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

RECUPERACIÓN DE IMÁGENES BASADA EN DIBUJOS CON COLOR MEDIANTE APRENDIZAJE AUTO-SUPERVISADO

MEMORIA PARA OPTAR AL TÍTULO
DE INGENIERO CIVIL ELÉCTRICO

MEMORIA PARA OPTAR AL TÍTULO
DE INGENIERO CIVIL EN COMPUTACIÓN

BRAULIO SALVADOR TORRES HENRIQUEZ

PROFESOR GUÍA:
JOSÉ SAAVEDRA RONDO

MIEMBROS DE LA COMISIÓN:
JUAN BARRIOS NÚÑEZ
FRANCISCO RIVERA SERRANO

SANTIAGO DE CHILE
2024

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO
Y AL TÍTULO DE INGENIERO CIVIL EN
COMPUTACIÓN
POR: BRAULIO SALVADOR TORRES HENRIQUEZ
FECHA: 2024
PROF GUÍA: JOSÉ SAAVEDRA RONDO

RECUPERACIÓN DE IMÁGENES BASADA EN DIBUJOS CON COLOR MEDIANTE APRENDIZAJE AUTO-SUPERVISADO

La recuperación de imágenes basada en dibujos es un problema en el contexto de búsqueda de imágenes y en el que hay un gran interés. Los dibujos son una forma natural de expresión para los seres humanos, mucho más que una descripción de una imagen. La irrupción en los últimos años de la tecnología de inteligencia artificial, ha permitido que varios problemas que antes se pensaba no eran posibles de resolver, ahora su solución se este convirtiendo en una realidad. En este contexto se encuentra la presente memoria.

La mayor parte de los trabajos hechos en este problema han sido con entrenamiento supervisado, lo que provoca problema de generalización de la red, haciendo que sea difícil para las redes discriminar dibujos que no hayan sido vistos en su entrenamiento. Por lo que en el presente trabajo se continuó con la investigación de entrenamientos auto-supervisados, esperando que las redes resultantes tuviesen mejor generalización que las obtenidas con entrenamientos supervisados.

Para esto sé probo con varias técnicas para mejorar la eficiencia del estado del arte, para encontrar cuál era el mejor camino para seguir con la investigación. Finalmente, se propuso un método que resultó ser el más prometedor, el filtro *Random liquify* que logró mejorar los resultados del problema de búsqueda de imágenes con color y sin color.

Por último, se concluye explicando por qué se obtuvieron los resultados observados y comentando sobre los futuros investigaciones que se podrían llevar a cabo en este problema.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

1.1. Motivación

La presente memoria se enfoca en el problema de recuperación de imágenes basadas en dibujos con color mediante aprendizaje auto-supervisado. La búsqueda basada en contenido ha ganado importancia debido a la posibilidad de realizar consultas sin perder expresividad. Un caso especial es cuando la consulta es un dibujo, ya que esta es una forma natural para expresar ideas o necesidades. Por ejemplo, si pensamos en una lámpara con cierta forma, una forma simple de expresarlo es a través de un dibujo.

El desarrollo explosivo de la inteligencia computacional que ha ocurrido en la última década ha provocado una revolución de tecnologías. La inteligencia computacional ha permitido que antiguos problemas computacionales, que resultaban muy complicados de resolver solo utilizando algoritmos creados por personas con un lenguaje de programación, por ejemplo, los problemas de identificar una especie de ave en una imagen, o lograr que un computador identificase de que se está hablando una línea de texto, han logrado un avance significativo en los últimos años.

Esto claramente ha provocado una revolución tecnológica, ya que de pronto las posibilidades de automatización de muchos procesos productivos se ha hecho una realidad y se ha abierto la puerta a servicios totalmente nuevos como la búsqueda por Internet asistida con inteligencia computacional. Es en este contexto en el cual la presente memoria se encuentra.

La búsqueda de imágenes mediante dibujos, de igual manera, es un problema que resulta difícil de resolver mediante el clásico desarrollo de algoritmos. Pero que la inteligencia computacional, y en particular *deep learning*, puede lograr solucionar. Por lo que el estudio de este problema ha florecido en los últimos años.

Los dibujos son una forma natural de comunicación y representan una modalidad conveniente, incluso más fácil que escribir textos, dada la masificación de dispositivos móviles. De hecho, recientemente la búsqueda por dibujo ha sido incorporada en buscadores de *eCommer-*

ce a través de las cuales pueden buscar productos, simplemente dibujando lo que desean. Sin embargo, los modelos actuales se basan en dibujos sin color, que no explota la creatividad de los consumidores. Más aún, no han aparecido modelos capaces de afrontar este problema de una manera eficiente. Una limitación es no contar con datos etiquetados, ya que la creación de conjuntos de datos etiquetados resulta lento y costoso.

Así, en este trabajo se plantea el desarrollo de modelos auto-supervisados, los cuales, no necesitan conjuntos de datos etiquetados para llevar a cabo el entrenamiento. Aumentando la cantidad de data útil para el problema y mejorar el uso práctico que este tiene.

1.2. Definición y relevancia del problema

En la presente memoria se pretende analizar métodos para mejorar el actual estado del arte en la búsqueda de imágenes mediante dibujos. Como existe una falta de datos etiquetados y además, se busca que las redes desarrolladas puedan generalizar sus resultados, que significa, identificar dibujos sin necesidad de que la clase a la que pertenezca este haya sido utilizada en el entrenamiento. Se va a continuar con el estudio de redes de entrenamiento auto-supervisado que se ha llevado a cabo en trabajos anteriores [1].

Como estas redes de entrenamiento auto-supervisadas no necesitan etiquetado de la data, hacen posible utilizar *datasets* disponibles más grandes sin la costosa actividad de etiquetar sus imágenes. Pero también poseen desventajas, esta forma de entrenamiento tiende a no crear redes tan eficaces como las entrenadas con métodos supervisados y necesitan ser inicializados con una red que ya haya sido entrenada de alguna manera. Generalmente, se utilizan redes que han sido entrenadas con el conjunto de datos *ImageNet*, el cual es uno de los conjuntos de datos de imágenes más grandes del mundo.

Cabe destacar que en este trabajo no se busca resolver el problema definitivamente, sino que se trata de una investigación para identificar cómo continuar con el estudio del problema, analizar qué métodos son los más prometedores y cuáles probablemente no podrán dar mejores resultados en un futuro.

Se aprovechará de la existencia de una gran cantidad de dibujo sin color, que expresan la forma de los objetos. Para este fin, trabajaremos en colaboración con Impresee Inc, empresa dedicada al rubro de motores de búsqueda para *eCommerce*.

1.3. Objetivos

1.3.1. Objetivo general

El objetivo es proponer métodos y estudiar su eficacia para mejorar los resultados de los modelos auto-supervisados en el problema de la recuperación de imágenes basada en dibujos con y sin color.

1.3.2. Objetivos específicos

1. Expandir o alterar los *datasets* existentes para poder adaptarlos al problema.
2. Encontrar una forma de agregar la información de color de los dibujos al entrenamiento.
3. Generar una estructura de redes convolucionales específicamente diseñada para resolver este problema, en el contexto de auto-supervisión.
4. Entrenar las redes con los *datasets* generados y poder comparar y analizar los resultados con el estado de arte en los problemas de búsqueda con y sin color.
5. Concluir sobre la eficacia de los métodos, por qué se obtuvieron los resultados y posibles trabajos futuros

Capítulo 2

Marco Teórico y Estado del Arte

En la actualidad, existe un gran avance en la investigación de búsqueda por dibujos sin color debido a la utilización de redes *deep*, en la siguiente figura se muestra un ejemplo de búsqueda con dibujo:

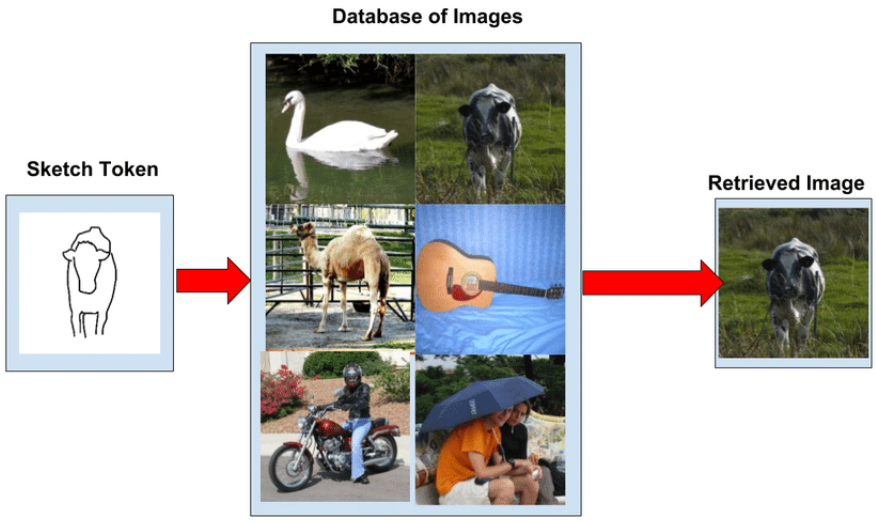


Figura 2.1: Consulta de imagen mediante un dibujo.

Pero esto tiene limitaciones, como el no utilizar la información extra que se podría extraer del color o textura, ya que esto puede dar más claridad sobre el objeto que se busca.

Otra forma en la que se realiza búsqueda por similitud es con texto. Se escribe una descripción de la imagen y un modelo previamente entrenado entrega la imagen más cercana a ella. Pero la búsqueda con dibujos presenta ventajas por sobre esta última porque un *sketch* permite representar de forma más precisa y eficiente una búsqueda que un texto, ya que posee forma, color, estilo y pose y cuando hay detalles muy específicos, las palabras pueden quedar cortas en su descripción. Por último, la popularidad y el uso de dispositivos táctiles facilita

el uso de este tipo de búsquedas.

Existen buscadores basados en redes convoluciones que resuelven bien este problema, pero están entrenados de forma supervisada, utilizando datos etiquetados con clases de objetos para su entrenamiento. Esto limita los casos donde se pueden utilizar, ya que solo tienen buen desempeño cuando se consulta por clases de objetos que estén en el *dataset* de entrenamiento y dificulta las consultas para datos más generales. Otro problema es que la mayoría de los *datasets* existentes no poseen color en los dibujos de las clases, porque solo tratan de resolver el problema de búsqueda con dibujos sin color, ignorando el potencial que este podría dar para la identificación de la imagen. Un ejemplo de estos *datasets* es *Quick draw!* [2], conjunto de datos que es generado mediante una página web creada por *Google*, allí los usuarios pueden hacer un dibujo de la imagen en un tiempo restringido de 15 segundos. Con esto, ya se han acumulado 50 millones de dibujos, lo que lo hace el *dataset* más grande de este tipo. Una página similar, pero que genere dibujos de color, ayudaría de gran manera al estudio de este proyecto. Otro problema es que varios dibujos pueden representar un mismo objeto, ya que dependen de la persona que los hizo y de la interpretación que se les dé. Un sistema eficiente debería ser robusto a estas variabilidades. Por último, el etiquetado es un proceso caro y lento, por lo que encontrar una forma de evitarlo o hacerlo más fácil sería un avance importante.

Debido a esto, resulta interesante explorar métodos de entrenamiento auto-supervisados, evaluando la posibilidad de utilizar dibujos sin etiquetas. Esto ayudaría a aumentar la cantidad posible de datos que se pueden utilizar en el entrenamiento y también aumentar los casos de uso de la tecnología, logrando que se utilice en consultas independientes de las clases con las que fue entrenada.

Un uso actual se puede ver en el *eCommerce*, donde se busca ofrecer al usuario realizar búsqueda de productos con dibujos. La empresa *Impresee Inc.* está investigando esta tecnología de manera de hacerla más eficiente y ofrecerla a sus clientes del *eCommerce*.

El trabajo realizado por Aníbal Fuentes sobre consultas de imagen mediante dibujos con color [3] corresponde al estado de arte sobre este tema, utilizando redes siamesas convolucionales, realizando pruebas con los *datasets* más famosos y creando *datasets* nuevos, uno para el entrenamiento y otro para el testeo. Aunque este fue realizado con entrenamiento supervisado, da una buena base para comparar y continuar su trabajo. Otro trabajo, pertinente al tema, corresponde al de Javier Morales [1], donde evalúa la eficacia de métodos auto-supervisados y semi-supervisados para consultas de imágenes mediante dibujos sin color, que da una idea de como enfrentar el problema con este tipo de entrenamiento. Aunque todavía existe la dificultad de cómo agregar la información del color al aprendizaje y la falta de *datasets* pertinentes al problema.

2.1. Redes convolucionales

Dado que este tipo de redes son cruciales para el trabajo, se va a realizar una breve explicación.

Las redes convolucionales fueron desarrolladas en un principio para el procesamiento de imágenes, se concibió como una forma de permitirles a las redes neuronales la habilidad de aprender a hacer preprocesamiento de imágenes.

El caso clásico de desarrollo de una AI se observa en la figura 2.2, el investigador selecciona las características de la imagen que considere sean pertinentes y ayuden a la red, lo que sería el preprocesamiento, con las cuales se entrenaría a la red.

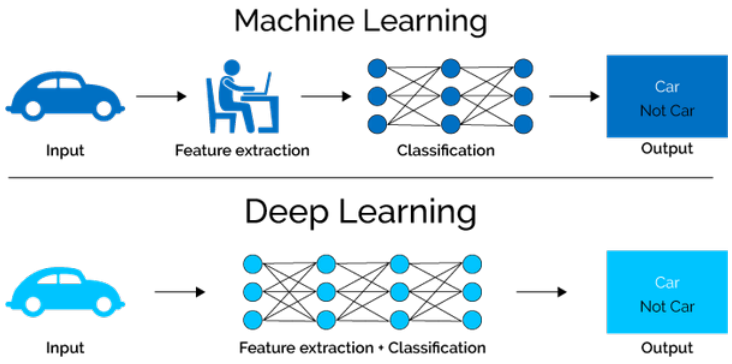


Figura 2.2: Diferencia en el desarrollo de AI, dando en cuenta que en deep learning no se realiza la extracción de características.

Pero con las redes convolucionales, la red puede aprender a extraer las características. Se logra utilizando la operación de convolución. Esta traslada un *kernel* a través de la imagen, multiplicando y sumando los valores, creando una nueva estructura de datos. Para estas redes, son los valores del *kernel* los que son entrenados. Un ejemplo de la operación se ve en 2.3

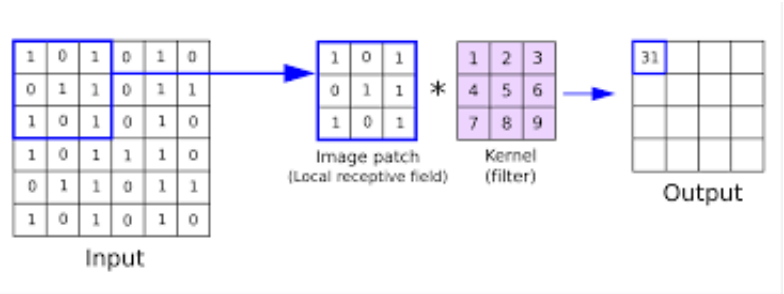


Figura 2.3: Operación de convolución.

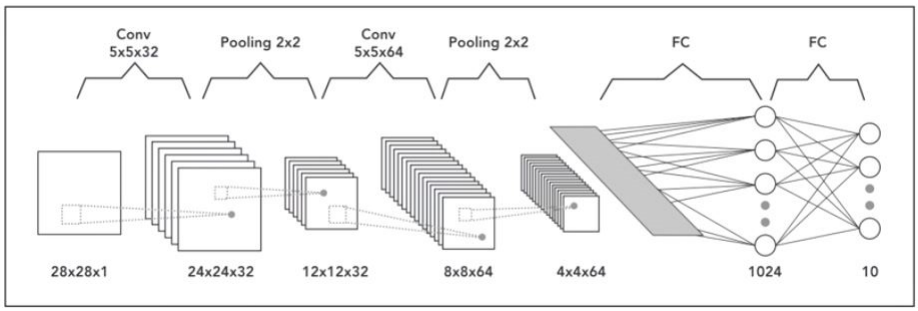


Figura 2.4: Operación de convolución en una red, por cada operación, se van agregando canales al resultado.

Cada vez que se realiza la convolución, se tienen tantos *kernels* como canales se quieran en la siguiente etapa (las imágenes comienzan con 3 canales, rojo, azul y verde), con lo que los canales van aumentando a través de la red y como el traslado del *kernel* le baja el tamaño a la imagen, al terminar el proceso se obtiene un vector unidimensional de características de la imagen, que generalmente se usa como *input* para una red neuronal, como se muestra en la figura 2.4.

Al igual que con las redes neuronales, estas redes deben minimizar la función de pérdida que se obtiene al evaluar los datos de los *dataset* de entrenamiento, la más común es la función de *cross entropy loss*:

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

Donde M, número de clases, $y_{o,c}$ indicador binario que dice si pertenece a la clase o no, y $p_{o,c}$ probabilidad predicha de que la observación o pertenezca a la clase c.

