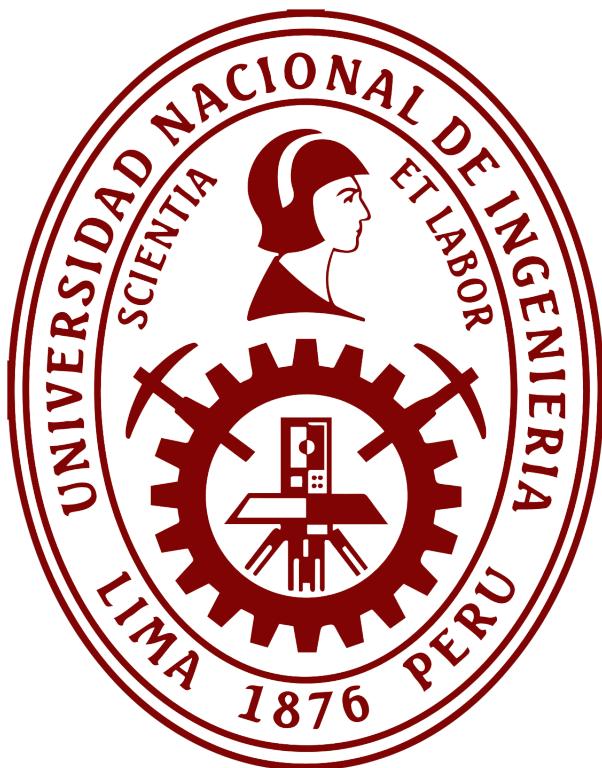


UNIVERSIDAD NACIONAL DE INGENIERÍA
ESCUELA DE CIENCIA DE LA COMPUTACIÓN



Seminario de tesis 1:
**Self Supervised Learning with Transformers networks focused
on Computer Vision tasks**

AUTOR
Cristhian Wiki Sánchez Sauñé

ASESOR
Prof. César Jesús Lara Ávila

LIMA-PERÚ

2021

...

Agradecimientos

A mis padres que me enseñaron a ser una persona humilde y dedicada a los estudios.

A todos mis compañeros de ACECOM por brindarme su hospitalidad y ganas de investigar de forma autodidacta.

A mi asesor César Lara-Avila por brindarme su opinión sincera y sugerencia de correcciones.

Y un agradecimiento especial a mi amiga Julissa García por brindarme soporte emocional y académico antes y durante la elaboración del presente trabajo.

Índice de figuras

2.1.	Visualización de la media para diferentes valores de β	11
2.2.	Arquitectura ResNet clásica	13
2.3.	Izquierda: Atención producto punto escalado. Derecha: Multi-head attention consiste de varias capas de atención ejecutándose de forma paralela. Imagen extraída de [10].	15
2.4.	Arquitectura Encoder-Decoder del modelo Transformer. Ambos están compuestos por módulos que se pueden apilar uno encima del otro varias veces, que se describe por Nx. Imagen extraída de [10].	16
2.5.	Arquitectura ViT presentada en [14]	17
2.6.	Diferentes transformaciones a una misma imagen [9].	18
2.7.	Flujo completo del proceso de destilación de un modelo	19
2.8.	Ejemplo de segmentación.	19
2.9.	Visualización de los mapas de atención en diferentes regiones de la imagen, las cuales se relacionan más con la palabra subrayada. Imagen extraída de [20]	21
2.10.	Visualización del mapa de gradientes usando GradCAM. Imagen extraída de [21].	21
3.1.	Plataforma <i>Papers With Code</i>	23
3.2.	Plataforma <i>Connected Papers</i>	23
3.3.	Comparación del rendimiento vs accuracy bajo el protocolo de clasificación lineal en ImageNet. Izquierda: rendimiento de todos los sistemas de visión SOA SSL, los tamaños de los círculos indican recuentos de parámetros del modelo; Derecha: rendimiento sobre recuentos de parámetros variados para modelos con una relación moderada (total / # parámetros). Extraído de [27].	24
3.4.	Benchmark de clasificación de imágenes en ImageNet usando <i>self-supervised</i> disponible en <i>Papers With Code</i> . Actualmente iBot [28] y EsViT [27] lideran dicho benchmark.	25
3.5.	Visualización del grafo de búsqueda para el paper [24]	25
4.1.	Representación del paso forward y backward usando el algoritmo DINO.	26

4.2. Self-distillation sin etiquetas. Ilustración de DINO en el caso de un solo par de vistas (x_1, x_2) para simplificar. El modelo pasa dos transformaciones aleatorias diferentes de una imagen de entrada a la red estudiante y a la red maestra. Ambas redes tienen la misma arquitectura pero diferentes parámetros. La salida de la red maestra se centra con una media calculada sobre el batch. Cada red genera una característica dimensional K que se normaliza con una temperatura softmax sobre la dimensión de la característica. Luego se mide su similitud con una pérdida de entropía cruzada. Aplicamos un operador stop-gradient (sg) en el maestro para propagar el gradiente solo a través del estudiante. Los parámetros de la red maestra se actualizan con una Media Móvil Exponencial (EMA) del parámetro del estudiante.	27
4.3. Algunas muestras del dataset ImageNet en baja resolución	32
4.4. Algunas muestras del dataset DAVIS.	33
4.5. Una muestra de segmentación en Pascal VOC.	33
5.1. <i>self-attention</i> de un Vision Transformer con parches 8×8 entrenados de forma <i>self-supervised</i> en una imagen de un gato con una capa.	36
5.2. <i>self-attention</i> de un Vision Transformer con parches 8×8 entrenados de forma <i>self-supervised</i> en una imagen de un gato durmiendo de forma extraña.	36
5.3. Mapas de atención de múltiples heads. Se han considerado los <i>heads</i> de la última capa de un DeiT-S/8 entrenado con DINO y se muestra el <i>self-attention</i> para la consulta del token <i>[CLS]</i> . Diferentes <i>heads</i> , materializados por diferentes colores, se enfocan en diferentes ubicaciones que representan diferentes objetos o partes.	37
5.4. Segmentación de forma supervisada versus DINO. En la parte superior, se muestran las máscaras resultantes de un DeiT-S/8 entrenado de forma supervisada y con DINO. Se muestra el mejor <i>head</i> para ambos modelos. La tabla en la parte inferior compara la similitud Jacard entre la segmentación target (usando PASCAL VOC12 [45]) y la segmentación predicha.	37
5.5. Ejemplo de segmentación de unos perros jugando.	38
5.6. Ejemplo de segmentación de un coche en la ciudad	39
5.7. Ejemplo de segmentación de un ciclista.	40
5.8. Efecto del tamaño del parche. Evaluación de k -NN en función del rendimiento para diferentes tamaños de parches de entrada con ViT-B y DeiT-S.	42
5.9. Accuracy Top-1 en el conjunto de validación de ImageNet con el clasificador k-NN. Comparación entre el desempeño de la red maestra momentum y el estudiante durante el entrenamiento. En la tabla inferior se hace una comparación entre las variantes de la red maestra. Usar <i>momentum encoder</i> conduce al mejor rendimiento, pero no es la única opción viable.	43
5.10. Estudio del colapso. Izquierda: Evolución de la entropía objetivo de la red maestra. Derecha: Evolución de la divergencia KL entre las salidas de la red maestra y el estudiante.	43

Índice general

1. Generalidades	9
1.1. Problemática	9
1.2. Objetivos	10
1.2.1. Objetivo General	10
1.2.2. Objetivos Específicos	10
1.3. Alcances	10
1.3.1. Hipótesis	10
1.3.2. Justificación	10
2. Marco Conceptual	11
2.1. Media Móvil Exponencial (EMA)	11
2.2. Promedio Polyak-Ruppert	12
2.3. Divergencia Kullback-Leibler (KL)	12
2.4. Redes Neuronales Convolucionales	12
2.4.1. ResNet	12
2.5. Redes Transformers	13
2.5.1. Mecanismos de Atención	13
2.5.2. Codificación posicional	14
2.5.3. Capa de normalización	14
2.5.4. Red totalmente conectada	14
2.5.5. Atención producto punto escalado	14
2.5.6. Multi-head attention	15
2.5.7. Arquitectura Transformer	15
2.6. Vision Transformer (ViT)	17
2.7. Self Supervised Learning (SSL)	17
2.7.1. Contrastive Learning	17

