UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

RECUPERACIÓN DE IMÁGENES BASADA EN DIBUJOS CON TÉCNICAS DE ATENCIÓN

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN COMPUTACIÓN

GONZALO ANDRÉS MONDACA WYMAN

PROFESOR GUÍA:
JOSE MANUEL SAAVEDRA RONDO

MIEMBROS DE LA COMISIÓN:
GONZALO NAVARRO BADINO
MARÍA CECILIA RIVARA ZÚÑIGA

SANTIAGO DE CHILE
2022

Resumen

En la última década han tenido lugar notables avances en técnicas de aprendizaje de máquinas para el reconocimiento de imágenes, basadas en *Deep Learning* y Redes neuronales, que buscan extraer representaciones abstractas por medio del entrenamiento de la red con grandes cantidades de datos de ejemplo.

La mayoría de las arquitecturas se han basado en la convolución discreta como bloque fundamental de construcción. Esta convolución discreta define filtros pequeños que "pasan" por la representación bidimensional de la imagen, de manera similar a como ocurre en la convolución continua entre dos funciones. Estos filtros se componen de parámetros entrenables que la red aprende y actúan como detectores de patrones visuales locales.

A pesar de su utilidad, la convolución presenta algunas desventajas: es ineficiente para relacionar puntos distantes de la imagen (por ser ventanas locales), es invariante a la rotación (el mismo patrón rotado es considerado como diferente), y los filtros resultante son fijos (los patrones que detectan no se adaptan en función de la imagen).

Una creciente de cantidad de investigación explora técnicas "atencionales" que, a grandes rasgos, imitan a la atención cognitiva de la que son capaces los seres humanos. Ilustrativamente, estas confieren la capacidad para atender de manera selectiva a elementos que son considerados más importantes, y permiten relacionarlos de manera contextual. Sin embargo, estas técnicas con frecuencia implican mayores costos de tiempo en la práctica, incluso si su eficiencia teórica se compara a la de modelos convolucionales.

En este trabajo comparamos modelos convolucionales y atencionales puros contra una variedad de modelos híbridos, que utilizan convoluciones y 3 tipos de operación atencional, en distintas proporciones, en tareas de clasificación de imágenes y recuperación de imágenes en base a dibujos. Mostramos que algunos modelos híbridos pueden igualar o incluso superar a sus contra partes convolucionales y atencionales puras, con menor cantidad de parámetros entrenables que las redes convolucionales, y menores costos de tiempo que las atencionales.

Tabla de Contenido

Capítulo 1

Introducción

1.1. Motivación

Si bien la computación ha permitido importantes avances en la automatización de tareas, aún hay una presencia importante de labores monótonas y repetitivas que se escapan a la capacidad de los sistemas computacionales tradicionales. Estos sistemas típicamente se desprenden de nuestro conocimiento en la disciplina respectiva: fórmulas matemáticas o leyes físicas que funcionan de manera consistente y confiable. Sin embargo, muchas tareas dependen de reglas o relaciones que resultan difíciles o infactibles de modelar manualmente. Un ejemplo de estas son las tareas visuales. Resulta fácil para un ser humano distinguir la imagen de un perro de aquella de un conejo, sin embargo, resulta difícil definir reglas automatizables para decidir cuándo un conjunto de pixeles representa uno o lo otro.

Durante la última década se han logrado importantes avances en el desarrollo de técnicas para abordar este tipo de problemas, lo que ha abierto múltiples oportunidades en las aplicaciones prácticas, desde la inteligencia de negocios, procesamiento del lenguaje natural, hasta el ya mencionado reconocimiento de imágenes. Dentro de esta última área, hay dos tareas principales que nos interesan en este trabajo: clasificación y recuperación de imágenes.

La clasificación es una tarea en la que pretendemos identificar la categoría o clase de una imagen entre un conjunto finito de opciones pre-definidas. Por ejemplo, reconocer imágenes de animales, objetos comunes, o monumentos conocidos. Como es de esperar, una limitación típica de los modelos de clasificación es que no pueden reconocer elementos fuera de las clases predefinidas. Si nuestro modelo reconoce solamente las clases “perro”, “pelota” y “guitarra”, obviamente no clasificará correctamente la imagen de un gato.

Por su parte, en la recuperación de imágenes, buscamos encontrar —de un conjunto dado— las imágenes que mejor coincidan con un contenido específico. Distintos sistemas pueden utilizar diferentes formas para especificar este contenido: mediante texto, grabaciones de sonido, fotos, etc. A diferencia de la clasificación, lo que obtenemos del sistema no es una clase o categoría, sino que imágenes, las que típicamente se ordenan según las que mejor coinciden con la consulta. Un ejemplo de sistema de recuperación de imágenes puede ser el buscador de *Google*, en el que podemos introducir cualquier texto y el sistema intentará

retornar las imágenes que mejor coincidan con el texto provisto. *Google* incluso nos permite utilizar imágenes para especificar lo que queremos recuperar.

Notemos que la *tarea* —en este caso, clasificación o recuperación de imágenes— solo describe el problema, y no necesariamente prescribe las técnicas que se utilizan para resolverlo.

De las distintas formas de recuperación de imágenes, existe en particular lo que se conoce como recuperación de imágenes en base a dibujos (SBIR, por sus siglas en inglés *sketch based image retrieval*). En este problema se pretende recuperar una imagen, típicamente de un objeto o lugar real, a partir de un dibujo de consulta. Este problema tiene interesantes aplicaciones comerciales para la búsqueda de contenido, especialmente debido al uso extendido de interfaces táctiles en dispositivos modernos, los que permiten dibujar con el dedo sin necesidad de herramientas adicionales. En el caso de un computador de escritorio, podemos utilizar el puntero. Podemos ver un ejemplo comercial en la Figura 1.1.

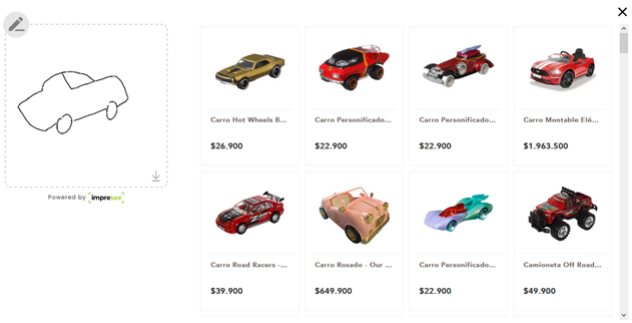

Figura 1.1: Ejemplo de búsqueda mediante dibujo en sitio pepeganga.com

Esta modalidad de recuperación no solo ofrece una forma llamativa y novedosa para buscar contenido, sino que también provee mecanismos alternativos en situaciones en las que una consulta textual equivalente podría ser menos intuitiva o más difícil de implementar, mantener o escalar. Consideremos, por ejemplo, la búsqueda "vestido sin mangas con líneas verticales delgadas y cinturón". En sistemas tradicionales, el éxito de esta consulta requeriría un apropiado etiquetado de la prenda (metadatos), proceso que es típicamente manual y suele ofrecer poca robustez ante variaciones del lenguaje (por ejemplo, si utilizamos la palabra "rayas" en lugar de "líneas", o "lazo" en lugar de "cinturón"). Por contraste, reflejar estas características no es difícil en un dibujo (ver Figura 1.2), el que nos permite evitar el problema de la concordancia de etiquetas, y puede proveer caracterización adicional para la búsqueda.

Con el aumento de popularidad de las técnicas basadas en *deep learning* en la última década, la investigación de múltiples tareas visuales se ha centrado en redes neuronales. Si bien las técnicas propuestas han variado en diferentes aspectos, la mayor parte se ha basado en la operación convolucional como bloque fundamental para la construcción de nuevas arquitecturas. Esta convolución es un análogo discreto a la convolución continúa tradicional, y permite a los modelos aprender y capturar diferentes patrones visuales abstractos mediante su uso repetido.

Las tareas de clasificación y SBIR no han sido la excepción al uso de la convolución. Sin embargo, aunque esta ha demostrado ser muy útil en tareas visuales, presenta algunas desventajas y limitaciones. La convolución no puede capturar eficientemente relaciones entre

puntos distantes de la imagen, y los patrones qué capturan son fijos: no se adaptan según la imagen específica que se esté procesando.

En este escenario, diferentes trabajos han propuesto variaciones o complementos a la operación convolucional [1], [2], [3], [4], [5]. En particular, se ha mostrado el potencial de operaciones basadas en autoatención como unidad fundamental para el procesamiento de imágenes [6], [7]. Las técnicas atencionales imitan a grades rasgos la atención cognitiva de los seres humanos, y en general permiten dar más peso o importancia a elementos que se consideran más importantes. Estas técnicas han llevado a mejoras importantes en otras áreas, y si bien estas han visto avance en clasificación de imágenes, su aplicación en SBIR ha sido comparativamente menos explorada.

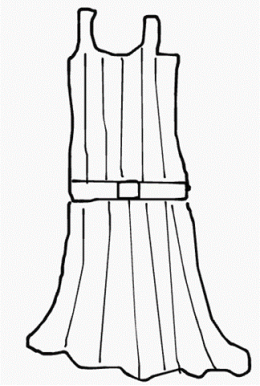

Figura 1.2: Posible dibujo para realizar una búsqueda equivalente a “vestido sin mangas con líneas verticales delgadas y cinturón”. A pesar de la falta de realismo, las características relevantes son fácilemente distinguibles.

En este trabajo buscamos comparar arquitecturas puras, tanto convolucionales como atencionales, contra arquitecturas mixtas, que combinan el uso de convolución y atención en una variedad de configuraciones. Evaluar y comparar arquitecturas de manera “justa” no es conceptualmente trivial. Hablaremos de esto más adelante, pero por ahora basta decir que esperamos que los modelos híbridos retengan la mayor cantidad de ventajas, tanto de los modelos convolucionales puros, como de los atencionales puros. En particular, que los modelos híbridos superen el rendimiento de los modelos convolucionales puros, requiriendo menos parámetros entrenables; y menores tiempos de entrenamiento y ejecución que los atencionales puros.

1.2. Hipótesis

El uso de mecanismos atencionales mejora el rendimiento para SBIR en comparación a modelos puramente convolucionales similares.

1.3. Objetivo general

Explorar el uso de técnicas de autoatención para mejorar el desempeño en la recuperación de imágenes basada en dibujos.

1.4. Objetivos específicos

- Definir mecanismos u operaciones atencionales a explorar.
- Definir arquitectura para línea base convolucional de comparación.
- Formular arquitecturas en que se reemplace parcial o complemente las operaciones convolucionales con componentes atencionales.
- Evaluar las arquitecturas propuestas en modelos de clasificación que utilicen datos de dibujos.
- Evaluar las arquitecturas propuestas en modelos de recuperación para SBIR.
- Evaluar estos módulos usando datos públicos.

Capítulo 2

Marco teórico

2.1. Procesamiento de imágenes y deep learning

Esta primera sección del marco teórico tiene como objetivo orientar al lector familiarizado con las ciencias de la computación, pero no con *deep learning* en particular. La caracterización de conceptos y la terminología específica usada en la literatura varían entre distintas fuentes, y si bien está fuera del alcance de este trabajo describir exhaustivamente el campo, consideramos conveniente ofrecer breves intuiciones, lenguaje y motivación teórica general antes de proceder.

2.1.1. Comprensión de alto nivel en contenido visual

Típicamente, el contenido visual digital se compone de pixeles que describen características de color e intensidad para cada posición en un plano bidimensional finito y discreto. Estas propiedades, al ser proyectadas en un monitor, permiten reproducir la imagen para que sea percibida por un humano. Naturalmente, la comprensión o interpretación de esta imagen no está codificada directamente en ella misma, sino que es el usuario que observa quien interpreta e identifica su contenido. En general, los patrones y formas que son reconocidos de forma natural y espontánea por seres humanos resultan difíciles de describir intuitivamente en una secuencia de pasos programables. La cantidad de posibles maneras en que pueden relacionarse los pixeles en una imagen son, simplemente, muchas y muy complejas como para ser declaradas manual y explícitamente.

De cierta manera, nos gustaría que el computador pudiera razonar sobre las imágenes de manera similar a como lo hace un ser humano. El área de estudio que investiga cómo los sistemas computacionales pueden obtener esta comprensión de alto nivel para imágenes y videos digitales se llama **visión de computadores** (*computer vision*). Esta comprensión nos permite automatizar tareas tradicionalmente dependientes del sistema visual humano.

2.1.2. Representación, ingeniería manual y aprendizaje de características

Para formular cualquier modelo de análisis, resulta importante contar con propiedades que describan los fenómenos que deseamos estudiar. Si queremos estudiar el estado de una partida de ajedrez, por ejemplo, podríamos registrar las coordenadas geográficas exactas de cada pieza, su forma, contorno, material y color. Sin embargo, una opción más conveniente podría ser imaginar una grilla cuadrada de 8×8 posiciones discretas, tipos de piezas, y jugador al que pertenecen. En otras palabras, la manera en que podemos representar la información varía inmensamente, y nuestra capacidad para procesar y obtener conclusiones útiles dependerá de la forma de esta representación.

En este contexto, una **representación** es un conjunto de **características** (*features*) medibles que describen un fenómeno y que utilizamos para derivar conclusiones (predicciones). Modelos de análisis tradicionales tienden a definir o seleccionar características relevantes a partir del conocimiento que poseemos en el dominio específico, así como nuestro conocimiento sobre el juego de ajedrez nos lleva a representar la información de manera más conveniente. Esta es una aproximación intuitiva, ya que, si reconocemos que una variable afecta el resultado del fenómeno que estudiamos, lo natural es incluirla en nuestro modelo. A este proceso de extracción "manual" de características —usando el conocimiento del dominio— se le conoce como **ingeniería de características**.

Sin embargo, a veces nuestro conocimiento del dominio no es suficiente como para formular una representación adecuada. Esto puede ocurrir cuando no sabemos qué características son relevantes o cómo contribuyen exactamente en el fenómeno. Pensemos en cómo distinguiríamos la imagen de un perro de la de un conejo. Quizás pensamos en características como el largo de las orejas, la forma de las patas o de la cola. Sin embargo, para un computador, una imagen digital no se constituye directamente de orejas, patas o colas, sino de pixeles, sus posiciones y colores. Saber de anatomía animal no parece ayudarnos en este caso.

En este sistema hipotético tenemos dos problemas. Por una parte, parece difícil enumerar explícitamente todas las reglas posibles —de anatomía, postura, perspectiva, iluminación, entorno, etc.— en todas sus posibles combinaciones y condiciones. Por la otra parte, aunque pudiésemos formular todas estas reglas, los datos de entrada no son de anatomía o perspectiva, sino de pixeles —bajo nivel—, y no se corresponden con las características de alto nivel que el conocimiento del dominio nos permite identificar.

En cierto sentido, podemos ver tanto la entrada como la salida de un sistema como representaciones que agrupan características de distintos niveles de *abstracción* y *especificidad*. En nuestro sistema que distingue perros de conejos, las entradas son imágenes digitales: representaciones de *bajo nivel* y de *propósito general*. Por su parte, la salida sería el animal reconocido —perro o conejo—, las que son representaciones de *alto nivel* y *específicas* para esta tarea de clasificación.

Una forma de enfrentar los problemas de representación mencionados es usando técnicas de **aprendizaje de características** (*feature learning*), también llamado aprendizaje de representaciones (*representation learning*), que nos permiten descubrir representaciones adecuadas de manera automática, a partir de contenido de bajo nivel. Una de estas técnicas,

basada en *deep learning*, son las redes neuronales.

2.1.3. Redes neuronales

Ejemplo de red neuronal simple

Para establecer conceptos e intuiciones generales sobre las redes neuronales, utilizaremos como ejemplo el tipo de red más simple: el perceptrón multicapa o MLP (*multi-layered perceptron*), también referido como red **fully connected**.

En la Figura 2.1 vemos un MLP simple de tres **capas**. X corresponde a la capa de entrada y es una representación numérica del *input* de la red. Por simplicidad, supongamos que X representa una imagen en escala de grises, en donde cada x_i es un valor de intensidad para un respectivo pixel de la imagen. Para mayor claridad visual, nuestro X solo tiene 5 pixeles, pero podemos imaginar el mismo ejemplo con vectores de entrada de mayor dimensión.

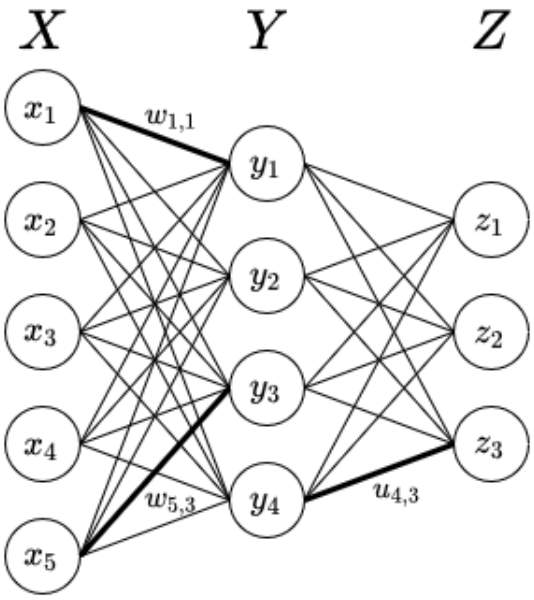

Figura 2.1: Red *fully connected* de 3 capas. X es la capa de entrada, Y es una capa escondida, Z es la capa de salida. Cada característica de Y y Z resulta de una suma ponderada de cada características de la capa anterior mediante los pesos w y u (cada línea) y la aplicación de una función no lineal *act*.

Supondremos que queremos clasificar las imágenes de entrada en 3 posibles categorías o clases: perros, conejos y gatos. Para definir la salida del modelo, diremos que cada una de esas categorías esta asociada a un elemento de Z, y que la predicción del modelo corresponde a la categoría con el valor más alto. Si z_1 es el mayor, la predicción es 'perro', si el mayor es z_2, la predicción es 'conejo', y es 'gato' si el mayor es z_3. Por ejemplo, si al procesar nuestro input de entrada nuestro Z resultante es $(z_1, z_2, z_3) = (2.4, -1.3, 0.2)$, diremos que la predicción del modelo es que la imagen corresponde a un perro. Podemos ver Z como una función que

toma X como entrada. Lo que nos gustaría es aproximar la función Z a aquella función, por ahora desconocida, que es la solución de nuestro problema.

Lo que las redes neuronales buscan hacer, a grandes rasgos, es permitir aproximarnos a cualquier función objetivo mediante el ajuste apropiado de una serie de parámetros. ¿Cómo es posible esto? Comencemos por definir y explicar Y:

$$y_j = act(\sum_i x_i w_{i,j} + b_j) \tag{2.1}$$

En la ecuación 2.1, *act* es una función no lineal, llamada **función de activación**, mientras que $w_{i,j}$ y b_j son **parámetros entrenables** que iremos ajustando durante el entrenamiento. Notemos que, matemáticamente, y_j no es mucho más que la aplicación de una función no lineal a una suma ponderada de las características x_i. Por ejemplo, $y_1 = act(x_1 w_{1,1} + x_2 w_{2,1} + ... + b_1)$.

Ilustrativamente, podemos ver cada y_j como una nueva característica que le asigna una importancia particular (pesos w) a las características previas. Si queremos obtener más características y_j, necesitaremos más pesos (parámetros entrenables) para modelar esas relaciones. Por esto, los parámetros entrenables se asocian al tamaño y **capacidad** de la red. Intuitivamente, una mayor cantidad de parámetros entrenables define una mayor cantidad de variables que podemos modificar de manera independiente para ajustar nuestro modelo. Dotar de una capacidad apropiada a nuestros modelos se considera un necesidad esencial para que estos alcancen el rendimiento esperado.

Sin embargo, solo dotar de capacidad a nuestro modelo no es suficiente. Notemos que la suma ponderada $x_1 w_{1,1} + x_2 w_{2,1} + ... + b_1$ no es más que la ecuación de una recta en varias dimensiones. No es difícil convencerse de que si nos limitamos a sumar rectas al interior de la red, la función Z sería equivalente también a una recta. Es decir, nuestro modelo solo podría ajustarse a problemas lineales, y probablemente no serviría para mucho.

Por esto, la aplicación de funciones no lineales es esencial. Ilustrativamente, la función no-lineal *act* transforma nuestra recta en una función con otra forma, y si sumamos suficientes y variadas instancias de esas formas, podemos aproximar la función objetivo. Esta no es una idea tan diferente de otras en el mundo de las matemáticas. Por ejemplo, las series de Fourier nos permiten aproximar cualquier función periódica utilizando y sumando solamente funciones sinusoidales. Conceptualmente, la no-linealidad se asocia a la **expresividad** de la red, pues nos permite ajustar la función final a formas mas variadas o "expresivas".

Luego, la relación entre Z e Y con U es la misma que entre Y y X con W. Es decir, cada característica de la capa Z se obtiene de una suma ponderada de las características de la capa anterior, y es transformada mediante la función *act*. Por su parte, añadir más **capas** a nuestra red nos ofrece más oportunidades para añadir **expresividad** y **capacidad**.

Vale notar que en nuestro trabajo, la función de activación utilizada en todos los casos es la función ReLU (*rectified linear unit*), la que corresponde simplemente a $ReLU(x) = max(0, x)$. Basta mencionar que es, probablemente, la función de activación más utilizada en tareas visuales.

Entrenamiento

Aunque existen técnicas de **inicialización** para escoger el valor inicial de los parámetros entrenables, en general, estos son escogidos mediante distribuciones aleatorias, y como es de esperar, al comienzo la red entrega resultados casi aleatorios para cualquier imagen de entrada. Con lo dicho en la sección anterior, lo único que nos falta para desmitificar las redes neuronales es explicar cómo ajustamos estos parámetros entrenables. Para esto utilizaremos la misma red *fully connected* de la sección anterior como referencia.

Supongamos que tenemos un gran conjunto de imágenes de perros, conejos y gatos en escala de grises, para los cuales ya tenemos asociaciones entre cada imagen y el resultado esperado. Por ejemplo, podríamos tener un archivo de texto simple en que cada linea corresponde a la ruta de una imagen, seguida de un índice que corresponde a una de las tres clases. Digamos: 0, 1 y 2 para perro, conejo y gato, respectivamente.

Lo primero que haremos es añadir un paso adicional al final de la red. Si bien sabemos que el valor más alto en el vector Z indica la predicción de la red, este podría tener cualquier valor, y nos gustaría tener un vector de salida algo más conveniente. Para esto utilizaremos la función ***softmax***. Basta decir que esta es ampliamente utilizada en redes de clasificación, y que transformará nuestro vector Z en un vector Z' que representa una distribución de probabilidades. Si z_1 era el mayor valor en Z, entonces z'_1 será el mayor valor de Z'. La diferencia será que los valores de Z' serán probabilidades (valores en $[0, 1]$), y que la suma de estos será igual a 1. En la Figura 2.2, ilustramos esta diferencia.