2.2. Tipos de entrenamiento

En los procesos de entrenamiento se utilizan grandes cantidades de datos para lograr que las redes logren aprender a discriminar los resultados. En este trabajo se va a hacer referencia a los tipos de entrenamiento supervisado y auto-supervisado.

2.2.1. Supervisado

El entrenamiento supervisado requiere que los conjuntos de datos tengan etiquetas o clases para poder identificar el tipo al que cada dato corresponde. Durante el entrenamiento la red aprende a conectar cierto tipo de data con una clase, después en las pruebas se utilizan datos pertenecientes a estas clases de entrenamiento para identificar la eficiencia de la red.

La desventaja de este entrenamiento es que cuando se intenta identificar objetos pertenecientes a clases diferentes a las utilizadas en el entrenamiento, la eficiencia baja. Por ejemplo, si se realiza un entrenamiento con las clases gato, barco y bicicleta y después se da como entrada una lámpara, la red no va a poder identificar el objeto, por lo que existe un problema de generalización.

En la figura 2.5 se ve un ejemplo de los pasos de este entrenamiento, donde un supervisor tiene que etiquetar los datos para que la red pueda aprender.

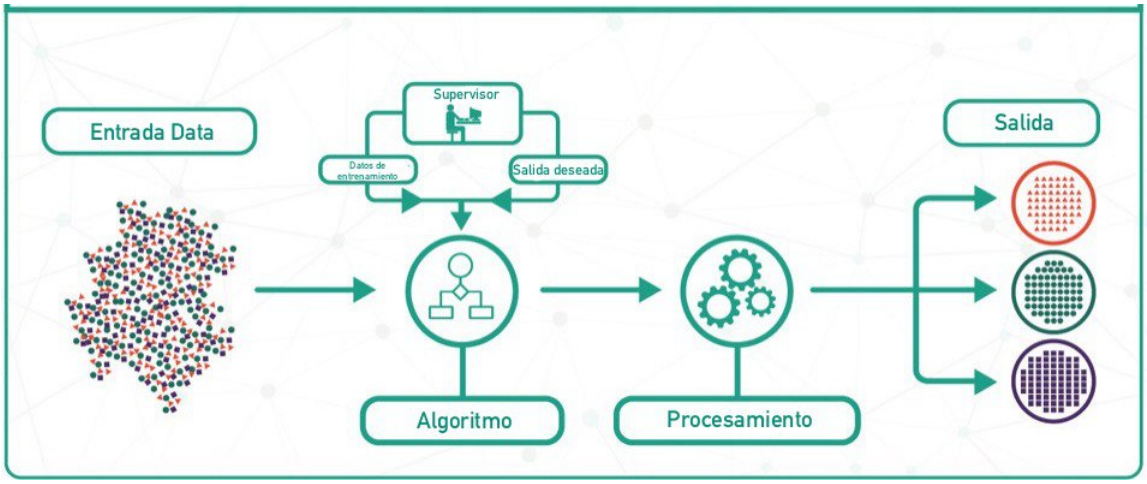


Figura 2.5: Cadena de pasos de un entrenamiento supervisado.

2.2.2. Auto-supervisado

El entrenamiento auto-supervisado no necesita clases para poder discriminar los datos de entrada, sino que por sí solo logra identificar los patrones en la data. Sin embargo, posee desventajas como la necesidad de inicializarse con pesos ya entrenados en otros conjuntos de

datos, y en general, no logra obtener la misma eficiencia que los resultados obtenidos con métodos supervisados. Pero sus ventajas son que, ya que no utiliza clases en el entrenamiento, logran generalizar de mejor manera los datos de entrada, potencialmente logrando identificar objetos de clases que no se utilizaron en el entrenamiento. También logran evitar el costoso proceso de etiquetado de los datos, lo que permite la utilización de conjuntos de datos más grandes. Debido a estas dos ventajas, el estudio de forma de entrenamiento ha florecido en los últimos años.

En la figura 2.6 se muestra un ejemplo del proceso de aprendizaje auto-supervisado.

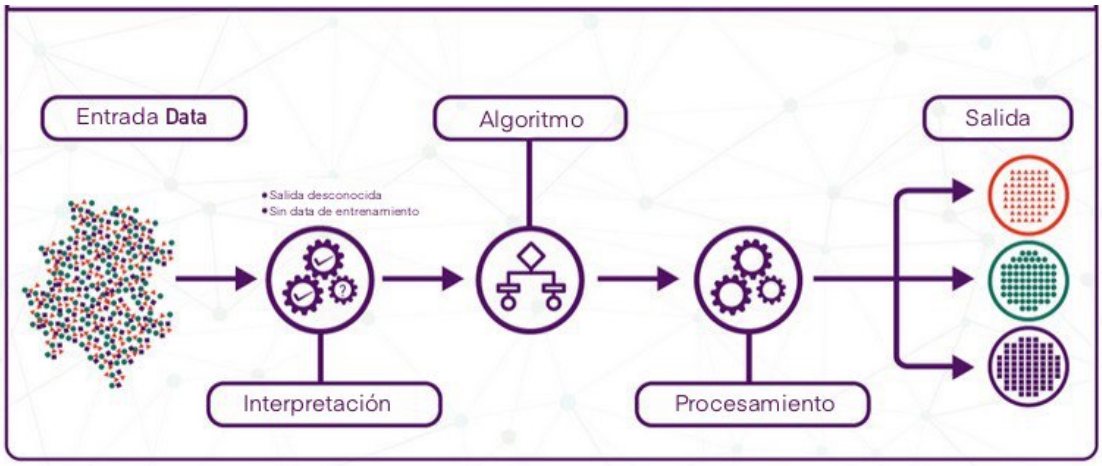


Figura 2.6: Cadena de pasos de un entrenamiento supervisado.

2.3. Modelos de representación de imágenes

Durante el presente trabajo se referencian dos modelos de colores, RGB (rojo, verde y azul) y HSV (Matiz, saturación y valor), los que se explicarán a continuación.

2.3.1. RGB

RGB corresponde a la composición de color de acuerdo a la intensidad de los 3 colores primarios, rojo, verde y azul. Este modelo es especialmente útil, ya que cada píxel de las pantallas de televisores o monitores poseen 3 luces led por cada píxel, donde cada una corresponde a un color primario. Debido a esto, es el más utilizado para mostrar imágenes y videos en dispositivos digitales.

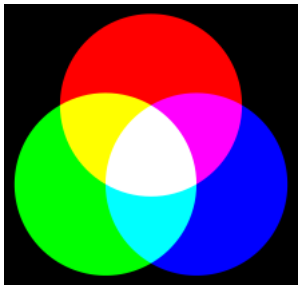


Figura 2.7: Modelo aditivo de colores rojo, verde, azul.

2.3.2. HSV

HSV corresponde a la composición de color de acuerdo al matiz, saturación y valor. Matiz corresponde al color de la imagen que va desde el púrpura al rojo. Saturación corresponde a la intensidad del color, con saturación cero, se pierde el color y se termina con una imagen gris. Valor define la intensidad del blanco de la imagen o diciéndolo de otra forma, qué tono de gris esta tiene.

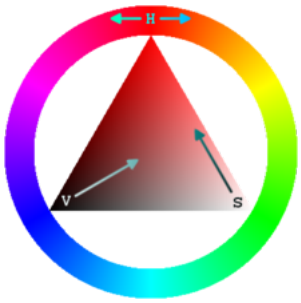


Figura 2.8: Diagrama del modelo HSV, el círculo define el color y el triángulo define la saturación y el valor.

La principal ventaja de HSV es que es más sencillo comprender como cada valor afecta a la imagen, a diferencia de RGB donde se necesita tener entrenamiento para comprender como afectan los cambios en los valores a los colores.

2.4. Redes

2.4.1. Resnet50

Resnet[4] es una abreviación para *Residual Network* o más específicamente, red neuronal residual. Lo que caracteriza a una red residual son las conexiones de identidad, las cuales toman el *input* y lo conectan directamente con el *output*.

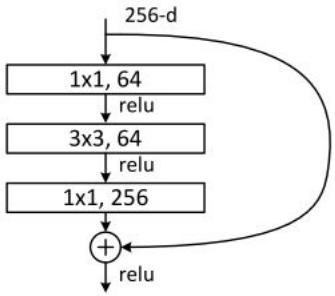


Figura 2.9: Conexión residual utilizada en la *Resnet50*.

La hipótesis detrás de esta conexión es darle la oportunidad a la red de utilizar o saltarse capas de convolución, mediante la utilización de la conexión de identidad. Esto porque tal vez diferentes problemas necesiten diferentes largos de redes convolucionales.

A la fecha, las redes residuales han demostrado ser inmensamente útiles en muchos problemas, superando a *Alexnet* en la competencia de *imageNet* en 2015.

La red en particular importante para este trabajo corresponde a la *ResNet50*, cuya arquitectura se puede observar en la figura 2.10

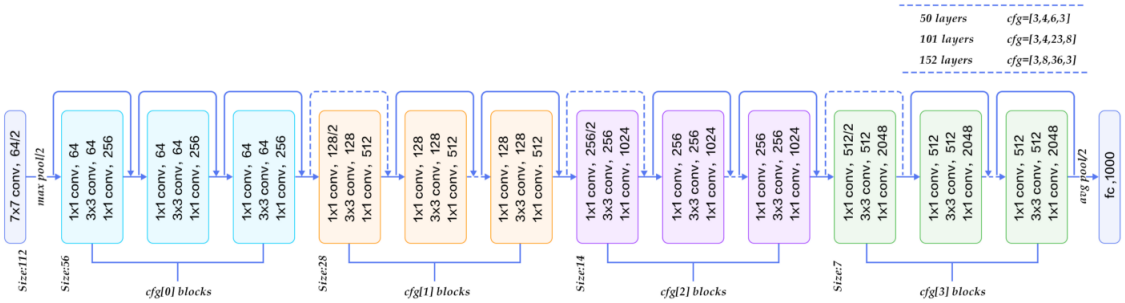


Figura 2.10: Arquitectura de la Resnet50.

Esta red tiene una profundidad de 50 capas convolucionales, de ahí su nombre, compuesto de 4 bloques principales y con aproximadamente 25 millones de parámetros entrenables.

2.5. Métricas

2.5.1. Mean Average Precision (map)

Esta es una métrica utilizada en el contexto de *object detectiony information retrieval* teniendo una definición distinta para cada método. En *information retrieval* es un valor entre 0 a 1 (mayor es mejor) que nos indica en promedio si las respuestas correctas se encuentran

más cercana a la primera posición, respecto al total de posibles respuestas. Este es igual a:

$$map = \frac{1}{|Q|} \sum_{i=1}^{|Q|} AP_i$$

Donde Q es la cantidad de consultas, por lo que corresponde al promedio de *average precision* de todas las consultas. El AP se calcula mediante la ecuación:

$$AP_i = \frac{1}{GTP_i} \sum_{j=1}^{M} \frac{1_{j=correct} * TP_{seen,j}}{j}$$

Esencialmente, es una métrica de que tan cercanas están las imágenes correctas al inicio de una lista ordenada de acuerdo a la similitud calculada. Primero se ordena todo el *dataset* por similitud con la imagen para recorrerlo; $1_{j=correst}$ es una indicatriz que vale 1 cuando la imagen es correcta y 0 cuando no, $TP_{seen,j}$ es la cantidad de imágenes correctas que se han visto hasta ese momento de la iteración, GTP_j es la cantidad de imágenes correctas para esa consulta y M es la cantidad de datos en el *dataset*. Entonces entre más se avance en la iteración, menos aportara a la métrica una imagen correcta.

2.5.2. Mean Reciprocal Rank (MRR)

El Mean Reciprocal Rank es una métrica de evaluación que toma valores entre 0 y 1 (1 es mejor) y nos indica en promedio para todas las consultas en qué posición se encuentra la primera respuesta correcta. La definición matemática es:

$$MRR = \frac{1}{Q} \sum_{i=1}^{|Q|} \frac{1}{rank_i}$$

Donde $|Q|$ es la cantidad de consultas, y $rank_i$ es la posición de la primera respuesta correcta para la consulta i.

2.6. Estado del Arte

Los trabajos más recientes realizados sobre recuperación de imágenes con dibujos y que se utilizarán como base para el proyecto corresponden a “Recuperación de imágenes basada en dibujos mediante redes convolucionales”[3] de Aníbal Fuentes Jara y “Evaluación de métodos auto-supervisados y semi-supervisados para la extracción de características visuales en el contexto de recuperación de imágenes basada en dibujos”[1] hecho por Javier Morales

Rodríguez. Se va a realizar una breve explicación de las arquitecturas y resultados obtenidos a continuación:

2.6.1. Recuperación de imágenes basada en dibujos mediante redes convolucionales.

En este trabajo se hace un estudio del desempeño de los mejores algoritmos y redes convolucionales para solucionar el problema de recuperación de imágenes mediante dibujos, donde estos podrían ser con o sin color. La principal diferencia entre los trabajos es que en este, los entrenamientos se realizaron de forma supervisada.

Para el caso de dibujos sin color, se implementaron 3 redes convolucionales de diferentes arquitecturas:

- Deep SBIR: que corresponde a entrenar una red ya pre-entrenada, como Alexnet, ResNet50, etc. pero utilizando un *dataset* pertinente al problema
- Siamese SBIR: esta arquitectura está formada por dos redes gemelas, donde una recibe el dibujo y la otra recibe la foto correspondiente al dibujo previamente procesado con un detector de bordes utilizando el algoritmo Canny[5]. La red se entrena mediante *constractive loss*, que considera que dos imágenes son similares si son de la misma clase y diferentes si pertenecen a clases diferentes.

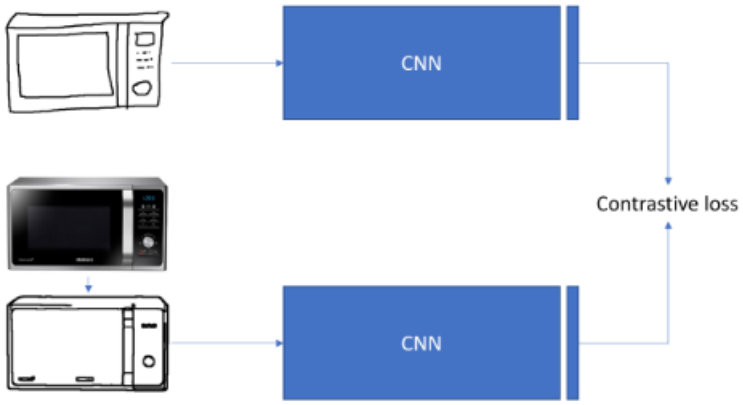


Figura 2.11: Diagrama del entrenamiento de la red siamesa.

- Multi Stage Regression SBIR: esta arquitectura plantea que para obtener mejores resultados, es necesario que existan dos redes independientes, una que extraiga características de las fotos y otra que extraiga de los dibujos, de modo que en las primeras capas se aprendan distintos filtros dependientes del dominio. Comparten los pesos de las últimas capas con el fin de extraer características comunes a ambos dominios. Además, se plantea un entrenamiento en múltiples etapas, donde se va aumentando la complejidad de la función de pérdida y del entrenamiento.

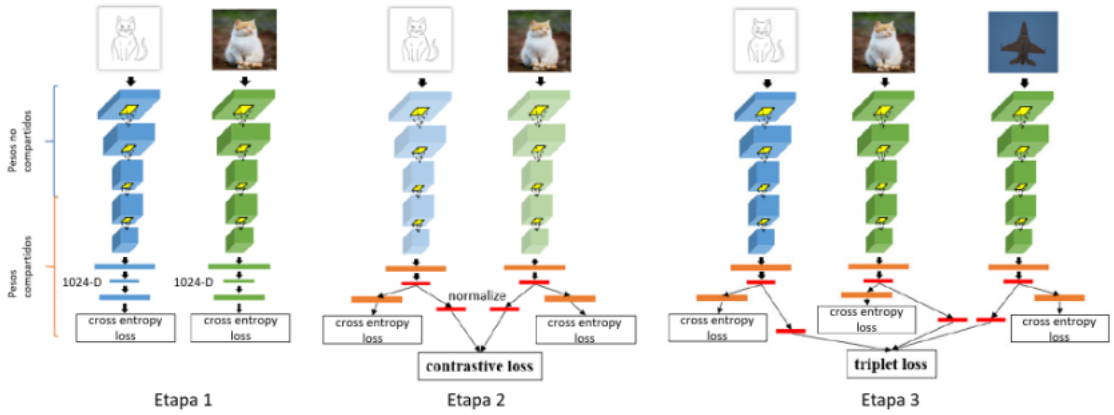


Figura 2.12: Diagrama de entrenamiento de Multi Stage Regression[3].

En la primera etapa se entrenan las redes de forma separada, cada una en su dominio correspondiente. En la segunda, se comparten los pesos de las últimas capas y se entrena de forma similar a una red siamesa. En la última etapa, se hace un *fine tuning* entrenando toda la red sin separar entre las primeras y últimas capas.