2.8. Destilación de modelos	18
2.9. Momentum Encoder	19
2.10. Métricas	19
2.10.1. Intersection-Over-Union (IoU, Índice de Jaccard)	19
2.11. Explicabilidad e Interpretabilidad	20
2.11.1. Interpretabilidad	20
2.11.2. Explicabilidad	20
2.11.3. Importancia	20
2.12. Mapas de atención	21
3. Revisión de la Literatura	22
3.1. Objetivo de la revisión	22
3.1.1. Cadenas de búsqueda	22
3.1.2. Preguntas de la Revisión	23
3.2. Resultados de la revisión	23
4. Métodología y Herramientas	26
4.1. Metodología	26
4.1.1. SSL con destilación del conocimiento	26
4.1.2. Implementación y protocolos de evaluación	30
4.2. Herramientas	32
4.2.1. Herramientas computacionales	32
4.2.2. Datasets	32
5. Experimentación y Resultados	34
5.1. Comparando otros enfoques SSL en ImageNet	34
5.1.1. Comparando con la misma arquitectura	34
5.1.2. Comparación entre arquitecturas	34
5.2. Propiedades de ViT entrenado con SSL	35
5.2.1. Descubriendo el diseño semántico de escenas	35
5.3. Despliegue	40

5.4. Estudio de ablación de DINO	41
5.4.1. Importancia de los diferentes componentes	41
5.4.2. Impacto de la elección de la red maestra	42
5.4.3. Evitando el colapso	43
5.4.4. Requerimientos de cómputo	44
5.4.5. Entrenando con pequeños batchs	44
6. Conclusiones y Trabajos Futuros	46
6.1. Conclusiones	46
6.2. Trabajos Futuros	46
Bibliografía	47

Generalidades

1.1. Problemática

Actualmente las redes Convolucionales (CNN) se desempeñan bien en muchas tareas de visión, con accuracy's que hasta hace un par de años parecían ser muy distantes. Estas tareas van desde el reconocimiento de imágenes a gran escala [1], como también a objetivos más específicos como la segmentación de imágenes médicas [2] de zonas específicas que antes solo eran posibles gracias a profesionales del área.

Cabe recalcar que estas CNN también se desenvuelven con facilidad en otros criterios no menos importantes como vienen a ser la facilidad de despliegue y una baja latencia de tiempo de inferencia [3, 4]. También han demostrado tener buenos métodos de explicabilidad respecto a las características que logran visualizar internamente [5].

Todos los logros de las CNN's anteriormente mencionados son bajo un enfoque supervisado utilizando grandes volúmenes de datos **etiquetados** [1]. Sin embargo, el enfoque *self-supervised* ha empezado a ganar fuerza estos últimos 2 años debido a que tiende a ser el futuro del Deep Learning [6], recalculando la escasez de los datos de entrenamiento y que muchas veces tiende a ser mucho más costoso su obtención comparado con el propio entrenamiento en sí. Entre las novedosos métodos aplicados a CNN's bajo el enfoque *self-supervised* podríamos mencionar a BYOL [7], MoCo [8] y SimCLR [9].

A pesar de todos los méritos ganados de las CNN's, se ha demostrado experimentalmente que tienen una saturación o bottleneck que los hacen dependientes de los datos [1] como también de su arquitectura, lo cual lleva a los investigadores a plantearse otras opciones. Y es aquí donde entran en escenario los Transformers, inicialmente planteados en 2017 [10] para tareas de NLP y que en los últimos años han logrado escalar con facilidad en otros campos como las GAN's [11], detección de objetos [12] y hasta segmentación [13], todo desde un enfoque supervisado.

El presente proyecto de tesis consiste en analizar más a fondo el desempeño de los Transformers en diversas tareas de Computer Vision pero desde un enfoque *self-supervised*. En ese contexto se describen las preguntas que motivan la presente investigación: ¿Podrán hacerlo mejor que las CNN's?, ¿Podríamos comprender lo que visualiza el Transformer?

1.2. Objetivos

1.2.1. Objetivo General

Evaluar el desempeño del algoritmo DINO frente a otros algoritmos *self-supervised* actuales basados en Transformers para diversas tareas de Computer Vision.

1.2.2. Objetivos Específicos

- O1) Analizar la explicabilidad e interpretabilidad de DINO frente a otros métodos.
- O2) Realizar un estudio de ablación del algoritmo DINO, analizando las consecuencias de modificar diversos hiperparámetros tales como el *batchsize*, función de pérdida, etc.

1.3. Alcances

1.3.1. Hipótesis

Es posible que la arquitectura Vision Transformer (ViT) y variantes entrenadas bajo un enfoque *self-supervised*, tengan un mejor desempeño y explicabilidad en tareas de Computer Vision, comparado con:

- CNNs, entrenadas bajo un enfoque supervisado
- ViT entrenado bajo un enfoque supervisado
- CNNs, entrenadas bajo un enfoque *self-supervised*

1.3.2. Justificación

Las redes Transformers carecen de los sesgos perceptuales de las CNN's, lo que le da un espectro más grande de análisis, pudiendo encontrar patrones muy elaborados debido a los mecanismos de atención que les permite prestar más atención a las zonas importantes de la imagen y menos atención al ruido de fondo.

Dosovitskiy et al. [14] demostraron que ViT entrenado en grandes datasets de manera supervisada supera a Big Transfer [1] (SOA 2020 basado en CNN's) en la tarea de clasificación de imágenes. Además encontraron patrones representativos (basados en los **heads attention**) que atienden a regiones de la imagen que son semánticamente relevantes para la clasificación, los cuales guardan relación con el campo receptivo de las CNN's. Los autores también exploran un enfoque *self-supervised* basado en la predicción de parches enmascarados, imitando la tarea de modelado de lenguaje enmascarado utilizada en BERT [15]. Con entrenamiento *self-supervised*, el modelo más reducido ViT-B/16 logró alcanzar un accuracy del 79,9% en ImageNet, una mejora significativa del 2% al entrenamiento desde cero, pero todavía un 4% por detrás del entrenamiento supervisado.

Para una revisión más detallada del código usado para la presente investigación, revisar [mi repositorio](#)¹.

¹<https://github.com/HiroForYou/Vision-2-Transformers>

Marco Conceptual

En el presente capítulo se hace un repaso por las principales definiciones que se consideró importante para una mejor comprensión del trabajo. Se han obviado aspectos más básicos.

2.1. Media Móvil Exponencial (EMA)

Este algoritmo es uno de los algoritmos más importantes actualmente en uso. Se usa en diversas áreas tales como series temporales, procesamiento de señales y hasta redes neuronales. El motivo de su fama es básicamente porque reduce el ruido ('suaviza') de los datos. La forma en que logra esto es esencialmente ponderando el número de observaciones y utilizando su promedio.

Ejemplo:

Digamos que queremos calcular la media móvil de la temperatura en los últimos N días. Lo que hacemos es comenzar con 0, y luego todos los días, vamos a combinarlo β veces con el valor acumulado hasta ahora (V_{t-1}) más $1 - \beta$ veces la temperatura de ese día (θ_t), entonces tenemos:

$$V_t = \beta * V_{t-1} + (1 - \beta) * \theta_t$$

En la figura 2.1, la línea roja muestra el promedio ponderado si β es 0.9 y la línea verde muestra el promedio ponderado si β es 0.98. Al elegir el valor de t y β , controlamos cuánta *importancia* dar durante los últimos N días en el cálculo del promedio. La fórmula $1/(1 - \beta)$ es una forma sencilla de recordar cómo se aplica esta *importancia*. Con β igual a 0.9, promediaremos los últimos 10 días de temperatura, mientras que con β igual a 0.98, tendremos en cuenta los últimos 50 días de temperatura y, por lo tanto, promediaremos más de los 50 días de temperatura.

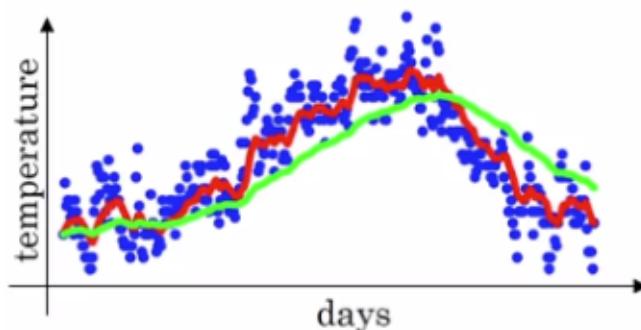


Figura 2.1: Visualización de la media para diferentes valores de β .

2.2. Promedio Polyak-Ruppert

El proceso de entrenamiento de las redes neuronales es un proceso de optimización desafiante que a menudo puede no converger. Esto puede significar que el modelo al final del entrenamiento puede no ser un conjunto de pesos estable o de mejor rendimiento para usar como modelo final.