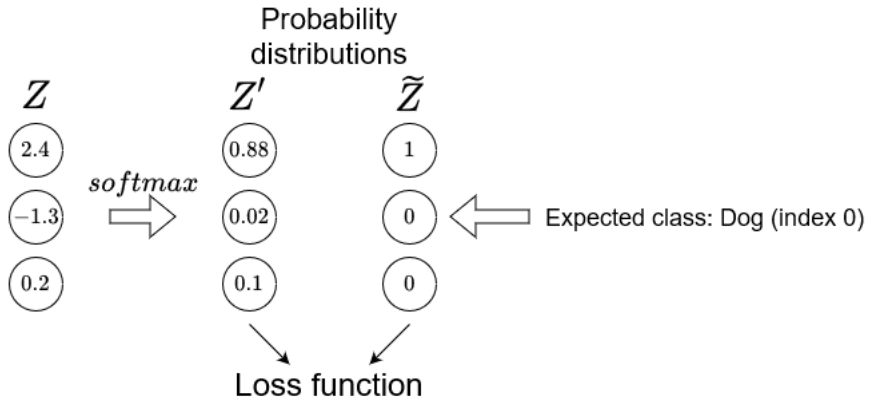

Figura 2.2: Uso de la función $softmax$ y representación de la categoría esperada como distribución de probabilidad.

Notemos que podemos transformar fácilmente los índices que mencionamos antes —para denotar el resultado esperado de cada imagen— en distribuciones de probabilidades. En efecto, si el índice es, por ejemplo, 0 (perro), entonces la distribución sería $(1, 0, 0)$ denotando que esperamos que la red nos entregue el resultado 'perro' con seguridad del 100 % y 0 % para las otras clases. Si el índice es 1 (conejo), el vector de probabilidades sería $(0, 1, 0)$, y si el índice es 2, entonces el vector sería $(0, 0, 1)$. Llamaremos $\widetilde{Z}$ a este vector que denota el resultado esperado para el input.

Luego, utilizamos una **función de pérdida** o **función de error**. En nuestro trabajo,

la función de pérdida usada para **clasificación** es **cross-entropy loss**. Nuevamente, para nuestros efectos, su caracterización matemática no es de interés y basta mencionar que en general las funciones de error nos entregan un valor **escalar** que cuantifica la diferencia entre la predicción del modelo y el resultado esperado. Este tipo de entrenamiento en que necesitamos usar y tener los valores esperados de antemano se conoce como **entrenamiento supervisado**.

Notemos que este valor de error tiene dependencia en los parámetros entrenables de la red. En efecto, si cambiamos los parámetros, cambia el error. Luego, el paso siguiente es calcular la gradiente del error en función de estos parámetros. Esto es decir, calcular las derivadas direccionales del error para cada parámetro entrenable de la red. Esto es equivalente a saber a como varía el error con la variación de cada parámetro entrenable, y como quizás sospeche el lector, eso nos ayudará a realizar pequeñas modificaciones iterativas sobre esos parámetros entrenables.

El cálculo de este gradiente se realiza mediante el uso extensivo de la regla de la cadena en un algorimo conocido como *backpropagation*. Ilustrativamente, este computa el gradiente 'propagando' el cálculo de las derivadas direccionales, desde aquellas funciones y expresiones analíticas que se encuentran más 'cerca' del error final (en las últimas capas), a aquellas que se encuentran más 'lejos' (o más cerca de la primera capa de la red).

Con la gradiente calculada, existen diversas estrategias para minimizar el error. Al conjunto de estrategias y algoritmos usados se les suele llamar **optimizador**. Entre otros elementos que suelen definir los optimizadores está la tasa de aprendizaje o **learning rate** (lr) que denota la magnitud de la modificación de los parámetros entrenables durante un paso de la optimización. Por ejemplo, una posible estrategia simple de optimización es redefinir cada parámetro entrenable mediante $w := w - lr \cdot g$ donde g es la derivada direccional para w. Así cada vez que procesemos una imagen, realizaremos una pequeña corrección sobre los parámetros entrenables, en la dirección que minimiza el error.

Por su puesto, las explicaciones anteriores constituyen solo una visión superficial de los desafíos que presentan las redes neuronales, y existen múltiples consideraciones y problemas que hacen que estas técnicas no funcionen tan bien como desearíamos. La descripción e inspección exhaustiva de estos temas se escapa al alcance de este trabajo, y solo mencionaremos algunos cuando sea apropiado.

Otros conceptos relevantes

A continuación, describiremos algunos conceptos e ideas adicionales que serán útiles para la comprensión de este trabajo, y que no hemos podido incluir sucintamente en los ejemplos anteriores:

Batch. Aunque en nuestros ejemplos hemos supuesto que procesamos una imagen a la vez para entrenar, en la práctica, esto resulta inconveniente. En su lugar, resulta mucho más eficiente procesar múltiples muestras a la vez. A un conjunto de muestras que se procesan simultáneamente se les llama **batch**. A su vez, lo más común es utilizar vectores, matrices o tensores para expresar las operaciones en una red neuronal. Por ejemplo, si utilizamos un

tamaño de batch 10, y utilizamos notación matricial, podemos expresar la ecuación 2.1 de manera más general como $Y = act(\sum_i XW + b)$, donde X es una matriz de 10×5, W una matriz de 5×4 y b una matriz de 10×4 que repite 10 veces el mismo vector de tamaño 4.

El tamaño del batch tiene efectos sobre el entrenamiento, y por lo mismo, sobre el rendimiento de nuestro modelo. Los tamaños de batch grandes suelen permitir tomar mayor provecho de la capacidad del hardware, y por lo tanto entrenar en menos tiempo. Algunos trabajos sugieren que lo tamaños de batch algo más pequeños resultan en un mejor rendimiento. Como referencia, no es raro ver tamaños de batch entre las 32 y 512 muestras en procesamiento de imágenes.

Época. Una época es una unidad de medida del entrenamiento que corresponde comúnmente a haber procesado o consumido todas las muestras del conjunto de datos de entrenamiento una vez. En algunos casos, especialmente en que generamos datos de entrenamiento de manera dinámica, resulta difícil o inconveniente definir una época de esta manera, y en su lugar se opta por utilizar otros criterios para definirla. Es común entrenar nuestros modelos por decenas o quizás cientos de épocas.

Scheduler. Este es un conjunto de estrategias y algoritmos utilizados para modificar la tasa de aprendizaje a lo largo del entrenamiento. En nuestro caso, utilizamos **cosine annealing**, el cual reduce paulatinamente la tasa de aprendizaje a lo largo del entrenamiento, siguiendo el descenso sinusoidal del coseno. Esto busca estabilizar el aprendizaje de la red, de manera similar a como supondríamos que mientras más estudiamos una materia, las correcciones que tendríamos que realizar serían menores y más finas.

Preprocesamiento del conjunto de datos. Es el conjunto de técnicas y transformaciones aplicadas a los datos con la intención de prepararlos y disponerlos para el procesamiento en un formato conveniente. Realizar recortes para que todas las imágenes sean del mismo tamaño, centrar el contenido, o normalizar los rangos de color RGB son formas de preprocesamiento común.

Data augmentation. Conjunto de técnicas que buscan incrementar artificialmente la cantidad de muestras diferentes disponibles en nuestro conjunto de datos. Por ejemplo, la rotación y zoom moderados son técnicas de *augmentation* comunes en conjuntos de datos de imágenes, y nos permiten generar más muestras a partir del conjunto original.

Pooling. Existen diferentes operaciones de *pooling*, y en general estas buscan reducir el tamaño de las representaciones sin transformar el contenido semántico subyacente. En el procesamiento de imágenes, se suele relacionar con la noción de 'reducir la resolución'.

Normalización. Al igual que en otros modelos estadísticos, la normalización de las entradas tiene un efecto sobre el aprendizaje. Por ejemplo, imaginemos que queremos explorar cómo influyen la edad y el ingreso económico en el endeudamiento, utilizando un modelo estadístico tradicional. Ya que la edad es un valor del orden de las decenas (años) y el ingreso está en el orden de los miles o cientos de miles (pesos chilenos), la variable 'ingreso' podría adquirir un peso mayor en el análisis matemático, que no se corresponde con las contribuciones cualitativas que esperaríamos descubrir.

Para corregir este problema, se suelen normalizar los valores de cada variable, para lle-

varlos a un rango comparable. Por ejemplo entre $[0, 1]$ o $[-1, 1]$. Esto ayuda a evitar efectos patológicos en el análisis matemático.

Este efecto también ocurre en las redes neuronales, pero no solo en la entrada. La transformación de características en cada capa puede introducir nuevas diferencias en los rangos de valores, y estos, a su vez, generar efectos matemáticos perjudiciales para el aprendizaje de la red. Para corregir esto, se suelen utilizar técnicas de normalización en la entrada y entre capas. En particular, se utiliza ampliamente la adición de capas de **batchnormalization** [8] entre otras capas de la red. Estas capas también incluyen parámetros entrenables, y su introducción implicó un avance importante en el entrenamiento de redes neuronales.

Caracterización de alto nivel

Podemos ver las **redes neuronales** como sistemas de varios niveles, en que cada nivel toma la representación entregada por el nivel anterior y la procesa mediante transformaciones lineales, funciones no lineales escalares (**funciones de activación**), parámetros ajustables, y lo entrega al nivel siguiente. Típicamente se entienden las representaciones subsecuentes como de mayor *nivel de abstracción*. El primer nivel sería la entrada de nuestro modelo, y el último sería la predicción o resultado.

Inicialmente, el modelo tiene mala capacidad predictiva, y es necesario entrenarlo. Esto es, ajustar los parámetros utilizados para procesar las representaciones. Para esto, típicamente se utilizan grandes cantidades de datos —para lo cuales ya conocemos el resultado esperado— que consumimos iterativamente para computar medidas del error de nuestro modelo en su estado actual. Con esta información, ajustamos los parámetros del modelo en cada iteración, minimizando el error.

Podemos decir, bajo la noción anterior, que todos los niveles de una red neuronal *extraen características* —en tanto todos los niveles arrojan una nueva representación—. Sin embargo, se suele hablar de **extracción de características** para referirse al proceso *general* de obtener representaciones de alto nivel que luego utilizamos para resolver alguna tarea específica —como clasificación o recuperación—. Lo anterior separa, cualitativamente, dos secciones de la red: aquella en que buscamos extraer una representación general y abstracta —posiblemente útil para una amplia variedad de problemas—, y aquella en que usamos esa representación para resolver nuestro problema específico.

Volviendo a nuestro modelo de perros y conejos, podríamos ejemplificar la extracción de características como sigue: supongamos que después del primer nivel obtenemos características de contornos, líneas y ángulos; más adelante, en otro nivel, características de figuras, cuerpos y texturas; y finalmente, características de orejas, patas y colas. Luego, los niveles finales de la red "razonan" sobre estas características de alto nivel, transformando la representación para formular el resultado final en un formato conveniente. Por ejemplo, un puntaje para el resultado "perro" y otro para "conejo", donde consideraríamos el más alto entre ambos como la predicción final de la red.

A la porción de la red que realiza la extracción "general", junto con las técnicas que lo caracterizan, se le suele llamar ***backbone*** o en algunas situaciones ***encoder***. Para referirse al

resultado de esta extracción de características se suelen utilizar distintos términos —dependiendo del contexto—, pero es común el uso de palabras como ***embedding***, aunque a veces se habla indistintamente de representación o características extraídas. A la segunda porción de la red a veces se le llama **cabeza** o ***decoder***. A los niveles —o **capas**— iniciales se les suele llamar ***stem***, el que suele ser algo diferente para lidiar con la entrada inicial. Existen otras partes de la red que pueden recibir caracterizaciones distintivas, pero las mencionadas son las más importantes para nuestro trabajo.

Vale mencionar que, si bien en nuestro ejemplo hemos asociado explícitamente las características de niveles *intermedios* con significados específicos —como formas, líneas, o partes de animales—, lo hemos hecho con un fin exclusivamente ilustrativo. En general, las representaciones *intermedias* de una red no son fácilmente interpretables; son conjuntos de números sin un aparente significado al leerlos. La interpretación de estos es en sí mismo objeto de estudio.

2.2. Caracterización del problema SBIR

2.2.1. Recuperación de imágenes

Podemos concebir la **recuperación de imágenes** dentro de la categoría más general de recuperación de información (*information retrieval*). En esta, el sistema de recuperación se describe mediante una consulta (*query*) que declara algún criterio o referencia para la información que se desea recuperar, y compara esta contra un conjunto de objetos según algún criterio de similitud, ranqueándolos. De estos objetos rankeados, se retorna una cantidad finita, típicamente ordenados según este criterio. Luego, dependiendo del dominio de la consulta y el criterio de similitud, se usan diferentes términos para referirse a variaciones específicas del problema. Aquí nos basamos en las categorías propuestas en [9]: en base a texto, contenido y semántica.

La **recuperación de imágenes en base a texto** (TBIR, por sus siglas en inglés *text based image retrieval*) se refiere al caso en que la consulta es una palabra o descripción escrita, y las imágenes a recuperar han sido anotadas con etiquetas u otros metadatos (por ejemplo, categoría, nombre de archivo, tamaño, etc.). Luego, se define el ranking de los resultados, típicamente usando técnicas basadas en palabras clave como *bag of words* [10]. La anotación manual de estos metadatos resulta impráctica para grandes conjuntos de imágenes, no solo por ser manual, sino también por la dependencia en el lenguaje, como el idioma específico y la variabilidad de formas para expresar contenido similar. En la Figura 2.3 presentamos un sistema típico para TBIR.

Las dificultades para la anotación manual motivan el uso de imágenes como elemento para formular la consulta. Es decir, podemos utilizar una imagen para especificar lo que esperamos recuperar. En este caso, nos referimos al problema como **recuperación de imágenes en base a contenido** (CBIR, por sus siglas en inglés *content based image retrieval*). En CBIR, las imágenes a recuperar no necesitan ser anotadas con información adicional. Es decir, no se usan metadatos manualmente anotados (texto), sino que se usa extracción de características

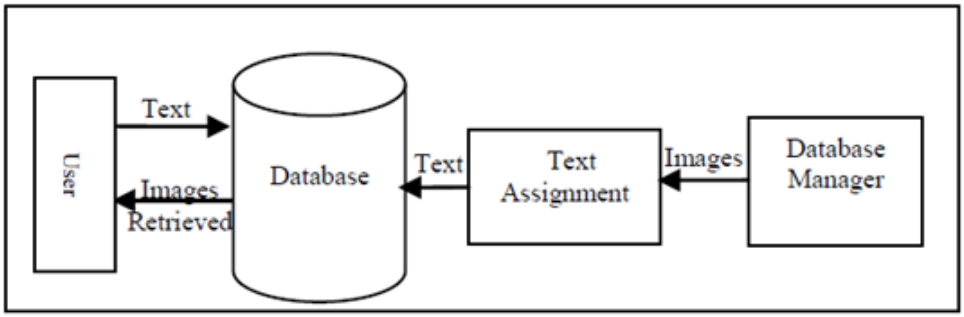

Figura 2.3: Sistema típico para TBIR. Fuente: [9].

para obtener una representación adecuada, y son estas características las que son utilizadas en la recuperación final. En la Figura 2.4, se describe el funcionamiento de un sistema típico para CBIR.

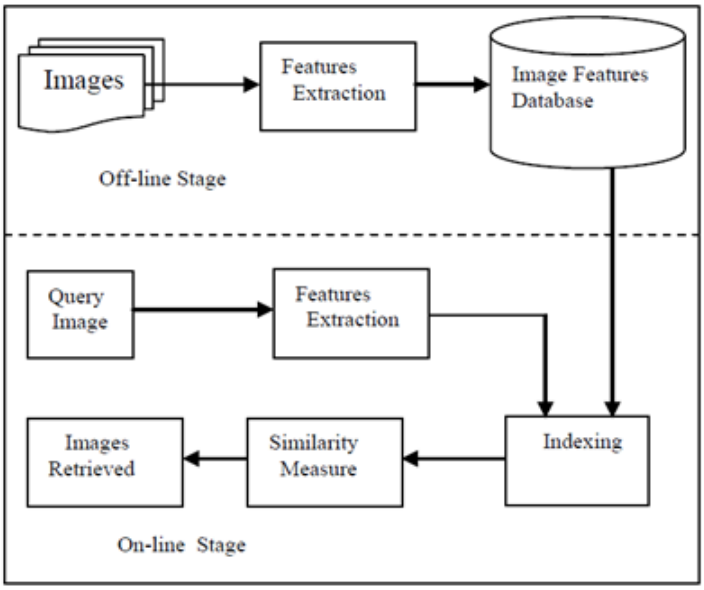

Figura 2.4: Sistema típico para CBIR. Notemos que, dependiendo de la implementación, la extracción de características del conjunto de imágenes a recuperar puede ser realizada de antemano (*off-line*). Fuente: [9].

Otras aproximaciones también utilizan texto como consulta, pero en lugar de anotar metadatos y etiquetas para las imágenes, extraen características semánticas del texto de consulta para comparar contra las características extraídas de las imágenes. A esta aproximación se le suele llamar **CBIR semántico** (*semantic* CBIR, o SBIR, pero no usaremos esta sigla debido a la ambigüedad con SBIR-*sketch based image retrieval*). La Figura 2.5 muestra un sistema típico para CBIR semántico.

2.2.2. Recuperación en base a dibujos (SBIR)

Podemos considerar SBIR como una forma específica de CBIR en que la imagen de entrada (consulta) es un dibujo. Al igual que en CBIR, el proceso implica extraer características de

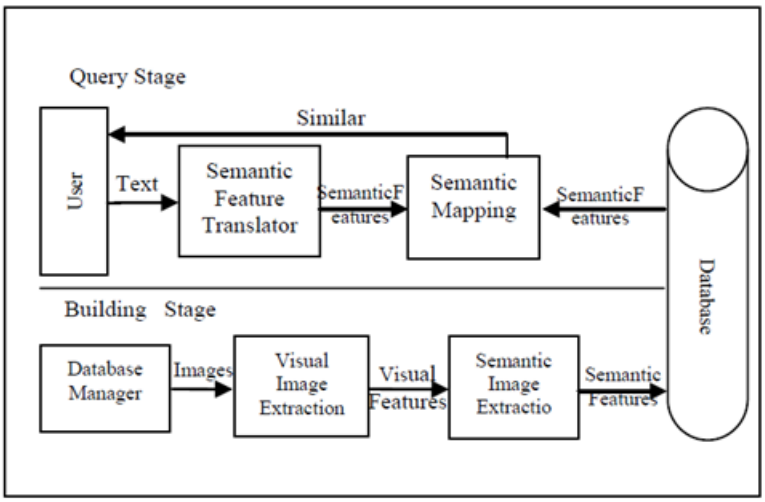

Figura 2.5: Sistema típico para CBIR semántico. Fuente: [9].

la imagen de entrada y usar estas para eventualmente comparar con características extraídas de las posibles imágenes a recuperar.

En ambos casos, la adecuada extracción de características es parte integral del problema, sin embargo, al usar dibujos como consulta se añade el problema de la distancia entre los dominios foto/dibujo. Los dibujos son representaciones abstractas y con frecuencia simplificadas de elementos reales, y como tal, muchas de las características extraídas de la imagen realista pueden no ser directamente comparables con aquellas del dibujo. Basta pensar: ¿En qué se parece el clásico dibujo de un corazón con un corazón real? Adicionalmente, la textura de las superficies, la iluminación y el color son con frecuencia omitidos en los dibujos. Por esto, parte de la investigación en SBIR tiende a buscar soluciones que lleven las características a un espacio de representación común antes de determinar la similitud.

2.2.3. Caracterización y variantes de SBIR

Además de los problemas de representación y extracción de características, la caracterización precisa del problema SBIR también depende de las necesidades del usuario, su comportamiento durante la búsqueda, y las expectativas que se tengan del sistema.

Por ejemplo, si se dibuja un corazón (♡), ¿qué suponemos que desea recuperar el usuario? Quizás busca resultados que incluyan ese patrón visual, o quizás se buscan elementos relacionados con el amor y el romance —en los cuales podría no aparecer explícitamente un corazón—. Esta intención no está codificada en la consulta misma, pero puede tener repercusiones relevantes en el criterio de éxito del sistema.

Aplicaciones comunes de SBIR en búsqueda de productos suelen asumir que los dibujos son representaciones de elementos "físicos" que el usuario busca en el comercio, por lo que se suelen favorecer las propiedades "físicas" durante la búsqueda. En las aproximaciones basadas en aprendizaje supervisado, en particular, el etiquetado de los datos de entrenamiento es de suma importancia para establecer las formas de representación que se desea favorecer.

Dos aspectos relevantes que motivan la investigación de variantes al problema son la **granularidad** y capacidad de **generalización**.

Granularidad

Dependiendo de lo específico que se espera que sean las consultas —y por extensión, los resultados—, se suele clasificar el problema como *coarse-grained* SBIR (CG-SBIR, a veces llamado *category-level* SBIR) y *fine-grained* SBIR (FG-SBIR). FG-SBIR se enfoca en la capacidad de diferenciar entre clases similares, así como pueden ser diferentes especies de un tipo de animal o diferentes modelos de zapatos. CG-SBIR puede verse como la recuperación en base a características más amplias y generales. FG-SBIR intenta ser consistente no solo con la categoría general de la imagen recuperada, sino también con la instancia específica dentro de dicha categoría, y por lo mismo se entiende como un problema que demanda representaciones más específicas.

Generalización

En general, las técnicas de entrenamiento y aprendizaje en *computer vision* esperan que el modelo obtenga una comprensión de alto nivel del problema. Informalmente, esperamos que el modelo pueda "entender" la información visual de manera similar a un humano. En nuestro caso, y dicho de forma simplificada, los datos de entrenamiento establecen correspondencia entre ciertas entradas y salidas del modelo, a partir de las cuales se espera que el modelo abstraiga el conocimiento de alto nivel necesario para lidiar correctamente con el problema. Sin embargo, en rigor, lo que el entrenamiento del modelo optimiza son esas correspondencias, y no el "entendimiento de alto nivel". Por lo tanto, puede ocurrir que el modelo termine aprendiendo "de memoria" dicha correspondencia en los datos de entrenamiento, sin lograr la representación abstracta que esperamos. Esto es similar a la idea de hacer trampa en un examen de alternativas que está disponible de antemano: podríamos estudiar, o también podríamos aprender de memoria las alternativas correctas; la calificación que obtendremos no distingue entre estos casos. Sin embargo, si no estudiamos, difícilmente podremos responder a preguntas que no hayamos visto antes. Esto se conoce como ***overfitting***.