Cabe destacar que con esta última red, se obtuvieron los mejores resultados, logrando un map de 0.553 en el *dataset* de evaluación *Flickr 15K*

Para el caso de dibujos con color, la arquitectura con mejores resultados fue la siguiente:

- CSBIR con *Quadruplet Networks*: esté método es similar a Multi Stage Regression, pero incorpora la información de forma y de color al entrenamiento. El objetivo es que imágenes de colores y forma similar estén más cerca en el embebido resultante, es decir, el vector de característica que se obtenga debe cumplir con la siguiente hipótesis:

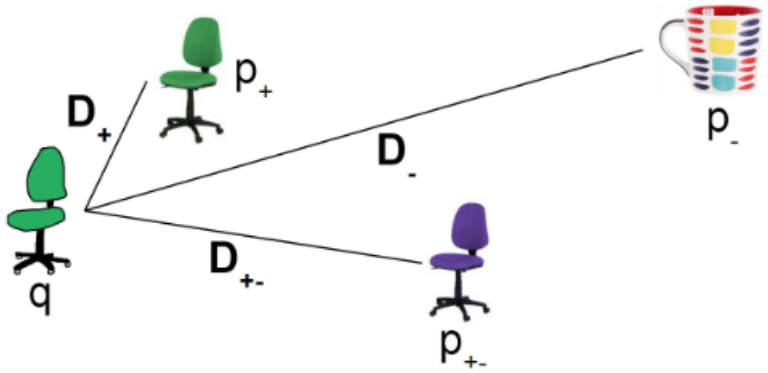


Figura 2.13: Diagrama de las distancias que se quieren obtener en el embebido de acuerdo a la imagen[3].

Como se ve, las imágenes de forma similar, pero con diferente color tienen una pequeña distancia y las imágenes sin parecido deben estar aún más alejados de la imagen de consulta. Para lograr esto se plantean dos funciones de perdida:

$$triplet\ loss_1 = max\{0, \mathrm{dist}(F(q), G(p_+)) + \alpha \cdot \lambda - \mathrm{dist}(F(q), G(p_{+-}))\}$$

$$triplet\ loss_2 = max\{0, \mathrm{dist}(F(q), G(p_{+-})) + (1-\alpha) \cdot \lambda - \mathrm{dist}(F(q), G(p_-))\}$$

Figura 2.14: Funciones de perdida.

Con esto el entrenamiento se hace de la siguiente forma:

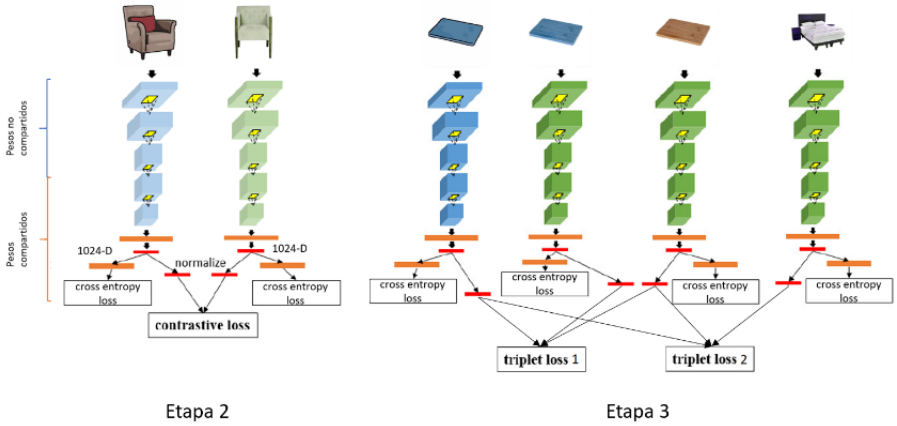


Figura 2.15: Etapas 2 y 3 de entrenamiento[3].

1. Entrenamiento de pesos no compartidos: las dos redes se entrenan de forma separada, utilizando la *cross entropy loss* como función de pérdida.
2. Entrenamiento de toda la red: el objetivo es aprender un embebido común, como función de pérdida se usa la suma de *cross entropy* más la *contrastive loss*.
3. Aprendizaje conjunto de color y forma: el objetivo de esta etapa es que la red sea capaz de aprender características de color y de forma, con el fin de alcanzar las desigualdades que se quieren obtener entre las imágenes. Para esto se entrena una *quadruplet network*, tal que la primera rama contenga los sketches (anchors), y las otras tres ramas contengan fotografías, donde la primera de estas ramas de fotografías tenga la foto de referencia para el sketch, la segunda tenga la misma foto, pero de un color diferente y la tercera rama tenga una foto de distinta clase. La función de perdida es la siguiente:

$$loss_{part3} = CE_1 + CE_2 + CE_3 + CE_4 + \beta * (tripletloss_1 + tripletloss_2)$$

Que corresponde a la suma del *cross entropy* para las 4 redes más un ponderador β multiplicado por la suma de los *triplet loss*

Con este método se llegó a un *Mean Reciprocal Rank* (MRR) de 0.352, el mejor de los métodos probados, para la evaluación se utilizó un *dataset* de 200 dibujos hechos a mano.

2.6.2. Evaluación de métodos auto-supervisados y semi-supervisados para la extracción de características visuales en el contexto de recuperación de imágenes basada en dibujos

En este trabajo se analizó el desempeño de redes con entrenamiento auto-supervisado y semi-supervisado con dibujos sin color, el trabajo actual se puede entender como una extensión de este al añadir color en el análisis y utilizar redes de clasificación más avanzadas que tal vez no existían cuando se realizó esta investigación. Solo se van a mencionar los modelos auto-supervisados que se utilizaron, ya que no se va a realizar un estudio sobre los semi-supervisados.

Variational Autoencoder (VAE)

Primero estas redes se basan en *Autoencoders* que corresponden a las redes auto-supervisadas más básicas, cuya arquitectura posee un encoder y un decoder. La imagen se codifica con la red neuronal obteniendo un vector, después con el decodificador se trata de obtener exactamente la misma imagen ingresada, con esto se espera crear un embebido que contenga toda la información de la imagen.

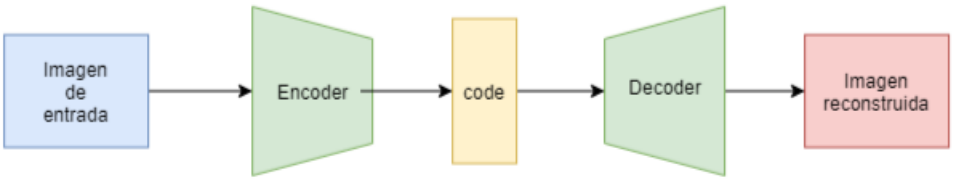


Figura 2.16: Arquitectura de un Autoencoder.

Un problema es que estas redes tienden a asignar un vector o punto en el espacio a cada imagen, por lo que no extraen realmente características. Los VAE[6] en vez de asignarle un vector, le asignan una distribución de probabilidad a la imagen.

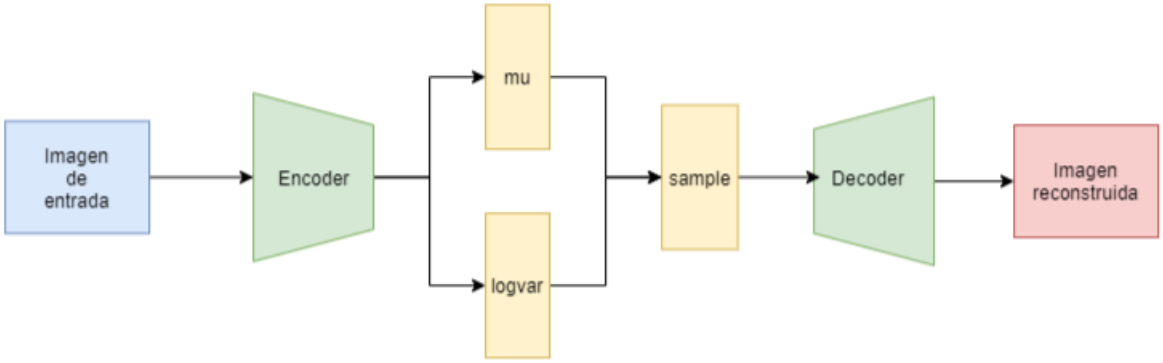


Figura 2.17: Arquitectura de un Variational Autoencoder.

Se descubrió que estas redes evitan el problema anterior de las Autoencoder. La diferencia es que al momento de reconstruir la red, no se utilizan los dos vectores *mu* y *logvar* sino que obtiene un punto aleatorio que se calcula con la siguiente distribución:

$$sample = mu + \mathcal{N}(0, 1) * logvar$$

Este modelo no logró obtener un resultado competitivo, obteniendo solo un *Mean Average Precision* (map) de 0.31 en comparación con el de 0.528 obtenidos por los modelos supervisados con *datasets* similares.

BYOL

Bootstrap Your Own Latent [7] Es una arquitectura diseñada para aprender a extraer características de imágenes de manera auto-supervisada. Es un modelo reciente, publicado a finales del 2020, que promete resultados competitivos en comparación a las técnicas supervisadas. BYOL es un modelo discriminativo, que a diferencia de los modelos generativos como VAE, solo busca obtener características que permitan determinar si dos elementos pertenecen a la misma clase. La idea principal detrás de BYOL es usar dos redes neuronales con una arquitectura casi idéntica, una online network y un target network, de forma que la primera busca predecir la salida de la segunda durante el entrenamiento, generando así su propia etiqueta.

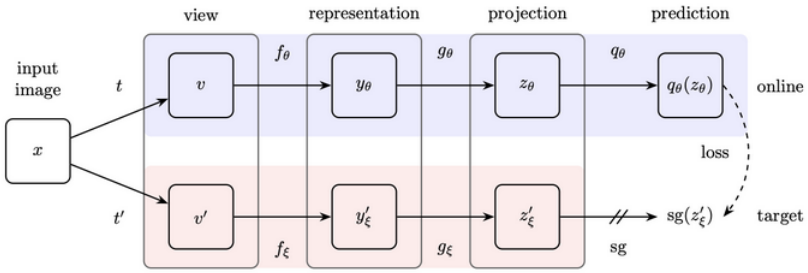


Figura 2.18: Arquitectura del modelo BYOL.

La idea detrás de BYOL es evitar la divergencia en el entrenamiento, la red online se entrena varias veces tratando de crear los embebido para las imágenes. Los valores de pesos del target network se calculan como el *exponential moving average* de la online, esta ecuación calcula el promedio de los pesos y le da más importancia a valores recientes, de esta manera la red target converge lentamente al punto óptimo.

Este modelo resultó muy exitoso, obteniendo un map de 0.590 comparable a los modelos supervisados, que obtuvieron 0.528 con los mismos *datasets*. En este trabajo se seguirá analizando este modelo.

Capítulo 3

Desarrollo de la solución

3.1. Formalización del problema

Sea un dataset de imágenes y un dibujo de un objeto cualquiera, el problema de *Sketch Based Image Retrieval*(SBIR) corresponde a encontrar las imágenes más parecidas al dibujo que se utiliza como entrada y *Color Sketch Based Image Retrieval*(CSBIR) corresponde al mismo proceso pero con la información adicional del color.

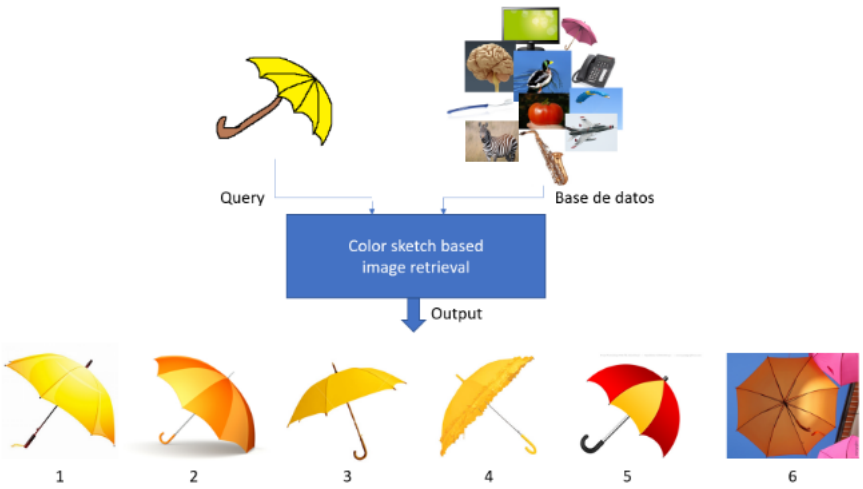


Figura 3.1: Ejemplo de *Image Retrieval* con color.

El problema formalmente corresponde a entrenar una red de forma auto-supervisada que resuelva el problema de *Image Retrieval* tratando que se obtengan los mejores resultados posibles y utilizando métricas que permitan comparar los resultados entre sí y con trabajos anteriores.

3.2. Conjunto de datos utilizados

3.2.1. Sketchy

Sketchy[8] corresponde al primer gran *dataset* de pares de dibujos e imágenes creados. En total, el conjunto de imágenes corresponden a 125 categorías con 75471 dibujos de 125000 objetos. Existe una gran correlación entre los dibujos y las imágenes, lo que significa que cada dibujo fue creado usando como referencia su correspondiente imagen, no se tiene un dibujo general de un zapato, por ejemplo.

Figura 3.2: Ejemplo de imágenes pertenecientes a *sketchy* y su correspondiente dibujo.

3.2.2. eCommerce

Este *dataset* fue propuesto por Pablo Torres y José Saavedra en el trabajo sobre representaciones compactas y efectivas para búsqueda de imágenes basada en dibujos[9]. Este conjunto de datos fue propuesto para lograr acercarse más a los casos de usos reales como en *eCommerce*.

El *dataset* está compuesto de un catálogo de 10600 fotos de productos relacionados con ecommerce distribuidas en 133 clases y con una gran diversidad de tamaño, color y objetos. También posee 665 dibujos hechos a mano, 5 por clase.

Figura 3.3: Ejemplos de imágenes y dibujos del *dataset* eCommerce.

3.2.3. Aumento de la data

Además de los datasets mencionados. También se realizó un aumento de la data con el dataset *eCommerce*, buscando que al aumentar la cantidad de datos que se utilizan en el input, mejoren los resultados de las redes entrenadas.

Para eCommerce, se crearon múltiples copias de una misma foto cambiando su color, lo que se logró primero expresando la foto mediante el formato HSV (*Hue*, *Saturation*, *Value*) y después se cambió el valor de *Hue*, esto provoca un cambio en el color de la imagen pero sin alterar su forma o brillo entre otras cosas. Generando 10 imágenes por cada foto. Así se consiguió crear aproximadamente 95000 fotos nuevas, con las que se realizaron varios experimentos.

3.2.4. Conjuntos de prueba

Para realizar la prueba de experimentos, se necesita un conjunto de datos lo más parecido al caso real de utilizar dibujos para buscar imágenes, por lo que estos datos están recolectados de dibujos hechos por personas, esto tiene la desventaja de que son conjuntos de datos pequeños.

Para probar el entrenamiento sin color, se utiliza el conjunto de datos de prueba de *eCommerce* del que se habla en el trabajo de Torres y Saavedra[9]. Para el entrenamiento de datos con color se utiliza el conjunto de pruebas desarrollado en el trabajo de Aníbal Fuentes [3].

Figura 3.4: Ejemplo de dibujo y su correspondiente imagen original utilizado en la prueba con color.

3.3. Generación de sketches sintéticos

Un problema existente para realizar el entrenamiento de las redes es la falta de *datasets* de entrenamiento pertinente al problema. Por lo que se exploraron las siguientes formas de generar dibujos artificialmente para el entrenamiento.

3.3.1. Pidinet

Una forma para crear un dibujo de forma artificial es utilizando PiDiNet [10], una red creada para la identificación de bordes en imágenes que trata de bajar la cantidad de ruido que típicamente se obtiene de estos procesos.

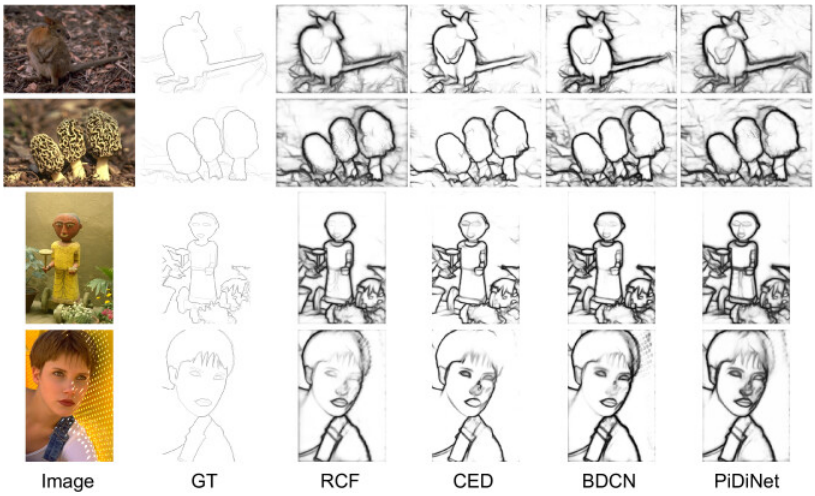


Figura 3.5: Ejemplo de identificación de bordes con varios métodos, en el último caso se puede ver los resultados de PiDiNet.