Un enfoque para abordar este problema es usar un promedio de los pesos del modelo en diferentes instantes casi al final del entrenamiento. Esto se denomina promedio de Polyak-Ruppert y originalmente fue presentado en [16].

Se ha demostrado que el promedio de Polyak-Ruppert mejora la convergencia del estándar SGD. Alternativamente, se puede usar una Media Móvil Exponencial (EMA) sobre los parámetros, dando mayor peso al valor del parámetro más reciente.

2.3. Divergencia Kullback-Leibler (KL)

En términos simples, la divergencia KL es una medida de cuán diferente es una distribución de probabilidad específica de una distribución de referencia.

A continuación se muestra la definición formal de la divergencia KL para dos distribuciones de probabilidad discretas P y Q dentro del espacio χ^n

$$D_{KL}(P \parallel Q) = \sum_{x \in \chi} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

La divergencia se usa a menudo como un término que se agrega a la función de coste de redes Autoencoder y GANs para poder generar resultados sintéticos más reales.

2.4. Redes Neuronales Convolucionales

Una red neuronal convolucional o CNN es un algoritmo de Deep Learning que puede tomar una imagen de entrada, asignar importancia (pesos y sesgos aprendibles) a varios aspectos/objetos en la imagen y ser capaz de diferenciar uno del otro. El pre-procesamiento requerido en una CNN es mucho menor en comparación con otros algoritmos de clasificación. Mientras que en los métodos primitivos los kernels son diseñados a mano, con suficiente entrenamiento, las CNN's tiene la capacidad de aprender estos kernels.

2.4.1. ResNet

Abreviatura de Residual Network, es un tipo específico de red neuronal convolucional que fue introducida en 2015 por Kaiming He, Xiangyu Zhang, Shaoqing Ren y Jian Sun en su artículo *Deep Residual Learning for Image Recognition* [17]. Estas redes hacen uso de conexiones de salto (ver figura 2.2) para combatir el problema del desvanecimiento del gradiente y pérdida de información contextual, un problema muy común en redes muy profundas. Los modelos resNet fueron extremadamente exitosos, ganando muchas competencias en su momento.

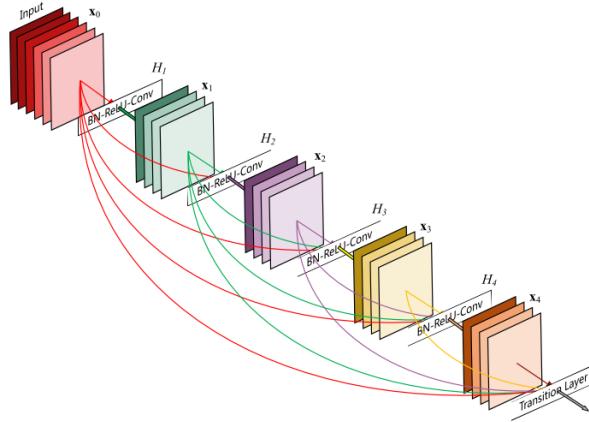


Figura 2.2: Arquitectura ResNet clásica.

2.5. Redes Transformers

2.5.1. Mecanismos de Atención

Los mecanismos de atención nacieron con la intención de resolver el problema de pérdida de información en secuencias largas para arquitecturas tipo Encoder-Decoder basadas en RNN's. Un mecanismo de atención tiene las siguientes entradas:

- Los ítems a atender, representados por los vectores $\{a_1, a_2, \dots, a_n\}$
- El ítem que presta atención a los ítems anteriores, representado por el vector x

Donde primero se calcula la relación entre x y $\{a_1, a_2, \dots, a_n\}$, resultando un vector de puntuaciones de tamaño n , donde una alta puntuación indica una fuerte relación. Esto se implementa a través de una función de atención denotada por f :

$$f(x, a_i) = x^T a_i = s_i$$

Posteriormente se normaliza por una función softmax para poder conseguir los pesos de atención:

$$\beta_{i,j} = \frac{e^{s_i}}{\sum_{j=1}^n e^{s_j}}$$

Finalmente, los pesos se usan para calcular la suma ponderada de los ítems atendidos, resultando vector de atención c .

Definimos el vector de atención como la combinación lineal de los ítems a ser atendidos $\{a_1, a_2, \dots, a_n\}$ y los pesos normalizados originados de la interacción entre el ítem que presta atención x y cada elemento que se atiende a_i .

$$c_j = \beta_{j,1}a_1 + \beta_{j,2}a_2 + \dots + \beta_{j,n}a_n$$

2.5.2. Codificación posicional

El Transformer aplica codificación posicional para preservar la información del orden de las palabras, ya que el orden es importante para entender la semántica de un texto. En la codificación posicional se asigna cada posición a una único vector de dimensión d_{model} . Por ejemplo la función trigonométrica para la codificación posicional presentada inicialmente para el Transformer fue :

$$PE_{pos,2i} = \sin(pos/10000^{2i/d_{model}})$$

$$PE_{pos,2i+1} = \cos(pos/10000^{2i/d_{model}})$$

Donde $PE_{pos,j}$ representa el valor del j_{th} componente del vector de la pos_{th} posición.

Para nuestros experimentos usaremos la codificación posicional relativa como una mejor alternativa.

2.5.3. Capa de normalización

Nos ayuda a evitar el problema de la extrema sensibilidad en las capas altas, que genera un entrenamiento inestable. Para ello el Transformer utiliza una capa de normalización que actúa sobre la entrada de cada capa, aplicando la media y desviación estandárd.

2.5.4. Red totalmente conectada

El Transformer contiene una red totalmente conectada (también conocida como red Feed Forward o Fully Connected) que comprende 2 capas lineales y una función de activación ReLU.

2.5.5. Atención producto punto escalado

Los bloques de construcción del Transformer son unidades de *Atención producto punto escalado*. Cuando se pasa una oración a un modelo Transformer, los pesos de atención se calculan entre cada token simultáneamente. La unidad de atención produce embeddings para cada token conteniendo información sobre el token en sí junto con una combinación ponderada de otros tokens relevantes, cada uno ponderado por su peso de atención.

El cálculo de atención para todos los tokens se puede expresar como el cálculo de una matriz grande utilizando la función softmax, que es útil para el entrenamiento debido a las optimizaciones computacionales en las operaciones de la matriz. Las matrices Q , K y V se definen como las matrices en las que las i filas son vectores q_i , k_i y v_i respectivamente.

De forma muy análoga a la ecuación presentada en la definición de mecanismo de atención, tendremos la siguiente ecuación:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

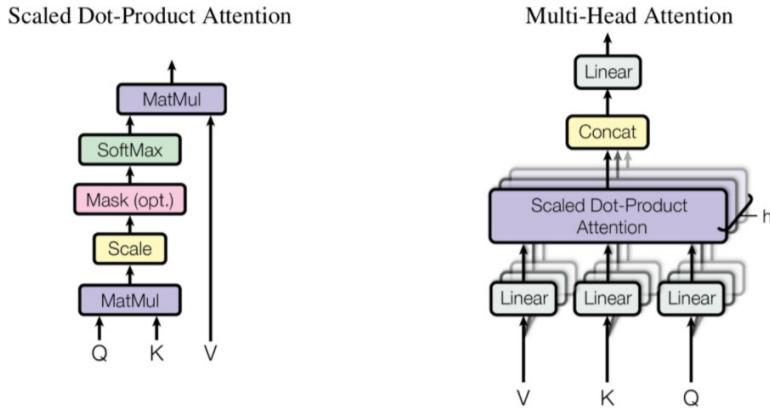


Figura 2.3: Izquierda: Atención producto punto escalado. Derecha: Multi-head attention consiste de varias capas de atención ejecutándose de forma paralela. Imagen extraída de [10].

2.5.6. Multi-head attention

La terna de matrices (W_Q, W_K, W_V) se denomina *head attention*, y cada capa en un modelo Transformer tiene varias *head attention*. Si bien cada *head attention* atiende a los tokens que son relevantes para cada token, con varias *head attention*, el modelo puede hacer esto para diferentes definiciones de **importancia**. Además, el campo de influencia que representa la importancia puede dilatarse progresivamente en capas sucesivas. Muchas *head attention* codifican relaciones de importancia que son significativas para los humanos. Por ejemplo, algunas *head attention* pueden prestar atención principalmente a la siguiente palabra, mientras que otras prestan atención principalmente a los objetos directos de los verbos. Las salidas de la capa de atención se concatenan para pasar a las capas de la red completamente conectada.