A menos que nuestros datos de entrenamiento abarquen razonablemente todos los casos posibles que puede enfrentar nuestro modelo, el *overfitting* es un problema que debemos tener en cuenta. Para el caso de SBIR, con frecuencia resulta infactible generar datos de todos los posibles casos, o resulta costoso añadirlos y re-entrenar el modelo cada vez que queramos añadir objetos a nuestra aplicación. Esto motiva el interés en desarrollar modelos capaces de generalizar a clases no vistas durante el entrenamiento.

La capacidad para generalizar a nuevas instancias de las clases ya vistas se considera un requisito elemental para cualquier modelo. El problema de interés particular, entonces, es la generalización a nuevas clases no vistas durante el entrenamiento, normalmente referido como ***zero-shot*** SBIR (ZS-SBIR), también referido como *cross-category* SBIR (CC-SBIR). Los métodos para abordar zero-shot generalmente funcionan asociando las clases observadas y no observadas por medio de alguna forma de información auxiliar, o por medio de características

generales de la representación.

Intuitivamente, un sistema de recuperación enfocado en esta modalidad debería poder recuperar nuevos elementos añadidos al catálogo, incluso si no son de la misma categoría que los ya existentes, así como suponemos que un humano podría reconocer la similitud entre un dibujo razonable y un objeto real al compararlos, incluso si nunca ha visto el objeto antes.

2.3. Arquitecturas backbone en procesamiento de imágenes

2.3.1. Redes convolucionales

En las redes ***fully connected*** (fc) tradicionales, visualizamos cada representacion como un listado de valores —vector de características—. Sin embargo, la naturaleza bidimensional en imágenes motiva organizar las representaciones mediante mapas bidimensionales —**mapas de características**—, que describen el valor de alguna propiedad para cada posición espacial de la imagen. Además, podemos apilar múltiples mapas de características para describir múltiples propiedades diferentes. En este contexto, cada mapa corresponde a un *canal*. Ilustrativamente, podemos identificar las dimensiones a lo "alto" y "ancho" de nuestra representación como posiciones espaciales asociadas a la imagen, mientras que la dimensión a lo "largo" indica las distintas propiedades extraídas. El término *vector de características*, en este contexto, denota el vector a lo "largo" de una posición particular (ver Figura 2.6). Cabe notar que es frecuente omitir algunas dimensiones (como los canales) en otras figuras ilustrativas, para simplificar, si bien no es el caso de la figura 2.6.

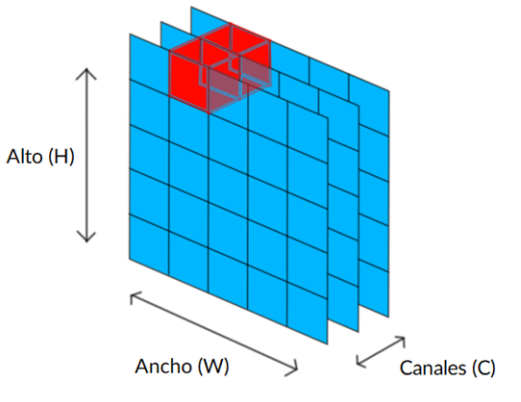

Figura 2.6: Representación típica en procesamiento de imágenes. Mapas de características de dimensiones H×W son apilados para formar un **tensor** de dimensiones H×W×C. En rojo: un vector de características a lo largo de los canales.

En una red *fc* tradicional, en cada representación consecutiva cada valor es obtenido relacionando todos los valores de la representación anterior. Es decir, cada valor resulta de agregar características a nivel *global*. A diferencia de estas redes *fc*, la convolución utiliza **filtros** que relacionan vecindades *locales* de la imagen. Es decir, cada vector de características

se obtiene a partir de relacionar características cercanas a esa posición particular en la representación precedente. Un filtro dado "pasa" por toda la imagen usando el mismo conjunto de pesos entrenables, actuando como un "detector" para una característica particular, sin importar la posición en que esta ocurra en el plano de entrada. Estas vecindades locales y pesos compartidos reducen el costo en memoria y procesamiento requerido.

Podemos definir la convolución discreta en nuestro plano bidimensional como:

$$(F * k)(p) = \sum_t F(p + t)\, k(t) \tag{2.2}$$

En donde las funciones discretas F y k representan un mapa de características y un filtro, respectivamente, y t y p son posiciones discretas en el plano (pares ordenados). Dependiendo de cómo deseemos manejar las condiciones de borde, podemos definir las coordenadas del filtro de diferentes maneras. En la Figura 2.7, mostramos un ejemplo de la convolución discreta considerando dimensiones entrantes de 5×5, un filtro de 3×3, y una salida de 3×3.

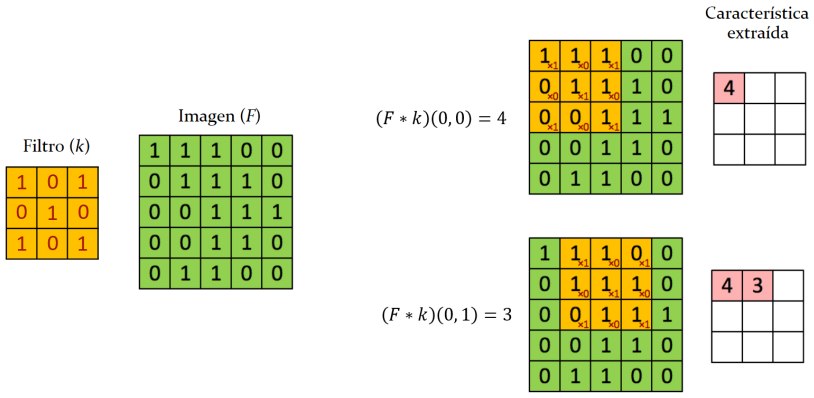

Figura 2.7: Ejemplo de cálculo de la convolución discreta para dos elementos, usando una entrada de tamaño 5×5 y filtro de tamaño 3×3. Definimos las coordenadas en ambas desde $(0, 0)$ a partir de la esquina superior izquierda, lo que resulta en un mapa de características de salida de 3×3. Fuente: adaptado de [11]

Esta técnica fue inicialmente aplicada para el reconocimiento de números en [12], y en 2012 fue popularizada tras su aplicación en el conjunto de datos ImageNet, superando ampliamente el estado del arte en ese momento [13]. Tras esto, una serie de arquitecturas convolucionales cada vez más potentes se han propuesto, como VGG [14], GoogLeNet [15], ResNet [16] y DenseNet [17], que han servido como la base de múltiples tareas en *computer vision* [7].

A pesar de estos avances, la convolución presenta algunas desventajas. Ya que las convoluciones procesan una vecindad local, capturar relaciones entre puntos distantes requiere aplicar múltiples capas convolucionales consecutivas [4]. Esto para que las características de orden superior incluyan indirectamente características previamente obtenidas de vecindades aledañas. La convolución carece de invarianza a la rotación —para el modelo, una imagen rotada es diferente—. La cantidad de parámetros necesarios escalan con el tamaño del filtro;

y los pesos aprendidos por los filtros son fijos: la agregación de información no puede adaptarse al contenido —es decir, un filtro dado procesa todas las porciones de la imagen de la misma manera—.

ResNet

Siguiendo lo expuesto lo expuesto en su publicación original [16], ResNet está motivada por las dificultades en el entrenamiento de redes cada vez más profundas (mayor cantidad de capas). Parte de estos problemas se atribuyen al desvanecimiento/explosión de gradiente. Este problema se relaciona con rangos de valores excesivamente bajos/altos para las gradientes calculadas durante el entrenamiento, y la magnificación de este efecto por uso de la regla de la cadena en la aplicación de ***backpropagation***. Esto puede ser abordado con técnicas de normalización en la entrada y en capas intermedias. Sin embargo, aun con estas técnicas, al estudiar el efecto de capas adicionales, notamos que el modelo se beneficia solo hasta cierto punto, en el cual, contra intuitivamente, añadir capas adicionales comienza a aumentar el error de entrenamiento y testing —efecto referido como ***problema de degradación***—.

ResNet aborda este problema introduciendo "atajos" o **conexiones residuales** que se saltan algunas capas. Esto facilita la propagación hacia las primeras capas y permite mayor flexibilidad en el proceso de aprendizaje. Las conexiones residuales son ampliamente utilizadas en arquitecturas posteriores [17], [2], [18], [4], [7], y ResNet continúa siendo utilizada en diferentes tareas visuales [19].

Estas conexiones residuales se nos presentan en [16] en dos tipos de bloques mostrados en la Figura 2.8, uno simple con dos convoluciones de 3×3, y una variante "cuello de botella" (*bottleneck*), que utiliza una convolución de 1×1 para reducir las dimensiones que procesará la siguiente convolución —de 3×3— y una convolución de 1×1 final para aumentar/restaurar las dimensiones de la entrada. Este bloque cumple una función similar al anterior, pero reduce el procesamiento necesario.

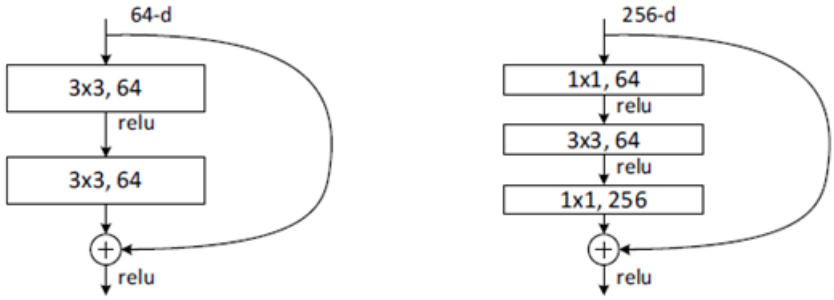

Figura 2.8: Bloques residuales en ResNet. A la izquierda: bloque simple. A la derecha: bloque *"bottleneck"* que utiliza convoluciones de 1×1 para reducción de canales. El bloque bottleneck puede manejar representaciones de mayor dimensionalidad, a un costo computacional comparable al bloque simple. Fuente: [16]

Tal y como es denotado en la Figura 2.8, la conexión residual se traduce formalmente en una suma entre el resultado de las convoluciones y la entrada del bloque (función identidad).

Esta suma requiere que las dimensiones de entrada y salida del bloque sean iguales. Sin embargo, es frecuente cambiar las dimensiones de la representación a lo largo de la red. Para permitir estos cambios de dimensión, podemos aplicar proyecciones lineales con pesos ajustables en la conexión (implementadas mediante convolución de 1×1, con *stride* variable para reducir dimensiones espaciales). En la Figura 2.9, mostramos variaciones del bloque residual simple que mantienen y luego varían las dimensiones de entrada.

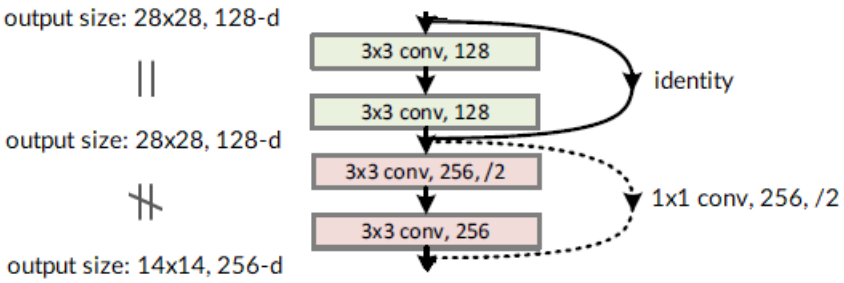

Figura 2.9: Manejo de dimensiones en bloque residual. El primer bloque aplica la identidad como conexión, el segundo aplica una proyección lineal. "-d" denota la cantidad de canales, "/2" denota *stride* 2. Fuente: adaptado de [16]

En la Tabla 2.1 podemos visualizar el diseño clásico de ResNet34 y ResNet50, las que utilizan la variante simple y *bottleneck*, respectivamente. La entrada de estas redes tiene dimensiones espaciales de 224×224. Previo a la aplicación de bloques residuales, reducimos las dimensiones de entrada mediante una convolución de 7×7 y *max pooling*. Posterior a los bloques residuales, se efectúa la clasificación mediante *average pooling*, una capa *fully connected*, y finalmente *softmax*. Si bien es posible usar proyecciones lineales en todas las conexiones residuales, se favorece el uso de la identidad, reservando la proyección para cuando es necesario corresponder dimensiones (Figura 2.9). Adicionalmente, se utiliza *batch normalization* [8] después de cada convolución y antes de la activación.

2.3.2. Atención

Intuitivamente, los mecanismos atencionales en redes neuronales imitan la atención cognitiva en los seres humanos. Dependiendo de la tarea que queramos realizar, no toda la información que percibimos parece igual de importante. Si queremos distinguir animales, por ejemplo, probablemente nos fijaremos más en ciertas características anatómicas, proporciones y texturas, que en el lugar físico en el que está el animal, el paisaje, u otros elementos que consideraríamos menos relevantes. Es decir, a pesar de que, como humanos, poseemos la capacidad para reconocer una amplia gama de patrones visuales en toda la imagen, no todas las partes de la imagen, ni todos esos patrones, son igual de relevantes ni son percibidos de igual manera siempre: las regiones de la imagen que contienen información importante pueden cambiar con la ubicación de los animales, la perspectiva no siempre nos permitirá ver todas las partes del animal, y las condiciones de iluminación y la claridad visual pueden cambiar significativamente la imagen. La atención nos ayuda a adaptarnos al contenido de cada imagen para extraer patrones de manera más eficiente.

Las formulaciones o modalidades en que los mecanismos atencionales pueden ser apli-

Output size	34-layer	50-layer
112×112	7×7, 64, stride 2	
56×56	3×3 max pool, stride 2	
56×56	$\begin{bmatrix} 3 \times 3, & 64 \\ 3 \times 3, & 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, & 64 \\ 3 \times 3, & 64 \\ 1 \times 1, & 256 \end{bmatrix} \times 3$
28×28	$\begin{bmatrix} 3 \times 3, & 128 \\ 3 \times 3, & 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, & 128 \\ 3 \times 3, & 128 \\ 1 \times 1, & 512 \end{bmatrix} \times 4$
14×14	$\begin{bmatrix} 3 \times 3, & 256 \\ 3 \times 3, & 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, & 256 \\ 3 \times 3, & 256 \\ 1 \times 1, & 1024 \end{bmatrix} \times 6$
7×7	$\begin{bmatrix} 3 \times 3, & 512 \\ 3 \times 3, & 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 3 \times 3, & 512 \\ 3 \times 3, & 512 \\ 1 \times 1, & 2048 \end{bmatrix} \times 3$
1×1	Global average pool $\rightarrow$ 1000-d fc $\rightarrow$ softmax	

Tabla 2.1: Arquitecturas clásicas para ResNet34 y ResNet50. Los bloques residuales son denotados con corchetes (ver Figura 2.8), indicando la cantidad de bloques apilados. Las conexiones residuales utilizan la identidad, excepto cuando se requiere corresponder dimensiones, en cuyo caso se utiliza la proyección lineal (siguiendo el manejo mostrado en la Figura 2.9). Se utiliza *batch normalization* después de cada convolución y antes de la activación. Fuente: adaptado de [16]

cados varían según el tipo de problema, pero en general la atención se expresa mediante la agregación de un conjunto de características ponderadas, donde los ponderadores son los **pesos** o **máscara** atencional que, ilustrativamente, usamos para "poner" atención sobre el conjunto de características de interés. Intuitivamente, estos pesos buscan capturar relaciones de correspondencia entre representaciones que, consideramos, guardan alguna relación pertinente. La "autoatención" simplemente se refiere al caso en que buscamos capturar relaciones internas en una misma representación.

La manera de relacionar estas representaciones para generar **pesos atencionales** es lo que caracteriza al mecanismo atencional. Buscamos generar estos pesos adaptativamente para cada entrada, y para lograr esta riqueza representacional, nos interesa capturar relaciones y correspondencias variadas en el espacio.

En la Figura 2.10, mostramos un esquema atencional típico, en correspondencia aproximada con la arquitectura Transformer [18]. En este, hay dos representaciones de las cuales

se computan los pesos atencionales, denotadas ***query*** (Q) y ***key*** (K). La función que relaciona estas representaciones para generar los pesos atencionales se denota como ***score***. La representación objetivo sobre la cual se aplican estos pesos se denota como ***value*** (V). Notar que θ, ϕ y g son proyecciones lineales con parámetros entrenables. Ya que tanto Q, K y V provienen de la misma entrada X, este corresponde a un esquema *autoatencional*. Una versión no autoatencional podría lograrse computando Q a partir de una entrada diferente a X. Aunque los esquemas pueden variar significativamente, el uso general de las nociones *query*, *key*, *value* y *score* son comunes en la literatura atencional.

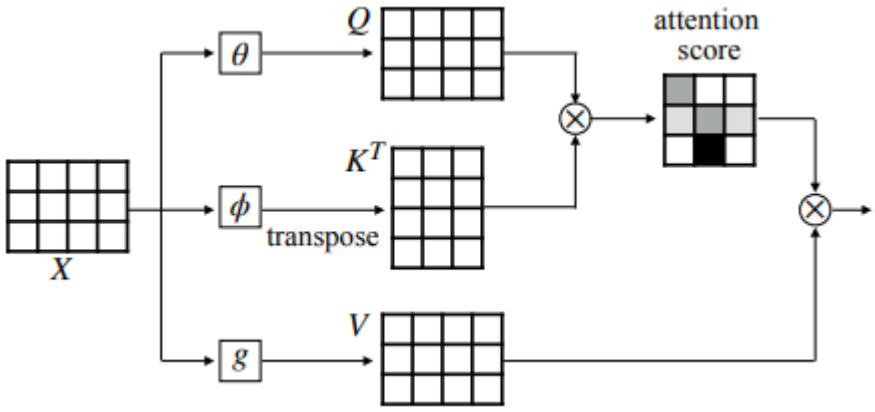

Figura 2.10: Esquema autoatencional típico usado en Transformer. Fuente: [20]

Surgimiento de la atención y su caracterización en tareas visuales

Inicialmente, los mecanismos atencionales surgen para lidiar con las limitaciones de arquitecturas neuronales recurrentes (RNN), como *seq2seq*, en la traducción de cadenas largas de texto [21], [22], en que el modelo debe comprimir toda la información necesaria de la cadena de entrada en un vector de características de largo fijo. Las primeras proposiciones atencionales inicialmente complementan las arquitecturas neuronales recurrentes [23], [24], [5], y luego las reemplazan completamente en arquitecturas como Transformer [18], la cual es ampliamente usada en modelos como GPT [25] y BERT [26].

Por su parte, podemos clasificar las aproximaciones en tareas visuales según una variedad de criterios de interés que, a grandes rasgos, podemos relacionar con el **patrón de agregación** y la modalidad de aplicación de los pesos atencionales sobre la representación objetivo. Los patrones de agregación usados pueden enfocarse en interdependencias entre canales (*channelwise attention*, *channel adaptive*) —como squeeze-and-excitation [2]—, y entre posiciones espaciales de la imagen (*content adaptive*). Pueden limitarse a vecindades locales a cada característica —como SANet [7] y SASA [6]—, o bien utilizar todo el vector/mapa (Non-Local net [4]). Cada peso atencional puede aplicarse escalarmente (un solo escalar pondera un vector/mapa de características) o puede operarse vectorialmente (cada elemento dentro del vector/mapa de características tiene su propio ponderador). En la Tabla 2.2 presentamos una variedad de modelos clasificados según estos criterios.

	Híbrido conv+atención	Atencional puro	Channel adaptive	Content adaptive	Atención global	Atención local	Atención escalar	Atención vectorial
Squeeze-and-excitation	✓		✓		✓		✓	
Non-local net	✓			✓	✓		✓	
SASA		✓	✓	✓		✓		✓
SANet		✓	✓	✓		✓		✓
ViT [1]		✓		✓	✓		✓	
BotNet [3]	✓			✓	✓		✓	

Tabla 2.2: Clasificación de varias arquitectura atencionales utilizadas en reconocimiento de imágenes.

2.3.3. SANet

SANet explora una variedad de formulaciones atencionales para construir redes atencionales puras e implementables a escala. La arquitectura sigue la estructura general de ResNet, reemplazando los bloques convolucionales usuales por operaciones autoatencionales. Para evitar el costo de computar atención sobre mapas grandes, SANet sigue la tendencia de arquitecturas como SASA, de restringir el alcance de la atención a regiones locales, en lugar de la atención global de trabajos como Non-Local net. SANet además busca formulaciones atencionales más generales, **desacoplando la agregación de características de la transformación de estas**.

SANet nos presenta dos formulaciones atencionales: por pares (***pairwise self-attention***) y por parches (***patchwise self-attention***).

Pairwise self-attention

Siguiendo lo expuesto en [7], la formulación por pares tiene la siguiente forma:

$$\mathrm{y}_i = \sum_{j \in \mathcal{R}(i)} \alpha(\mathrm{x}_i, \mathrm{x}_j) \odot \beta(\mathrm{x}_j) \tag{2.3}$$

donde $\odot$ es el producto componente a componente, i es el índice que denota la posición en el mapa de características, y $\mathcal{R}(i)$ es el conjunto de índices que componen la región local de la agregación. En este contexto, x_i e y_i son los vectores formados a lo largo de los canales, en la posición i del mapa de características.

La función β es un mapeo lineal que produce los vectores $\beta(\mathrm{x}_j)$ que son agregados por los pesos atencionales $\alpha(\mathrm{x}_i, \mathrm{x}_j)$. La función α se descompone a su vez en:

$$\alpha(\mathrm{x}_i, \mathrm{x}_j) = \gamma(\delta(\mathrm{x}_i, \mathrm{x}_j)) \tag{2.4}$$

donde δ arroja un vector que representa las características x_i y x_j. En [7] se exploran varias formas para δ. En este trabajo, consideramos la variante $\delta(x_i, x_j) = \varphi(x_i) - \psi(x_j)$, donde φ

y ψ son mapeos lineales. La función γ realiza un mapeo lineal, seguido de ReLU, seguido de otro mapeo lineal.