Gracias a esto se puede generar un trazado de un dibujo utilizando la imagen original. Una clara desventaja es que la generación mediante bordes genera dibujos demasiado "perfectos" o muy parecidos a la imagen original, lo que va en contra del objetivo de acercarse lo más posible a un dibujo real.

3.3.2. Cuantización del color

Los trazados no son suficientes para definir un dibujo por completo, estos también deberían contar con el color de la imagen original. Para agregar color a los dibujos artificiales, se optó por cuantizar los valores de colores originales. A continuación se va a explicar como funciona. Sea una imagen expresada mediante RGB (*Red*, *Green*, *Blue*), cada píxel tiene 3 valores con un rango de 0 a 255, para cuantizar estos valores se escogen ciertos valores del rango a los que se va a aproximar cada valor de los píxeles, por ejemplo si se divide en 3 partes el rango 0 - 255, sea, 0, 85 y 170, y se tiene el píxel con valores $10, 220, 160$, se aproximará a los siguientes valores: $0, 170, 170$ en los 3 canales.

Con esto se obtiene un color más plano y uniforme a través de la imagen, simulando el color homogéneo que generalmente se tiene en los dibujos, en la siguiente imagen se muestra un ejemplo dividiendo en 16 veces el rango.

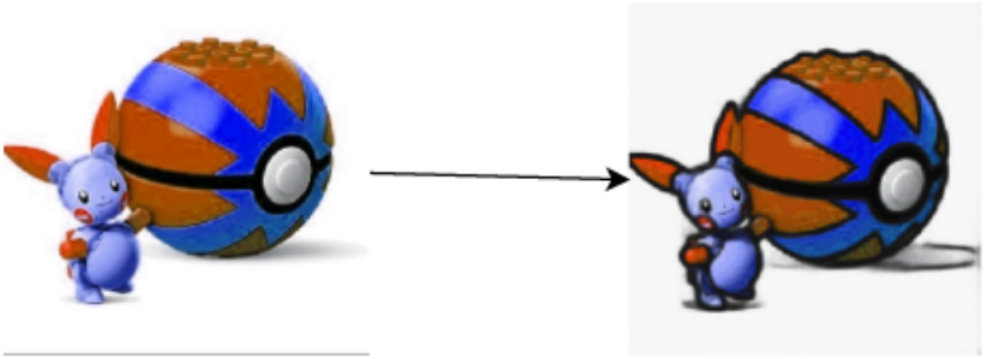


Figura 3.6: Ejemplo de cuantización del color de una imagen dividiendo en 16 el rango de los canales, también se agregó el resultado del filtro Pidinet a la imagen final.

3.3.3. Segmentación del color

Una segunda forma de agregar color a las imágenes es utilizando un segmentador. Estos dividen la imagen en áreas basadas en características de los píxeles. Por ejemplo, se puede separar una imagen en regiones donde el color es relativamente homogéneo. Como se observa en la imagen 3.7.

Figura 3.7: Ejemplo de segmentación hecho por el modelo de *segment anything* [11].

Para esto se utilizó el modelo de segmentación *segment anything* 3.7 para calcular las regiones de segmentación. Posteriormente, se calcula el promedio del valor de color de cada área y se reemplaza el color de esta por el color promedio, logrando que se obtenga una imagen parecida a un dibujo coloreada de forma uniforme. Como se observa en la imagen 3.8.

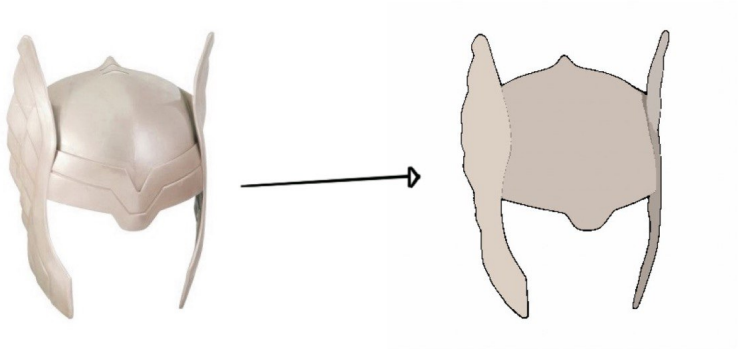


Figura 3.8: Ejemplo de proceso de creación de dibujos artificiales con segmentación.

3.3.4. Licuamento de los sketchs (Random liquify)

Para resolver el problema de que los trazados obtenidos de Pidinet tengan algo de variabilidad, se optó por licuar los sketchs generados, el licuamento de las imágenes se inspiró en la herramienta licuar de *photoshop*.

Figura 3.9: Ejemplo de la herramienta Licuar de *photoshop*, donde esta mueve los píxeles de la imagen correspondiente.

Entonces, al aplicar un filtro que logre una transformación parecida a las imágenes generadas con Pidinet, se puede obtener un dibujo artificial más parecido a los reales. Se va a referir a este tipo de transformaciones como elástica porque la idea de la forma de la imagen original se mantiene.

Figura 3.10: Ejemplo de licuamento sobre la imagen generada con Pidinet.

A continuación una explicación de como se logra esto.

Para realizar el filtro de *Random Liquify*, se utilizó la librería *pytorch* de python, esta posee el método *grid_sample*, que aplica un campo de vectores a una imagen, efectivamente moviendo los píxeles de esta en forma similar a como lo hace la herramienta *liquify* de photoshop. Entonces todo lo que queda a hacer es generar estos campos de vectores de forma que generen *sketchs* parecidos a dibujos.

Para esto primero se genera un círculo donde los vectores del campo van a apuntar hacia el centro y mientras más alejados estén de este, menos intensidad tendrán. Entonces, para crear las imágenes licuadas, se generan múltiples centros al azar repartidos sobre la imagen con sus correspondientes campos de vectores, así, generando un campo vectorial *random* que puede alterar la imagen de forma elástica.

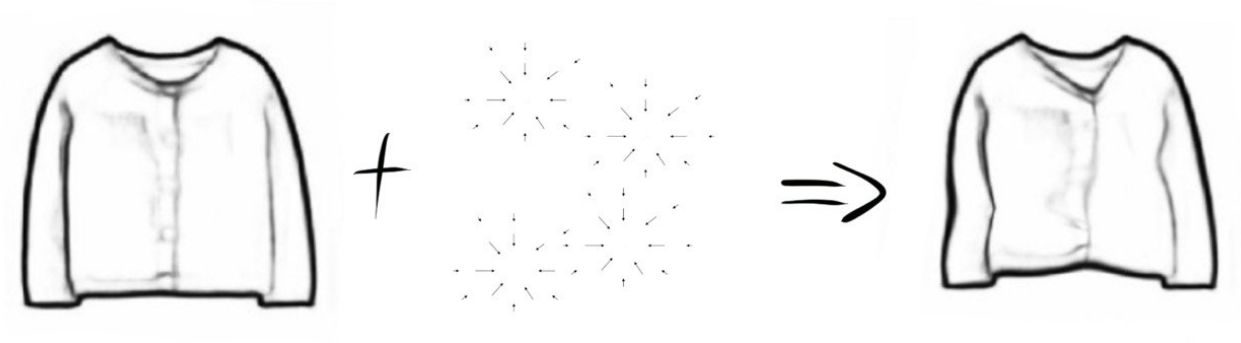


Figura 3.11: Ejemplo de uso de campos vectoriales para aplicar una transformación elástica.

Para aplicar la función *grid_sample* el campo de vectores debe estar en coordenadas cartesianas, por lo que se inicia el campo en estas coordenadas, después se le realiza una transformación a coordenadas polares utilizando como centro un punto seleccionado aleatoriamente y se le aplica la siguiente ecuación:

$$M = e^{-\rho c} + \ln(s)$$

Donde M corresponde a la magnitud del vector, c es un parámetro que define que tan rápido baja la magnitud a 0 y s define el valor de magnitud en el centro. La siguiente curva muestra la ecuación anterior para valores de $c = 1.1$ y $s = 0.03$

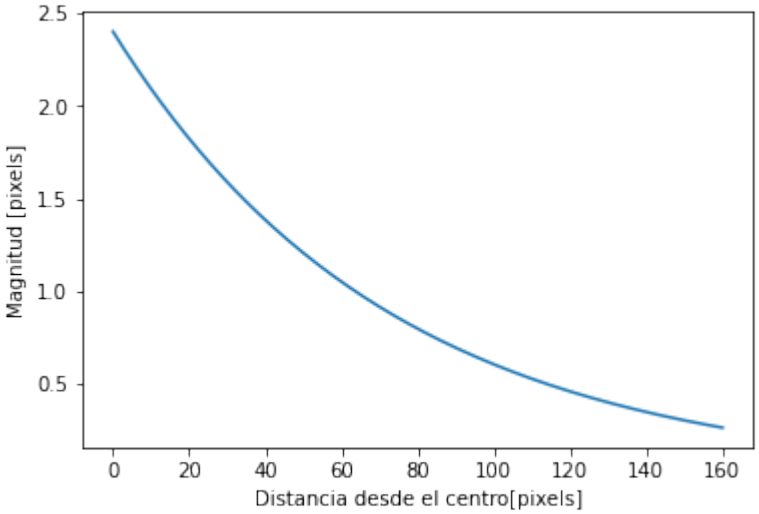


Figura 3.12: Curvatura representando el cambio de magnitud de los vectores al alejarse del centro.

Para los experimentos se utilizaron los valores de c y s mencionados anteriormente, ya que se comprobó empíricamente que lograban las imágenes más parecidas a un dibujo.

3.4. Filtros

Durante los experimentos también se va a utilizar filtros para agregar variabilidad a los datos y ayudar a que los dibujos sintéticos sean más parecidos a un dibujo real. En la mayoría de los filtros se utilizó un hiperparámetro que define la posibilidad de que el filtro se aplique sobre el input de la red, en la mayoría de los casos se escogió un valor de 50 % de que el filtro se aplicara.

3.4.1. Escalamiento y Traslación

Este filtro corresponde a alterar el dibujo generado artificialmente, cambiado el tamaño y su posición en la imagen.

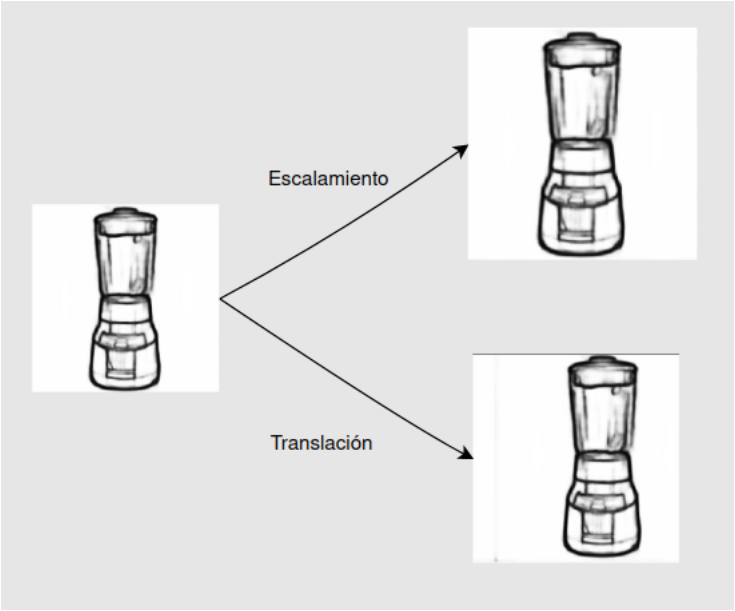


Figura 3.13: Ejemplo del escalamiento y traslación, durante el entrenamiento existe una posibilidad de que la imagen sea alterada de una u otra forma .

Existen dos formas en la que esto se realiza, escalamiento y traslación. En escalamiento la imagen es aumentada o disminuida en tamaño un 10 % con respecto a la imagen original antes de ingresarla a la red. En traslación, el dibujo es trasladada en una dirección al azar un 10 % con respecto al tamaño de la imagen.

3.4.2. Desenfoque Gaussiano

Corresponde a un filtro que, utilizando un kernel que se traslada por la imagen (de la misma forma que en una capa convolucional) y con valores discretizados de una campana gaussiana de dos dimensiones en el kernel. Esto provoca un efecto de desenfoque en la imagen

como se puede ver en la figura 3.14.

Figura 3.14: Ejemplo del desenfoque gaussiano, a la izquierda la imagen original y a la derecha la imagen filtrada.

El único hiperparámetro a definir es sigma, que entre más grande aumenta el desenfoque que sufre la imagen.

3.4.3. Cambio de saturación

Corresponde a cambiarle el valor de saturación a la imagen cuando esta expresada en el formato *HSV*. El objetivo es tomar en cuenta el caso donde el color de los dibujos hechos por las personas tiende a no ser el mismo al de las imágenes originales, sino que este es un poco diferente a este. Por lo que el filtro le da más variabilidad a los datos.

3.5. Obtención de Resultados

3.5.1. Mejor resultado para dibujos sin color

Como primer objetivo, se presentan los mejores resultados obtenidos por Javier Morales sobre el Dataset eCommerce para SBIR en la tabla 3.1.

Tabla 3.1: Mejores resultados para este problema obtenidos por Javier Morales.

Dataset de evaluación	map
eCommerce	0.253

Estos resultados se obtuvieron con el framework BYOL, utilizando como base la red

Resnet50 con pesos pre-entrenados en ImageNet. El entrenamiento se realizó de la siguiente forma, se comienza utilizando el *dataset* sketchy entrenando de forma auto-supervisada y con pesos pre-entrenados en ImageNet, después se realiza un segundo y último entrenamiento con eCommerce, utilizando sketches generados artificialmente con PidiNet y sin Color.

A continuación se presenta un diagrama del proceso con el que se obtuvo este resultado

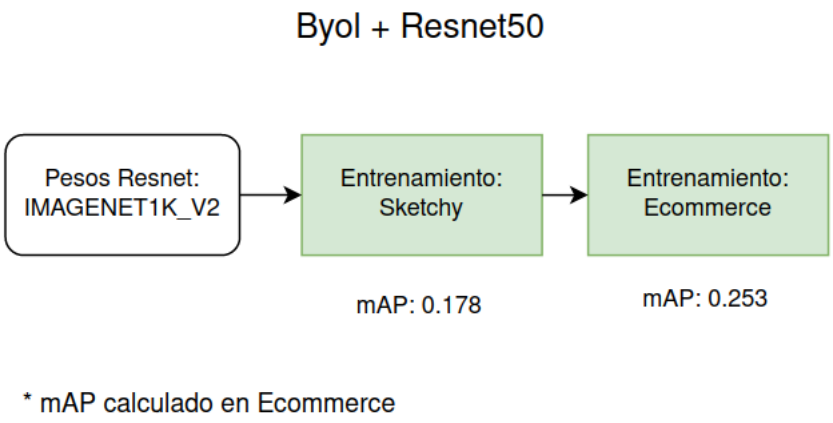


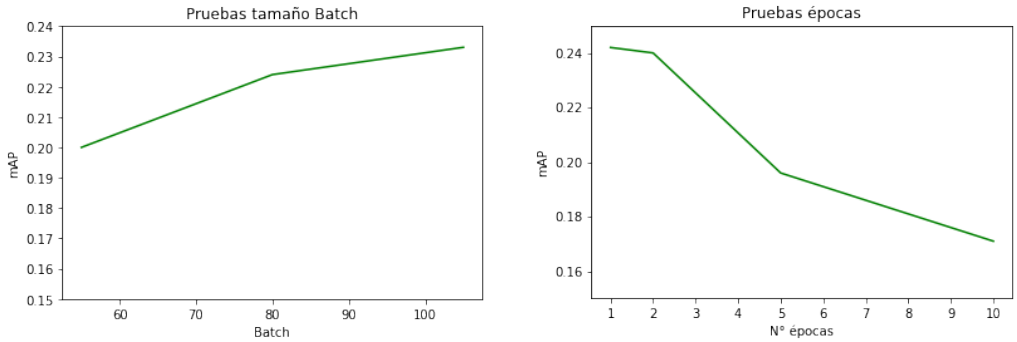
Figura 3.15: Diagrama del proceso para obtener el resultado de estado del arte.

Capítulo 4

Experimentos y resultados

4.1. Resultados Preliminares

Antes de comenzar, se presentan resultados preliminares que se realizaron para identificar los mejores valores de *Batch* y épocas.



(a) Pruebas de valores de Batch.

(b) Pruebas de valores de época.

Figura 4.1: Pruebas de batch y épocas, estas se realizan en la segunda etapa de entrenamiento, con el *dataset eCommerce.*

Como se puede observar en los gráficos anteriores, entre mayor sea el valor de*Batch*, mejor es el resultado que se obtiene sobre el *dataset* eCommerce, el mayor valor de Batch posible es 105 por limitaciones de hardware, por lo que todos los siguientes experimentos se realizaran con este valor de Batch.