2.5.7. Arquitectura Transformer

2.5.7.1. Encoder

Cada Encoder consta de dos componentes principales: un mecanismo de *multi-head attention* y una red totalmente conectada. El mecanismo de *multi-head attention* acepta codificaciones de entrada del Encoder anterior y pondera su importancia entre sí para generar codificaciones de salida. La red neuronal completamente conectada procesa además cada codificación de salida de forma individual. Estas codificaciones de salida se pasan luego al siguiente Encoder como su entrada, así como a los Decoders.

El primer Encoder toma información posicional y también información de los embeddings de la secuencia de entrada como entrada, en lugar de codificaciones. La información de posición es necesaria para que el Transformer haga uso del orden de la secuencia, porque ninguna otra parte del Transformer hace uso de ella.

2.5.7.2. Decoder

Cada Decoder consta de tres componentes principales: un mecanismo *multi-head attention* para sus propias codificaciones, un mecanismo *multi-head attention* para las codificaciones (resultado

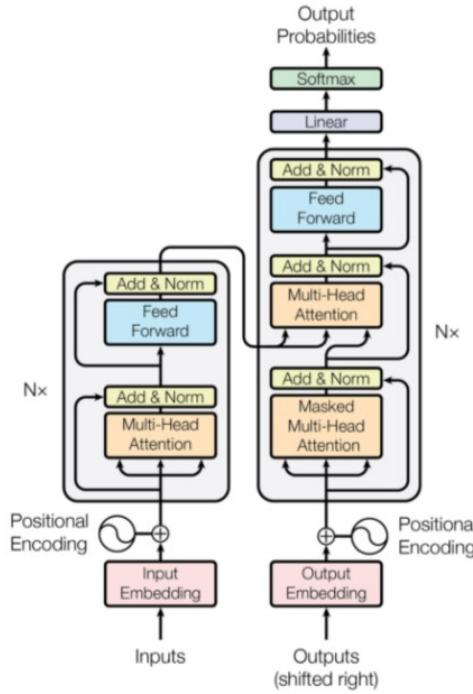


Figura 2.4: Arquitectura Encoder-Decoder del modelo Transformer. Ambos están compuestos por módulos que se pueden apilar uno encima del otro varias veces, que se describe por Nx . Imagen extraída de [10].

del Encoder) y una red totalmente conectada.

Al igual que el primer Encoder, el primer Decoder toma información posición posicional y también de los embeddings de la secuencia de salida como entrada, en lugar de las codificaciones. El Transformer no debe utilizar la salida actual o futura para predecir una salida, por lo que la secuencia de salida debe estar parcialmente enmascarada para evitar este flujo de información inversa. El último Decoder es seguido por una transformación lineal final y una capa softmax, para producir las probabilidades de salida sobre el vocabulario.

2.6. Vision Transformer (ViT)

Vision Transformer, o ViT, es un modelo para la clasificación de imágenes que emplea una arquitectura similar al Transformer original [10] sobre parches de una imagen. La imagen se divide en parches de tamaño fijo, cada uno de ellos se proyectan linealmente, se añaden embeddings de posición y la secuencia de vectores resultante se alimenta a un Encoder de Transformer estándar. Para realizar la clasificación, se utiliza el enfoque estándar de agregar un *token de clasificación* extra que se pueda aprender a la secuencia.

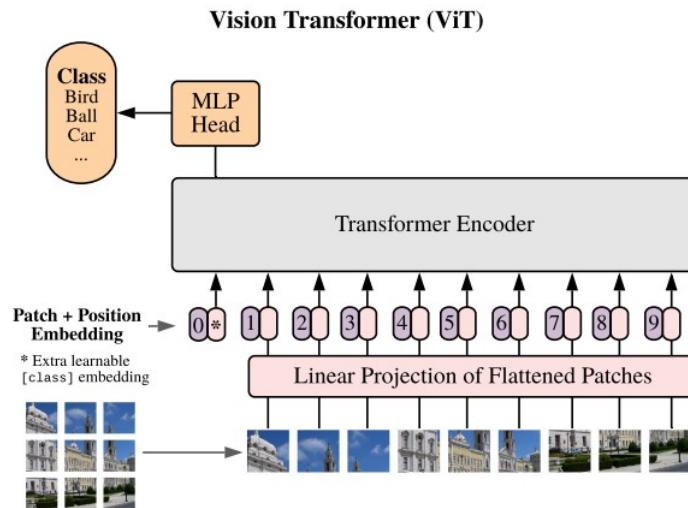


Figura 2.5: Arquitectura ViT presentada en [14]

2.7. Self Supervised Learning (SSL)

Self Supervised Learning es una técnica de aprendizaje relativamente reciente que a diferencia del aprendizaje supervisado, no requiere ningún dato etiquetado. En su lugar, crea pseudoetiquetas autodefinidas como señales de supervisión y aprende representaciones, que luego se utilizan en tareas posteriores.

Self Supervised Learning utiliza muchas más señales de supervisión que el aprendizaje supervisado, y enormemente más que el aprendizaje de refuerzo. Por eso llamarlo *sin supervisión* es totalmente engañoso. Por eso también se puede aprender más conocimiento sobre la estructura del mundo a través de Self Supervised Learning que de los otros dos paradigmas: los datos son ilimitados, y la cantidad de retroalimentación proporcionada por cada ejemplo es enorme. Self Supervised Learning ha tenido un enorme éxito en NLP. Por ejemplo, el modelo BERT [15] y técnicas similares producen excelentes representaciones de texto.

2.7.1. Contrastive Learning

Es un paradigma perteneciente a SSL que tiene como objetivo agrupar muestras similares más cercanas y muestras diferentes lejos unas de otras.

La principal motivación para Contrastive Learning proviene de los patrones de aprendizaje

humanos. Los humanos reconocen objetos sin recordar todos los pequeños detalles. Por ejemplo, es fácil para nosotros mirar una imagen y encontrar una tabla en ella basada en el color, la forma y ciertas otras características. En términos generales, creamos algún tipo de representaciones en nuestras mentes, y luego las usamos para reconocer nuevos objetos. El objetivo principal de SSL y Contrastive Learning son, respectivamente, crear y generalizar estas representaciones.

Las tareas *self-supervised* se denominan tareas de pretexto y tienen como objetivo generar automáticamente pseudoetiquetas. Hay muchas formas diferentes de crear tareas pretexto, como (ver figura 2.6):

- Aumento de color
- Rotación y recorte de imágenes
- Otras transformaciones geométricas

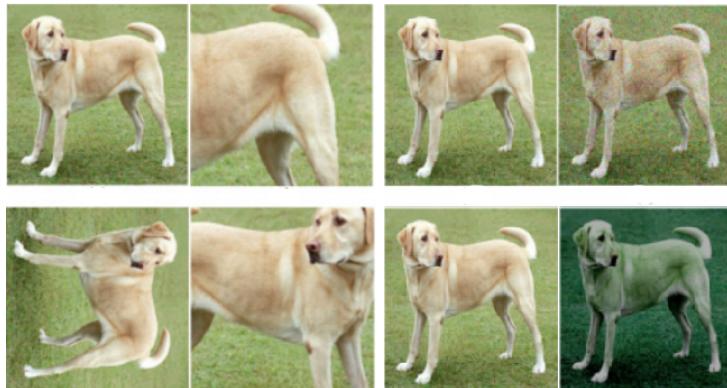


Figura 2.6: Diferentes transformaciones a una misma imagen [9].

Tres de los trabajos más populares en este dominio son:

- SimCLR [9]: un algoritmo creado por Google Research. Para su entrenamiento se han utilizado TPUs, con un tamaño de lote de 4096 o más, que está lejos de las GPU comerciales ordinarias.
- SwAV [18]: el reciente algoritmo de vanguardia (SOA) publicado por Facebook Research.
- MoCo [19]: también fue publicado por Facebook Research. Viene con la simplicidad y potencia de SimCLR pero con menos requisitos de cómputo.

También existe BYOL [7] (publicado por investigadores de DeepMind) como un enfoque no enteramente Contrastive Learning. Mientras que los trabajos anteriores utilizan ejemplos positivos y negativos en sus funciones de pérdida, BYOL utiliza sólo ejemplos positivos.

2.8. Destilación de modelos

Es un procedimiento para la compresión de un modelo, en la que se entrena un modelo pequeño (estudiante) para que intente replicar la salida de un gran modelo pre-entrenado (maestro). El conocimiento se transfiere del modelo de maestro al estudiante al minimizar una función de pérdida, dirigida a coincidir con la salida de la red maestra (soft logits), así como las etiquetas verdaderas.