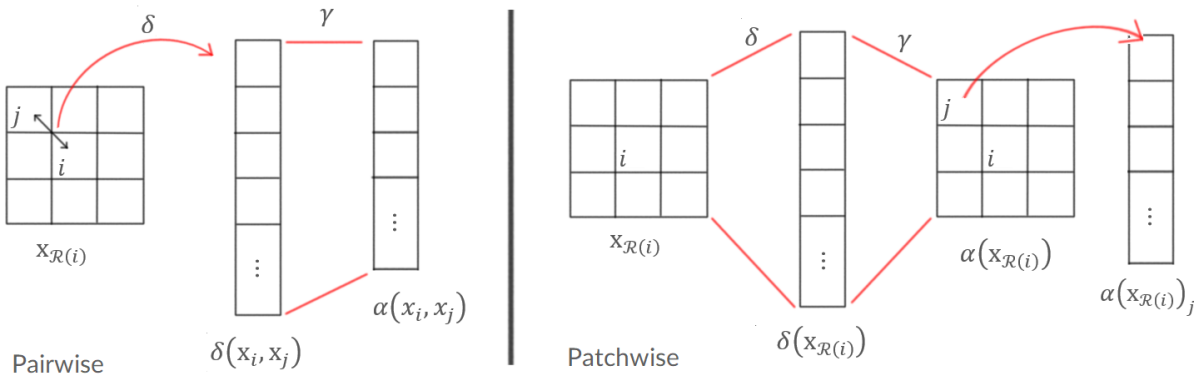

Figura 2.11: Izquierda: en la autoatención por pares, el vector $\alpha(x_i, x_j)$ solo incorpora información de i y j. Derecha: en la autoatención por parches, cada vector de $\alpha\left(x_{\mathcal{R}(i)}\right)$ puede incorporar información de todo el parche $x_{\mathcal{R}(i)}$.

Notemos que la función α relaciona cada posición j solo con respecto a la posición central i de la región local (ver Figura 2.11, izquierda). Como consecuencia, al computar y_i, la ubicación de los vectores x_j dentro del parche $\mathcal{R}(i)$ no cambia el resultado. Para proveer contexto espacial, esta variante *pairwise* computa y añade información posicional (***position encoding***) al resultado de la función δ, antes del mapeo γ.

Parchwise self-attention

En la formulación por parches, cambia la forma de α:

$$y_i = \sum_{j \in \mathcal{R}(i)} \alpha(x_{\mathcal{R}(i)})_j \odot \beta(x_j) \tag{2.5}$$

donde $x_{\mathcal{R}(i)}$ es el parche de vectores de características de la región $\mathcal{R}(i)$. $\alpha(x_{\mathcal{R}(i)})$ es un tensor de las mismas dimensiones que $x_{\mathcal{R}(i)}$, y $\alpha(x_{\mathcal{R}(i)})_j$ es el vector en la posición j en este tensor, que se corresponde espacialmente con el vector x_j en $x_{\mathcal{R}(i)}$.

A diferencia de la versión por pares, esta versión permite a cada vector α_j incorporar información de todas las posiciones dentro de la región $\mathcal{R}(i)$ (ver Figura 2.11, derecha), lo que permite capturar información posicional. Es decir, cada permutación espacial de los vectores x_j genera, potencialmente, una respuesta y_i diferente, haciendo innecesaria la adición manual de información posicional.

De forma similar a la versión por pares, α se descompone para separar transformación y agregación:

$$\alpha(x_{\mathcal{R}(i)}) = \gamma(\delta(x_{\mathcal{R}(i)})) \tag{2.6}$$

donde la función δ combina los vectores de características x_j del parche $x_{\mathcal{R}(i)}$ en un vector, el que es mapeado a un tensor de dimensiones apropiadas por γ. De las formas exploradas en [7]

para la función δ, aquí consideramos la concatenación: $\delta\left(x_{\mathcal{R}(i)}\right) = \left[\varphi\left(x_i\right), \quad [\psi\left(x_j\right)]_{\forall j \in \mathcal{R}(i)}\right]$, donde los corchetes denotan conjuntos de vectores concatenados en uno solo.

Bloque autoatencional y arquitectura SAN10

En la Figura 2.12 se ilustra el bloque autoatencional residual construido a partir de las de las variaciones *pairwise* y *patchwise*. El tensor de entrada es transformado linealmente mediante los mapeos entrenables φ, ψ y β. De estos, φ y ψ reducen los canales por un factor r_1, y β por un factor r_2. La función α es computada a partir del resultado de φ y ψ, primero aplicando la relación δ, y luego la transformación γ. Los resultados de α y β son entonces agregados según 2.3 y 2.5. Las características resultantes son procesadas con normalización y no-linealidad (ReLU) y luego mediante una última transformación lineal que reestablece la dimensionalidad de los canales.

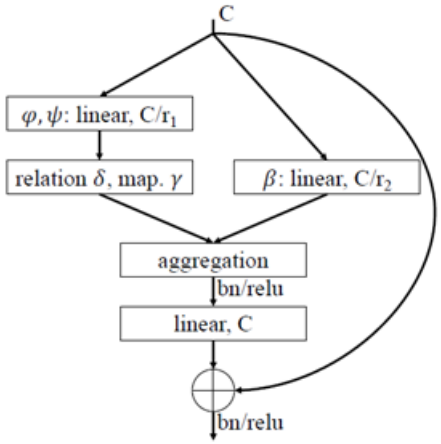

Figura 2.12: Bloque autoatencional SANet. C es la dimensión de los canales. El camino izquierdo computa los pesos atencionales α y el camino derecho la transformación lineal β. Ambos caminos reducen dimensionalidad por motivos de eficiencia. Ambos caminos son agregados mediante producto componente a componente y luego las dimensiones son aumentadas de vuelta a C. Fuente: [7]

La arquitectura SANet sigue una estructura similar a ResNet (ver Tabla 2.1) en cuanto a la reducción de dimensiones espaciales y expansión de canales en cada etapa, pero la aplicación de los bloques autoatencionales aparece antes que en sus equivalentes residuales de ResNet, dando lugar a 5 etapas de bloques, en lugar de 4 en ResNet. Adicionalmente, SANet utiliza etapas de transición para reducir la dimensionalidad espacial y expandir los canales, a diferencia de ResNet (ver Figura 2.9). Esta transición consiste en una capa de normalización, ReLU, 2×2 *max-pooling* con *stride* 2, y un mapeo lineal que aumenta los canales. De las variaciones presentadas en [7], en la Tabla 2.3 comparamos SAN10 (llamado así por sus 10 bloques autoatencionales) con ResNet26, con quien se corresponde aproximadamente en cantidad de parámetros y complejidad.

Stage	Output size	ResNet26	SAN10
	$224 \times 224 \times 64$		64-d linear
	$112 \times 112 \times 64$	7×7, 64, stride 2	2×2 max pool, stride 2 $\rightarrow$ 64-d linear
	$112 \times 112 \times 64$		$\begin{bmatrix} 3 \times 3, \ 16 \text{ sa} \\ 64\text{-d linear} \end{bmatrix} \times 2$
	$56 \times 56 \times 64$	3×3 max pool, stride 2	
Transition	$56 \times 56 \times 256$		2×2 max pool, stride 2 $\rightarrow$ 256-d linear
Block	$56 \times 56 \times 256$	$\begin{bmatrix} 64\text{-d linear} \\ 3 \times 3, \ 64 \\ 256\text{-d linear} \end{bmatrix} \times 1$	$\begin{bmatrix} 7 \times 7, \ 64 \text{ sa} \\ 256\text{-d linear} \end{bmatrix} \times 1$
Transition	$28 \times 28 \times 512$		2×2 max pool, stride 2 $\rightarrow$ 512-d linear
Block	$28 \times 28 \times 512$	$\begin{bmatrix} 128\text{-d linear} \\ 3 \times 3, \ 128 \\ 512\text{-d linear} \end{bmatrix} \times 2$	$\begin{bmatrix} 7 \times 7, \ 128 \text{ sa} \\ 512\text{-d linear} \end{bmatrix} \times 2$
Transition	$14 \times 14 \times 1024$		2×2 max pool, stride 2 $\rightarrow$ 1024-d linear
Block	$14 \times 14 \times 1024$	$\begin{bmatrix} 256\text{-d linear} \\ 3 \times 3, \ 256 \\ 1024\text{-d linear} \end{bmatrix} \times 4$	$\begin{bmatrix} 7 \times 7, \ 256 \text{ sa} \\ 1024\text{-d linear} \end{bmatrix} \times 4$
Transition	$7 \times 7 \times 2048$		2×2 max pool, stride 2 $\rightarrow$ 2048-d linear
Block	$7 \times 7 \times 2048$	$\begin{bmatrix} 512\text{-d linear} \\ 3 \times 3, \ 512 \\ 2048\text{-d linear} \end{bmatrix} \times 1$	$\begin{bmatrix} 7 \times 7, \ 512 \text{ sa} \\ 2048\text{-d linear} \end{bmatrix} \times 1$
Classification	$1 \times 1 \times 1000$	Global average pool $\rightarrow$ 1000 fc linear $\rightarrow$ softmax	

Tabla 2.3: Arquitecturas para ResNet26 y SAN10. ResNet26 se construye a partir de ResNet50, removiendo 2 bloques residuales de cada etapa (ver Tabla 2.1). 'C-d linear' denota una capa lineal con C canales de salida, implementada como una convolución de 1×1. 'C-d sa' denota una operación autoatencional con C canales de salida. El tipo de operación atencional es la que define a SAN10 *pairwise* o *patchwise*.

2.3.4. Non-local net

Non-local net [4] nos presenta bloques autoatencionales globales que complementan la operación convolucional. Se define la siguiente operacion no-local genérica:

$$\mathrm{y}_i = \frac{1}{\mathcal{C}\left(\mathrm{x}\right)} \sum_{\forall j} f(\mathrm{x}_i, \mathrm{x}_j)\, g(\mathrm{x}_j) \tag{2.7}$$

donde f es una función escalar que relaciona los vectores x_i y x_j y tiene un rol aproximadamente equivalente al de α en la formulación atencional de SANet. De las funciones f propuestas originalmente, en este trabajo usamos el producto punto $f(\mathrm{x}_i, \mathrm{x}_j) = \theta(\mathrm{x}_i)^T \phi(\mathrm{x}_j)$, siendo θ y ϕ transformaciones lineales similares a φ y ψ en SANet. g es, a su vez, equivalente a β en SANet.

Como podemos ver, tanto non-local net como SANet *pairwise* utilizan un patrón de agregación similar por pares. Sin embargo, mientras que α computa vectores en SANet *pairwise*, f computa escalares, y carece de efectos atencionales que recalibren cada canal por separado, lo que sí posee SANet con su atención vectorial. Adicionalmente, y más importante, la agregación en 2.7 se realiza sobre todo el mapa de características, y no solo sobre una región local. Esto es lo que le da su carácter global, o no-local.

Otra diferencia relevante es la ausencia de un equivalente a la función γ en SANet, que añade transformaciones y expresividad. Esto tiene sentido considerando que non-local está concebido más como un complemento a las convoluciones, por lo que se enfoca en los patrones atencionales de agregación. Los bloques en SANet, por su parte, buscan construir redes atencionales puras, y por lo mismo, necesitan atender también las necesidades expresivas y la capacidad de la red completa de manera consistente. Adicionalmente, γ actúa sobre vectores en SANet, permitiendo recalibrar pesos para canales. En non-local, debido al carácter escalar de f, la adición de un equivalente a γ añadiría capacidad y expresividad mínimas (con un solo canal de entrada y salida, cada convolución de 1×1 solo añade un parámetro entrenable adicional).

Por último, el factor $\mathcal{C}(\mathrm{x})$ normaliza las respuestas. Si bien no es denotado en la ecuación 2.3, SANet incluye la aplicación de softmax en su implementación pairwise. En la Figura 2.13, ilustramos de manera más explícita la implementación de los bloques SANet *pairwise* y *Non-local*.

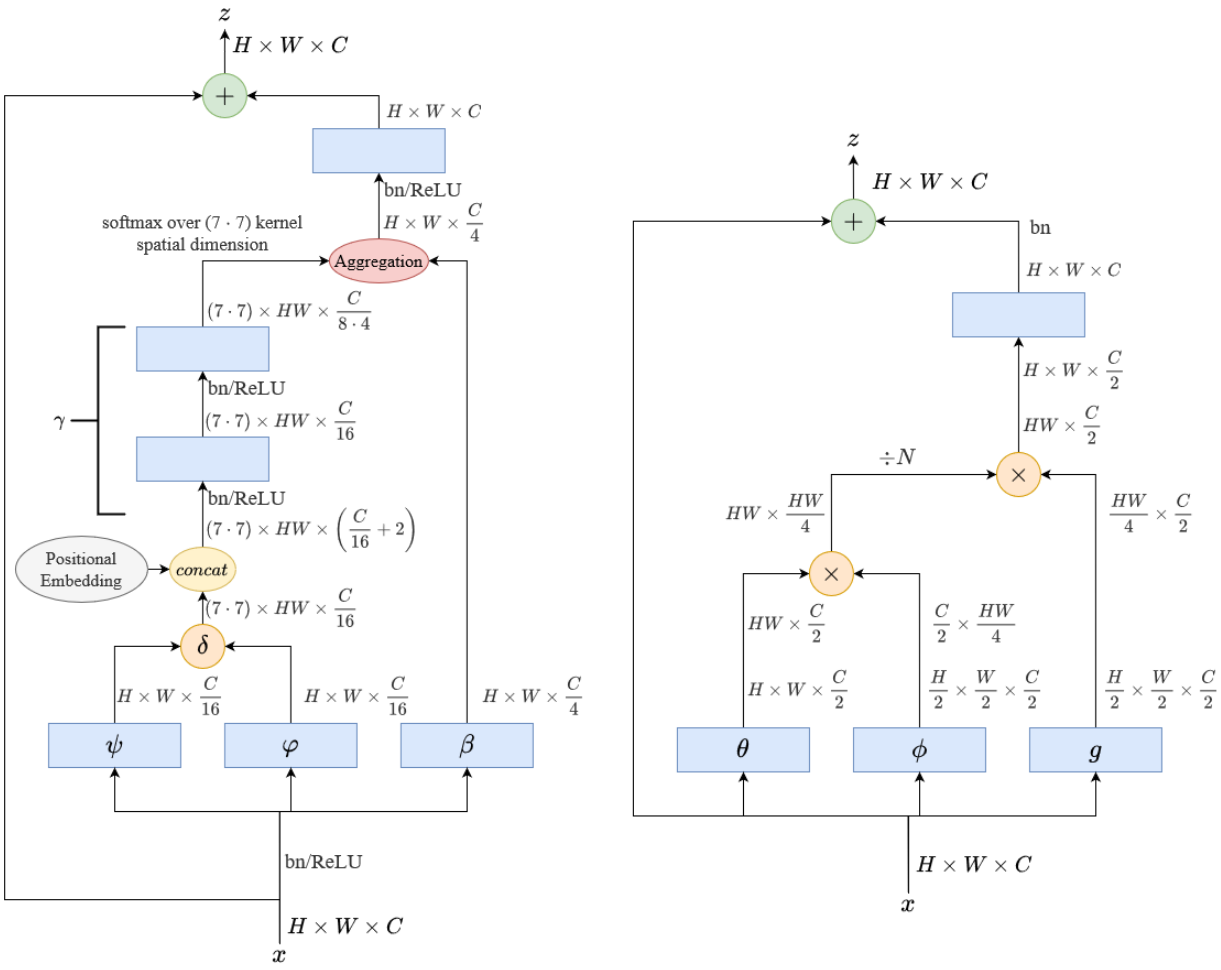

Figura 2.13: Bloques atencionales SAN *pairwise* (izquierda) y *non-local* (derecha). Los tensores de entrada tienen dimensiones de alto x ancho x canales ($H \times W \times C$). Los recuadros azules denotan convoluciones de 1×1. *Non-local*: se ilustra su versión con $f(\mathrm{x}_i, \mathrm{x}_j) = \theta(\mathrm{x}_i)^T \phi(\mathrm{x}_j)$ producto punto, reducción de canales a la mitad, subsampleo de dimensiones espaciales para θ y ϕ, y capa de batch normalization. Se denotan explícitamente las operaciones y redimensionado necesario en cada paso. SAN *pairwise*: utilizamos como ejemplo un kernel típico de 7×7, factores $r_1 = 16$, $r_2 = 4$, y 8 planos compartidos (final de la sección γ). Por simplicidad abstraemos las redimensiones explícitas, así como también algunas operaciones (δ, *positional embedding* y la agregación).

2.4. Entrenamiento SBIR

Como ya hemos comentado antes, el problema de la distancia entre los dominios foto/-dibujo hace del problema SBIR uno en que el aprendizaje resulta más difícil que en otras tareas tradicionalmente más simples, como clasificación. De los diferentes esquemas de entrenamiento propuestos para este problema, es de particular importancia para este trabajo aquel presentado en [27]. Este propone dividir el entrenamiento en varias fases con tareas incrementalmente más complejas, para llegar a la tarea final de recuperación SBIR, utilizando modelos con dos ramas: una para dibujos y otra para fotos. En la Figura 2.14 ilustramos el esquema de entrenamiento.

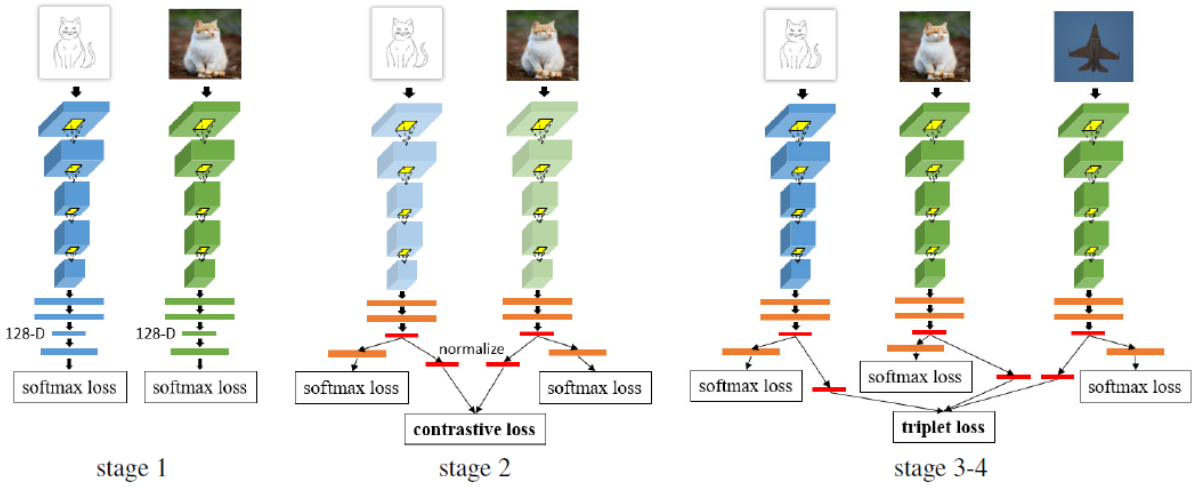

Figura 2.14: Régimen de entrenamiento para SBIR en múltiples etapas. Los bloques tridimencionales corresponden a segmentos del backbone de cada rama, y los rectangulos planos denotan vectores de características de la cabeza de la red, relacionados mediantes capas *fully connected*. Los *embeddings* a utilizar en las funciones de perdida *contrastive* y *triplet loss* provienen de vectores normalizados (normalización euclideana) de largo 128. Aunque no es explícito, una porción de los pesos en la etapa 2 y 3-4 pueden ser compartidos entre ambas ramas. Una serie de configuraciones son probadas en el trabajo original, variando la cantidad de pesos compartidos, tamaño de los vectores de embedding y las arquitecturas para cada rama. Notemos que en la etapa 3-4, ambas ramas verdes corresponden en realidad a un mismo sub-modelo. Fuente: Adaptado de [27]

Etapa 1. Esta etapa busca aprender pesos para las primeras capas del modelo (pesos no compartidos). Se entrenan 2 modelos de clasificación independientes usando *softmax*: uno para los dibujos y otro para las fotos. Idealmente, este entrenamiento utiliza como punto inicial pesos preentrenados sobre datasets grandes, como ImageNet.

Etapa 2. En la segunda etapa, formamos un solo modelo de dos "ramas": las primeras capas de cada rama utilizan los pesos obtenidos de los respectivos modelos de la etapa anterior. Estos pesos no compartidos son congelados. Las siguientes capas de este modelo doble-rama utilizan pesos compartidos. En esta etapa, el entrenamiento se realiza proveyendo pares de imágenes: un dibujo *anchor* o ancla, y una foto que puede ser un "par positivo" si el dibujo y la foto corresponden a la misma clase, o un "par negativo" si no. Cada dibujo es procesado

y clasificado por la rama "dibujo" y análogamente, cada foto es procesada y clasificada por la rama "foto". Adicionalmente, cada rama entrega un *embedding* para la foto/dibujo. Estos dos *embeddings* se utilizan para computar una función de pérdida contrastiva entre el dibujo y la foto, o ***contrastive loss***, que busca minimizar la distancia entre ejemplos similares, y maximizar la distancia entre ejemplos entre ejemplos diferentes:

$$\mathcal{L}_C(Y, \mathrm{x}^S, \mathrm{x}^I) = \frac{1}{2}(1-Y)D^2(\mathrm{x}^S, \mathrm{x}^I) + \frac{1}{2}Y\{max(0, m - D^2(\mathrm{x}^S, \mathrm{x}^I))\} \tag{2.8}$$

donde x^S es el vector *embedding* del dibujo, x^I es el vector *embedding* de la foto, m es un margen que define un límite aceptable para que x^S y x^I sean considerados diferentes o disimilares, y $D(\cdot, \cdot)$ denota la distancia euclideana entre dos vectores. Por su parte Y denota la correspondencia entre la foto y dibujo:

$$Y = \begin{cases} 0 & \text{si } (\mathrm{x}^S, \mathrm{x}^I) \text{ son similares} \\ 1 & \text{si } (\mathrm{x}^S, \mathrm{x}^I) \text{ son disimilares} \end{cases} \tag{2.9}$$

La pérdida total se define como la suma de la pérdida constrastiva y aquellas dos obtenidas mediante la clasificación con *softmax*. Esta etapa busca el aprendizaje de pesos iniciales para las capas compartidas.

Etapa 3. En la tercera etapa, descongelamos todas las capas, y entrenamos entregando tripletas de imágenes a la red: un dibujo *anchor* o ancla, una foto "par positivo", y una foto "par negativo". De manera similar a la etapa 2, el dibujo es procesado y clasificado mediante la rama de "dibujo", y las dos fotos son procesadas y clasificadas utilizando la misma rama de "foto". Cada una de las entradas genera un *embedding* de salida, el que esta vez se procesa mediante una función de pérdida ***triplet loss***:

$$\mathcal{L}_T(\mathrm{x}^S, \mathrm{x}^I_+, \mathrm{x}^I_-) = \frac{1}{2}\{max(0, m + D^2(x^S, x^I_+) - D^2(x^S, x^I_-))\} \tag{2.10}$$

que busca que la distancia entre el *anchor* y el par negativo sean mayor a la distancia entre el *anchor* y el par positivo por un margen m.

Etapa 4 (opcional). Una cuarta etapa puede repetir el entrenamiento de la etapa 3, con algún otro dataset auxiliar, para refinar el modelo.