Para el caso de las épocas, se puede observar que el valor de map comienza a bajar inmediatamente después de la primera época, con esto se puede concluir que existe **overfitting** en el entrenamiento, algunas maneras de combatir este *overfitting* se verán en los siguientes experimentos.

4.2. Experimentos con dibujos sin color

4.2.1. Entrenamiento particionado entre Skecthy y eCommerce

Primero se plantea que en el segundo proceso de entrenamiento, en vez de realizarlo con eCommerce, se realice un entrenamiento particionado, donde por ejemplo, en la primera época se utiliza un *dataset* conformado por un 80 % de Sketchy y un 20 % de eCommerce, en la segunda época un 50 % de Sketchy y un 50 % de eCommerce, así aumentando la proporción de eCommerce, la idea detrás de este método, es que la red pueda aprender de forma más gradual el segundo *dataset* utilizado y tal vez aumente su eficacia. A continuación se muestra un ejemplo.

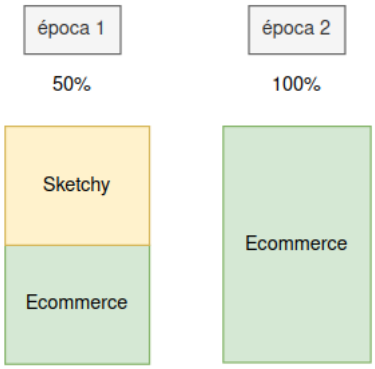


Figura 4.2: Ejemplo de entrenamiento particionado donde primero se entrena con un 50 % de *Sketchy* y 50 % de *eCommerce* para después en una segunda época, entrenar 100 % con *eCommerce*.

Entonces, en este experimento se implementaron diferentes porcentajes para las distintas épocas, de modo de observar como se comportaba la red y también como prueba inicial para observar si vale la pena experimentar con particiones más complicadas. En la tabla 4.1 se pueden ver los resultados.

Tabla 4.1: Cada porcentaje representa una época y a la cantidad de datos correspondiente a eCommerce, por ejemplo, la primera línea muestra el entrenamiento ejemplificado en la figura 4.2. Esto se realiza después de realizar el pre-entrenamiento con *sketchy*.

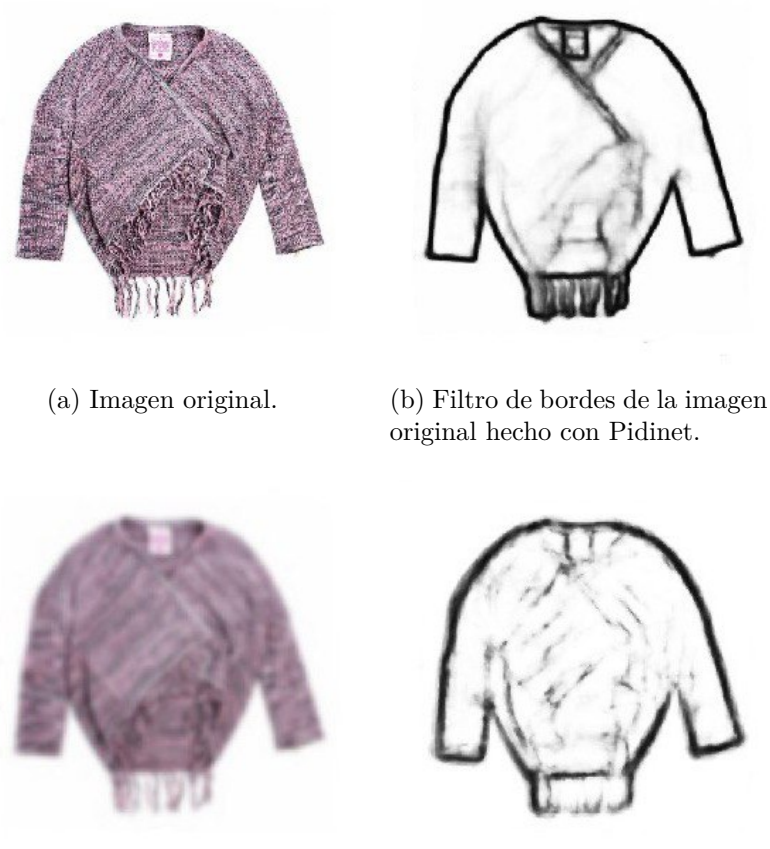
Porcentaje correspondiente al dataset eCommerce para cada época	map
50 % - 100 %	0.215
30 % - 80 % - 100 %	0.230
80 %	0.240
80 % - 100 %	0.233

Analizándolos, se puede observar que ninguna de las particiones escogidas logró superar

el valor del estado de arte, 0.253 map, una hipótesis de por qué se obtuvo este resultado es que, ya que hay un gran *overfitting* en el entrenamiento como se discutió en la sección 4.1, cualquier entrenamiento con más de una época tenderá a bajar la eficiencia de la red, independiente de los porcentajes de partición que estas tengan. Como se ve en la tabla, el mejor resultado fue el tercero con una época y 80 % del *dataset* correspondiente a eCommerce, pero se da el caso que es el más cercano al entrenamiento original de los 4 experimentos, por lo que lo único que se puede concluir es que el mejor tipo de entrenamiento particionado era el original para este problema en particular. Hay que notar que el entrenamiento particionado podría aún funcionar y estos resultados no son definitivos en su eficiencia, pero si muestran que debería ser utilizado en redes o *frameworks* que no tengan demasiado *overfitting*.

4.2.2. Desenfoque Gaussiano de los sketchs

Este experimento corresponde a que previamente a realizar el filtrado con Pidinet, se realizará un procesamiento con filtro Gaussiano en la imagen, la hipótesis detrás de esto es que al obtener una imagen más borrosa, Pidinet identificará menos bordes innecesarios en la imagen y el dibujo artificial será más parecido a uno real.



(a) Imagen original.

(b) Filtro de bordes de la imagen original hecho con Pidinet.

(c) Imagen desenfocada.

(d) Filtro de bordes de la imagen desenfocada.

Figura 4.3: Ejemplo del proceso de desenfoque y posterior filtrado.

En el entrenamiento se utiliza la imagen original y el dibujo artificial hecho de la imagen desenfocada. Por ejemplo, en las figuras anteriores, se utilizaría 4.3.a como foto y 4.3.d como dibujo. El valor de sigma para el desenfoque de la imagen es 3. A continuación se presentan los resultados:

Tabla 4.2: Mejor resultado para el experimento de filtro de desenfoque gaussiano en el último entrenamiento con eCommerce.

	map
Desenfoque gausiano	0.241

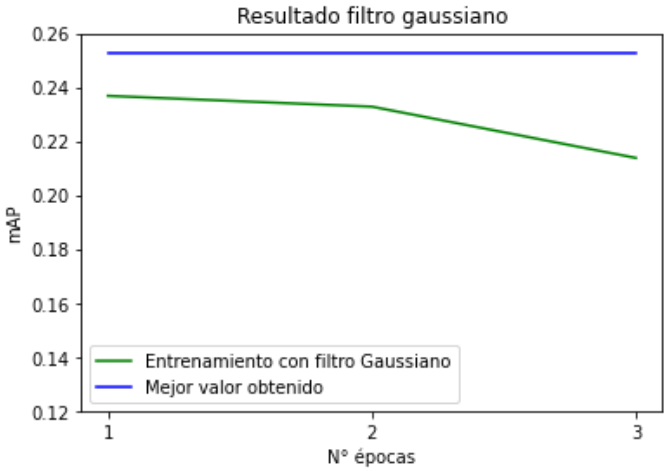


Figura 4.4: Evolución del map en el entrenamiento de la red, la línea azul corresponde al resultado de estado de arte.

En general, el proceso de desenfocar las imágenes no logró superar los resultados del estado del arte, como se observa en la figura 4.4. Una idea de por qué esto ocurre es analizando la figura 4.3, se puede ver que el dibujo artificial creado a partir de la imagen desenfocada tiene más ruido que el dibujo artificial original, los trazos son más gruesos y menos definidos, lo que los hace muy diferente a un trazo de un dibujo real, segundo, a pesar de que se intentaba identificar menos bordes innecesarios, se crearon trazos al azar sobre la imagen, haciendo que la forma sea más confusa para la red.

4.2.3. Entrenamiento con solo 3 bloques de Resnet50

Este experimento se formuló como la primera idea para enfrentar el problema del *overfitting* que sufre la red. La hipótesis detrás de esta es que al utilizar menos bloques en la red de *backbone* (Resmet50) esta tendría menos capacidad de aprender los datos del *dataset* y debido a esto generalizar mejor los datos. Esto se basa en que, ya que los *sketchs* son artificiales, la red está aprendiendo demasiado bien como discriminar estos tipos de imágenes artificiales y no logra generalizar al probar con datos reales. Una idea similar a la técnica de *dropout* (que elimina neuronas al azar durante el entrenamiento de una red neuronal) ayuda a que las redes generalicen mejor. Entonces se van a utilizar los primeros 3 bloques de la red Resnet50 ejemplificada en la figura 2.10.

A continuación se presentan los resultados:

Tabla 4.3: Mejores resultados para el entrenamiento solo utilizando 3 bloques de Resmet50.

	map
Entrenamiento con 3 bloques de Resnet50	0.184

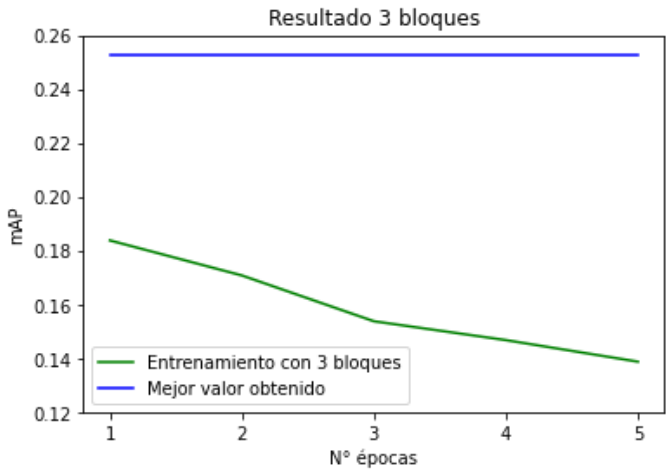


Figura 4.5: Resultado de map para el entrenamiento usando de *backbone* 3 bloques de la Resnet50, la línea azul corresponde al estado del arte.

Como se puede observar, los resultados obtenidos son bastante lejanos al del estado del arte y con una curva de aprendizaje similar al del experimento preliminar, donde la mejor época corresponde a la primera y después comienza a bajar de forma inmediata. Se barajan dos razones de por qué esto podría haber pasado. Primero, tal vez este problema necesita los aproximadamente 25 millones de parámetros de la Resnet50 original para poder expresar de buena manera el problema. Una segunda razón es que el experimento podría no haberse realizado en las mismas condiciones que en el estado del arte, por ejemplo un diferente valor de *batch* o diferente *hardware* utilizado, aunque esta posibilidad es poco probable, ya que los resultados fueron considerablemente diferentes al estado del arte.

4.2.4. Escalamiento y Traslación

Este experimento corresponde a utilizar el filtro discutido en 3.4.1. La hipótesis detrás es la de agregar más variabilidad a los dibujos artificiales y que gracias a esto se parezcan más a los reales. Como la mayoría de los dibujos están hechos por personas sin entrenamiento, lo más probable es que estos tengan problemas de forma, escala o discontinuidades. Por lo que este experimento es el primero para enfrentar este problema.

Como se mencionó en la sección 3.4 de filtros, este filtro se utiliza con una probabilidad de 50 %, ósea, que en el entrenamiento, la mitad de los dibujos van a pasar por el filtro.

A continuación se presentan los resultados, con 50 % de probabilidad de que se aplique el filtro, con una traslación del 10 % en comparación al tamaño de la imagen y con un rango de escalamiento de 0.8 - 1.2, lo que significa que el dibujo podría ser 0.8 veces más pequeño o hasta 1.2 veces más grande en comparación al original:

Tabla 4.4: Mejor resultado para el experimento de agregar escalamiento y traslación en el último entrenamiento con eCommerce.

	map
Entrenamiento con escalamiento y traslación	0.257

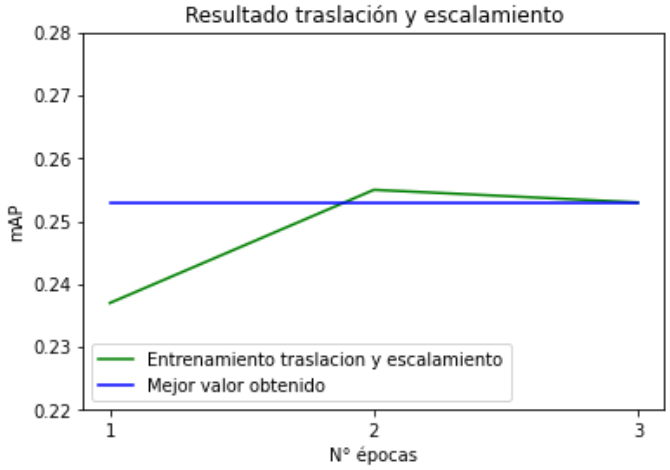


Figura 4.6: Comparación del aprendizaje de la red con filtro de traslación y escalamiento y el estado del arte.

Se puede observar un primer caso donde el experimento logró superar el estado del arte. Aunque la diferencia fue mínima, se observa que aumentar la variabilidad de los datos y lograr que los *sketchs* artificiales se parezcan más a un *sketch* real, pareciera ser donde más potencial existe para mejorar el funcionamiento de la red. Con esta idea, se procede al siguiente experimento que podría pensarse como una expansión de esta.

4.2.5. Licuamento de los sketchs

En este experimento se utiliza el filtro de licuamento de la sección 3.3.4. Como ya se mencionó, la idea es que al mover al azar los píxeles de la imagen, esta se parezcan más a un dibujo real hecho por una persona. Agregar estas variabilidades a los dibujos artificiales es crucial para poder entrenar una red que sea resiliente a *sketchs* mal hechos. Por lo que se va a realizar un experimento con dibujos alterados de la manera como se muestra en la figura 3.10.

A continuación los resultados de este experimento:

Tabla 4.5: Resultado agregando licuamento en el último entrenamiento con eCommerce.

	map
Entrenamiento con licuamento	0.257

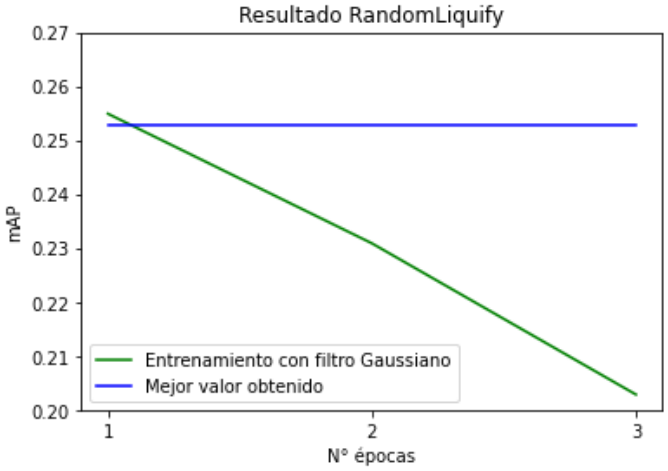


Figura 4.7: Comparación de la curva de aprendizaje con filtro de licuamento y el valor de estado del arte.

Al igual que con la traslación de los *sketchs*, se observa una leve mejoría de los resultados, aunque esta sigue siendo mínima como para lograr una diferencia sustancial, de todas maneras, la idea de lograr que los sketch se parezcan más a los reales sigue mostrando potencial. Es importante notar que el algoritmo de *Random Liquify* para generar los *sketchs* alterados podría mejorar sustancialmente, ya que este se implementó de forma rápida para poder realizar el experimento, por lo que aún existe mucho potencial para explorar esta idea con mejores algoritmos de generación.

4.2.6. Licuamento con traslación y escalamiento

Siguiendo con la idea de lograr que los *sketchs* se parezcan cada vez más a un dibujo real, se plantea el siguiente experimento que agrupa los dos filtros de licuamento con traslación y escalamiento, estos dos filtros fueron los que mejor resultaron de los experimentos previos por lo que se formó la hipótesis de que al juntarlos se podría mejorar aún más la eficiencia de la red.

Los experimentos se llevaron haciendo algunas alteraciones a la probabilidad de que un filtro en particular se aplicará. A continuación se presentan los resultados:

Tabla 4.6: Mejores valores de map para diferentes probabilidades de que el filtro licuamento se utilizara.