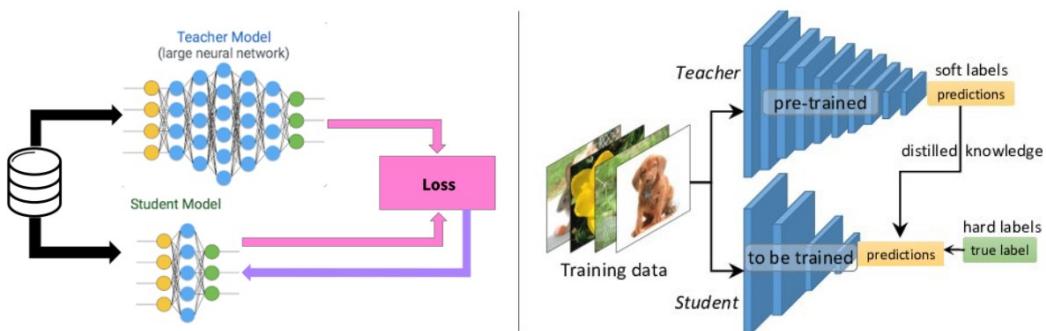


Figura 2.7: Flujo completo del proceso de destilación de un modelo

2.9. Momentum Encoder

Inicialmente propuesto en la primera versión de MoCo [19], no es más que aplicar el Media Móvil Exponencial sobre los parámetros entrenables θ de dos redes neuronales para poder reducir el 'ruido' de dichos parámetros y así obtener un modelo más robusto.

Por ejemplo, sean 2 redes neuronales s (red estudiante) y t (red maestra), y queremos obtener una versión Momentum Encoder de t , entonces:

$$\theta_t \leftarrow l * \theta_t + (1 - l) * \theta_s$$

Donde l es un hiperparámetro que podremos configurar de acuerdo a nuestros requerimientos.

2.10. Métricas

2.10.1. Intersection-Over-Union (IoU, Índice de Jaccard)

El Intersection-Over-Union (IoU), también conocido como el Índice Jaccard, es una de las métricas más utilizadas en la segmentación semántica. IoU se calcula como el área de superposición entre la segmentación predicha y la segmentación target dividida por el área de unión entre la segmentación predicha y la segmentación target, como se muestra en la figura 2.8. Esta métrica varía de 0 a 1 (0 a 100 %) con 0 que significa que no hay superposición y 1 que significa una segmentación perfectamente superpuesta.

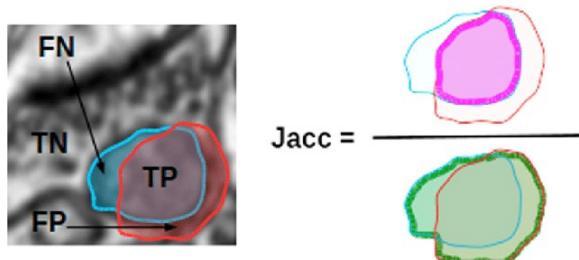


Figura 2.8: Ejemplo de segmentación.

2.11. Explicabilidad e Interpretabilidad

En el contexto de Machine Learning e IA, la explicabilidad y la interpretabilidad a menudo se utilizan indistintamente. Si bien están muy estrechamente relacionados, vale la pena desentrañar las diferencias, aunque solo sea para ver qué tan complicadas pueden ser las cosas una vez que comience a profundizar en los sistemas de Machine Learning.

2.11.1. Interpretabilidad

La interpretabilidad se trata de la medida en que se puede observar una causa y un efecto dentro de un sistema. O, para decirlo de otra manera, es la medida en que puede predecir lo que va a suceder, dado un cambio en la entrada o los parámetros algorítmicos. Es poder mirar un algoritmo y decir sí, *puedo ver lo que está sucediendo aquí*.

2.11.2. Explicabilidad

La explicabilidad, por su parte, es la medida en que la mecánica interna de un sistema de Deep Learning puede explicarse en términos humanos. Es fácil pasar por alto la sutil diferencia con la interpretabilidad, pero considéralo así: la interpretabilidad se trata de ser capaz de discernir la mecánica sin necesariamente saber por qué. La explicabilidad es ser capaz de explicar literalmente lo que está sucediendo.

Digamos que estamos haciendo un experimento científico en la universidad. El experimento puede ser interpretable en la medida en que puedes ver lo que estás haciendo, pero solo es realmente explicable una vez que profundizas en la química detrás de lo que puedes ver que sucede. Eso puede ser un poco crudo, pero sin embargo es un buen punto de partida para pensar en cómo los dos conceptos se relacionan entre sí.

2.11.3. Importancia

La tecnología ciertamente se puede poner en uso dudoso, hay muchas maneras en que puede producir resultados pobres, discriminatorios, sin intención de causar daño.

A medida que dominios como la atención médica buscan implementar sistemas de IA y Deep Learning, donde las cuestiones de responsabilidad y transparencia son particularmente importantes, si no podemos ofrecer adecuadamente una mejor interpretabilidad y, en última instancia, explicabilidad en nuestros algoritmos, limitaremos seriamente el impacto potencial de la IA.

Pero aparte de las consideraciones legales y profesionales que deben hacerse, también hay un argumento de que mejorar la interpretabilidad y la explicabilidad son importantes incluso en escenarios comerciales. Comprender cómo funciona realmente un algoritmo puede ayudar a alinear mejor las actividades de los científicos y analistas de datos con las preguntas y necesidades clave de su organización.

2.12. Mapas de atención

Es una técnica de interpretabilidad que consiste en un matriz escalar que representa la importancia relativa que da la red Transformer en diferentes ubicaciones espaciales 2D con respecto a la tarea objetivo. Es decir, es un mapa de atención es una cuadrícula de números que indica qué ubicaciones 2D son importantes para una tarea. Las ubicaciones importantes corresponden a números más grandes y suele representarse como un mapa de calor. Esta matriz se extrae directamente del módulo *multi-head attention* y se deja a criterio del investigador el hacer uso de otros módulos del Transformer para intentar obtener una matriz más precisa.

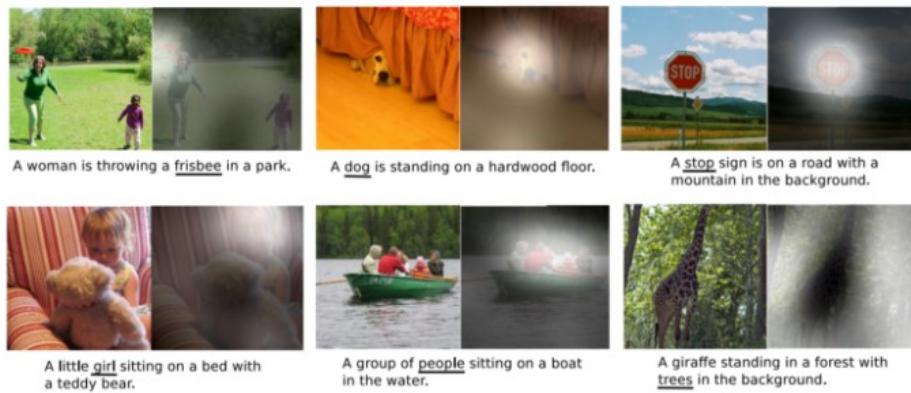


Figura 2.9: Visualización de los mapas de atención en diferentes regiones de la imagen, las cuales se relacionan más con la palabra subrayada. Imagen extraída de [20]

Existen otras técnicas de interpretabilidad basadas en mapas de gradientes (GradCAM) [21] que ya vienen integrados en paquetes como **Captum**, que es una extensión de Pytorch para poder visualizar características aprendidas por las diferentes capas de nuestra red neuronal.

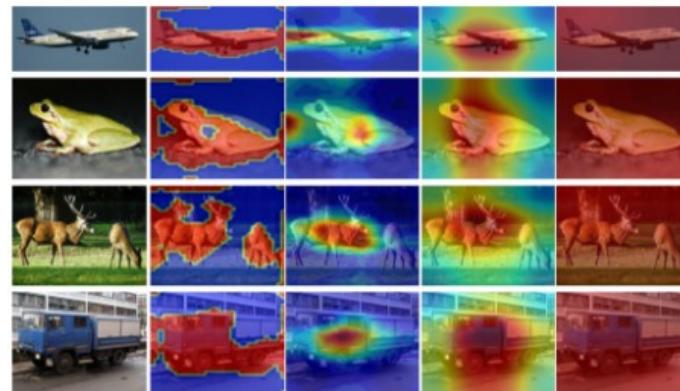


Figura 2.10: Visualización del mapa de gradientes usando GradCAM. Imagen extraída de [21].

