" está altamente correlacionado con el atributo "bigote").

Una tercera opción es la presentada en [77], obteniendo vectores de atributos en la representación de la capa de features de un clasificador, y luego optimizando la imagen generada para que se mueva en esa dirección.

Es decir, el gradiente permite relacionar diferentes representaciones de la imagen, donde: $-\nabla_z \left\|G(z) - x\right\|_2$ representa una dirección en espacio latente para producir una modificación dada a nivel de píxeles (x). De manera similar, $-\nabla_z \left\|E(G(z)) - h\right\|_2$ representa una dirección en espacio latente para producir una modificación dada a nivel de features extraídos por un extractor de features E. Dependiendo de la aplicación, puede resultar más simple expresar una modificación dada en una representación distinta de la imagen (por ejemplo, *píxeles*), y luego traducirla en una dirección en espacio latente a través del gradiente.

Sin embargo, todo este tipo de aplicaciones están limitadas por el nivel de reconstrucción de la imagen objetivo, qué tan parecida sea a la imagen original. Por lo tanto, en estos casos resulta muy relevante los resultados obtenidos en esta tesina, que nos permiten obtener para las imágenes reales una

representación muy buena en la primera capa del generador, reconstruyendo la misma imagen. La propuesta es que se pueden replicar todos estos experimentos, pero en el espacio de la primera capa, que resulta ser mucho más expresivo y, al mismo tiempo, permite obtener interpolaciones significativas.

8.4. Edición de video

De manera similar a las imágenes, es posible utilizar el mismo procedimiento para editar los diferentes frames de un video: $f_1, f_2, ...f_n$. Un posible enfoque inicial consistiría en considerar cada frame (f_i) como una imagen independiente e invertirla hasta la representación en la primera capa densa ($h_i/G_2(h_i) \approx f_i$). Sin embargo, al realizar el proceso de optimización de manera independiente para cada frame, las representaciones de frames consecutivos (h_i, h_{i+1}) pueden terminar siendo innecesariamente distantes, porque cada optimización toma un camino distinto y es posible que haya muchas formas de representar la misma imagen. Esto puede dificultar el trabajo posterior con las representaciones obtenidas, por ejemplo, en un generador condicional como BigGAN, si queremos reinterpretar esta misma secuencia de representaciones en una clase distinta, puede resultar que esas diferencias entre frames que eran irrelevantes para la clase original resulten diferencias significativas en la nueva clase, y por lo tanto, las reconstrucciones en la nueva clase sean imágenes muy distintas para frames consecutivos.

En cambio, sería más adecuado obtener una secuencia de representaciones: $h_1, h_2, ..., h_n$, donde la diferencia entre la representación de dos frames consecutivos f_{i-1} y f_i, $\Delta_i = h_i - h_{i-1}$, se minimice para representar las mínimas modificaciones necesarias para ir de una imagen a la otra (Figura 8.4). Esto se puede lograr agregando un nuevo término a la función de costo de la optimización que represente la distancia entre las representaciones de frames consecutivos.

Entonces, al invertir el frame f_i, se minimiza la siguiente función:

$$\mathcal{L}_{frame}(f_i, h_i) = \mathcal{L}(f_i, h_i) + \lambda_{frame} \left\| h_i - h_{i-1} \right\|_1$$

donde λ_{frame} determina la relación entre minimizar el error original y disminuir la distancia entre las representaciones de los frames consecutivos.

De esta manera, primero se invierte el primer frame, y luego, en cada iteración se invierte un nuevo frame, minimizando la distancia respecto a la representación del frame anterior.

Sin embargo, procesar una imagen a la vez termina siendo ineficiente. El proceso puede ser extendido para invertir lotes de frames consecutivos al mismo tiempo, partiendo la optimización desde la representación del último frame invertido, y luego buscando la representación de todos los frames en el lote al mismo tiempo. El algoritmo es el siguiente:

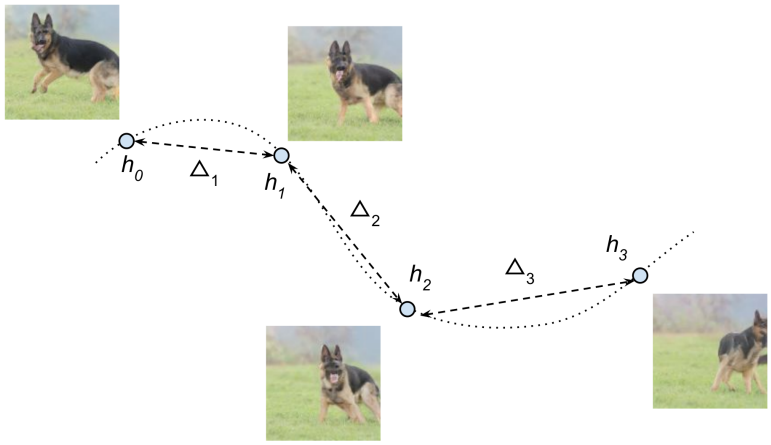


Figura 8.4: Visualización en la primera capa del proceso de inversión de un conjunto de frames consecutivos.