Este trabajo utilizó los datasets TU-Berlin y Flickr25k para entrenamiento (etapas 1-3), y Sketchy (etapa 4) logrando resultados de hasta 41.13 % de mAP (etapa 3) y 53.26 % (etapa 4), evaluando en Flickr15k.

Capítulo 3

Metodología

Si bien modelos atencionales como SANet tienen una eficiencia teórica comparable con ResNet, sus implementaciones actuales en hardware moderno tienden a ser menos eficientes en la práctica, debido al uso de patrones atencionales especializados. Esto, junto con el uso de conjuntos de datos de gran tamaño, hacen que el entrenamiento repetido y la exploración casual de diferentes ajustes experimentales sea excesivamente lenta para algunos experimentos, especialmente aquellos con datasets de mayor tamaño.

Para aleviar estos problemas, primero realizamos experimentos en tareas más sencillas, y con conjuntos de datos más pequeños. Esto nos permite estudiar el rendimiento tentativo de los mecanismos atencionales y explorar diferentes ajustes experimentales y arquitectónicos para utilizar en pruebas posteriores, con mayor cantidad de datos y esquemas de entrenamiento más complejos.

Con esta metodología, antes de abordar el problema SBIR final (recuperar imágenes de objetos reales usando dibujos como consulta), probamos modelos autoatencionales puros, convolucionales puros, e híbridos, en tareas de clasificación con dibujos y recuperación de dibujos utilizando también dibujos como consulta. Para evitar ambigüedades, llamaremos a esta última modalidad SBIR "unimodal", por su único dominio del contenido, a diferencia del SBIR bimodal que utiliza dibujos como entrada y fotos reales como valor de recuperación.

3.1. Hardware y software

Durante el transcurso de nuestro trabajo, utilizamos dos configuraciones de hardware diferentes: **(1)** Una con dos tarjetas TITAN X (12Gb cada una, 24Gb total) y **(2)** otra con dos tarjetas TITAN RTX (24Gb cada una, 48Gb total). La segunda configuración de hardware estuvo disponible más tarde en el transcurso del trabajo, y fue utilizada para repetir múltiples experimentos realizado en la primera configuración de hardware, pero con algunas diferencias de diseño (ver sección 3.4.1 este capítulo). Adicionalmente, tanto el entrenamiento con ImageNet y los resultados finales expuestos para SBIR bimodal corresponden exclusivamente a la configuración **(2)**.

Para la implementación, utilizamos como base el código original de SANet [7] (PyTorch), de Zhao et al (disponible en https://github.com/hszhao/SAN). Debido a dificultades para replicar resultados existentes en la literatura para entrenamiento SBIR, también se utilizó temporalmente un proyecto basado en *tensorflow*, el cual finalmente fue desechado por enfrentar dificultades similares. En el capítulo 5 discutimos algunas de estas dificultades. Nuestro código se encuentra disponible en https://github.com/FearsomeTuna/Hybrid.

3.2. Datasets

Los datasets seleccionados para nuestros experimentos son ampliamente utilizados en SBIR y reconocimiento de imágenes. La mayoría de ellos son utilizados en [27], cuya metodología usamos como base para el entrenamiento SBIR. Para pruebas de clasificación, adicionalmente, incluimos un conjunto de dibujos de mayor tamaño extraído de QuickDraw [28].

TU-Berlin. Presentado en el trabajo "How Do Humans Sketch Objects" [29], TU-Berlin es un dataset de 20.000 dibujos en escala de grises, distribuidos igualmente entre 250 clases. Estos dibujos fueron realizados por personas bajo instrucciones generales que buscan hacerlos más reconocibles y aptos para su uso en modelos de recuperación.

Mini QuickDraw. QuickDraw es un conjunto de datos obtenido por Google mediante el juego "Quick, Draw!" disponible online [28]. El dataset incluye 50 millones de dibujos en escala de grises, distribuidos en 345 clases diferentes. Para nuestros experimentos, hemos extraído un subconjunto de estos datos, que hemos llamado Mini-QuickDraw, obtenido muestreando 1000 elementos al azar de cada clase, para un total de 345.000 dibujos.

Flickr15k. Flickr15k [30] incluye 14660 imágenes de 33 clases, extraídas de Flickr, y 10 dibujos para cada clase (330 dibujos en total) para su uso en SBIR. Estos dibujos fueron realizados por 10 participantes a quienes se les mostraron imágenes de referencia brevemente, antes de dibujarlas.

Flickr25k. Flickr25k [27] es un dataset que combina los dibujos de TU-Berlin con 25.000 fotos (100 de cada clase) obtenidas de Google, Bing y Flickr para su uso en SBIR.

ILSVRC dataset. Imagenet [31] es un dataset con más de 14 millones de imagenes a color, de las cuales el subconjunto más utilizado es aquel de ILSVRC o *ImageNet Large Scale Visual Recognition Challenge*, que considera 1000 clases y contiene 1.281.167 imágenes de entrenamiento, 50.000 imágenes de validación y 100.000 imágenes de test. En nuestro trabajo, utilizamos solo los conjuntos de entrenamiento y validación.

3.3. Comparación con otros trabajos

Vale notar que, si bien podemos trazar algunas comparaciones tentativas con las métricas alcanzadas por otros trabajos, la comparación directa de estos valores nominales con

frecuencia no es conceptualmente informativa, en tanto no siempre se controlan razonablemente variables que influyen de manera importante en el rendimiento de los modelos. Por ejemplo, un modelo con 5 millones de parámetros entrenables podría tener un rendimiento significativamente menor a otro con 20 millones, incluso si el segundo utiliza una arquitectura normalmente reconocida como menos potente. Tampoco es extraño que un modelo entrenado por una cantidad de épocas significativamente menor tenga un menor desempeño. Otros factores que afectan los resultados son la complejidad del dataset, su tamaño, cantidad de clases, la cantidad de capas de la arquitectura y otros ajustes experimentales como el optimizador, *scheduler*, y tasa de aprendizaje.

Por otra parte, decidir qué parámetros se deben controlar para una comparación "justa" tampoco es un proceso trivial. Puede ser que una arquitectura propuesta esté pensada para ser usada con ciertos ajustes experimentales particulares. En este caso, igualar estos ajustes podría llevar, contraintuitivamente, a una comparación injusta. Es decir, buscamos mantener los elementos que caracterizan a la arquitectura y confieren —o conjeturamos que confieren— las propiedades de interés que queremos comparar.

Posiblemente, el método más común para realizar comparaciones es el uso de datasets de entrenamiento y validación convenidos, con frecuencia sin importar otras condiciones experimentales, mientras el modelo alcance su mejor rendimiento posible. Esta provee una comparación de rendimiento absoluto. Sin embargo, puede traducirse en barreras de entrada para la investigación con hardware menos potente. En otras situaciones, el uso de estos marcadores absolutos no refleja apropiadamente la comparación de interés, en cuyo caso una aproximación alternativa es definir e implementar una línea base "a medida" para el trabajo en cuestión, usualmente a base de arquitecturas ampliamente utilizadas, que actúa como un marcador auxiliar de comparación indirecta con otros trabajos.

Si bien en nuestro trabajo empleamos datasets de uso estándar para mejorar la comparación, hemos decidido favorecer la última aproximación mencionada, especialmente para los experimentos de clasificación. Esto es debido al alcance del trabajo, condiciones de hardware, y especialmente a restricciones de tiempo, dado —como ya mencionamos— el carácter prohibitivo que nos impone el entrenamiento con datasets de gran tamaño para la exploración casual de parámetros y ajustes.

En general, para los experimentos de clasificación seguimos una aproximación similar a la utilizada en el trabajo asociado a SANet [7] en lo que respecta a comparación —en tanto también busca comparar modelos atencionales con modelos convolucionales—, solo que nosotros usamos datasets más pequeños en la mayoría de los casos. En ese mismo trabajo basamos también la construcción general de nuestros modelos, y por lo mismo proveemos comparaciones con él en los pocos casos en que utilizamos el mismo dataset (ImageNet).

En lo que respecta a los experimentos de recuperación, algunas de las restricciones anteriores no aplican, dado que los datasets utilizados como marcadores estándar de rendimiento son de menor tamaño en este caso. Incluso sin ser comparaciones "justas", proveemos valores referenciales que nos aportarán perspectiva para interpretar nuestros resultados. En particular, comparamos con el trabajo de Bui et al. [27] en que basamos la metodología de entrenamiento (ver sección 2.4) y con Fuentes en su trabajo de título [32]. Este último también se basa en el mismo esquema de entrenamiento, y a pesar de utilizar modelos de mayor

capacidad, estos se basan en arquitecturas derivadas de ResNet más similares a las nuestras que aquellas usadas por Bui et al.

3.4. Esquema experimental

3.4.1. Clasificación

Para las pruebas de clasificación, realizamos un conjunto de experimentos en dos versiones. En una de ellas, los bloques ResNet y Non-local siguen un diseño con **post-activación**, diferente del diseño con **pre-activación** de SANet. Esto se corresponde con las formas de activación originales de cada bloque. En el otro conjunto de experimentos, modificamos el diseño de los bloques ResNet y Non-local para que todos los bloques utilicen pre-activación.

La pre y post-activacion se relacionan con la forma de ordenar componentes dentro de un bloque residual para incluir o excluir la última función de activación en el bloque (ver Figura 3.1). La formulación ResNet original corresponde a post-activación. En cambio, el trabajo realizado en [33] sugiere un diseño ligeramente diferente para incluir todos los componentes dentro del bloque residual (antes de la suma). Ilustrativamente, esto hace que la propagación del gradiente mediante las conexiones residuales encuentre menos 'obstaculos' en el camino.

En nuestro caso, la combinación de estos diferentes diseños en bloques sucesivos puede dar lugar a secuencias de capas redundantes. En un esquema de entrenamiento dejamos estas redundancias sin atender. En el otro, las evitamos optando por el diseño preactivación para todos los bloques.

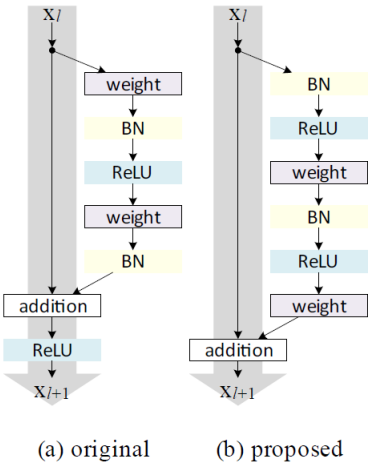

Figura 3.1: Comparación entre bloques con diseño preactivación y postactivación. Izquierda (a): formulación original postactivación. Derecha (b): formulación preactivación. El diseño con preactivación permitiría una propagación menos obstaculizada del gradiente durante el entrenamiento. Fuente: [33]

A continuación describiremos el esquema experimental para la versión que utiliza post-activación para ResNet y Non-local. Si bien solo dos de los tres tipos de bloques utilizan

post-activación, por conveniencia denotaremos este conjunto de experimentos como la versión "post-activación", y la otra, la versión "pre-activación".

Modelos de clasificación Post-activación

Utilizamos varias categorías de modelos: puramente atencionales, puramente convolucionales, e híbridos. En la Figura 3.2 resumimos la construcción general de estos modelos, y en la Tabla 3.1 comparamos la cantidad de parámetros, eficiencia teórica y tiempos de ejecución referenciales. En los apartados a continuación aclaramos su construcción con mayor detalle.

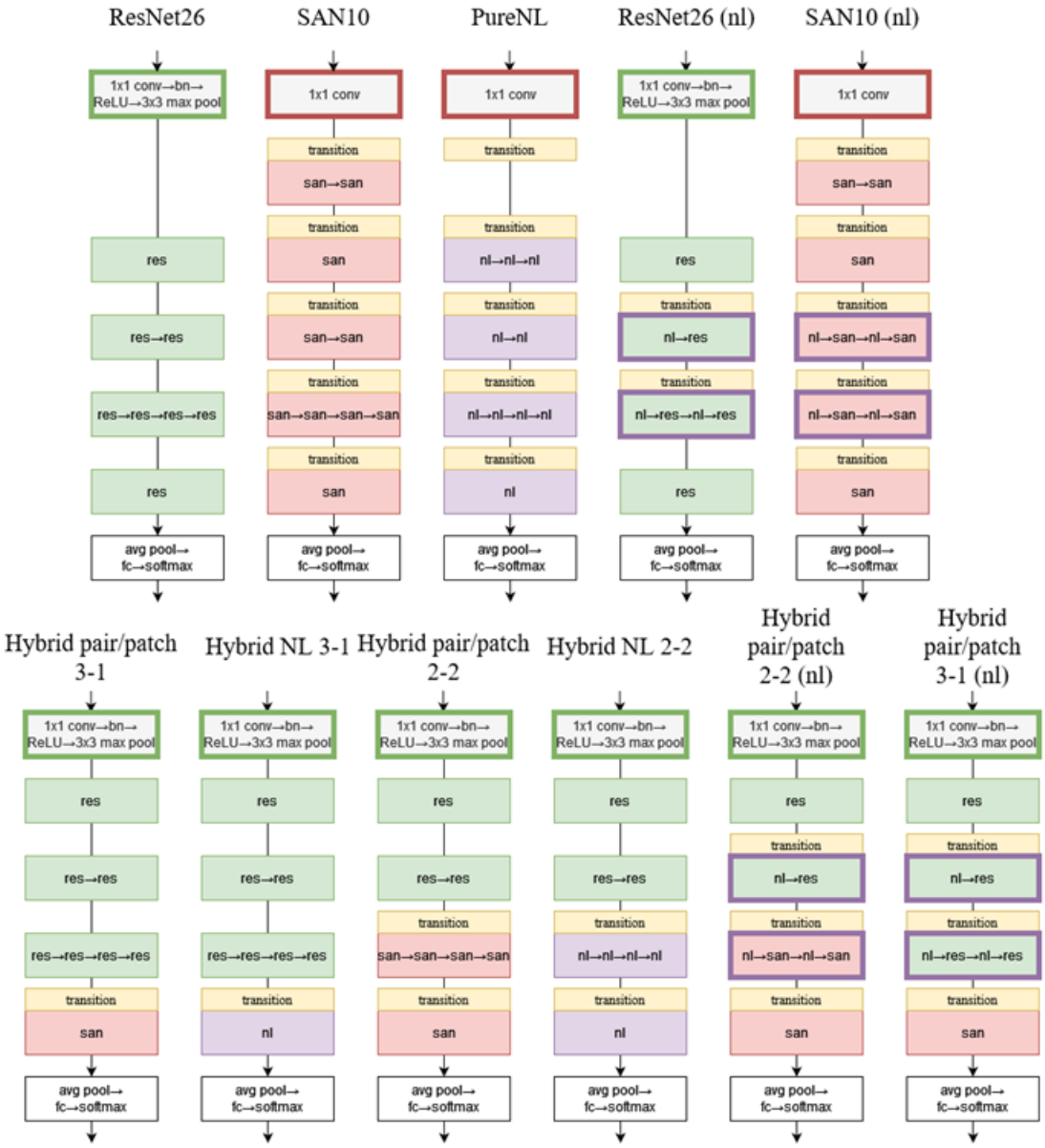

Figura 3.2: Construcción general de los modelos de clasificación. Los modelos se componen de una *stem* (proveniente de *ResNet* o de *SANet*), 4 o 5 etapas de bloques de diferentes tipos —posiblemente precedidas por etapas de transición—, y una fase final que clasifica las características extraídas.

Model	Params	Flop	Relative train time	Relative inference time	Absolute train time (ms)	Absolute inference time (ms)
ResNet26	12.15M	2.28G	1	1	129*	50*
ResNet26 (nl)	11.58M	2.0G	0.98	1	127	50
SAN10 pairwise	8.99M	2.15G	4.24	3.2	547	160
SAN10 pairwise (nl)	9.27M	2.03G	3.59	2.76	464	138
SAN10 patchwise	10.3M	1.86G	3.06	2.68	395	134
SAN10 patchwise (nl)	10.23M	1.84G	2.62	2.32	338	116
HIB pair 2-2	9.2M	2.06G	1.65	1.6	214	80
HIB pair 2-2 (nl)	9.4M	1.88G	1.36	1.36	176	68
HIB patch 2-2	10.41M	1.9G	1.48	1.6	191	80
HIB patch 2-2 (nl)	10.3M	1.78G	1.24	1.4	161	70
HIB pair 3-1	10.87M	2.2G	1.06	1.04	138	52
HIB pair 3-1 (nl)	10.3M	1.92G	1.03	1.08	134	54
HIB patch 3-1	11.47M	2.17G	1.06	1.08	138	54
HIB patch 3-1 (nl)	10.9M	1.89G	1.02	1.12	132	56
Pure NL	10.17M	2.28G	2.13	2	275	100
HIB NL 2-2	10.2M	1.87G	0.97	1.04	126	52
HIB NL 3-1	11.37M	2.17G	0.97	1.04	126	52

Tabla 3.1: Cantidad de parámetros (millones), eficiencia teórica (usando como referencia modelos de clasificación de 250 clases) en operaciones de multiplicación-acumulación (MAC —multiply-accumulate operations), y tiempos de ejecución relativos y absolutos de entrenamiento e inferencia para batch de tamaño 64. Para las mediciones de tiempo, se utilizó la configuración de hardware **(1)** (ver sección 3.1). El tiempo relativo denota la relación entre el tiempo absoluto del modelo, comparado con el tiempo absoluto de la línea base ResNet26 (marcado con *, es decir, $\frac{t_{model}}{t_{resnet26}}$). Nota: la medición de tiempo puede variar según el entorno de ejecución, procesos activos y hardware utilizado. Los valores mostrados buscan dar una idea aproximada.

Variantes puras. Las variantes puras (atencionales y convolucionales) corresponden a las mostradas en la Tabla 2.3. A estas añadimos una variante puramente atencional basada en non-local blocks, indicada en la Tabla 3.2. El diseño de esta última sigue el de SANet, utilizando fases de transición. Debido a restricciones de memoria al aplicar estos bloques *non-local* sobre mapas de mayor resolución, hemos omitido una de las etapas de bloques, dejando solo 4 de las 5 intencionadas, y aumentando la cantidad de bloques de la primera etapa, para compensar. Debido a la falta de expresividad de los bloques non-local —no

aplican ReLU—, hemos añadido batch normalization y ReLU antes de cada bloque para esta variante (omitido en la Tabla 3.2 por simplicidad). Los resultados preliminares de este modelo fueron significativamente más bajos, por lo que fue omitido en las pruebas de clasificación con "pre-activación" y experimentos subsiguientes.

Para asemejar la cuenta de parámetros y complejidad con la de los bloques SANet, hemos modificado el diseño con cuello de botella del bloque non-local original, aplicando una reducción de canales más agresiva, y permitiendo un factor de reducción separado para la transformación g (ver Figura 3.3). A pesar de que esta reducción más agresiva facilita la comparación en nuestro trabajo, no realizamos pruebas para controlar por el efecto de esta modificación en relación a la formulación original.

Cabe mencionar que, a diferencia de su formulación original en [16], utilizamos la implementación de bloques residuales ResNet de PyTorch, que aplica la reducción de dimensiones en la segunda convolución (3×3) en lugar de la primera convolución (1×1). Esta variante se conoce como ResNet 1.5 y mejora el rendimiento según [34].

Stage	**Output size**	**PureNL**
	$224 \times 224 \times 64$	64-d linear
Transition	$112 \times 112 \times 64$	2×2 max pool, stride $2 \rightarrow$ 64-d linear
Block	$112 \times 112 \times 64$	
Transition	$56 \times 56 \times 256$	2×2 max pool, stride $2 \rightarrow$ 256-d linear
Block	$56 \times 56 \times 256$	[256-d nl] $\times$ 3
Transition	$28 \times 28 \times 512$	2×2 max pool, stride $2 \rightarrow$ 512-d linear
Block	$28 \times 28 \times 512$	[512-d nl] $\times$ 2
Transition	$14 \times 14 \times 1024$	2×2 max pool, stride $2 \rightarrow$ 1024-d linear
Block	$14 \times 14 \times 1024$	[1024-d nl] $\times$ 4
Transition	$7 \times 7 \times 2048$	2×2 max pool, stride $2 \rightarrow$ 2048-d linear
Block	$7 \times 7 \times 2048$	[2048-d nl] $\times$ 1
Classification	$1 \times 1 \times 1000$	Global average pool $\rightarrow$ 1000 fc linear $\rightarrow$ softmax

Tabla 3.2: Modelo atencional puro basado en non-local blocks. 'C-d' denota una salida de C dimensiones. 'C-d nl' denota un bloque non-local. En gris: fase de bloques omitida por restricciones de memoria.

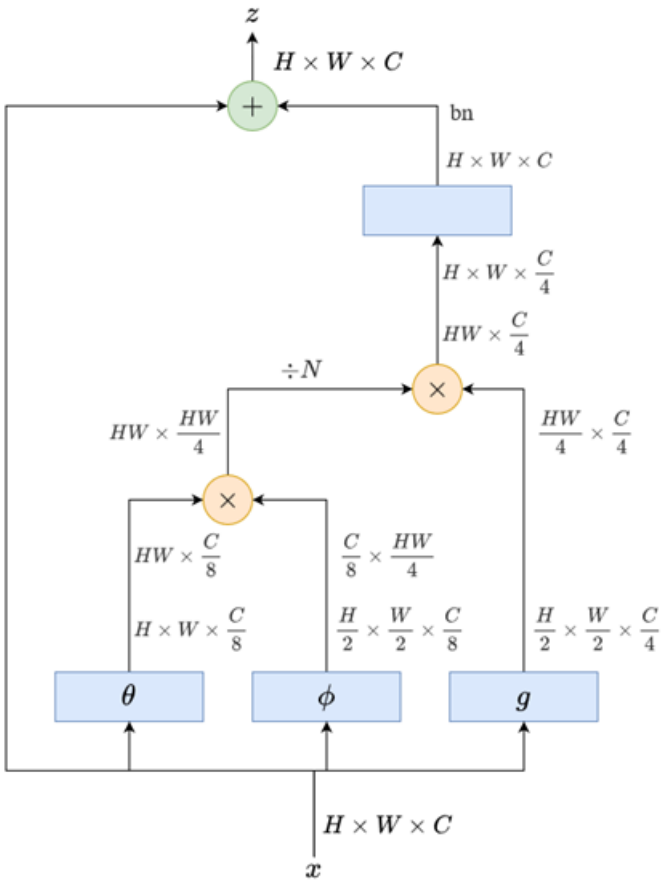

Figura 3.3: Bloque Non-local modificado. La única diferencia con la implementación original es la reducción de canales más agresiva para θ y ϕ, y la reducción separada, y también más agresiva, para g, equivalente al factor r_2 en SANet.