Probabilidad de usar de cada filtro	map
50 % traslación - 100 % licuamento	0.263
50 % traslación - 60 % licuamento	0.262
50 % traslación - 90 % licuamento	0.265

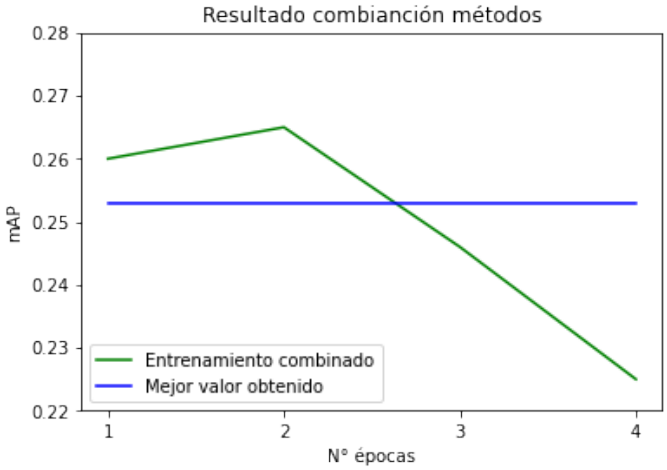


Figura 4.8: Gráfico de aprendizaje para el tercer caso de la tabla 4.6.

Utilizando estos dos filtros juntos, se pudo obtener el mejor resultado hasta ahora con 0.265 de map en el tercer caso de la tabla 4.6, aunque ninguno de los 3 casos mostró una diferencia demasiado importante, lo que pareciera mostrar que la probabilidad no afecta la eficiencia de la red.

Observando el gráfico, se puede observar que a diferencia de la mayoría de los gráficos de aprendizaje mostrados, el *overfitting* empieza a partir de la tercera época, lo que implica que la red tuvo más dificultad de aprender con los filtros utilizados. Esto da más apoyo a la idea de que generar técnicas para que los dibujos artificiales se parezcan más a los reales posee potencial para este problema.

4.2.7. Mejor resultado

Entonces, tomando el mejor resultado de las secciones previas, el diagrama original quedará de la siguiente forma:

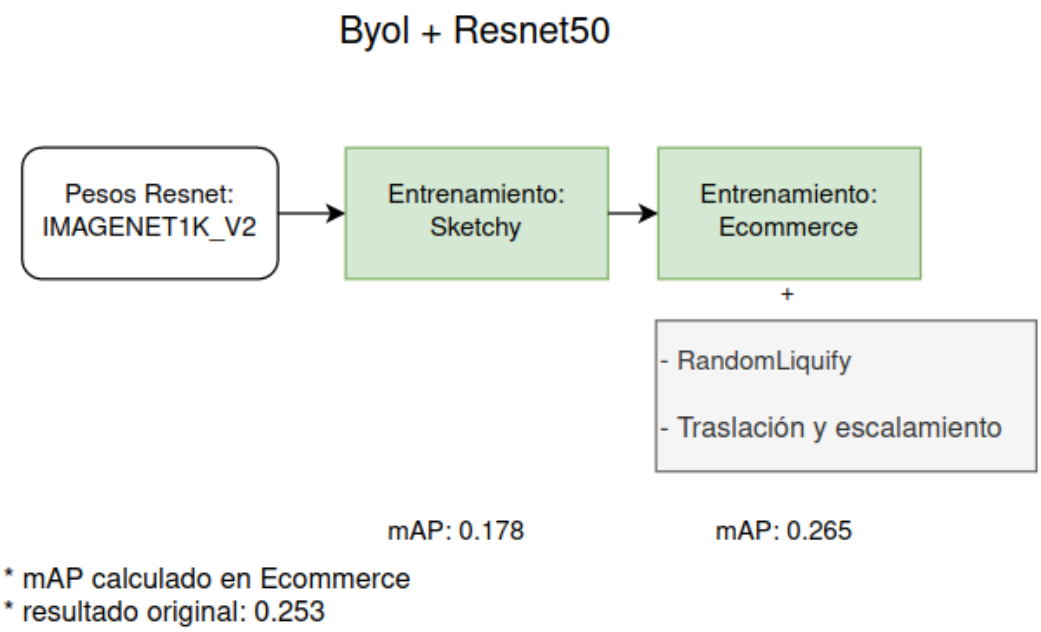


Figura 4.9: Diagrama del mejor método encontrado.

Algo importante a notar es que el filtro de traslación y escalamiento podría verse como una forma de licuar la imagen, para el caso donde se agranda la imagen, se podría utilizar un campo vectorial circular que irradiaría desde el centro de la imagen o para el caso de traslaciones, se trataría de un campo vectorial donde todos los vectores apuntan hacia alguna dirección. Entonces los dos filtros podrían verse como el mismo filtro con diferente composición.

Con esto, una forma de mejorar este resultado sería creando una implementación más sofisticada de *Random liquify*, que pudiera generar formas de dibujos artificiales más diversas, como por ejemplo agregar discontinuidades, espirales, traslaciones, escalamientos, etc. en los dibujos. Se discutió la idea de utilizar algoritmos de generación de mapas al azar de videojuegos, para la creación de campos vectoriales más sofisticados, pero esto tomaría demasiado tiempo y se escapa del objetivo de la memoria, de todas maneras, la creación de un mejor filtro *Random liquify* sería un punto de partida perfecto para una investigación futura.

4.3. Experimentos de dibujos con color

En los siguientes experimentos, se usará una versión alterada de eCommerce. Utilizando el método discutido en la sección 3.3.2, se generó un *dataset* de igual tamaño a eCommerce con color agregado a todas los dibujos artificiales.

4.3.1. Resultados preliminares

A continuación se presenta el resultado de la mejor red encontrada en la sección previa de experimentos con dibujos sin color evaluada en el *dataset* de prueba con color.

Tabla 4.7: Resultados obtenidos de evaluar la mejor red encontrada en la sección de dibujos sin color evaluada en los datos de prueba con color.

	map	MRR
Mejor experimento obtenido sin color	0.092	0.138

4.3.2. Experimento sin pesos pre-entrenados

En este primer experimento se busca observar la eficiencia de la red cuando no existen pesos pre-entrenados en otros conjuntos de datos. En este caso, la red Resnet50 solo tendría los pesos entrenados en ImageNet como se ha hecho en experimentos previos. No se espera que se obtengan buenos resultados, si no, como una forma exploratoria para observar el comportamiento de la red con diferentes pesos iniciales en el *dataset* con color.

Tabla 4.8: Resultado utilizando solo pesos pre-entrenados en ImageNet y entrenando en los datos de eCommerce con color.

	map	MRR
Resultado	0.076	0.169

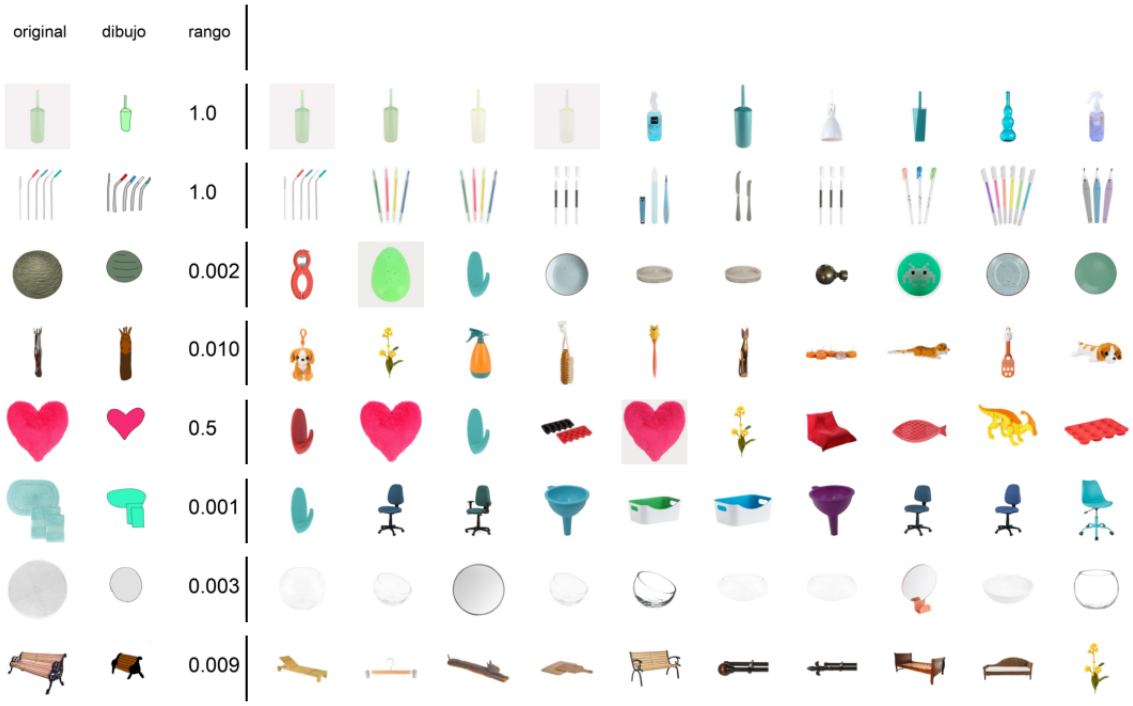


Figura 4.10: Ejemplos de recuperación de imágenes realizada con la red, la columna a la izquierda muestra la imagen original, la siguiente columna muestra el dibujo de la imagen, después el rango que la imagen original obtuvo. Las siguientes columnas muestran las imágenes recuperadas ordenadas de izquierda a derecha de acuerdo a la similitud.

Primero observando los resultados de la tabla, se observa que no hubo una mejor eficiencia de la red en comparación al resultado preliminar. A primera vista se podría decir que realizar pre-entrenamientos con otros conjuntos de datos como *sketchy* o eCommerce no añadió mucha eficiencia a la red, pero si se observan los resultados en la figura 4.10, se puede ver que la red tiende a favorecer el color por sobre la forma de la imagen. Esto ayuda en algunos casos, como en la quinta imagen del corazón, donde la foto original quedo segunda, pero en el cuarto ejemplo, solo se encontraron imágenes con color café pero no la foto original. Entonces cuando hay muchas imágenes en el *datasets* con color parecido al *input*, tendería a ser menos eficiente.

4.3.3. Experimento prentrenando con sketchy

A continuación se hará un experimento similar, pero solamente usando los pesos pre-entrenados con *sketchy*, ósea se realizó un entrenamiento con el *dataset sketchy*, para después entrenar con el conjunto de datos con color.

Tabla 4.9: Resultado entrenando con eCommerce con color y utilizando pesos pre-entrenados en *sketchy*.

	map	MRR
Resultado	0.086	0.135

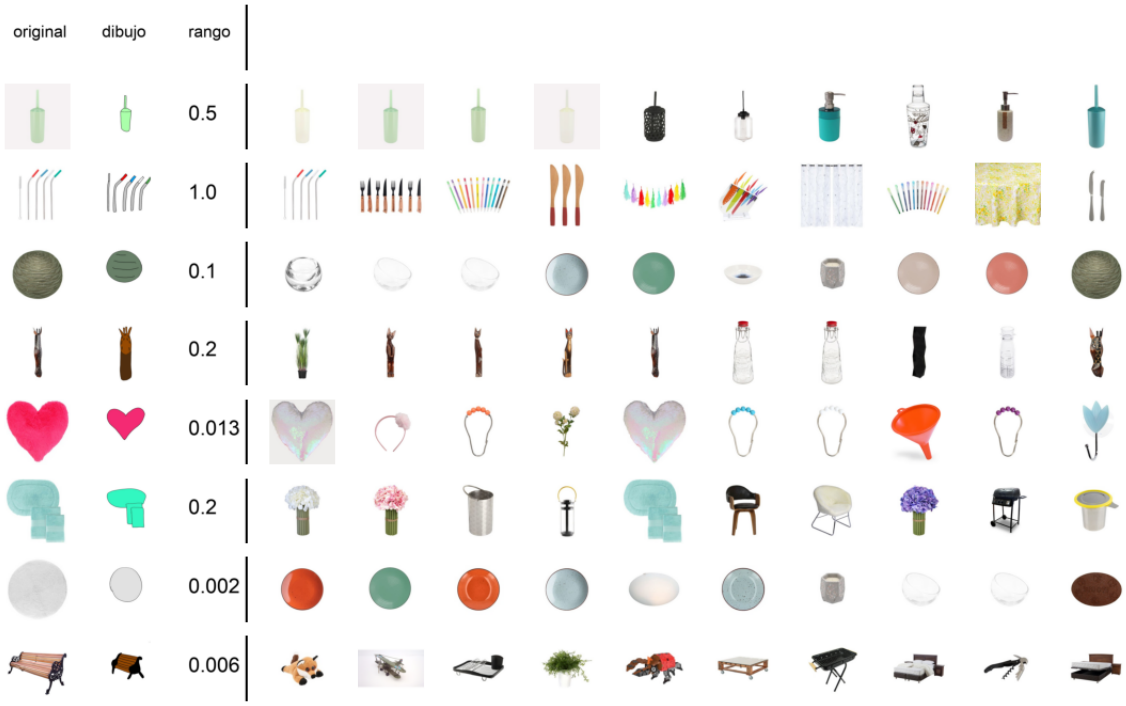


Figura 4.11: Ejemplos de recuperación de imágenes realizados con la red prentrenada con *sketchy*.

Los resultados de la tabla muestran que la eficiencia baja en comparación al primer experimento y se acercaron a los resultados preliminares, pero el map subió, lo que significa que la red está identificando la clase a la que pertenece el objeto mejor, pero identifica peor el objeto original del que se creó el dibujo.

Observando los ejemplos de la figura 4.11, se puede ver que la red recupero imágenes con objetos de formas más parecidas al *input*, por ejemplo, para el corazón, se recuperaron imágenes de forma triangular y algunos corazones, pero empeoro la habilidad de identificar el color. Esto muestra que un entrenamiento previo con *sketchy* logró dar a la red la habilidad de identificar mejor las formas, pero se perdió algo la habilidad para identificar color, lo que tiene sentido, ya que *sketchy* es un conjunto de datos de dibujos sin color.

4.3.4. Agregando filtros de licuamento y traslación con escalamiento

A continuación se presentan los resultados de agregar los filtros de licuamento y traslación con escalamiento al entrenamiento de la red. Al igual que en el caso anterior, se utiliza solamente la red pre-entrenada en *sketchy*, al igual como sucedió en los experimentos sin color, se espera que los filtros logren acercar los dibujos artificiales de mejor manera a un dibujo real, creando variabilidad en la data.

Tabla 4.10: Resultado utilizando pesos pre-entrenados en *sketchy* más filtros de *Random liquify* y traslación con escalamiento.

	map	MRR
Resultado	0.095	0.151

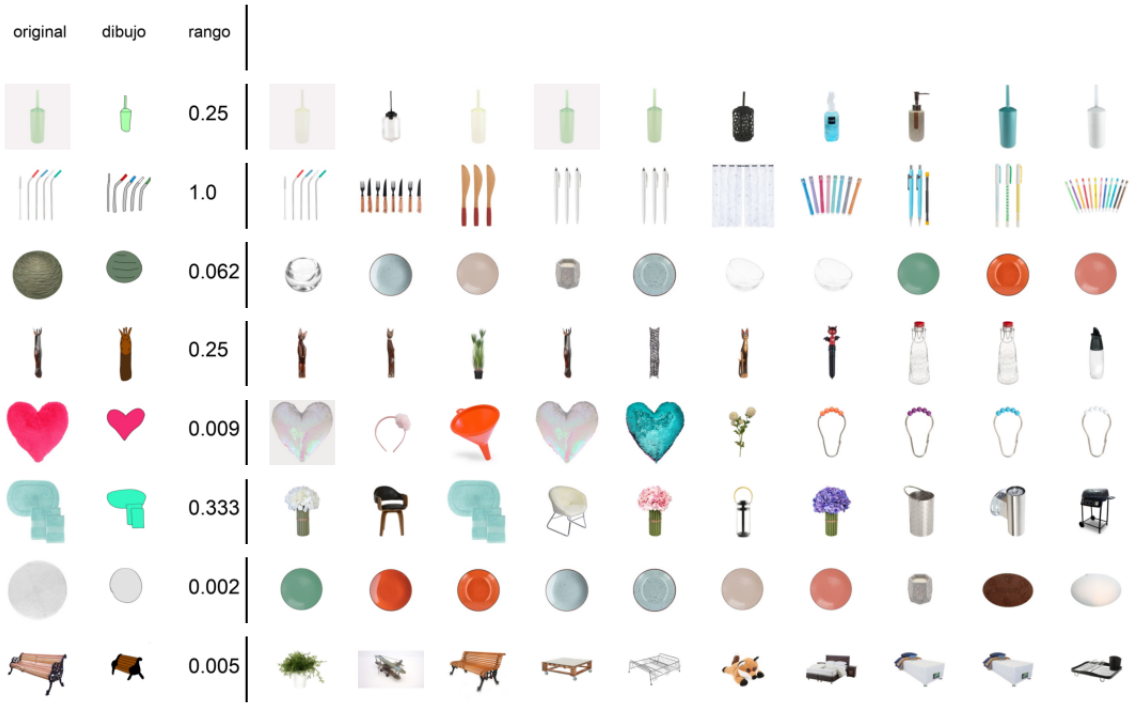


Figura 4.12: Ejemplos de recuperación de imágenes realizados con la red entrenada con filtros de licuamento y traslación.