Algoritmo 2: Inversión de una secuencia de frames.

$h_0 \sim P^l_{init}$
mientras *no converge* **repetir**
  $h_0 \leftarrow h_0 - \eta \, \nabla_{h_0} \mathcal{L}(f_0, h_0)$
fin
para cada *batch de k frames* $f_{i+1}, ..., f_{i+k}$ **:**
  $h_{i+1}, ..., h_{i+k} \leftarrow h_i$
  mientras *no converge* **repetir**
    $L \leftarrow \sum_{j=i+1}^{i+k} \mathcal{L}_{frame}(f_j, h_j)$
    para cada $j \in \{i+1, ..., i+k\}$ **:**
      $h_j \leftarrow h_j - \eta \, \nabla_{h_j} L$
    fin
  fin
fin
resultado: $h_0, ..., h_n$

Donde P^l_{init} determina la inicialización aleatoria en el espacio de la capa l.

Este algoritmo se repite dos veces de acuerdo a la optimización en dos pasos. Primero sobre el espacio latente, agregando un término de regularización en la log-likelihood de z y luego sobre el espacio de la primera capa densa, agregando un término de regularización en la norma l_1 de δ.

Como se puede ver en la Figura 8.5, al extender el proceso de inversión hasta la primera capa densa, se pueden reconstruir los frames del video con gran fidelidad.

La Figura 8.6 muestra las diferencias entre las representaciones obtenidas

(a) Secuencia original.

(b) Reconstrucción en el espacio latente.

(c) Reconstrucción en el espacio de la primera capa densa.

Figura 8.5: Reconstrucción de una secuencia de frames para diferentes niveles de representación en la red BigGAN.

con y sin el término de regularización en la distancia de frames consecutivos. Se puede observar que al optimizar sobre $\mathcal{L}_{frame}$, los frames obtenidos al cambiar la clase son más coherentes.

En la Figura 8.7 se pueden ver más ejemplos de interpolación del video a otras clases.

Figura 8.6: Resultado de evaluar el generador sobre las representaciones obtenidas para la secuencia de frames de la Figura 8.5a, cambiando el valor de la clase de entrada a una clase distinta. Primer y tercer secuencias, resultado sobre la representación obtenida al invertir los frames independientemente. Segunda y cuarta secuencia, resultado sobre la representación obtenida al invertir la secuencia de frames en conjunto minimizando las distancia entre frames consecutivos. Se puede observar una mejora considerable en este caso, donde el fondo de la escena se modifica coherentemente frame a frame, y de igual manera, la estructura del animal en movimiento.

Figura 8.7: Ejemplos de interpolación en la clase luego de invertir una secuencia de video. Primera fila: reconstrucción de la secuencia original. Filas restantes: resultado al cambiar el valor de la clase de entrada.

Capítulo 9

Conclusiones

La posibilidad de acceder a representaciones de alto nivel para las imágenes naturales y sus respectivas reconstrucciones permite innumerables aplicaciones de procesamiento y edición de imágenes basadas en la manipulación de rasgos semánticos. El éxito de estas aplicaciones está limitado por la calidad de la reconstrucción y la calidad de la representación alcanzada.

En este trabajo hemos demostrado que es posible alcanzar *buenas* representaciones para las imágenes naturales en el espacio de llegada de la primera capa densa de un generador GAN. Son buenas representaciones en el sentido de que, por un lado, podemos obtener reconstrucciones de alta calidad de las imágenes y, por otro, podemos realizar experimentos de interpolación en el espacio de representación obteniendo una secuencia completa de imágenes significativas desde cualquier imagen real a una imagen objetivo (real o generada).

A diferencia de trabajos previos, que solo consideran invertir el generador hasta el espacio latente, en este trabajo demostramos que es posible explotar la representación interna del generador y mostramos que esto permite una mayor capacidad de representación que no es capturada desde el espacio latente.