Variantes híbridas. Construimos las variantes híbridas combinando directamente una porción inicial de etapas convolucionales obtenida de ResNet26 y una porción de etapas atencionales obtenida de SAN10 o de nuestra variante Non-local. De las 4 etapas totales, solo la última, o últimas dos, son atencionales. Las etapas atencionales están precedidas por etapas de transición, al igual que en SANet. Dependiendo de la proporción de etapas, denotamos estas variantes con la notación "HIB *c*-*a*" donde *c* indica la cantidad de etapas convolucionales, y *a* indica la cantidad de etapas atencionales. Probamos estas variantes tanto con bloques SANet *pairwise* como *patchwise*. En la Tabla 3.3 mostramos las variantes híbridas HIB 3-1 y HIB 2-2 que combinan ResNet con SANet. Podemos construir las combinaciones ResNet-NonLocal reemplazando las fases de bloques atencionales de la Tabla 3.3 por aquellas definidas para la Tabla 3.2.

Esta aproximación al diseño híbrido es similar a la de redes como BotNet [3]. La intuición detrás de esta construcción supone que las convoluciones son suficientemente potentes —y más eficientes— para capturar patrones de bajo nivel en imágenes, y que entonces la atención es mejor aprovechada al aplicarse sobre características de mayor nivel de abstracción.

Variantes con bloques non-local añadidos. Siguiendo el planteamiento original de Non-local net [4], también probamos integrando estos bloques en pequeñas cantidades en

Stage	Output size	HIB 3-1	HIB 2-2
	$112 \times 112 \times 64$	7×7, 64, stride 2	7×7, 64, stride 2
	$56 \times 56 \times 256$	3×3 max pool, stride 2	3×3 max pool, stride 2
Block	$56 \times 56 \times 256$	$\begin{bmatrix} \text{64-d linear} \\ 3 \times 3, 64 \\ \text{256-d linear} \end{bmatrix} \times 1$	$\begin{bmatrix} \text{64-d linear} \\ 3 \times 3, 64 \\ \text{256-d linear} \end{bmatrix} \times 1$
Block	$28 \times 28 \times 512$	$\begin{bmatrix} \text{128-d linear} \\ 3 \times 3, 128 \\ \text{512-d linear} \end{bmatrix} \times 2$	$\begin{bmatrix} \text{128-d linear} \\ 3 \times 3, 128 \\ \text{512-d linear} \end{bmatrix} \times 2$
Transition	$14 \times 14 \times 1024$		2×2 max pool, stride 2 $\rightarrow$ 1024-d linear
Block	$14 \times 14 \times 1024$	$\begin{bmatrix} \text{256-d linear} \\ 3 \times 3, 256 \\ \text{1024-d linear} \end{bmatrix} \times 4$	$\begin{bmatrix} 7 \times 7, \text{ 256 sa} \\ \text{1024-d linear} \end{bmatrix} \times 4$
Transition	$7 \times 7 \times 2048$	2×2 max pool, stride 2 $\rightarrow$ 2048-d linear	2×2 max pool, stride 2 $\rightarrow$ 2048-d linear
Block	$7 \times 7 \times 2048$	$\begin{bmatrix} 7 \times 7, \text{ 512 sa} \\ \text{2048-d linear} \end{bmatrix} \times 1$	$\begin{bmatrix} 7 \times 7, \text{ 512 sa} \\ \text{2048-d linear} \end{bmatrix} \times 1$
Classification	$1 \times 1 \times c$	Global average pool $\rightarrow$ fc $\rightarrow$ softmax	

Tabla 3.3: Modelos híbridos que combinan ResNet26 con SAN10. La entrada inicial es de 224×224. La capa *fully connected* (fc) final varía su dimensión dependiendo de la cantidad de clases a clasificar C. En amarillo: etapas derivadas de SAN10.

las etapas intermedias, en lugar de utilizar etapas completas basadas en bloques non-local. Exceptuando los modelos que ya incluyen etapas de bloques non-local, probamos todos los modelos anteriores reemplazando 3 bloques (ya sean bloques ResNet o SANet) por bloques non-local: dos de estos en la penúltima etapa, y otro en la antepenúltima, de manera intercalada con otros tipos de bloques, de forma similar a su configuración en [4]. Con esto, la secuencia de bloques queda como NL→■→NL→■ para la penúltima etapa y NL→■ para la antepenúltima, donde 'NL' denota un bloque non-local, y ■ denota el tipo de bloque que tenía anteriormente la etapa en cuestión. En el caso ■=ResNet, precedemos la etapa con una etapa de transición SANet. Esto es debido a la falta del primer bloque ResNet de la etapa, que es precisamente el que realiza la reducción de dimensiones espaciales y la expansión de canales.

Añadir estos bloques en etapas intermedias es importante por dos motivos. Primero: en todos los modelos probados, el tamaño de la entrada de la etapa final es pequeño (7×7), con lo que cada kernel cubre una porción significativa del mapa total, acercándose al efecto buscado con la atención global, pero gozando de los beneficios de la atención vectorial (más potente). Segundo: tal y como es mencionado en [4], las dimensiones espaciales reducidas de la última etapa podrían ser insuficientes para proveer información espacial precisa (siendo estas relaciones espaciales el principal aporte de los bloques NL).

Estas variantes —con bloques non-local añadidos— no solo buscan replicar los resultados cualitativos de [4], sino también explorar el complemento de la atención local de SANet con la atención global de Non-local net para capturar interdependencias espaciales distantes. Las denotamos en gráficos y tablas añadiendo "(nl)" luego del nombre del modelo base sobre el cual se construyen.

Modelos de clasificación Pre-activación

El uso de bloques con pre y post-activación en un mismo modelo da lugar a secuencias de capas redundantes al pasar de bloques con un tipo de activación a otro diferente. Por ejemplo: bn→ReLU→bn→ReLU al pasar de post-activación a pre-activación; o dos convoluciones de 1×1 seguidas, al pasar de pre a post-activación.

Con la intención de evitar estas transiciones redundantes —y adicionalmente, eliminar fuentes de variabilidad indeseada en la comparación de las arquitecturas—, planteamos versiones modificadas de las arquitecturas formuladas en la sección anterior como sigue:

- Los bloques ResNet y Non-local utilizan un diseño con pre-activación (ver Figuras 3.4 y 3.5). Notemos que el diseño pre-activación del bloque Non-local no solo añade o desplaza capas de *batchnormalization*, sino que añade expresividad (ReLU), antes ausente en el diseño del bloque. Por lo mismo, las capas bn→ReLU antes introducidas entre bloques Non-local sucesivos ya no son necesarias.
- El diseño de la *stem* proveniente de ResNet (1×1 conv→bn→ReLU→ 3×3 max pool) es reemplazado por "conv→ 3×3 max pool". Si la última etapa de bloques es ResNet o Non-local, esta es sucedida por *batch normalization* y ReLU antes de proceder a la cabeza del modelo (el clasificador).

Omitimos la variante pura NL en este conjunto de experimentos, debido a su evidente rendimiento inferior en los experimentos "post-activación" en pruebas preliminares, además de su tiempo de entrenamiento comparativamente alto, y uso de memoria inesperadamente alto para su versión "pre-activación" en la configuración de hardware **(2)** (ver sección 3.1).

Régimen de entrenamiento clasificación

Para clasificación, utilizamos los conjuntos TU-Berlin y Mini QuickDraw. El resto de los conjuntos de datos se reservan para SBIR bimodal.

Experimentos "post-activación". Para la clasificación con TU-Berlin, el 80 % de los datos fue asignado al conjunto de entrenamiento, y el 20 % restante al conjunto de validación. Se entrenó cada modelo por 100 épocas. Para Mini-QuickDraw, el conjunto de datos se separó de la siguiente manera: de las 345 clases, utilizamos solo 300, escogidas al azar, para entrenar en clasificación, reservando las otras 45 clases para pruebas de recuperación (ver sección 3.4.2). De estas 300.000 imágenes, el 80 % fueron asignadas al conjunto de entrenamiento y el 20 % restante al conjunto de validación. Se entrenó cada modelo por 40 épocas.

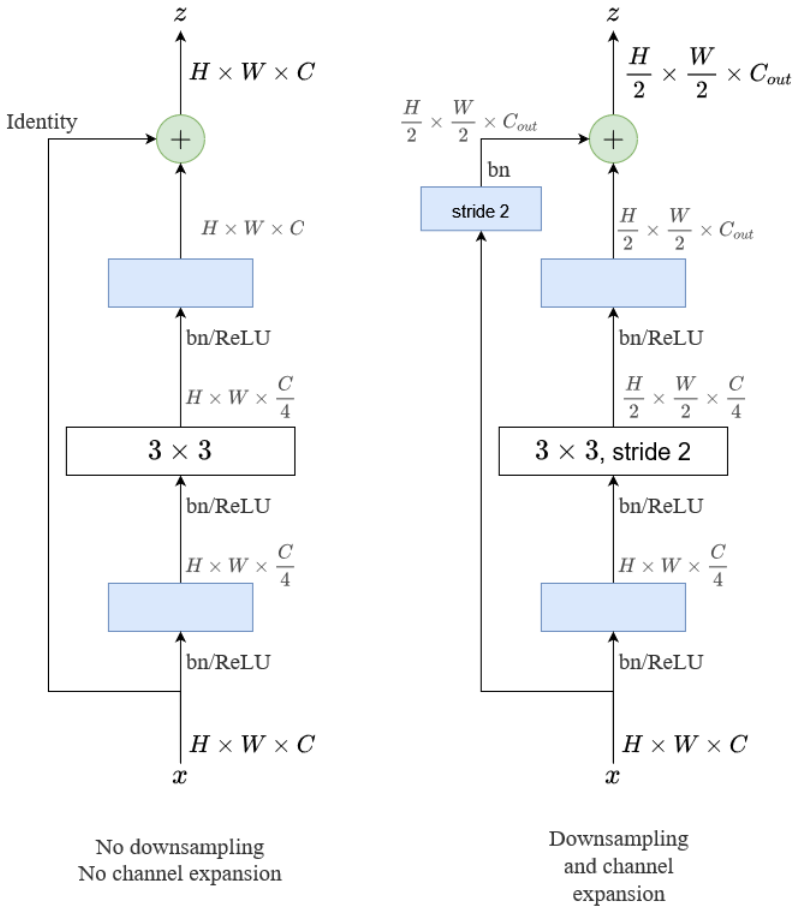

Figura 3.4: Bloque residual ResNet con diseño pre-activación [33]. La conexión residual puede variar según las dimensiones de salida. Este diseño se basa en ResNet1.5 [34].

Experimentos "pre-activación". El régimen de entrenamiento para las versiones post y pre-activación de los experimentos formulados en la sección anterior es idéntico, salvo por el mayor tiempo total de entrenamiento, debido al uso de la configuración de hardware **(2)** más potente (ver sección 3.1): cada modelo fue entrenado con TU-Berlin dos veces, por 100 épocas cada vez, y dos veces con Mini-QuickDraw, por 50 épocas cada una.

3.4.2. Recuperación de dibujos unimodal

Antes de pasar a la recuperación en base a dibujos bimodal, utilizamos los mismos modelos de clasificación descritos en la sección 3.4.1 para estudiar la calidad de las representaciones logradas, mediante un esquema de recuperación sobre clases no vistas en el entrenamiento (zero-shot).

Para esto, utilizamos las 45 clases de mini QuickDraw que no fueron utilizadas durante el entrenamiento en clasificación. De las 45.000 imágenes que comprenden estas clases, seleccionamos al azar 500 de cada una (22.500 en total) para formar el conjunto en el cual buscamos, el que llamaremos catálogo. De estas, escogemos a su vez 50 imágenes al azar de cada clase para actuar como consultas (2.250 consultas en total).

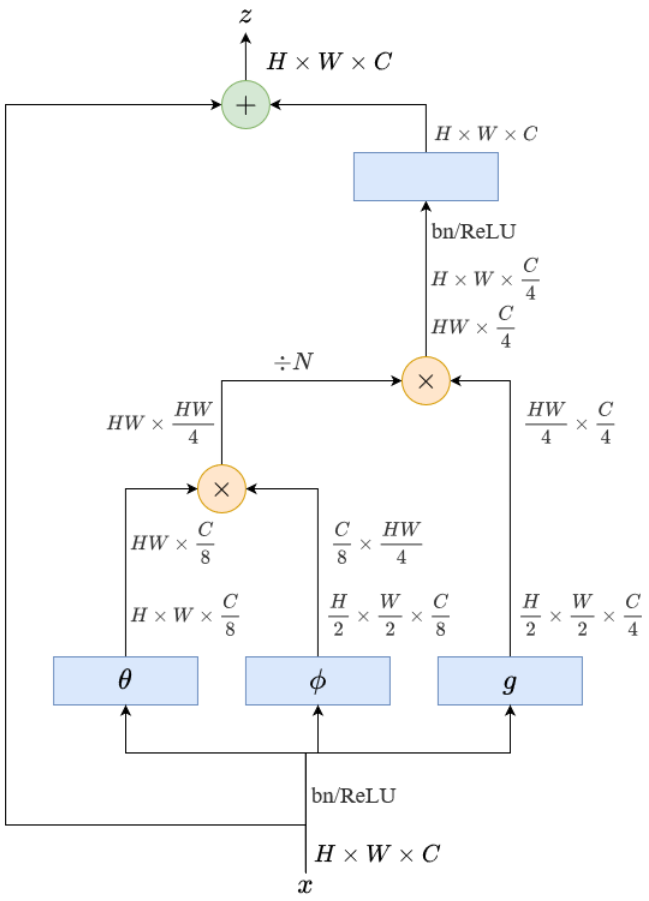

Figura 3.5: Bloque Non-local con diseño pre-activación. Sigue el mismo diseño general que aquel mostrado en la Figura 3.3, pero añadiendo/desplazando capas de *batchnormalization* y ReLU al inicio del bloque, y antes de la convolución final.

Usando la mejor época para los modelos de clasificación ya entrenados, procesamos las imágenes del catálogo y las consultas, y extraemos la representación resultante de la última etapa de bloques (es decir, luego de aplicarle *global average pool*) para formar *embeddings* de tamaño 2048. Luego, calculamos la similitud entre cada *embedding* de consulta con todas las imágenes del catálogo, utilizando la similitud coseno (normalización de los vectores y producto punto). A partir de los puntajes resultantes, calculamos mAP y mRR.

3.4.3. Recuperación en base a dibujos bimodal

Para los experimentos en SBIR bimodal, nos basamos en la metodología indicada en [27] (ver sección 2.4). Debido al uso de datasets de gran tamaño (ImageNet) y problemas para replicar resultados similares al trabajo original en [27], nuestros experimentos en esta modalidad solo comprenden la línea base ResNet y la variante híbrida HIB patch 2-2 (nl) —utilizando sus versiones pre-activación—. Es decir, uno de los modelos SBIR bimodal comprende dos ramas ResNet26, y el otro dos ramas HIB patch 2-2 (nl). La variante híbrida fue escogida por su buen rendimiento en los experimentos descritos en las secciones anteriores.

Nuestros modelos para esta modalidad SBIR comparten los pesos de las últimas 2 etapas de bloques (2 de las 4 totales), además de la cabeza o clasificador. Esta configuración de pesos compartidos está motivada por el trabajo realizado en [32] y podemos justificar este, a su vez, en las configuraciones con mejor rendimiento de [27] para los modelos basados en InceptionV1, que son los más cercanos a nuestros modelos.

Para los pesos iniciales, a diferencia de [27], entrenamos ambas ramas (dibujo y foto) usando ImageNet, en lugar de solo hacerlo con la rama de fotos. Realizamos esto en un intento por asemejar las representaciones iniciales para cada rama. Este entrenamiento se realiza por 50 épocas. Los pesos pre-entrenados solo son cargados en las porciones de la red que no comparten pesos. Esto es decir que los pesos compartidos de las últimas dos fases de bloques, así como las capas finales, son entrenadas de cero en la etapa 1 del entrenamiento descrita en la sección 2.4.

Utilizamos solo 3 de las 4 etapas de entrenamiento, ignorando la etapa opcional. Para estas tres etapas, utilizamos el conjunto de datos Flickr25k para entrenar. Tanto para el conjunto de dibujos de este (correspondientes a TU-Berlin) como las fotos, utilizamos un 80 % para entrenamiento, y un 20 % para validación de métricas de clasificación después de cada época. Para validar métricas de recuperación después de cada época, utilizamos un subconjunto extraído del 20 % de TU-Berlin para actuar como dibujos de consulta, utilizando el 20 % completo del conjunto de fotos como catálogo de búsqueda. Si bien estas métricas no se corresponden necesariamente a aquellas obtenidas al validar con Flickr15k (gran diferencia en la cantidad de clases), las utilizamos para seleccionar la mejor época y estudiar el comportamiento del modelo durante el entrenamiento. Por su parte, las métricas finales de recuperación utilizan como conjunto de test el dataset Flickr15k.

Para la primera etapa de entrenamiento, entrenamos por 50 épocas cada rama, de manera separada. Luego, al formar el modelo doble-rama para las etapas 2 y 3, añadimos a la cabeza del modelo dos capas *fully connected* adicionales antes del clasificador final. De la segunda de estas —aquella justo antes de la capa clasificadora final— extraemos los *embeddings* que usaremos para recuperación. La primera, ausente en [27], tiene la intención de aumentar la expresividad de la representación antes de producir los *embeddings*, y por lo mismo se incluye normalización y ReLU entre ambas capas. Entre la segunda capa añadida y la capa final no hay activación ni normalización, por lo que la relación entre ellas es lineal, lo que a su vez significa que el clasificador final es teóricamente equivalente con o sin esta capa de *embedding*.

Las etapas 2 y 3 entrenan por 35 y 10 épocas, respectivamente. La formación de pares y tripletas es como sigue: para cada época, se forma un par/tripleta por cada dibujo del 80 % de entrenamiento de TU-Berlin. Todos los dibujos son utilizados en cada época, pero los pares positivos/negativos asociados a cada dibujo son escogidos al azar. La mitad de los pares (etapa 2) corresponden a pares positivos, y la otra mitad a pares negativos. Las tripletas (etapa 3) siempre incluyen un dibujo, una foto de la misma clase y una foto de una clase diferente.

3.5. Preprocesamiento y *data augmentation*

Clasificación y SBIR unimodal. A todas las imágenes de entrenamiento se les aplica:

- *RandomResizedCrop*: Se recorta una región rectangular que ocupa entre el 8 % y el 100 % del área total de la imagen. Esta región rectangular tiene una relación de aspecto entre 3/4 y 4/3. Luego esta se redimensiona a 224×224.
- *RandomHorizontalFlip*: Voltea la imagen horizontalmente con una probabilidad del 50 %.

Para la validación, las transformaciones son más simples:

- *Resize*: La imagen es redimensionada de manera que su lado más corto sea de tamaño 256, conservando la relación de aspecto.
- *CenterCrop*: Se aplica un recorte central de 224×224.

Este procesamiento sigue, por la mayor parte, aquel utilizado en [7]. Adicionalmente, todos los dibujos son son normalizados utilizando los valores de media y desviación estándar del conjunto de datos MNIST [35], por considerarlo una aproximación razonable para dibujos en escala de grises.

SBIR bimodal. Para estas pruebas, utilizamos transformaciones similares a las anteriores, pero utilizamos un recorte menos agresivo para *RandomResizedCrop* (70-100 % para fotos, 90-100 % para dibujos). Esto se debe a que esperamos que el modelo aprenda a relacionar patrones entre ambos dominios, y los recortes agresivos podrían resultar en imágenes que no retienen esta relación. Esto es especialmente cierto para los dibujos, que pueden incluir amplias regiones sin ningún tipo de contenido (espacio en blanco). Adicionalmente para el dibujo, incluimos una rotación aleatoria entre -20 y 20 grados. Esta transformación no es aplicada sobre las fotos para evitar la generación de artefactos visuales en bordes (para los dibujos, se puede rellenar con color blanco, reteniendo la consistencia visual). Para normalizar la entrada, utilizamos los valores de media y desviación estándar de ImageNet, tanto para los dibujos como las fotos. Esto es debido a que realizamos una fase de pre-entrenamiento con este dataset.

3.6. Ajustes experimentales

Pruebas iniciales en el entrenamiento —hardware **(1)**— mostraron un uso de memoria GPU significativamente mayor en las variantes atencionales. Para mantener una comparación justa, todos los modelos entrenados para el esquema de clasificación (sección 3.4.1) fueron entrenados con un tamaño de batch 64, que fue la máxima potencia de 2 alcanzada para los modelos atencionales de mayor uso de memoria en ese hardware.

El pre-entrenamiento con ImageNet utiliza un tamaño de batch 256, y el entrenamiento SBIR bimodal utiliza tamaño de batch 128. Adicionalmente, todos los modelos fueron entrenados con optimizador SGD; *momentum* 0,9; *weight-decay* 0,0001 y tasa de aprendizaje 0,1 con variación basada en coseno (*cosine annealing*). Estos ajustes, con la excepción del tamaño del batch, siguen aquellos usados en [7].

Capítulo 4

Resultados

4.1. Clasificación TU-Berlin

En la Tabla 4.1 resumimos los resultados de validación obtenidos con el dataset TU-Berlin para la mejor época de cada modelo. Para las versiones "pre-activación", los resultados corresponden al promedio de dos mediciones. Si bien la mayoría de los modelos tienen un rendimiento semejante, tanto ResNet26 como su variación "(nl)" tienen un rendimiento similar o mejor que los atencionales e híbridos. Por su parte, la variante atencional pura *Pure NL* es el modelo con el peor rendimiento. En la mayoría de los casos, las variantes **(nl)** tienen un rendimiento menor o igual que sus equivalentes sin bloques *non-local*, con la excepción de ResNet26 y HIB patch 3-1, para las versiones "post-act"; y SAN10 pairwise, para las versiones "pre-act".

En las Figuras 4.1 y 4.2 mostramos las curvas de entrenamiento para validación para la métrica *top1* en las variantes post-activación y pre-activación, respectivamente. Para mayor claridad visual, en ambas figuras seleccionamos una variante con convergencia estable y la repetimos como referencia visual para facilitar la comparación. Con el mismo objetivo, graficamos solo uno de los dos entrenamientos para las variantes "pre-act". En la mayoría de los casos el comportamiento es aproximadamente equivalente entre ambas mediciones. Sin embargo, vale notar que observamos mayores variaciones en en algunas de las variantes "(nl)". Con esto dicho, las versiones "post-act" y "pre-act" son comparables en la estabilidad de la convergencia en la mayoría de los casos.

Notemos que las variantes con mayor presencia atencional tienden a ser más estables en la convergencia, con la excepción de las variante pura SAN10 pairwise, y las variantes que integran etapas completas de bloques non-local (HIB NL 2-2 y HIB NL 3-1).