Revisión de la Literatura

En el presente capítulo se desarrollan los criterios de búsqueda utilizados para identificar el estado del arte (SOA), y las arquitecturas alternativas desarrolladas con respecto a tareas de visión usando Transformers y con enfoque *self-supervised*.

3.1. Objetivo de la revisión

El objetivo de la revisión es identificar y seleccionar los papers desarrollados para resolver la problemática planteada.

Para la revisión hay unos criterios importantes que se tuvieron en cuenta:

- En cuanto a la búsqueda de los SOA se usó la plataforma **Papers With Code** como punto de partida, por su facilidad de uso y además de brindar cuadros comparativos muy interesantes.
- Como extensión adicional, se usó la plataforma **Connected Papers** para búsqueda de papers más especializados y que no son tan fácilmente accesibles mediante *Papers With Code*.
- El criterio de búsqueda de *Connected Papers* se basa de acuerdo a la similitud de los papers. Eso significa que incluso los documentos que no se citan directamente entre sí pueden estar fuertemente conectados y muy estrechamente posicionados.
- Tanto *Connected Papers* y *Papers With Code* son ampliamente usados por la comunidad científica especializada en Deep Learning y Data Science.
- También se consultó material adicional en **Scopus** y **Arxiv**.
- Para búsquedas avanzadas, se usarán criterios de filtrado específicos, como las cadenas de búsqueda.

3.1.1. Cadenas de búsqueda

Las consultas hechas fueron las siguientes:

- TITLE-ABS-KEY ((self supervised learning OR SSL) and (Convolutional Neural Network OR CNN))
- TITLE-ABS-KEY ((self supervised learning OR SSL) and (Vision Transformer OR ViT))
- TITLE-ABS-KEY ((Vision Transformer OR ViT) AND (computational complexity))
- TITLE-ABS-KEY (((Vision Transformer OR ViT) AND (Convolutional Neural Network OR CNN)) AND (Interpretability OR Explainability))

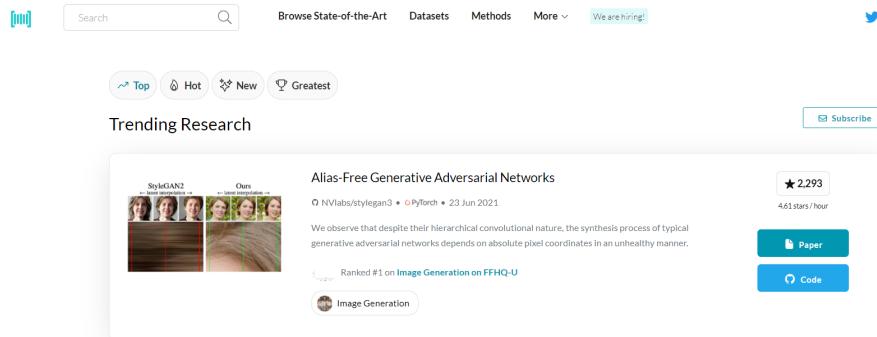


Figura 3.1: Plataforma *Papers With Code*

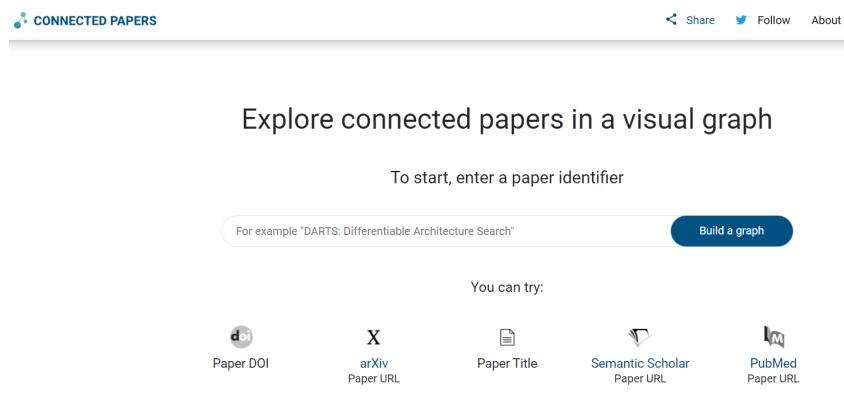


Figura 3.2: Plataforma *Connected Papers*

3.1.2. Preguntas de la Revisión

- ¿Cuáles son las arquitecturas de Vision Transformer más actuales que utilizan el enfoque self-supervised?
- ¿Cuál es la forma más optima de entrenar Vision Transformers?
- ¿Qué arquitectura tiene una mejor explicabilidad respecto a los patrones aprendidos ?

3.2. Resultados de la revisión

Se encontraron alrededor de 10 artículos de los cuales 6 se han considerado como relevantes al momento de responder las preguntas de la revisión.

Para intentar evaluar la explicabilidad de las CNN's y Vision Transformers, en el trabajo *Are Convolutional Neural Networks or Transformers more like human vision?* [22] se hace un estudio detallado (desde el enfoque supervisado) de las similitudes que comparten estas arquitecturas respecto a la visión humana, demostrando que los Vision Transformers son en gran medida más consistentes con los errores humanos. Esto posiblemente podría explicarse por la naturaleza de los modelos de atención que permiten enfocarse en la parte de la imagen que es importante para la tarea dada y descuidar el fondo ruidoso para hacer predicciones.

Para poder optimizar el costo computacional que implica entrenar un Vision Transformer

y a la vez teniendo en cuenta la necesidad de usar pocos datos se encontró un trabajo de parte de un equipo de **Facebook AI Research** titulado *Data-efficient image Transformers (DeiT)* [23] en donde optimizan considerablemente la red ViT para ser entrenado en pequeños datasets, pero desde un enfoque supervisado.

También encontramos otro brillante trabajo del mismo equipo, titulado *Emerging Properties in Self-Supervised Vision Transformers* [24] en donde haciendo uso de la arquitectura DeiT proponen un algoritmo llamado DINO para poder entrenar de forma *self-supervised*, al mismo tiempo que lograban conseguir una mejor explicabilidad de los patrones que aprenía el modelo.

Posterior a esto, publicaron otro trabajo titulado *XCI{T}: Cross-Covariance Image Transformers* [25], donde introducen un método de covarianza cruzada que permite a DINO poder manejar imágenes de muy alta resolución, mejorando con creces la explicabilidad y escalabilidad del mismo.

Por otro lado, otro equipo de **Microsoft AI Research** publicó recientemente un trabajo titulado *Swin Transformer: Hierarchical Vision Transformer using Shifted Windows* [26] en donde hacen uso de una variante de DeiT llamada *Swin Transformer* que mediante una técnica que denominaron *ventanas deslizantes* buscan reducir el costo cuadrático que inicialmente tenían los Vision Transformers, pero manteniendo el enfoque supervisado

Y al igual que el equipo de **Facebook AI Research**, publicaron casi inmediatamente la variante *self-supervised* en un paper titulado *Efficient Self-Supervised Vision Transformers (EsViT)* [27] en donde se basan de los trabajos anteriormente mencionados, añadiendo mejoras a ViT, Swin Transformer y al algoritmo DINO, logrando posicionarse como el SOA actual al momento de redactarse esta tesis. Sin embargo, dichos investigadores han observado que se logra perder cierta explicabilidad, y que podría mejorarse en el futuro.

A continuación se muestra un cuadro comparativo de las diferentes arquitecturas antes mencionadas:

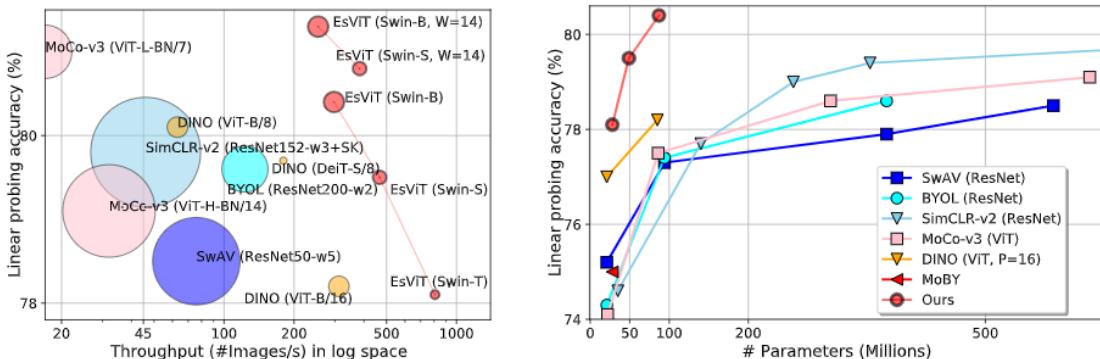


Figura 3.3: Comparación del rendimiento vs accuracy bajo el protocolo de clasificación lineal en ImageNet. **Izquierda:** rendimiento de todos los sistemas de visión SOA SSL, los tamaños de los círculos indican recuentos de parámetros del modelo; **Derecha:** rendimiento sobre recuentos de parámetros variados para modelos con una relación moderada (total / # parámetros). Extraído de [27].