Validamos nuestros resultados en tres formulaciones distintas de GAN: Unconditional GAN sobre la arquitectura DCGAN para el dataset CIFAR-10, Conditional GAN con la arquitectura BigGAN para el dataset ImageNet (representa el estado del arte en generación de imágenes), y sobre un generador ProGAN entrenado con Crecimiento Progresivo sobre el dataset CelebA-HQ de rostros. Demostramos que en todos estos casos se repiten los resultados respecto al desempeño de la primera capa, donde el algoritmo propuesto de optimización en dos pasos permite mejorar la calidad de las reconstrucciones al mismo tiempo que se obtienen representaciones significativas como se demostró en múltiples experimentos de interpolación.

Al permitir reconstruir imágenes reales con gran fidelidad, nuestro trabajo tiene un impacto directo sobre trabajos previos en edición de imágenes a alto nivel usando el generador GAN [52] [53] [78].

Como contribución adicional, demostramos que un generador de la com-

plejidad de BigGAN [4] puede ser invertido con un enfoque no paramétrico, mientras que trabajos previos [52] [56] [60] únicamente consideran modelos simples DCGAN y datasets de baja variabilidad.

Finalmente, como nueva aplicación práctica de invertir el generador, mostramos que es posible explotar la representación aprendida en el mapa de atención del generador para obtener una segmentación no supervisada de imágenes reales. También, presentamos ejemplos de potenciales aplicaciones en la edición de video, relacionando las representaciones obtenidas para secuencias de frames en conjunto.

Bibliografía

[1] Ian Goodfellow y col. "Generative adversarial nets". En: *Advances in neural information processing systems*. 2014, págs. 2672-2680.

[2] Alec Radford, Luke Metz y Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks". En: *arXiv preprint arXiv:1511.06434* (2015).

[3] Alex Krizhevsky y Geoffrey Hinton. *Learning multiple layers of features from tiny images*. Inf. téc. Citeseer, 2009.

[4] Andrew Brock, Jeff Donahue y Karen Simonyan. "Large scale gan training for high fidelity natural image synthesis". En: *arXiv preprint arXiv:1809.11096* (2018).

[5] Olga Russakovsky y col. "Imagenet large scale visual recognition challenge". En: *International journal of computer vision* 115.3 (2015), págs. 211-252.

[6] Kevin P Murphy. *Machine learning: a probabilistic perspective*. 2012.

[7] Omnicore Inc. Salman Aslam. *YouTube by the Numbers: Stats, Demographics & Fun Facts*. 2019. URL: `https://www.omnicoreagency.com/youtube-statistics` (visitado 01-06-2019).

[8] Zephoria Inc. Dan Noyes. *The Top 20 Valuable Facebook Statistics – Updated April 2019*. 2019. URL: `https://zephoria.com/top-15-valuable-facebook-statistics` (visitado 01-06-2019).

[9] Tom Mitchell y col. "Machine learning". En: *Annual review of computer science* 4.1 (1990), págs. 417-433.

[10] Ian Goodfellow, Yoshua Bengio y Aaron Courville. *Deep learning*. MIT press, 2016.

[11] Xavier Glorot, Antoine Bordes y Yoshua Bengio. "Deep sparse rectifier neural networks". En: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. 2011, págs. 315-323.

[12] Alex Krizhevsky, Ilya Sutskever y Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". En: *Advances in neural information processing systems*. 2012, págs. 1097-1105.

[13] Kurt Hornik. "Approximation capabilities of multilayer feedforward networks". En: *Neural networks* 4.2 (1991), págs. 251-257.

[14] David H Wolpert. "The lack of a priori distinctions between learning algorithms". En: *Neural computation* 8.7 (1996), págs. 1341-1390.

[15] Yann N Dauphin y col. "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization". En: *Advances in neural information processing systems*. 2014, págs. 2933-2941.

[16] David E Rumelhart, Geoffrey E Hinton y Ronald J Williams. *Learning internal representations by error propagation*. Inf. téc. California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[17] Yann LeCun, Yoshua Bengio y Geoffrey Hinton. "Deep learning". En: *nature* 521.7553 (2015), pág. 436.

[18] Diederik P Kingma y Jimmy Ba. "Adam: A method for stochastic optimization". En: *arXiv preprint arXiv:1412.6980* (2014).

[19] Yoshua Bengio, Aaron Courville y Pascal Vincent. "Representation learning: A review and new perspectives". En: *IEEE transactions on pattern analysis and machine intelligence* 35.8 (2013), págs. 1798-1828.

[20] Marvin Minsky y Seymour A Papert. *Perceptrons: An introduction to computational geometry*. MIT press, 2017.