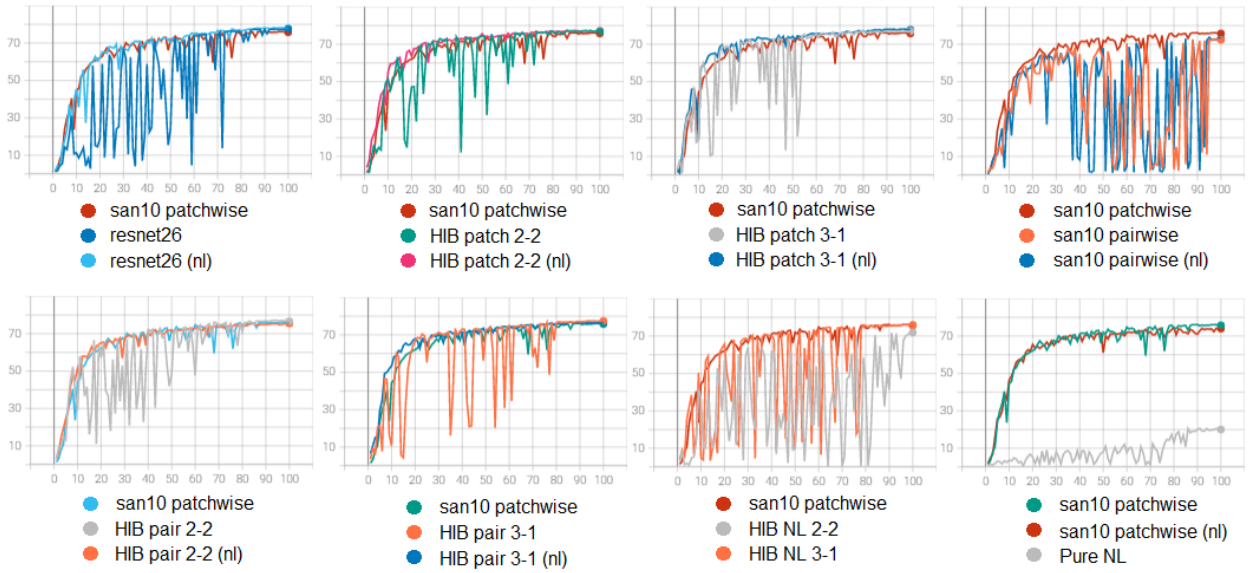

Figura 4.1: Comportamiento en el conjunto de validación para modelos de clasificación con dataset TU-Berlin en su versión "post-activación". En cada gráfico se repite como referencia visual SAN10 *patchwise*. El eje horizontal denota la cantidad de épocas, y el vertical el porcentaje alcanzado para top1.

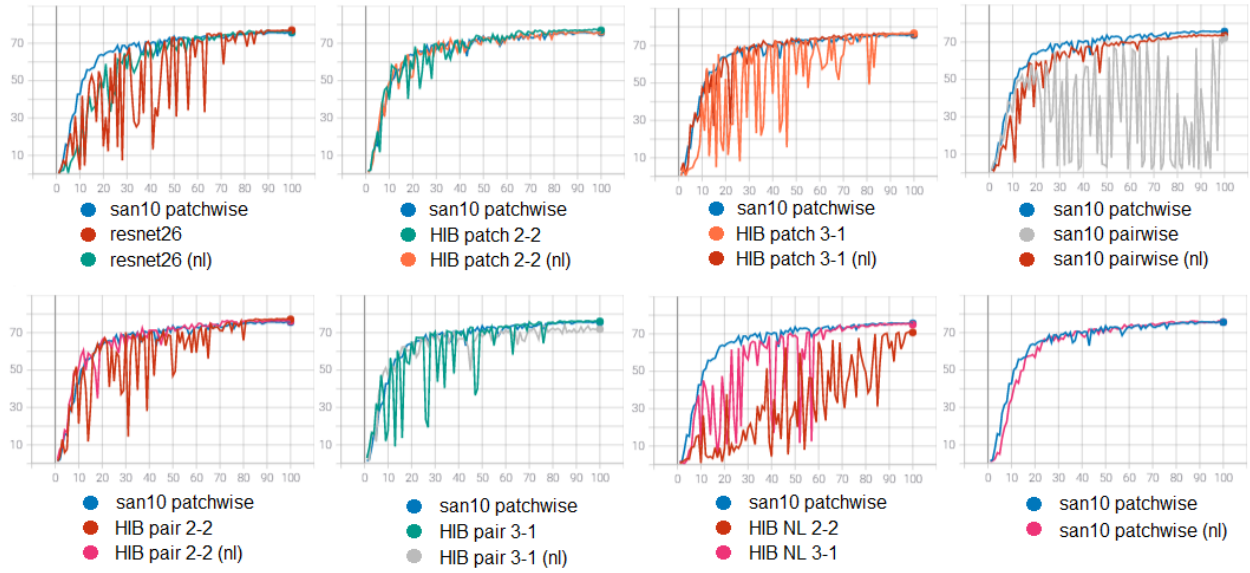

Figura 4.2: Comportamiento en el conjunto de validación para modelos de clasificación con dataset TU-Berlin en su versión "pre-activación". En cada gráfico se repite como referencia visual SAN10 *patchwise*. El eje horizontal denota la cantidad de épocas, y el vertical el porcentaje alcanzado para top1.

	Setup (1) PostAct		**Setup (2) PreAct**	
Model	**Top1 val**	**Top5 val**	**Top1 val**	**Top5 val**
Resnet26	77.52	93.12	**77.45**	**92.86**
Resnet26 (nl)	**78.20**	**93.22**	77.31	92.58
SAN10 pairwise	72.85	90.37	72.80	89.98
San10 pairwise (nl)	73.70	90.40	73.20	89.58
SAN10 patchwise	76.12	91.75	76.22	91.46
SAN10 patchwise (nl)	74.55	90.85	74.61	90.38
HIB pair 2-2	77.20	91.67	77.15	92.10
HIB pair 2-2 (nl)	75.47	91.07	76.20	91.73
HIB patch 2-2	77.42	92.12	**77.4**	92.3
HIB patch 2-2 (nl)	77.45	92.02	76.57	91.93
HIB pair 3-1	77.50	92.50	76.62	92.28
HIB pair 3-1 (nl)	76.80	91.95	75.62	90.92
HIB patch 3-1	76.25	91.50	77.17	92.21
HIB patch 3-1 (nl)	77.07	91.92	76.68	92.10
Pure NL	20.60	44.82		
HIB NL 2-2	72.07	89.42	70.43	88.72
HIB NL 3-1	76.20	91.97	75.76	91.86

Tabla 4.1: Rendimiento de los modelos de clasificación sobre dataset TU-Berlin (validación), en porcentaje. En la configuración (1) se utilizan bloques ResNet y non-local con postactivación. En la configuración (2), se utiliza preactivación. Los bloques provenientes de SANet siempre utilizan preactivación. Para la configuración (2), los valores corresponden al promedio de dos entrenamientos para cada modelo.

4.2. Clasificación Mini QuickDraw

En la Tabla 4.2 resumimos los resultados de validación obtenidos para la mejor época de cada modelo. A diferencia de los resultados obtenidos con TU-Berlin, para Mini-QuickDraw la mayoría de los modelos atencionales e híbridos presentan un rendimiento igual o mayor que ResNet26 y ResNet26 (nl). Múltiples de las variantes con presencia *pairwise* presentan un rendimiento algo menor que sus contra partes *patchwise*, como es el caso en los modelos SAN10 pairwise, SAN10 pairwise (nl) y HIB-pair 3-1. Nuevamente, las variantes basadas en etapas completas de bloques non-local presentan un rendimiento decididamente inferior a la mayoría de las variantes.

De manera equivalente a la sección anterior, en las Figuras 4.3 y 4.4 mostramos las curvas de entrenamiento para Mini-QuickDraw.

Para las versiones "post-act", la estabilidad en la convergencia de las variantes con mayor presencia atencional es menos aparente que en los experimentos con TU-Berlin, pero aun distinguible. En particular, podemos notarlo entre las variantes SAN10 patchwise y ResNet26, y entre las variantes híbridas 2-2 y las híbridas 3-1, exceptuando las variantes que integran etapas completas de bloques non-local (HIB NL).

Para las versiones "pre-act", la estabilidad en la convergencia es decididamente mayor para múltiples de las variantes. En particular, las variantes "(nl)" presentan un rendimiento superior a sus contra partes sin bloques non-local añadidos, con la excepción de ambas variantes atencionales puras Pairwise y Patchwise.

	Setup (1) PostAct		Setup (2) PreAct	
Model	**Top1 val**	**Top5 val**	**Top1 val**	**Top5 val**
Resnet26	76.57	93.00	76.63	93.02
Resnet26 (nl)	77.10	93.29	76.23	92.91
SAN10 pairwise	76.62	93.00	75.12	92.41
San10 pairwise (nl)	73.72	91.75	62.97	86.78
SAN10 patchwise	77.20	93.17	77.32	93.06
SAN10 patchwise (nl)	77.08	93.07	77.10	92.94
HIB pair 2-2	76.35	92.89	**77.48**	93.25
HIB pair 2-2 (nl)	77.39	93.19	77.25	93.12
HIB patch 2-2	77.43	93.28	77.40	93.16
HIB patch 2-2 (nl)	**77.57**	93.28	**77.46**	93.06
HIB pair 3-1	76.85	93.01	75.94	92.76
HIB pair 3-1 (nl)	77.10	92.98	77.11	93.08
HIB patch 3-1	76.63	93.00	77.02	93.06
HIB patch 3-1 (nl)	77.25	93.19	77.34	93.18
Pure NL	15.68	37.74		
HIB NL 2-2	66.70	88.63	70.69	90.47
HIB NL 3-1	76.73	92.98	70.96	90.76

Tabla 4.2: Rendimiento de los modelos de clasificación sobre dataset Mini QuickDraw (validación). En la configuración (1) se utilizan bloques ResNet y non-local con postactivación. En la configuración (2), se utiliza preactivación. Los bloques provenientes de SANet siempre utilizan preactivación. Para la configuración (2), los valores corresponden al promedio de dos entrenamientos para cada modelo.

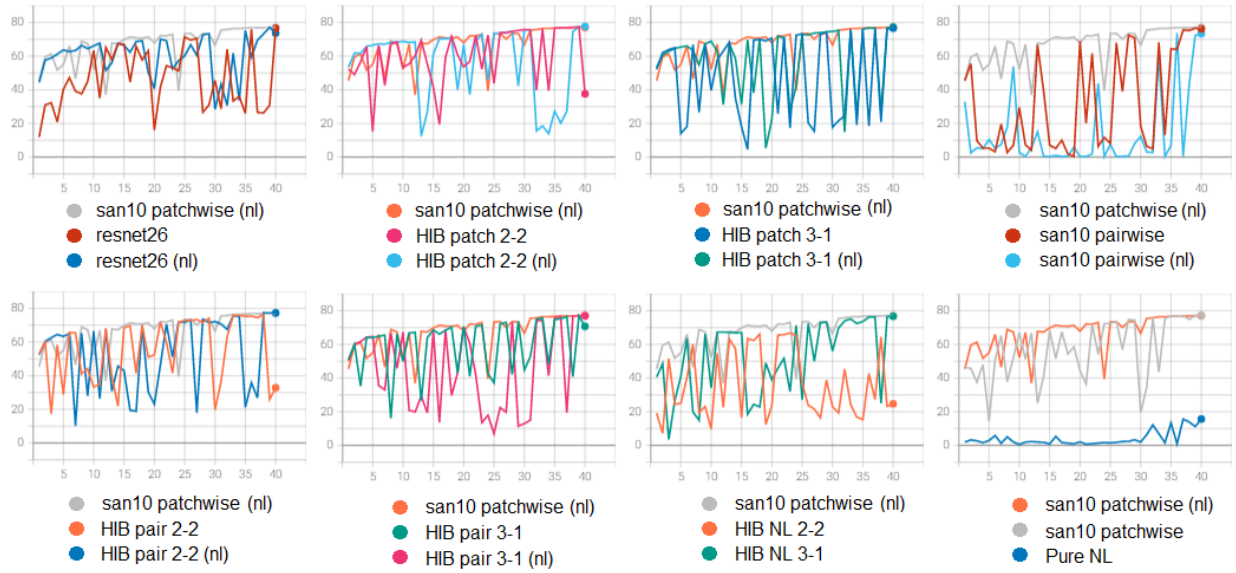

Figura 4.3: Comportamiento en el conjunto de validación para modelos de clasificación con dataset Mini-QuickDraw en su versión "post-activación". En cada gráfico se repite como referencia visual SAN10 *patchwise* (nl). El eje horizontal denota la cantidad de épocas, y el vertical el porcentaje alcanzado para top1.

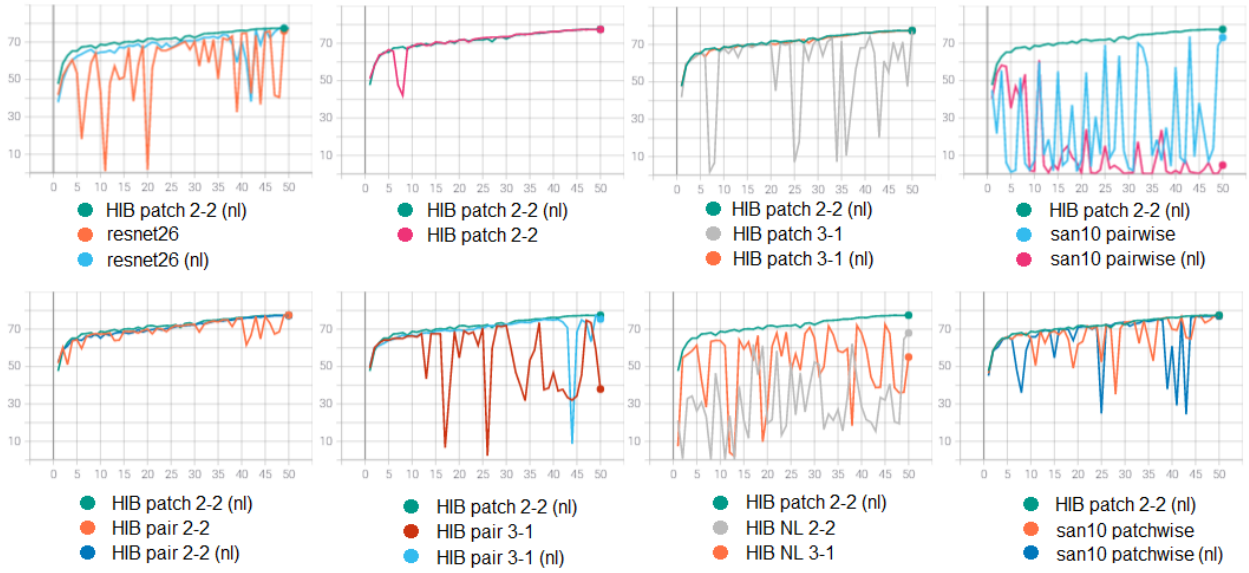

Figura 4.4: Comportamiento en el conjunto de validación para modelos de clasificación con dataset Mini-QuickDraw en su versión "pre-activación". En cada gráfico se repite como referencia visual la variante HIB patch 2-2 (nl). El eje horizontal denota la cantidad de épocas, y el vertical el porcentaje alcanzado para top1.

4.3. SBIR unimodal mini QuickDraw

En la Tabla 4.3 mostramos el desempeño de los modelos para el esquema de recuperación SBIR unimodal. Guiándonos principalmente por el mAP, observamos que los modelos con presencia atencional desempeñan mejor, exceptuando la variantes SAN10 pairwise, su versión "(nl)", y las variantes que con etapas completas basadas en bloques non-local. Incluso modelos con presencia atencional reducida, como HIB patch 3-1 y HIB pair 3-1, presentan resultados consistentemente mejores que ResNet26. La variante con mejor rendimiento es HIB patch 2-2.

	Setup (1) PostAct			**Setup (2) PreAct**		
Model	**mAP**	**mRR**	**P@10**	**mAP**	**mRR**	**P@10**
Resnet26	43.4	87.6	79.3	42.8	87.1	79.3
Resnet26 (nl)	44.3	88.0	79.7	44.7	88.2	79.9
SAN10 pairwise	45.6	87.1	79.3	42.4	86.4	77.8
San10 pairwise (nl)	42.1	86.3	77.3	39.6	84.1	74.5
SAN10 patchwise	45.3	88.1	79.6	44.6	88.1	79.5
SAN10 patchwise (nl)	45.2	87.1	79.5	45.0	87.6	79.3
HIB pair 2-2	45.1	87.4	79.2	44.7	88.0	79.8
HIB pair 2-2 (nl)	45.0	87.6	79.4	44.7	87.8	79.4
HIB patch 2-2	**45.7**	87.5	79.3	44.4	87.8	79.2
HIB patch 2-2 (nl)	44.6	87.2	79.4	44.7	87.6	79.5
HIB pair 3-1	44.8	87.7	79.3	**44.8**	87.8	79.5
HIB pair 3-1 (nl)	44.3	87.3	78.9	44.3	87.2	79.1
HIB patch 3-1	44.3	87.6	79.3	**44.8**	87.2	79.8
HIB patch 3-1 (nl)	45.0	87.9	79.5	44.6	87.2	79.5
Pure NL	9.1	40.7	24.0			
HIB NL 2-2	37.3	84.2	73.6	36.8	83.9	73.4
HIB NL 3-1	44.3	87.7	79.5	40.8	85.7	76.3

Tabla 4.3: Métricas de mAP, mRR y P@10 para SBIR unimodal sobre dataset Mini-QuickDraw, en porcentajes.

4.4. SBIR bimodal

En la Tabla 4.4 presentamos los resultados para el entrenamiento SBIR bimodal, en cada etapa. Por conveniencia, hemos añadido como referencia valores obtenidos en algunos trabajos adicionales.

El primero de estos trabajos es el de Zhao et al. [7]. Como ya hemos mencionado antes, de este proviene la formulación de SAN10 en sus dos variantes (pairwise y patchwise), pero también lo hemos utilizado como base para la metodología de comparación (ResNet26 como línea base). Por lo mismo, provee la comparación más cercana a nuestro trabajo en cuanto a ajustes experimentales, cantidad de parámetros entrenables y construcción de los modelos. Este trabajo no contiene experimentos comparables para la tarea de recuperación, pero sí incluye experimentos para clasificación con ImageNet, por lo que comparamos con nuestra fase de preentrenamiento. A pesar de ser la comparación más cercana, vale notar que en ese trabajo el entrenamiento se realizó por el doble de épocas, aspecto que influye en el mayor rendimiento alcanzado. A pesar de esto, vemos que el modelo híbrido supera a SAN10 pairwise original.

El segundo trabajo con el que comparamos es el de Bui et al. [27], del cual tomamos la metodología de entrenamiento en varias etapas. Mostramos los valores alcanzados para la variante que utiliza InceptionV1 [15] (también llamada GoogLeNet), que representa los mejores resultados alcanzados en ese paper. El trabajo no reporta métricas para las fases de preentrenamiento y entrenamiento de las ramas por separado (fase 1), sin embargo, utilizamos como referencia para el preentrenamiento los valores alcanzados por la implementación de PyTorch entrenada con ImageNet. Nuestros modelos son decididamente superiores a InceptionV1 en clasificación. Sin embargo, a pesar de utilizar una metodología de entrenamiento aproximadamente equivalente, los resultados para recuperación de nuestros experimentos son muy inferiores, y están lejos de ser competitivos.

Un tercer trabajo con el que comparamos corresponde al de Fuentes [32] en su trabajo de título. Este obtiene resultados mucho mayores que los nuestros usando aproximadamente la misma metodología de entrenamiento que Bui et al. Vale mencionar que este utiliza modelos de capacidad mucho mayor —más del doble de parámetros entrenables que los nuestros—. A pesar de esto, lo incluimos por utilizar SE ResNext, una arquitectura basada en ResNet más similar a la nuestra que el trabajo de Bui et al. Este trabajo tampoco ofrece métricas para las primeras fases de entrenamiento.

Por otra parte, al comparar nuestras variantes entre ellas, vemos que los resultados para la variante híbrida son decididamente superiores para el pre-entrenamiento y para las etapas 2-3 del entrenamiento (*contrastive* y *triplet loss*). En particular, el mAP para la variante híbrida es 5 puntos mayor que la variante convolucional pura.

En la Figura 4.5 presentamos la curva de entrenamiento para las distintas etapas del entrenamiento SBIR bimodal. Es importante notar que el mAP de validación expuesto en esta figura fue calculado sobre Flickr25k durante el entrenamiento, solo para estudiar su comportamiento y seleccionar la mejor época. Luego, sobre esta época calculamos mAP sobre Flickr15k, presentado en la Tabla 4.4, que representa el resultado final.

	Pre-train	Stage 1		Stage 2-3		
Model	**top1 ImageNet**	**top1 sketch**	**top1 photo**	**mAP**	**mRR**	**P@10**
ResNet26	72.02	79.30	88.84	11.0	24.1	16.6
HIB patch 2-2 (nl)	75.49	79.6	89.74	**16.3**	36.3	25.5
Original SAN10 pairwise	74.9					
Original SAN10 patchwise	77.1					
SANet paper ResNet26	73.6					
Tu Bui [27] InceptionV1	69.7*			41.1		
Fuentes [32] SE ResNext50				55.3		

Tabla 4.4: Resultados para validación SBIR bimodal, separado en las distintas fases del entrenamiento (mejor época en cada uno). Hemos añadido como referencia los resultados para clasificación top1 obtenidos en [7] para ResNet26, SAN10 pairwise y SAN10 patchwise (doble de épocas); y los resultados de recuperación mAP alcanzados por Bui et al. con InceptionV1 [27] y por Fuentes con SE ResNext50 [32]. El valor para top1 incluido para InceptionV1 corresponde a la implementación de PyTorch entrenada con ImageNet, debido a que [27] no reporta métricas para esta fase.

Podemos observar que a partir de la fase 3 (época 35 en la tercera fila) el error aumenta significativamente y el mAP decae. Esto nos indica que el valor mAP indicado en la Tabla 4.4 proviene exclusivamente de las últimas épocas de la fase 2.