Cabe recalcar que todos estos papers tiene una antigüedad no mayor a 1 año y que desde el momento de la publicación de ViT [14] se han producido alrededor de **más de 200 publicaciones** relacionadas al mismo en diferentes conferencias como IA, por lo que es un tema de especial interés para mí.

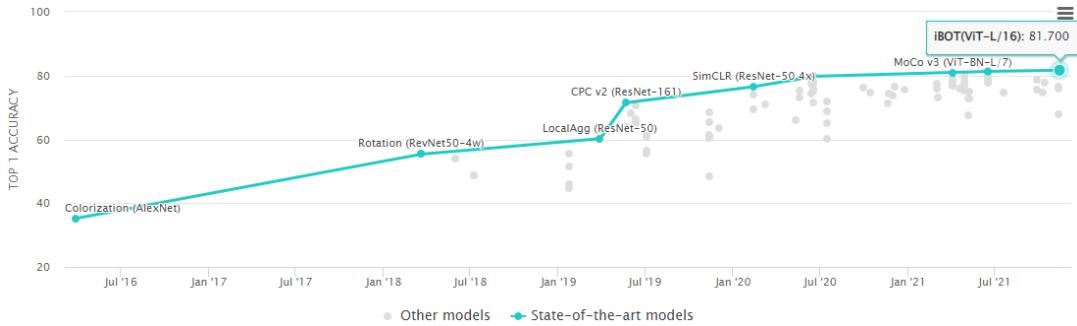


Figura 3.4: Benchmark de clasificación de imágenes en ImageNet usando *self-supervised* disponible en [Papers With Code](#). Actualmente **iBot** [28] y **EsViT** [27] lideran dicho benchmark.

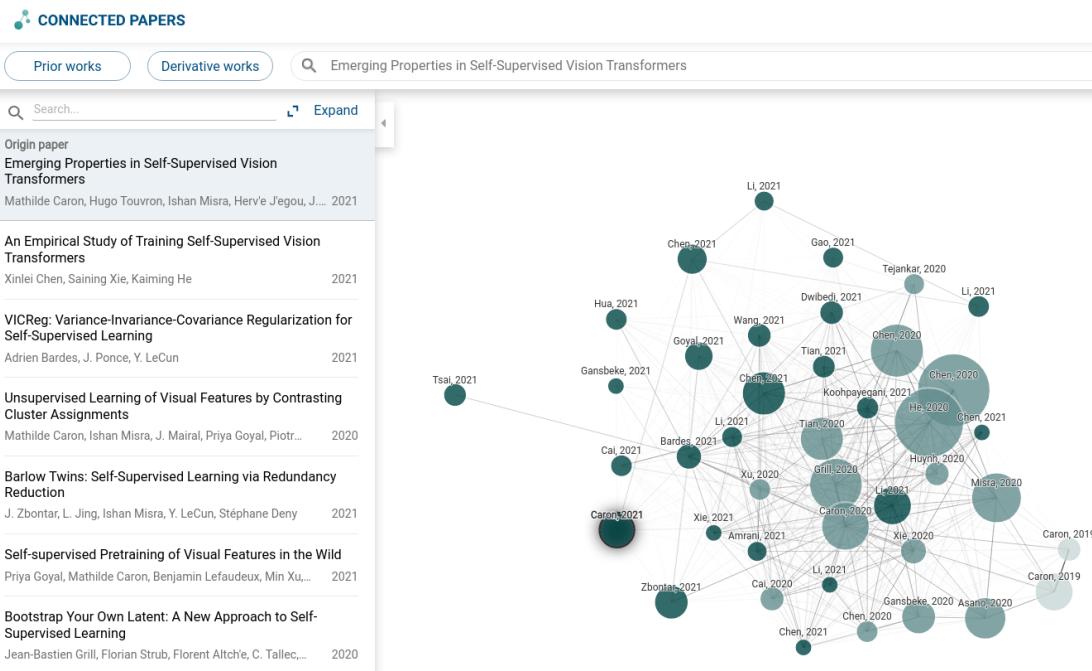


Figura 3.5: Visualización del grafo de búsqueda para el paper [24]

Métodología y Herramientas

4.1. Metodología

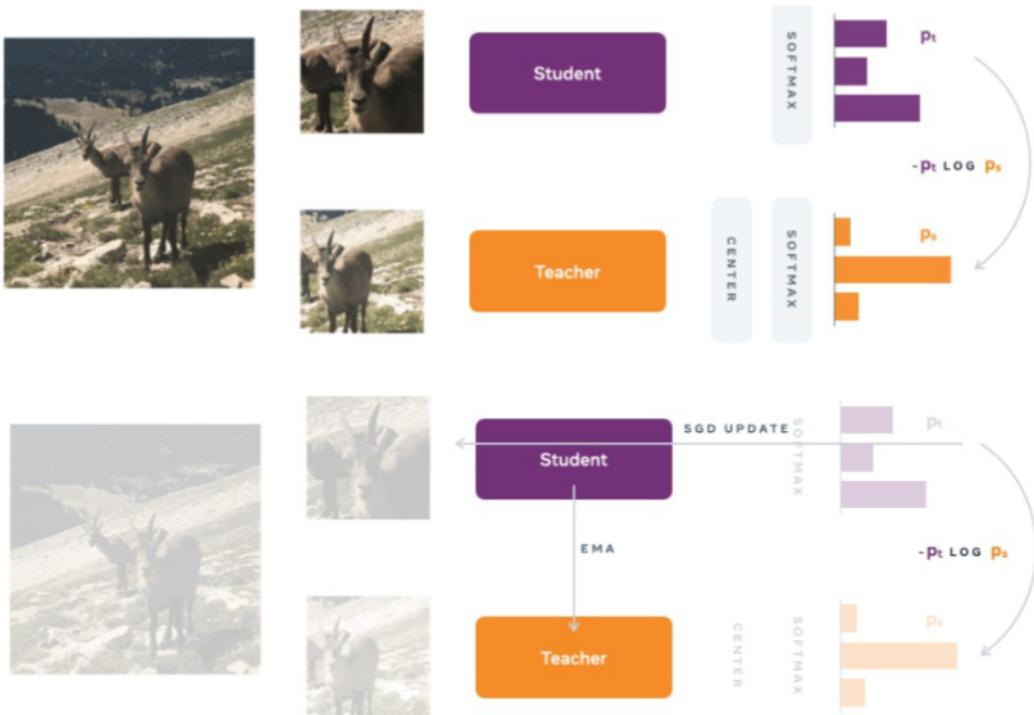


Figura 4.1: Representación del paso forward y backward usando el algoritmo DINO.