[21] Geoffrey E Hinton, Simon Osindero y Yee-Whye Teh. "A fast learning algorithm for deep belief nets". En: *Neural computation* 18.7 (2006), págs. 1527-1554.

[22] Yann LeCun y col. "Gradient-based learning applied to document recognition". En: *Proceedings of the IEEE* 86.11 (1998), págs. 2278-2324.

[23] Kaiming He y col. "Mask r-cnn". En: *Proceedings of the IEEE international conference on computer vision*. 2017, págs. 2961-2969.

[24] Karen Simonyan y Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". En: *arXiv preprint arXiv:1409.1556* (2014).

[25] Pierre Sermanet y col. "Overfeat: Integrated recognition, localization and detection using convolutional networks". En: *arXiv preprint arXiv:1312.6229* (2013).

[26] Jost Tobias Springenberg y col. "Striving for simplicity: The all convolutional net". En: *arXiv preprint arXiv:1412.6806* (2014).

[27] Sergey Ioffe y Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". En: *arXiv preprint arXiv:1502.03167* (2015).

[28] Kaiming He y Jian Sun. "Convolutional neural networks at constrained time cost". En: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, págs. 5353-5360.

[29] Rupesh Kumar Srivastava, Klaus Greff y Jürgen Schmidhuber. "Highway networks". En: *arXiv preprint arXiv:1505.00387* (2015).

[30] Kaiming He y col. "Deep residual learning for image recognition". En: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, págs. 770-778.

[31] Xiaolong Wang y col. "Non-local neural networks". En: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, págs. 7794-7803.

[32] Ashish Vaswani y col. "Attention is all you need". En: *Advances in neural information processing systems*. 2017, págs. 5998-6008.

[33] Han Zhang y col. "Self-attention generative adversarial networks". En: *arXiv preprint arXiv:1805.08318* (2018).

[34] Phillip Isola y col. "Image-to-image translation with conditional adversarial networks". En: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, págs. 1125-1134.

[35] Jun-Yan Zhu y col. "Unpaired image-to-image translation using cycle-consistent adversarial networks". En: *Proceedings of the IEEE international conference on computer vision*. 2017, págs. 2223-2232.

[36] Casper Kaae Sønderby y col. "Amortised map inference for image super-resolution". En: *arXiv preprint arXiv:1610.04490* (2016).

[37] Christian Ledig y col. "Photo-realistic single image super-resolution using a generative adversarial network". En: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, págs. 4681-4690.

[38] Scott Reed y col. "Generative adversarial text to image synthesis". En: *arXiv preprint arXiv:1605.05396* (2016).

[39] Han Zhang y col. "Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks". En: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, págs. 5907-5915.

[40] Vincent Dumoulin y Francesco Visin. "A guide to convolution arithmetic for deep learning". En: *arXiv preprint arXiv:1603.07285* (2016).

[41] Mehdi Mirza y Simon Osindero. "Conditional generative adversarial nets". En: *arXiv preprint arXiv:1411.1784* (2014).

[42] Augustus Odena, Christopher Olah y Jonathon Shlens. "Conditional image synthesis with auxiliary classifier gans". En: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, págs. 2642-2651.

[43] Vincent Dumoulin, Jonathon Shlens y Manjunath Kudlur. *A Learned Representation For Artistic Style*. 2016.

[44] Harm De Vries y col. "Modulating early visual processing by language". En: *Advances in Neural Information Processing Systems*. 2017, págs. 6594-6604.

[45] Martin Arjovsky y Léon Bottou. *Towards Principled Methods for Training Generative Adversarial Networks*. 2017.

[46] Martin Arjovsky, Soumith Chintala y Léon Bottou. "Wasserstein gan". En: *arXiv preprint arXiv:1701.07875* (2017).

[47] Takeru Miyato y col. "Spectral normalization for generative adversarial networks". En: *arXiv preprint arXiv:1802.05957* (2018).

[48] Tero Karras y col. "Progressive growing of gans for improved quality, stability, and variation". En: *arXiv preprint arXiv:1710.10196* (2017).

[49] Tim Salimans y col. "Improved techniques for training gans". En: *Advances in neural information processing systems*. 2016, págs. 2234-2242.

[50] Vincent Dumoulin y col. "Adversarially learned inference". En: *arXiv preprint arXiv:1606.00704* (2016).

[51] Jeff Donahue, Philipp Krähenbühl y Trevor Darrell. "Adversarial feature learning". En: *arXiv preprint arXiv:1605.09782* (2016).