Como se puede observar, los resultados mejoraron en las dos métricas con respecto al experimento anterior, también el map logró superar por poco al resultado preliminar y el MRR subió algunas décimas de porcentaje. Lo que implica que los filtros sí ayudaron al entrenamiento de la red. Esto demuestra que para los dibujos con color, los filtros usados siguen ayudando a aumentar la eficiencia.

4.3.5. Experimento utilizando la mejor red del entrenamiento sin color como base

Aquí lo que se busca es identificar como se comporta el entrenamiento cuando se utiliza la mejor red entrenada en los experimentos sin color. Esta misma red corresponde a la que se mostró en los resultados preliminares, por lo que esos datos son importantes como punto de comparación de la eficiencia que se tenga. También se muestran los experimentos con filtros y sin filtros para que se observe el efecto que estos tengan.

Tabla 4.11: Resultados utilizando la mejor red encontrada para la sección de dibujos sin color, con y sin filtros de *Random liquify* y traslación con escalamiento.

	map	MRR
Entrenamiento sin filtros	0.063	0.166
Entrenamiento con filtros	0.060	0.174

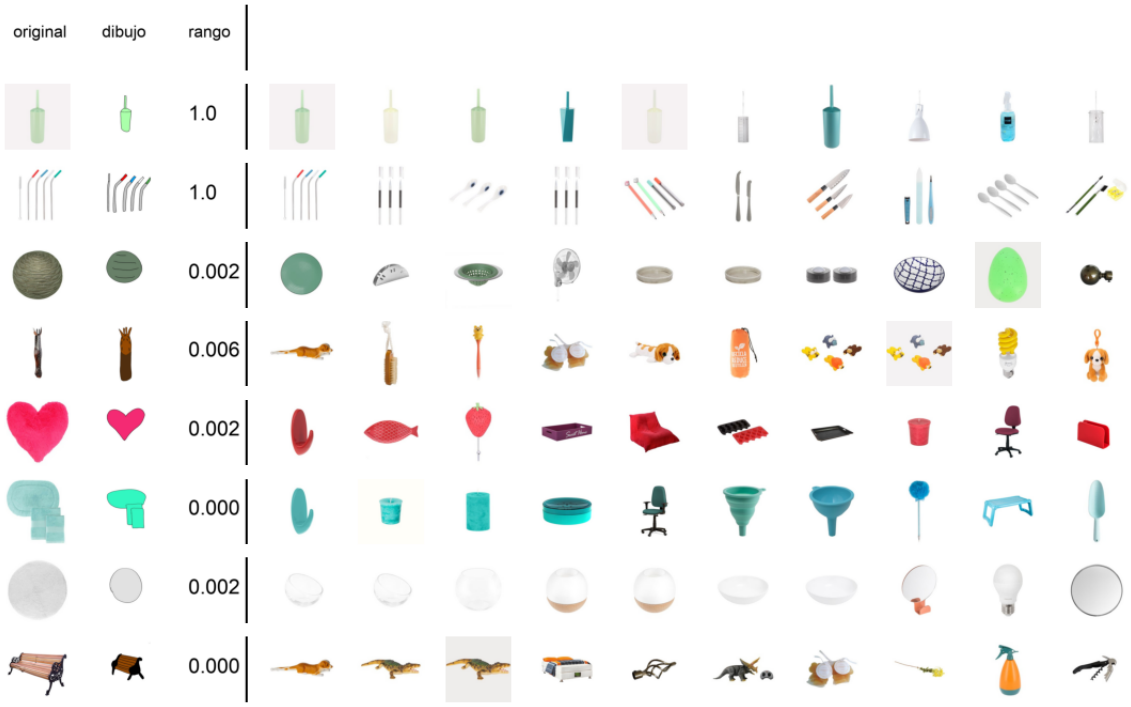


Figura 4.13: Ejemplos de recuperación de imágenes realizados con la red entrenada con la mejor red encontrada para el caso sin color como base, se muestran los resultados del segundo modelo de la tabla 4.11.

Otra vez se observa una mejora en el MRR obtenido. Donde en los dos casos, con filtro y sin filtro, mejoró con respecto a los resultados preliminares, para el caso con los filtros, hubo una mejora de algunas décimas de porcentaje, por lo que este proceso le da una cierta ayuda a la red.

Aunque se obtuvieron buenos resultados, es importante notar que hubo algunos detalles en el entrenamiento que son importantes de mencionar. En el entrenamiento la primera época generalmente se comportaba de forma normal, con el loss bajando mientras se entrenaba, pero en la segunda época hubo un desvanecimiento del gradiente, el loss se volvió prácticamente cero y se mantuvo allí hasta que terminó la época. Después probando la eficiencia de la red de la segunda época, se obtuvo un MRR de 0 y un map de 0. Esto no ocurre en todos los entrenamientos, sino que a veces termina en la solución trivial como en el caso anterior y otras veces esta termina bien su segunda época. Esto podría haber ocurrido porque ya se está es el tercer entrenamiento por la que pasa esta red, primero con *sketchy*, después con eCommerce y por último eCommerce con color agregado, estos pre-entrenamientos seguidos

podrían haber provocado el desvanecimiento del gradiente.

4.3.6. Cambios de saturación

Como la variabilidad de los datos ha demostrado que logra mejorar los resultados, se plantea que entrenando con colores ligeramente distintos a los que se tiene en el conjunto de datos logrará que la red sea más robusta a cambios de colores en los dibujos reales. Ya que como ya se ha dicho, los dibujos hechos por personas tienden a tener errores, en este caso se trata de errores en el color del objeto. Es importante notar que en este experimento es similar al anterior, pero no se están ocupando los filtros de licuamento ni traslación.

Tabla 4.12: Resultado para el entrenamiento con cambios de saturación en la data con probabilidad 50 %.

	map	MRR
Resultado	0.089	0.140

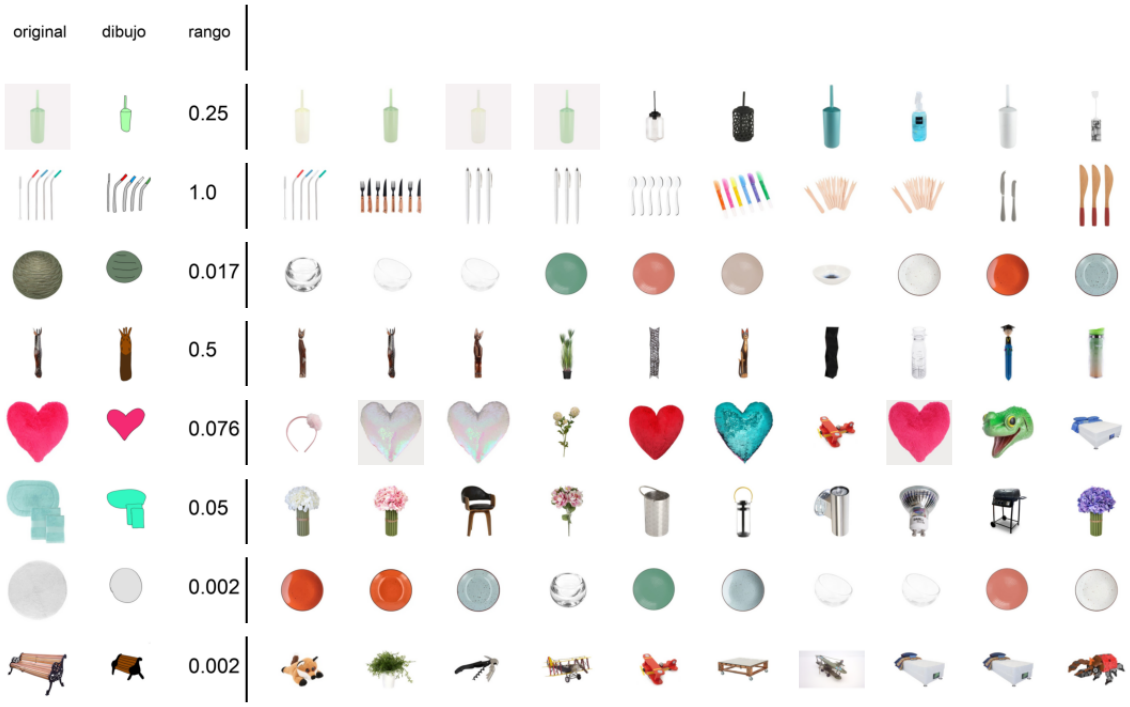


Figura 4.14: Ejemplos de recuperación de imágenes realizados con la red entrenada con cambios de saturación.

Comparando estos resultados con la primera fila de la tabla 4.11, se observa una pequeña mejora en el map, pero el MRR empeoró. Esto generalmente indica que la red tuvo mejor capacidad para identificar la clase del objeto, pero peor para identificar el objeto original. El resultado se podría explicar con que los objetos de una misma clase tienden a tener una forma similar y es la forma la que la red logró aprender un poco mejor. Esto implica que de hecho el cambiar la saturación de la data provoco algo similar a lo que ocurrió en el experimento

4.3.3, pero en este caso, como el color era un poco diferente, la red tuvo que aprender más de la forma de los objetos que del color.

4.3.7. Experimentos con datos aumentados

En el siguiente experimento se utiliza una aumento de la data, como se discutió en la sección 3.2.3, donde se habló de agregar más datos al entrenamiento, cambiando el color de las imágenes originales y sus correspondientes dibujos artificiales.

Se muestran varios experimentos similares a los ya vistos pero con el conjunto de datos aumentados:

Tabla 4.13: Resultados de entrenamiento con datos aumentados del *eCommerce* , en las 3 primeras filas se utilizan como valores iniciales de la red los pesos de la red preliminar y en la última fila se inicializa solamente con los valores entrenados en ImageNet.

	map	MRR
Red base sin filtros	0.084	0.135
Filtros de licuamento con escalamiento y traslación	0.082	0.119
Filtro de saturación	0.084	0.129
Entrenamiento sin pesos pre-entrenados	0.069	0.163

Los resultados de imágenes encontradas se encuentran en la sección de anexo 5.1.

En general, los resultados fueron peores en comparación a los experimentos hechos en secciones anteriores, excepto por la última fila de la tabla 4.13, pero en ese caso se obtuvo un map menor al de los otros, lo que implicaría que hubo peor discriminación de la forma de los dibujos.

La explicación a la que se llegó de estos resultados, es que la gran mayoría de los experimentos realizados en la presente memoria, tienen problemas de *overfitting*, donde ya en la segunda época, existe una baja en la eficiencia de las redes entrenadas. Entrenar una época con los datos aumentados es similar a entrenar dos épocas con los datos no aumentados y como eso bajaba la eficiencia en experimentos anteriores, esto provoco que la efectividad bajara para los datos de la tabla anterior. Este fenómeno también es consecuencia de que se hayan usado dibujos artificiales, si todos los dibujos fueran reales, entre más se tendría en el conjunto de datos, mejor para el entrenamiento. Pero al ser artificiales, existe una cantidad máxima de datos que pueden entregar resultados mejores, después la eficacia empieza a bajar debido al *overfitting*, ósea que la red está aprendiendo a identificar mejor los dibujos artificiales que los reales.

4.4. Experimento con segmentación de imágenes para la generación de dibujos

En el siguiente experimento se prueba el utilizar dibujos artificiales generados con el método de segmentación de imágenes mencionado en la sección 3.3.3. Se utilizan como pesos iniciales la red que entregó los mejores resultados para el caso sin color.

A continuación se presentan los resultados:

Tabla 4.14: Resultados de entrenamiento utilizando segmentación de colores para la creación de dibujos artificiales. Utilizando la mejor red encontrada para la sección de dibujos sin color como base. Experimentando con y sin filtros de *Random liquify* y traslación con escalamiento.

	map	MRR
Entrenamiento sin filtros	0.088	0.132
Entrenamiento con filtros	0.093	0.146

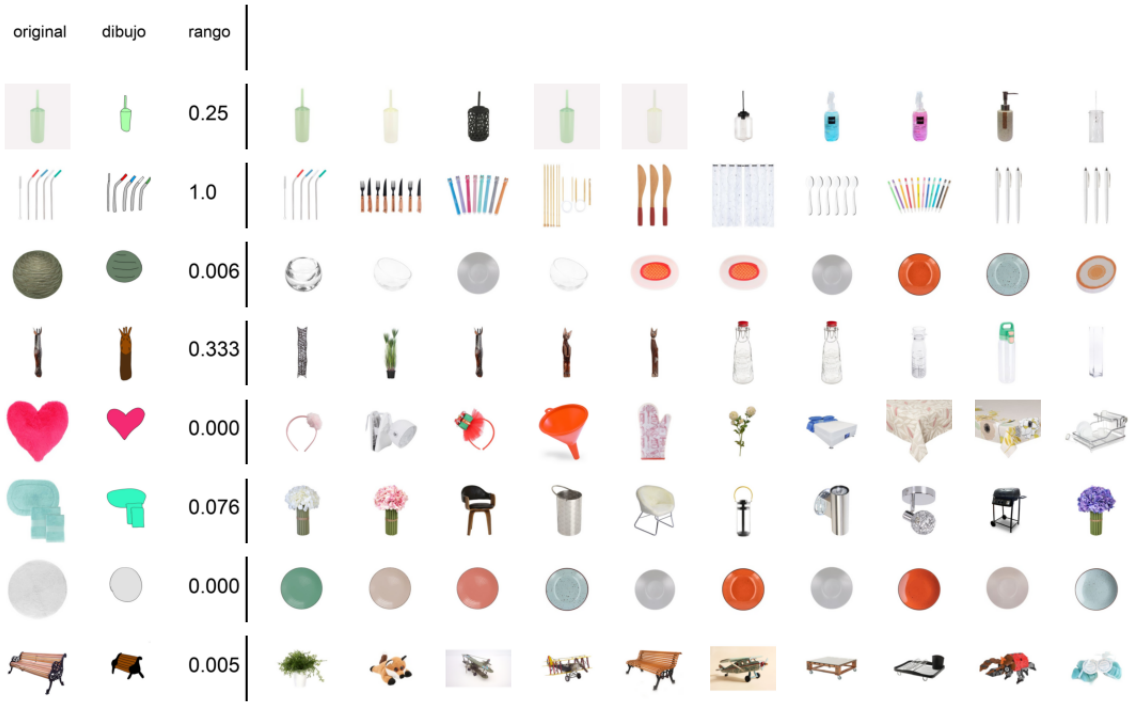


Figura 4.15: Ejemplos de recuperación de imágenes realizados con la red entrenada con dibujos artificiales creados con segmentación.

En general, no se obtuvieron resultados muy diferentes a los experimentos anteriores, lo que puede implicar que el utilizar segmentación en vez de cuantización de color no provoca diferencias importantes en el entrenamiento.

Pero un detalle importante es que el resultado de map para el entrenamiento con filtros resulto un poco mejor a los de los otros experimentos obtenidos, aunque la diferencia fue

mínima. Esto puede indicar que la segmentación combinada con filtros tiene mejor capacidad de entregar imágenes con forma parecidas a la imagen de consulta.

Capítulo 5

Conclusiones

Para los experimentos sin color, el problema de *overfitting* fue una constante razón de que los experimentos fallaran o que no demostraran tanta eficiencia como podrían mostrar en otros casos. Por ejemplo, para el experimento de entrenamiento particionado, el *overfitting* impedía que este mostrara mejores resultados, porque la red inmediatamente bajaba su eficiencia después de la primera época. El desenfoque gaussiano no demostró generar diferencias muy grandes con respecto a los resultados de estado de arte, ya que los dibujos artificiales tendían a parecerse menos a un dibujo real, que era lo contrario a lo que se buscaba. Por último, el entrenamiento con 3 bloques no logró obtener suficiente expresividad para este problema.

Los mejores métodos encontrados corresponden principalmente a los filtrados de licuamento y traslación con escalamiento. Como ya se mencionó en secciones anteriores. Estos dos filtros pueden verse como diferentes casos de un mismo filtro, que altera la imagen sin destruirla. Si el filtro de *Random liquify* utiliza un campo de vectores para realizar la trasformación de la imagen, el filtro de escalamiento puede entenderse como una trasformación con campo de vectores, en este caso con los vectores apuntando desde el centro de la imagen hacia afuera o en la dirección contraria y en la traslación es similar, donde los vectores todos apuntan hacia una dirección. Este tipo de filtrado de campos de vectores, demuestra que logra mejorar la eficiencia de las redes en los dos tipos de problemas, con color o sin color, como se vio en los experimentos de las secciones 3.3.4 y 4.3.4.