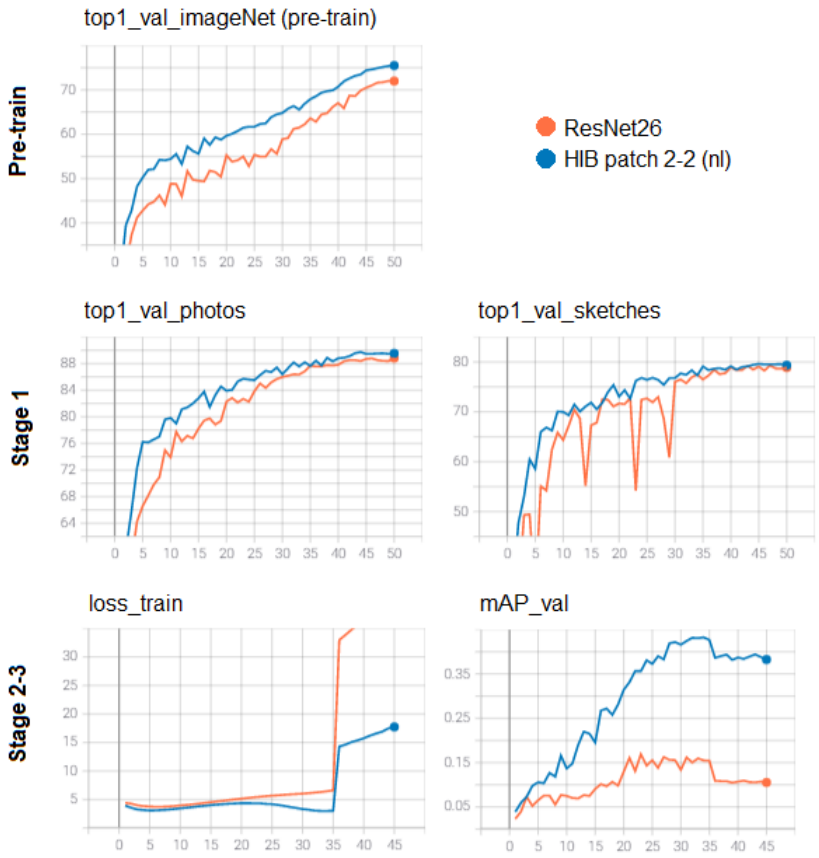

Figura 4.5: Curvas de entrenamiento para las distintas etapas del entrenamiento SBIR bimodal. La primera fila corresponde al pre-entrenamiento con ImageNet (50 épocas). La segunda fila corresponde a la fase 1, en que entrenamos cada rama por separado (50 épocas cada una), con los pesos pre-entrenados de ImageNet. La tercera fila corresponde a las fases 2 (*contrastive loss*, 35 épocas) y 3 (*triplet loss*, 10 épocas). El eje horizontal corresponde a la cantidad de épocas. El eje vertical corresponde al valor de la métrica.

Capítulo 5

Discusión

5.1. Clasificación

Antes de discutir y comparar nuestro trabajo con otros, comenzaremos analizando y comparando el rendimiento de nuestros propios modelos:

Una primera observación importante corresponde a notar la variabilidad en los resultados obtenidos. Si bien se identifican algunas tendencias tentativas, ninguna de estas parece ser simultáneamente consistente con todas las configuraciones (pre-act vs post-act), datasets, y modelos. En TU-Berlin, ambas configuraciones tienen una estabilidad similar, pero no así en Mini-QuickDraw, en que la configuración pre-act presenta una estabilidad marcadamente diferente, a pesar de que las arquitecturas probadas en ambos datasets son las mismas. Así mismo, podríamos vernos tentados a relacionar definitivamente los bloques SANet pairwise con una menor estabilidad, y sin embargo, modelos como HIB pair 2-2 se salen de esta tendencia. Lo mismo ocurre con el rendimiento *top1* de los modelos "(nl)", en los cuales los bloques non-local parecen complementar de manera positiva a algunas arquitecturas, pero a otras de manera negativa. Por su parte, los resultados para TU-Berlin favorecen el rendimiento de los modelos convolucionales puros por sobre los atencionales y con Mini QuickDraw sucede lo contrario. Esto podría sugerir que si el dataset es lo suficientemente grande, los modelos híbridos son una mejor opción tanto en estabilidad como rendimiento absoluto. Sin embargo, es posible que el volumen no sea la causa de la diferencia. Hay otros factores que también pueden influir, como la abstracción de las categorías de cada conjunto, y la abstracción y calidad de los dibujos mismos.

Vale la pena estresar la importancia de esta variabilidad, que en parte podría ser resultado de una falta de robustez y consistencia en la metodología experimental —por ejemplo, los cambios de hardware, la baja cantidad de ejecuciones para los experimentos finales, y la inhabilidad para replicar resultados conocidos en los experimentos SBIR bimodales—. Con lo anterior dicho, a pesar de la variabilidad, los modelos con convergencia más estable y mejor rendimiento incluyen, casi todos, algún grado de presencia atencional, y podemos elaborar algunas conjeturas generales para esta tendencia.

Primeramente, podemos abordar el caso del modelo atencional "Pure NL" de manera

aparte. Este modelo tuvo un rendimiento significativamente menor al resto, y por más de 50 puntos (ver Tabla 4.1), incluso con normalización y expresividad añadida sobre el diseño non-local original. Atribuimos este resultado a la ausencia de operaciones de agregación local (convolución u operaciones atencionales SANet). Si bien están fuera de nuestro esquema experimental formal, pruebas adicionales en que cambiamos solo la *stem* de esta arquitectura por la misma usada en ResNet26 (convoluciones de mas de 1x1) muestran mejorías del orden de 20 puntos. Estas pruebas no son robustas (y por eso no son formalmente incluidas en los resultados), pero, sumadas a los valores obtenidos por los modelos HIB NL 2-2 y HIB NL 3-1 —que integran porciones crecientes de etapas con operaciones convolucionales—, sugieren la importancia de los sesgos inductivos espaciales y el uso de operaciones locales como herramienta principal en el procesamiento de imágenes.

Para el caso general, podemos notar que para ambos datasets y configuraciones, encontramos variantes híbridas con buen rendimiento. Algunos modelos híbridos, incluso, presentan un rendimiento superior a todas las variantes puras, tanto convolucionales como atencionales. Para dar cuenta de este resultado, ofrecemos una hipótesis tentativa: Cada tipo de operación de agregación puede capturar una variedad de posibles relaciones o patrones. Si este conjunto de posibles relaciones es menor, entonces la operación es menos potente, lo que se traduce en un espacio de representación de menor tamaño, o menor 'riqueza' o 'diversidad' representacional. Sin embargo, debido a la menor cantidad de posibilidades, puede resultar más sencillo ajustarse a este espacio de representación más pequeño. Dicho de otra manera, es más fácil aprender si la 'profundidad' del conocimiento es menor. A partir de esto, especulamos que la combinación de diferentes tipos de agregación ayuda a definir un tamaño y riqueza adecuado para el espacio de representación de cada sección de la arquitectura, optimizando el aprendizaje de esa sección, y simultáneamente aumentando selectivamente la riqueza 'total' del espacio de representación final.

Comparación con paper SANet. Si bien fueron parte del esquema de entrenamiento de recuperación, resulta conveniente incluir la fase de preentrenamiento con ImageNet dentro del análisis de clasificación.

Tal y como mencionamos en la sección de resultados, los experimentos en el trabajo de Zhao et al. con SAN10 proveen la comparación más cercana a nuestros modelos de clasificación para el preentrenamiento con ImageNet, tanto en cuanto a dataset, como a ajustes experimentales y cantidad de parámetros entrenables. A pesar de esto, nuestros modelos fueron entrenados por la mitad de épocas, lo que constituye una diferencia relevante. Normalmente, para este tipo de comparaciones buscamos entrenar los modelos por la misma cantidad de épocas, o bien entrenarlos hasta que el rendimiento converja. Esto es decir, hasta un punto en que el entrenamiento adicional no ofrece aumentos significativos en el rendimiento. En este caso, no se cumple ninguna de las dos condiciones. Esta es una inconveniencia que surge de limitaciones de tiempo y hardware. Sin embargo, podemos ofrecer algunas observaciones cualitativas.

Podemos partir observando que nuestra implementación de ResNet26 tiene un rendimiento menor al de Zhao et al., lo que corresponde al resultado esperado considerando el tiempo de entrenamiento. Sin embargo, bajo esas mismas condiciones nuestro modelo híbrido supera a SAN10 pairwise original.

Por su parte, la comparación con SAN10 patchwise es menos clara. Podríamos vernos tentados a proyectar el rendimiento de nuestro modelo híbrido usando la diferencia entre ambos modelos ResNet26 como referencia. Esta es de aproximadamente 1,5 puntos. Si proyectamos el rendimiento de nuestro modelo híbrido por esta cantidad, su rendimiento es comparable a SAN10 patchwise original. Sin embargo, en necesario estresar que esta proyección no tiene ninguna base rigurosa, más allá de suponer, razonablemente, que el rendimiento de nuestro modelo híbrido aumentaría en alguna cantidad no despreciable con el doble de entrenamiento. En cambio, la conclusión cualitativamente relevante es notar que nuestro modelo posiblemente se aproxima al modelo atencional puro, con un diseño híbrido menos costoso de entrenar y usar en la práctica, tanto en tiempo absoluto (ver Tabla 3.1), como en memoria.

Por su parte, la implementación de InceptionV1 referida en la Tabla 4.4 solo provee un valor de referencia. No es un modelo competitivo, ni popular en comparación a ResNet. En cambio, lo proveemos solo para contextualizar los resultados de recuperación, que analizaremos más adelante.

5.2. SBIR unimodal

La recuperación unimodal realizada tiene como objetivo probar los modelos entrenados para Mini-QuickDraw sobre un conjunto de clases no vistas en el entrenamiento. Los resultados siguen variaciones similares a aquellas observadas para los resultados de clasificación con Mini-QuickDraw. Esto es decir, los resultados altos para clasificación también son altos para esta recuperación. Esta consistencia sugiere que sugiere la métrica *top1* puede ser un buen indicador de la capacidad de predicción en SBIR bimodal. A partir de estos resultados, y aquellos de clasificación, se seleccionó la variante HIB patch 2-2 (nl) para las pruebas SBIR bimodal por considerarla un buen representante de las arquitecturas híbridas que deseamos explorar y por mantener un rendimiento y comportamiento comparativamente consistente en todas las pruebas.

5.3. SBIR bimodal

En la exploración de ajustes y esquemas experimentales para SBIR bimodal, encontramos dificultades persistentes para obtener resultados comparables a otros de la literatura, en particular aquellos del trabajo original en que basamos la metodología, de Bui et al. [27], y otros similares que también utilizan modelos basados en ResNet, de Fuentes [32] (aunque con más del doble de parámetros entrenables que los nuestros).

Los resultados de la Tabla 4.4 muestran como nuestros resultados de recuperación son mucho menores a los de Bui et al., incluso a pesar de ver mejores resultados en el preentrenamiento, y utilizar como base modelos con mejor rendimiento nominal y mayor cantidad de parámetros entrenables. Por otra parte, incluso con una cuenta de parámetros entrenables mucho mayor, el uso exitoso de arquitecturas basadas en ResNet en [32] sugiere la viabilidad que debería tener nuestro ResNet26 para alcanzar mejores resultados, lo que nos hace

especular sobre posibles problemas o variables no controladas en nuestra implementación.

Entre los ajustes que variamos durante fases exploratorias están: la tasa de aprendizaje, optimizador, capacidad total (probamos con ResNet50), mayores tiempos de entrenamiento, diferentes ponderaciones para cada tipo de error en las fases 2 y 3, otros datasets, tamaño de *embedding* de recuperación, formas de *augmentation* y preprocesamiento. Adicionalmente, probamos otra base de código en TensorFlow, en la cual, a pesar de algunas mejorías, enfrentamos dificultades similares. Bajo esta exploración de ajustes y variaciones, nuestros resultados variarion entre el 10 % y 30 % de mAP. Aunque resultados cercanos al 30 % de mAP pueden parecer razonables, los resultados en [27] y [32] —que utilizan la misma metodología base— alcanzan valores sobre el 40 % y 50 %, respectivamente.

Por lo anterior, optamos por simplificar el esquema experimental, descartar la base de código en TensorFlow, y concentrarnos en la comparación convolucional-atencional, a pesar de que los valores absolutos de mAP en ese esquema simplificado fueron bastante menores (hasta 16 %).

Si bien no hemos logrado identificar de manera definitiva la causa por la cual nuestro modelo no pudo alcanzar los rendimientos absolutos esperados, hay una variedad de factores de interés que hemos observado:

- Incluso al utilizar un subconjunto del propio dataset de entrenamiento para testear, el modelo muestra dificultades importantes para ajustarse a los datos. Estas dificultades parecen reducirse significativamente al eliminar las técnicas de *augmentation*, pero esta aparente mejoría no se extiende para nuestro conjunto de test formal (Flickr15k), lo que no es una sorpresa, considerando que estas técnicas ayudan a la generalización, y sin ellas nuestros conjuntos de datos son poco representativos de la variabilidad real que se espera que enfrenten los modelos.
- Recurrentemente observamos que el error y las métricas de validación mejoran inicialmente, y luego de una cierta cantidad de tiempo esta mejoría se detiene, o derechamente se revierte. Esto es cierto tanto si utilizamos solo *contrastive loss* o *triplet loss*. Ya que las métricas en el conjunto de entrenamiento siguen una tendencia similar, no es claro si esto corresponde a *overfitting*, sin embargo, especulamos que las componentes del error provenientes de *softmax* (clasificadores) en las fases 2 y 3 podrían generar algún tipo de sobre-ajuste o tener una influencia mayor en el aprendizaje, en comparación a las pérdidas contrastivas y por tripletas, llevando los *embeddings* a un espacio de representación similar al de clasificación. Si bien observamos que las métricas de clasificación aumentan mucho más rápido que las de recuperación, es difícil confirmar esta hipótesis. Con esto dicho, variaciones en las ponderaciones de estos errores, tanto fijas como dinámicas —en función de la época— muestran algunas mejorías.
- Al examinar el mAP por cada clase y los resultados visuales, notamos una gran variabilidad en la recuperación. Para algunas clases, la recuperación es consistentemente razonable, e incluso si no es correcta, se puede apreciar algún grado de relación semántica entre la consulta y las imágenes recuperadas. Para otras clases, sin embargo, la recuperación parece guardar poca o ninguna relación semántica entre el dibujo y la foto. Más aún, en algunos casos, la clase de las fotos recuperadas incorrectamente es

consistente. Esto es decir que, para algunos dibujos de consulta, el modelo recupera consistentemente fotos de una clase particular, sin ninguna relación semántica aparente con el dibujo. Esto parece ocurrir de manera independiente al grado de abstracción del dibujo. Esta 'confusión' de clases sugiere errores lógicos en la implementación. Por ejemplo, la generación de pares positivos/negativos incorrectos, o el mapeo inconsistente de clases entre ambos datasets dibujo/foto —o sea, que el mismo índice corresponda a clases diferentes en ambos clasificadores—. Sin embargo, no hemos encontrado errores concretos de la implementación en estos ámbitos.

- Otra explicación para la variabilidad entre clases consiste en el aprendizaje no uniforme de recuperación para diferentes categorías. Con esto nos referimos a que la optimización local de la solución lleve al modelo a reducir el error de manera selectiva, para algunas clases 'fáciles', ignorando las clases 'difíciles'. Probamos algunas variantes de la función de pérdida constrastiva y por tripletas, seleccionando porciones de pares negativos 'difíciles' en cada batch (*hard negative mining*), sin aumentos importantes en el rendimiento absoluto. Sin embargo, estas pruebas no fueron exhaustivas.

A pesar de los problemas encontrados para alcanzar resultados competitivos, la comparación entre ResNet26 y la variante híbrida confirma la aparente superioridad de los modelos híbridos. En conjunto con las observaciones discutidas para Clasificación, proponemos que estos experimentos comprueban el potencial de las redes híbridas: alcanzando rendimientos iguales o superiores en algunas de sus variantes, utilizando menos parámetros totales que sus contra partes convolucionales, y tiempos de entrenamiento e inferencia solo ligeramente mayores, pero decididamente menores a las variantes atencionales puras.

Capítulo 6

Conclusión

En este trabajo, exploramos la utilidad de distintas formulaciones atencionales en modelos híbridos. Para esto utilizamos bloques convolucionales en conjunto con operaciones atencionales locales y globales, y las combinamos en distintas proporciones en tareas de clasificación y recuperación de imágenes basada en dibujos.

Los resultados obtenidos sugieren fuertemente la viabilidad y potencial superioridad de los diseños híbridos en algunos casos, especialmente en tareas de clasificación, presentando maneras sencillas de dotar de potencia adicional a arquitecturas tradicionales, a bajo costo, y presentando formas interesantes para estudiar el efecto de distintos patrones de agregación. Así mismo, también nos permite estudiar sesgos inductivos estructurales de manera selectiva en distintas porciones de la red. En este respecto, reinstamos las hipótesis presentadas: (1) sobre el efecto —tanto de la atención como de la combinación de mecanismos de agregación— sobre la riqueza del espacio de representación; y (2) sobre los beneficios de la aplicación selectiva en distintas porciones de la red.

Finalmente, en vista de las dificultades mencionadas, y la variabilidad de los resultados obtenidos, creemos que son necesarios más estudios que controlen de manera exhaustiva las variadas configuraciones experimentales, e instamos a incluir otros mecanismos atencionales en las variantes propuestas. En particular, recomendamos las siguientes mejoras y adiciones:

- Utilizar un esquema experimental más uniforme y robusto, en que el esquema final escogido (después de explorar ajustes) se ejecute múltiples e igual cantidad de veces para cada experimento. También recomendamos un esquema simplificado que use solo pre-activación, o solo post-activación para todos los diseños.
- En el caso de los experimentos de clasificación ImageNet, entrenar por extensiones de tiempo mayores si es posible, que faciliten la comparación directa con otros resultados de la literatura.
- Ejecutar las pruebas de recuperación unimodal en los modelos entrenados con el conjunto TU-Berlin, para controlar por el efecto del conjunto de datos.
- Probar una mayor cantidad de variantes para SBIR bimodal.

- Explorar mejoras en el diseño de bloques, especialmente para el bloque *non-local*, para el cual, si bien formulamos modificaciones teóricamente fundadas, no exploramos ni comparamos exhaustivamente muchas variaciones en el diseño.
- Explorar y combinar otros mecanismos atencionales en el diseño híbrido, como puede ser el enfoque basado en bloques *transformer* de BotNet [3].
- Utilizar esquemas de entrenamiento que favorezcan la generalización y aplicación en sistemas reales, como esquemas no-supervisado o semi-supervisados.

Bibliografía

[1] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations*, 2021. 3, 23

[2] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Conference on Computer Vision and Pattern Recognition*, 2018. 3, 19, 22

[3] A. Srinivas, T.-Y. Lin, N. Parmar, J. Shlens, P. Abbeel, and A. Vaswani, "Bottleneck transformers for visual recognition," in *Conference on Computer Vision and Pattern Recognition*, 2021. 3, 23, 38, 62

[4] X. Wang, R. Girshick, A. Gupta, and K. He, "Non-local neural networks," in *Conference on Computer Vision and Pattern Recognition*, 2018. 3, 18, 19, 22, 26, 38, 39, 40

[5] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," in *International Conference on Machine Learning*, 2015. 3, 22

[6] P. Ramachandran, N. Parmar, A. Vaswani, I. Bello, A. Levskaya, and J. Shlens, "Standalone self-attention in vision models," in *Advances in Neural Information Processing Systems*, 2019. 3, 22

[7] H. Zhao, J. Jia, and V. Koltun, "Exploring self-attention for image recognition," in *Conference on Computer Vision and Pattern Recognition*, 2020. 3, 18, 19, 22, 23, 24, 25, 32, 33, 44, 45, 53, 54

[8] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning*, ser. ICML'15. JMLR.org, 2015, p. 448–456. 12, 20

[9] M. Alkhawlani, M. Elmogy, and H. El-Bakry, "Text-based, content-based, and semantic-based image retrievals: A survey," *International Journal of Computer and Information Technology*, vol. 4, pp. 58–66, Jan 2015. 13, 14, 15

[10] J. van Gemert, "Retrieving images as text," 2003. 13

[11] S. Saha, "A comprehensive guide to convolutional neural networks-the ELI5 way," Towards Data Science, Dec 2018, last accessed: Jul 2021. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 18

[12] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989. 18

[13] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," *Neural Information Processing Systems*, vol. 25, 01 2012. 18

[14] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv 1409.1556v6*, 2015. 18

[15] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9. 18, 53

[16] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778. 18, 19, 20, 21, 37

[17] G. Huang, Z. Liu, L. V. D. Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, jul 2017, pp. 2261–2269. 18, 19

[18] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. 19, 21, 22

[19] "Resnet explained," Papers with Code, last accessed: Jul 2021. [Online]. Available: https://paperswithcode.com/method/resnet#tasks 19

[20] "Multi-head self-attention in nlp," Oracle AI & Data Science Blog, last accessed: Aug 2021. [Online]. Available: https://blogs.oracle.com/ai-and-datascience/post/multi-head-self-attention-in-nlp 22

[21] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1724–1734. 22

[22] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proceedings of the 27th International Conference on Neural Information*

Processing Systems, ser. NIPS'14, vol. 2. Cambridge, MA, USA: MIT Press, 2014, p. 3104–3112. 22

[23] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2016. 22

[24] T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, Sep. 2015, pp. 1412–1421. [Online]. Available: https://aclanthology.org/D15-1166 22

[25] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018. 22

[26] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, J. Burstein, C. Doran, and T. Solorio, Eds. Association for Computational Linguistics, 2019, pp. 4171–4186. [Online]. Available: https://doi.org/10.18653/v1/n19-1423 22

[27] T. Bui, L. Ribeiro, M. Ponti, and J. Collomosse, "Sketching out the details: Sketch-based image retrieval using convolutional neural networks with multi-stage regression," *Computers & Graphics*, vol. 71, pp. 77–87, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0097849317302194 29, 32, 33, 42, 43, 53, 54, 58, 59

[28] "Quick, draw! the data," Google LLC, last accessed: Aug 2021. [Online]. Available: https://quickdraw.withgoogle.com/data 32

[29] M. Eitz, J. Hays, and M. Alexa, "How do humans sketch objects?" *ACM Trans. Graph. (Proc. SIGGRAPH)*, vol. 31, no. 4, pp. 44:1–44:10, 2012. 32

[30] R. Hu and J. Collomosse, "A performance evaluation of gradient field hog descriptor for sketch based image retrieval," *Computer Vision and Image Understanding*, vol. 117, no. 7, pp. 790–806, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1077314213000349 32

[31] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255. 32

[32] "Recuperación de imágenes basada en dibujos mediante redes convolucionales," 2020, B.S thesis, FCFM, Universidad de Chile. 33, 43, 53, 54, 58, 59

[33] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham: Springer International Publishing, 2016, pp. 630–645. 34, 41

[34] "Resnetv1.5 for pytorch," NVIDIA NGC, last accessed: Mar 2022. [Online]. Available: https://catalog.ngc.nvidia.com/orgs/nvidia/resources/resnet_50_v1_5_for_pytorch 37, 41

[35] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database," ATT Labs, last accessed: Aug 2021. [Online]. Available: http://yann.lecun.com/exdb/mnist/ 44