[52] Jun-Yan Zhu y col. "Generative visual manipulation on the natural image manifold". En: *European Conference on Computer Vision*. Springer. 2016, págs. 597-613.

[53] Andrew Brock y col. "Neural photo editing with introspective adversarial networks". En: *arXiv preprint arXiv:1609.07093* (2016).

[54] Anders Boesen Lindbo Larsen y col. "Autoencoding beyond pixels using a learned similarity metric". En: *arXiv preprint arXiv:1512.09300* (2015).

[55] Tom White. "Sampling generative networks". En: *arXiv preprint arXiv:1609.04468* (2016).

[56] Antonia Creswell y Anil Anthony Bharath. "Inverting the generator of a generative adversarial network". En: *IEEE transactions on neural networks and learning systems* (2018).

[57] Chunyuan Li y col. "Alice: Towards understanding adversarial learning for joint distribution matching". En: *Advances in Neural Information Processing Systems*. 2017, págs. 5495-5503.

[58] Junyu Luo y col. "Learning inverse mapping by autoencoder based generative adversarial nets". En: *International Conference on Neural Information Processing*. Springer. 2017, págs. 207-216.

[59] Aravindh Mahendran y Andrea Vedaldi. "Understanding deep image representations by inverting them". En: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, págs. 5188-5196.

[60] Zachary C Lipton y Subarna Tripathi. "Precise recovery of latent vectors from generative adversarial networks". En: *arXiv preprint arXiv:1702.04782* (2017).

[61] Hariharan Narayanan y Sanjoy Mitter. "Sample complexity of testing the manifold hypothesis". En: *Advances in Neural Information Processing Systems*. 2010, págs. 1786-1794.

[62] Lawrence Cayton. "Algorithms for manifold learning". En: *Univ. of California at San Diego Tech. Rep* 12.1-17 (2005), pág. 1.

[63] Yoshua Bengio y col. "Better mixing via deep representations". En: *International conference on machine learning*. 2013, págs. 552-560.

[64] Martín Abadi y col. "Tensorflow: A system for large-scale machine learning". En: *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 2016, págs. 265-283.

[65] Leon A Gatys, Alexander S Ecker y Matthias Bethge. "A neural algorithm of artistic style". En: *arXiv preprint arXiv:1508.06576* (2015).

[66] Alex Lamb, Vincent Dumoulin y Aaron Courville. "Discriminative regularization for generative models". En: *arXiv preprint arXiv:1602.03220* (2016).

[67] Justin Johnson, Alexandre Alahi y Li Fei-Fei. "Perceptual losses for real-time style transfer and super-resolution". En: *European conference on computer vision*. Springer. 2016, págs. 694-711.

[68] Alexey Dosovitskiy y Thomas Brox. "Generating images with perceptual similarity metrics based on deep networks". En: *Advances in neural information processing systems*. 2016, págs. 658-666.

[69] Ishaan Gulrajani y col. "Improved training of wasserstein gans". En: *Advances in Neural Information Processing Systems*. 2017, págs. 5767-5777.

[70] Zhou Wang y Alan C Bovik. "Mean squared error: Love it or leave it? A new look at signal fidelity measures". En: *IEEE signal processing magazine* 26.1 (2009), págs. 98-117.

[71] Ian Goodfellow y col. "Measuring invariances in deep networks". En: *Advances in neural information processing systems*. 2009, págs. 646-654.

[72] Matthew D Zeiler y Rob Fergus. "Visualizing and understanding convolutional networks". En: *European conference on computer vision*. Springer. 2014, págs. 818-833.

[73] Karen Simonyan, Andrea Vedaldi y Andrew Zisserman. "Deep inside convolutional networks: Visualising image classification models and saliency maps". En: *arXiv preprint arXiv:1312.6034* (2013).

[74] Christian Szegedy y col. "Rethinking the inception architecture for computer vision". En: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, págs. 2818-2826.

[75] Kilian Q Weinberger y Lawrence K Saul. "Unsupervised learning of image manifolds by semidefinite programming". En: *International journal of computer vision* 70.1 (2006), págs. 77-90.

[76] Haw-Minn Lu, Yeshaiahu Fainman y Robert Hecht-Nielsen. "Image manifolds". En: *Applications of Artificial Neural Networks in Image Processing III*. Vol. 3307. International Society for Optics y Photonics. 1998, págs. 52-64.

[77] Paul Upchurch y col. "Deep feature interpolation for image content changes". En: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, págs. 7064-7073.

[78] Raymond A Yeh y col. "Semantic image inpainting with deep generative models". En: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, págs. 5485-5493.