Para los resultados con color, se encontró con un nuevo problema donde las redes podían alternar entre identificar mejor las imágenes con la forma más parecida a la consulta o las imágenes con el color más parecido al de la consulta. Diferentes redes con diferentes parámetros o técnicas de entrenamiento tienden a favorecer a uno con respecto a otro. En general, la mejor solución sería una red que le dé algo más de importancia a la forma que al color, esto porque los dibujos reales hechos por personas, tienden a tener colores diferentes con respecto a los objetos que representan. Con este criterio, el mejor método encontrado fue el de agregar filtros de licuamento y traslación con escalamiento, visto en la sección 4.3.4, pero de todas maneras esto no implica que otros experimentos no sean aconsejables de seguir, ya que los resultados no fueron tan diferentes al mejor. Los métodos que no son aconsejable de seguir, son los que aumentan la cantidad de datos cuando se tienen imágenes creadas

artificialmente, por lo menos se tiene que tomar en cuenta que existe una cantidad máxima de datos a partir de la cual la eficacia empezará a bajar. Esto puede ser diferente para cada problema. Si una red no demuestra demasiado *overfitting* en el entrenamiento, esto no sería un problema. Pero es importante estar consciente de que este problema podría aparecer. En cambio, métodos que aumentan la variabilidad de los datos y que mejoran su parecido a los dibujos, como los discutidos en el párrafo anterior, demuestran ser la mejor forma de mejorar la eficacia.

Con respecto al objetivo general del trabajo, se logró estudiar la eficacia de varios métodos sobre el problema de recuperación de imágenes con dibujo y se analizó las razones de por qué cada uno obtenía los resultados encontrados. Se propuso un método, licuamento, que logró una leve mejora de la eficacia del problema en cuestión.

Con respecto a los objetivos especificados, se logró crear dibujos artificiales y añadir color a estos utilizando Pidinet, cuantización del color, segmentación y aumento de la data cambiando los colores. Las imágenes creadas a partir de estos procesos fueron utilizadas para el entrenamiento de los modelos, por lo que se lograron los objetivos primero y segundo.

El tercer objetivo no se logró, ya que no se tuvo el suficiente tiempo para realizar la implementación de una red convolucional.

En cuanto al cuarto y quinto, se logró realizar el análisis de todos los métodos propuestos, comparándolos entre ellos y entre trabajos anteriores, concluyendo sobre su eficacia.

5.1. Trabajos futuros

- Un posible trabajo futuro, como ya se habló, es el de implementar el filtro de *Random liquify* de nuevo con algoritmos más sofisticados. La primera idea es agregar más filtros que hagan trasformaciones elásticas a la figura, como filtros que agreguen discontinuidades, que tengan espirales, o, por ejemplo, que provoquen traslaciones en solo una parte de la imagen. Agregando todos estos filtros de forma seguida se puede lograr formas más diversas de alterar los dibujos artificiales. Una segunda forma para realizar este objetivo, es realizando una investigación sobre algoritmos de generación de mapas usados en videojuegos, ya que las transformaciones pueden ser expresadas como campos de vectores. Un algoritmo que genere estos campos al azar podría ayudar de gran manera. Como la generación de mapas al azar crea cosas como ríos, montañas o poblados en videojuegos, es posible realizar una implementación que genere campos de vectores con estructuras complicadas, como ejemplo círculos que se expanden, discontinuidades o movimientos parciales de traslación. Una posible desventaja de esto es que tal vez estos algoritmos tengan una complejidad prohibitiva para este problema.
- Otra posible idea es la de utilizar imágenes en el formato HSV como entrada de la red, esto para analizar comó es la eficacia que se logra con esta entrada y, además, se podría prescindir del valor, ya que este no contiene información del color de la imagen.
- También, se podría realizar un cambio al funcionamiento del *framework* de entrena-

miento BYOL. Como se explicó en la metodología, este *framework* está formado por dos redes; la *teacher* y la *student*, entonces se plantea el combinar los vectores de características en la salida de las dos redes para crear un nuevo loss que después se utilizará para realizar el *back propagation* de la red.

Como recomendación para trabajos futuros que sean similares a este. Es importante tomar nota de cada experimento realizado, escribiendo en primer lugar la hipótesis del experimento, los valores de los parámetros y el lugar o archivo donde se guardan los resultados. Esto para evitar el olvido información importante en trabajos donde se realizan una gran cantidad de experimentos.

Bibliografía

[1] Rodríguez, J. M., "Evaluación de métodos auto-supervisados y semi-supervisados para la extracción de características visuales en el contexto de recuperación de imágenes basada en dibujos," 2021, https://repositorio.uchile.cl/handle/2250/184264.

[2] Lab, G. C., "The quick, draw! dataset." https://github.com/googlecreativelab/quickdraw-dataset, 2015.

[3] Jara, A. F., "Recuperación de imágenes basada en dibujos mediante redes convolucionales," 2020, https://repositorio.uchile.cl/handle/2250/175585.

[4] He, K., Zhang, X., Ren, S., y Sun, J., "Deep residual learning for image recognition," 2015, doi:10.48550/ARXIV.1512.03385.

[5] Canny, J., "A computational approach to edge detection," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-8, no. 6, pp. 679–698, 1986, doi:10.1109/TPAMI.1986.4767851.

[6] Kingma, D. P. y Welling, M., "Auto-encoding variational bayes," 2013, doi:10.48550/ARXIV.1312.6114.

[7] Grill, J.-B., Strub, F., Altché, F., Tallec, C., Richemond, P. H., Buchatskaya, E., Doersch, C., Pires, B. A., Guo, Z. D., Azar, M. G., Piot, B., Kavukcuoglu, K., Munos, R., y Valko, M., "Bootstrap your own latent: A new approach to self-supervised learning," 2020, doi:10.48550/ARXIV.2006.07733.

[8] Sangkloy, P., Burnell, N., Ham, C., y Hays, J., "The sketchy database: Learning to retrieve badly drawn bunnies," ACM Transactions on Graphics (proceedings of SIGGRAPH), 2016.

[9] Torres, P. y Saavedra, J. M., "Compact and effective representations for sketch-based image retrieval," 2021, doi:10.48550/ARXIV.2104.10278.

[10] Su, Z., Liu, W., Yu, Z., Hu, D., Liao, Q., Tian, Q., Pietikäinen, M., y Liu, L., "Pixel difference networks for efficient edge detection," 2021, doi:10.48550/ARXIV.2108.07009.

[11] Kirillov, A., Mintun, E., Ravi, N., Mao, H., Rolland, C., Gustafson, L., Xiao, T., Whitehead, S., Berg, A. C., Lo, W.-Y., Dollár, P., y Girshick, R., "Segment anything," 2023.

Anexo

Ejemplos de recuperación de imágenes para los experimentos de con datos aumentados

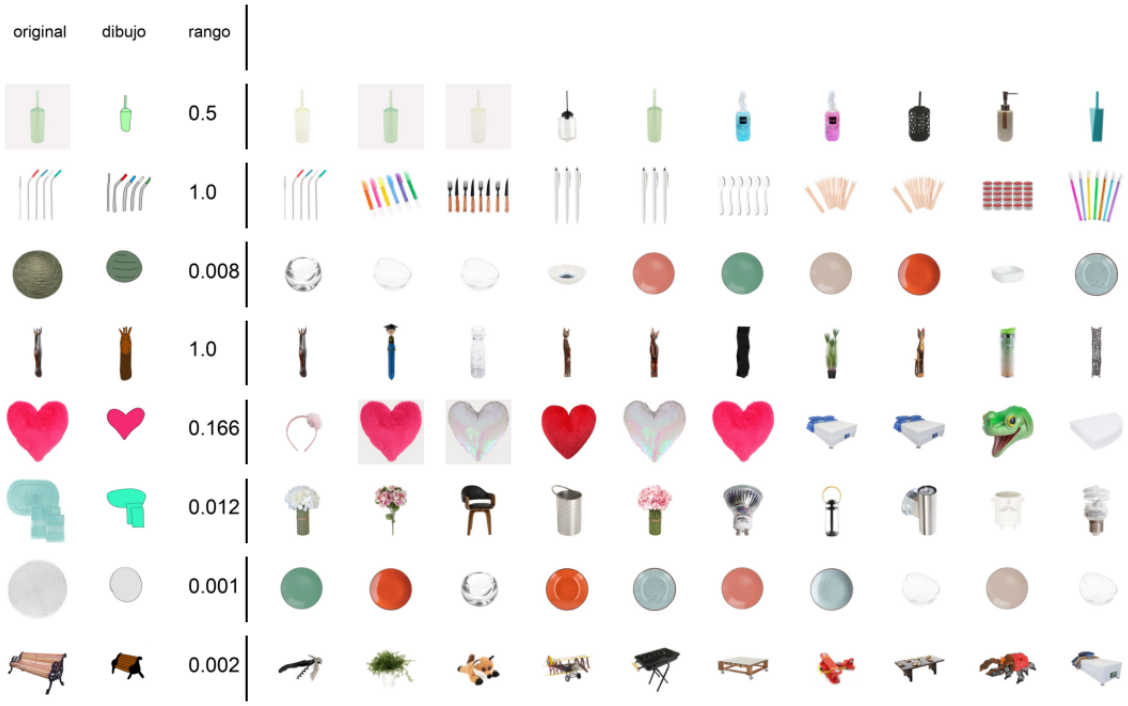


Figura .1: Ejemplos de recuperación de imágenes para la red base sin filtros.

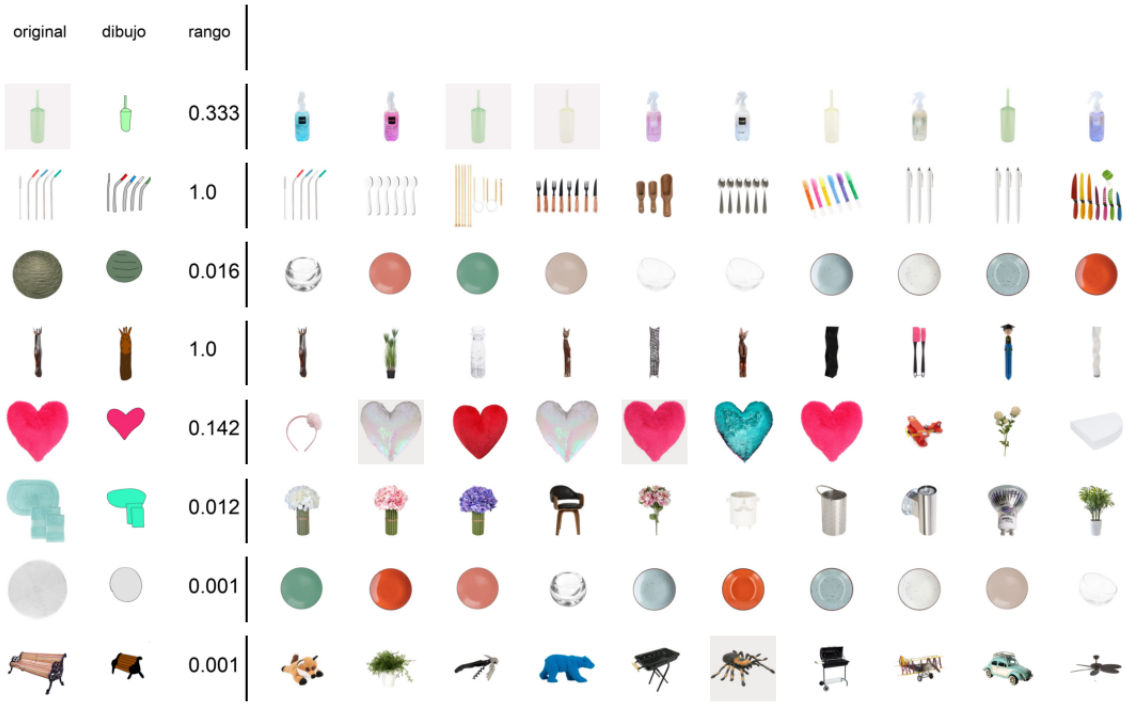


Figura .2: Ejemplos de recuperación de imágenes para filtros de licuamento con escalamiento y traslación.

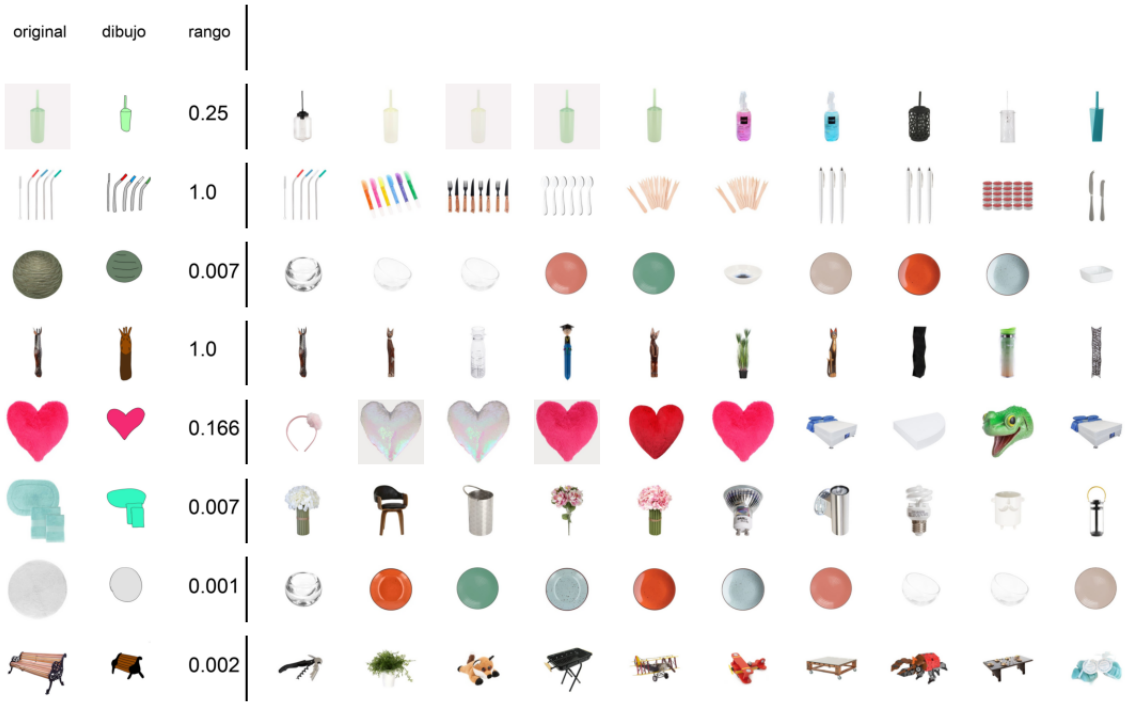


Figura .3: Ejemplos de recuperación de imágenes para filtros de saturación.

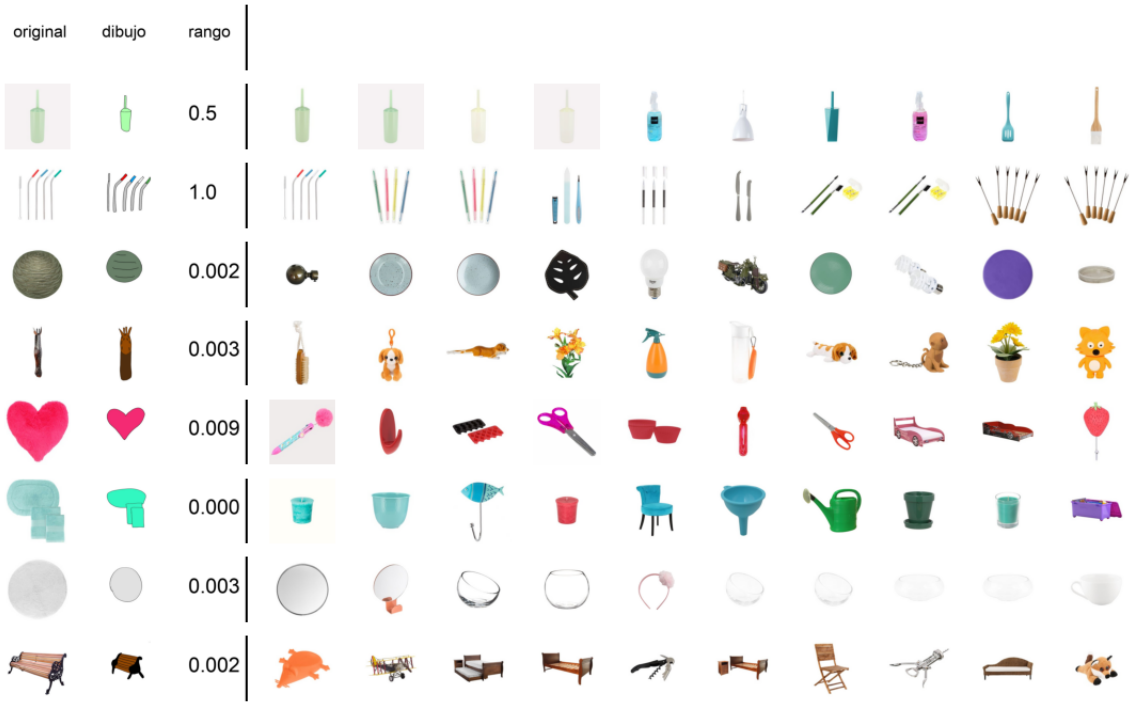


Figura .4: Ejemplos de recuperación de imágenes para entrenamiento sin pesos pre-entrenados.