4.1.1. SSL con destilación del conocimiento

El algoritmo DINO, comparte la misma estructura general que los enfoques *self-supervised* recientes [18, 29, 30, 7, 8]. Sin embargo, DINO también comparte similitudes con la destilación de conocimiento original propuesta en [31] y se presenta bajo este enfoque. Se ilustra DINO en la figura 4.1 y 4.2, y también se detalla su funcionamiento en el algoritmo 1. La probabilidad P se obtiene normalizando la salida de la red con una función softmax. Más precisamente,

$$P_s(x)^{(i)} = \frac{\exp(g_{\theta_s}(x)^{(i)}/\tau_s)}{\sum_{k=1}^K \exp(g_{\theta_s}(x)^{(k)}/\tau_s)} \quad (4.1)$$

donde $\tau_s > 0$ es un parámetro de temperatura que controla la agudeza de la distribución de salida, y una fórmula similar se mantiene para P_t con temperatura τ_t . Dada una red maestra

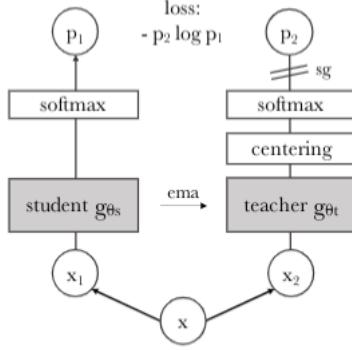


Figura 4.2: Self-distillation sin etiquetas. Ilustración de DINO en el caso de un solo par de vistas (x_1, x_2) para simplificar. El modelo pasa dos transformaciones aleatorias diferentes de una imagen de entrada a la red estudiante y a la red maestra. Ambas redes tienen la misma arquitectura pero diferentes parámetros. La salida de la red maestra se centra con una media calculada sobre el batch. Cada red genera una característica dimensional K que se normaliza con una temperatura softmax sobre la dimensión de la característica. Luego se mide su similitud con una pérdida de entropía cruzada. Aplicamos un operador stop-gradient (sg) en el maestro para propagar el gradiente solo a través del estudiante. Los parámetros de la red maestra se actualizan con una Media Móvil Exponencial (EMA) del parámetro del estudiante.

fija, aprendemos a igualar estas distribuciones minimizando la pérdida de entropía cruzada con respecto a los parámetros de la red estudiante θ_s :

$$\min_{\theta_s} H(P_t(x), P_s(x)) \quad (4.2)$$

donde $H(a, b) = -a \log b$.

El algoritmo 2 detalla el funcionamiento de la entropía cruzada.

A continuación, se detalla cómo se adaptó el problema de la ecuación 4.2 al aprendizaje *self-supervised*. Primero, se construyen diferentes vistas distorsionadas, o recortes, de una imagen con la estrategia *multi-crop* [18]. Más precisamente, a partir de una imagen dada, se genera un conjunto de vistas diferentes. Este conjunto contiene dos vistas globales, x_1^g y x_2^g y varias vistas locales de menor resolución. Todos los recortes se envían al estudiante, mientras que solo las vistas globales se envían al maestro, por lo que se fomenta la correspondencia 'local a global'.

Minimizamos la pérdida:

$$\min_{\theta_s} \sum_{x \in \{x_1^g, x_2^g\}} \sum_{\substack{x' \in V \\ x' \neq x}} H(P_t(x), P_s(x')) \quad (4.3)$$

Esta pérdida es general y se puede usar en cualquier número de vistas, incluso solo 2. Sin embargo, se sigue la configuración estándar para *multi-crops* utilizando 2 vistas globales con una resolución de 224^2 que cubren un área más grande (por ejemplo, más del 50%) de la imagen original, y varias vistas locales de resolución 96^2 que cubren solo áreas pequeñas (por ejemplo, menos del 50%) de la imagen original. Esta configuración será la parametrización básica de DINO. Ambas redes comparten la misma arquitectura con parámetros θ_s y θ_t . Los parámetros

Algorithm 1 DINO

```

 $augment \leftarrow$  función de data augmentation
 $H \leftarrow$  función de entropía
 $backpropagation \leftarrow$  función de retropopagación
 $update \leftarrow$  función de actualización de parámetros mediante el método SGD
 $mean \leftarrow$  función de cálculo de la media
 $concatenation \leftarrow$  función de concatenación de tensores
 $ns, nt \leftarrow$  red estudiante y maestra
 $C \leftarrow$  centro
 $l, m \leftarrow$  ratios de momentum de la red y del centro
 $imgs \leftarrow$  imágenes de entrenamiento
 $ns.params = nt.params$  ▷ Condición inicial necesaria

for  $x$  in  $imgs$  do
     $x_1, x_2 \leftarrow augment(x)$ 

     $s_1, s_2 \leftarrow ns(x_1), ns(x_2)$ 
     $t_1, t_2 \leftarrow nt(x_1), nt(x_2)$ 

     $loss \leftarrow H(t_1, s_2)/2 + H(t_2, s_1)/2$ 
     $backpropagation(loss)$ 

     $update(ns)$ 
     $nt.params \leftarrow l * nt.params + (1 - l) * ns.params$  ▷ Método media móvil exponencial
     $C \leftarrow m * C + (1 - m) * mean(concatenation(t_1, t_2))$ 
end for

```

Algorithm 2 Entropía H

```

 $softmax \leftarrow$  función softmax
 $sum \leftarrow$  función suma
 $log \leftarrow$  función logaritmo
 $mean \leftarrow$  función de cálculo de la media
 $tps, tpt \leftarrow$  temperatura del estudiante y de la red maestra
 $C \leftarrow$  centro
 $s, t \leftarrow$  entradas

function  $H(t, s)$ 
     $s \leftarrow softmax(s / tps)$ 
     $t \leftarrow softmax((t - C) / tpt)$ 
    return  $-mean(sum(t * log(s)))$ 
end function

```

θ_s se aprenden minimizando la ecuación 4.3 con el descenso de gradiente estocástico (SGD).

4.1.1.1. Red maestra

A diferencia de la destilación del conocimiento original, no se tiene una prioridad de la red maestra y, por lo tanto, se construye a partir de iteraciones pasadas de la red estudiante. Se han estudiado diferentes reglas de actualización para el maestro en la sección 5.4.2.1 y se demuestra que congelar la red maestra durante una época funciona sorprendentemente bien, mientras que copiar los pesos del estudiante para el maestro no logra converger. De particular interés, el uso de una Media Móvil Exponencial (EMA) en los pesos del estudiante, es decir, un *momentum encoder* [8], es particularmente adecuado para DINO. La regla de actualización es $\theta_t \leftarrow \lambda\theta_t + (1 - \lambda)\theta_s$, con λ siguiendo una regla coseno de 0,996 a 1 durante el entrenamiento [7].

Originalmente, *momentum encoder* se introdujo como sustituto para una cola en *contrastive learning* [8]. Sin embargo, en DINO su papel difiere, y puede estar más cerca del papel de la red maestra promediada. De hecho, se observa que DINO realiza una forma de modelo ensamblado similar al promedio de Polyak-Ruppert con un decaimiento exponencial [16, 32]. El uso de promedios de Polyak-Ruppert para el ensamblaje de modelos es una práctica estándar para mejorar el rendimiento de un modelo [33]. Se observa que el maestro tiene un mejor desempeño comparado con el estudiante a lo largo del entrenamiento y, por tanto, guía el entrenamiento del estudiante proporcionando características objetivo de mayor calidad. Esta dinámica no se ha observado en trabajos anteriores [7, 34].

4.1.1.2. Arquitectura de red

La red neuronal se compone de un backbone f (ViT [14] o ResNet [35]) y de un *head projection* $h : g = h \circ f$. Las características que se utilizan en las tareas posteriores son la salida del backbone f . El *head projection* consta de un MLP de 3 capas con dimensión oculta 2048 seguido de normalización l_2 y una capa completamente conectada con pesos normalizados [36] con K dimensiones, que es similar al diseño de SwAV [18]. Se han probado otros *head projection* y este diseño en particular parece funcionar mejor para DINO. No se usa un predictor [7, 29], lo que da como resultado exactamente la misma arquitectura en la red estudiante y maestra. De particular interés, tengamos en cuenta que, a diferencia de las CNN's estándar, las arquitecturas ViT no utilizan *Batch Normalization* (BN) de forma predeterminada. Por lo tanto, al aplicar DINO a ViT no se utiliza ningún BN en los *head projection*, lo que hace que el sistema esté completamente libre de BN.

4.1.1.3. Evitando el colapso

Entiéndase por colapso al proceso que sufre la red de generar predicciones ruidosas a partir de determinada época del entrenamiento. En modelo se ve imposibilitado de mejorar si se continúa entrenando. DINO es muy susceptible a sufrir este tipo de colapso, por lo que hay que tener especial cuidado al momento de escoger los hiperparámetros.

Varios métodos *self-supervised* difieren según la operación utilizada para evitar el colapso, ya sea por el *contrastive loss* [37], predictores [7] o normalizaciones por lotes [7, 34]. Si bien DINO se puede estabilizar con múltiples normalizaciones [18], también puede funcionar con solo centrar y agudizar los resultados de la red maestra momentum para evitar el colapso del modelo. Como se muestra experimentalmente en la sección 5.4.3, el centrado evita que una dimensión domine pero fomenta el colapso de la distribución uniforme, mientras que la agudeza tiene el efecto

opuesto. La aplicación de ambas operaciones equilibra sus efectos, lo que es suficiente para evitar el colapso en presencia de un maestro momentum. Elegir este método para evitar el colapso de la estabilidad de las operaciones se puede interpretar como la adición de un término de sesgo a la red maestra: $g_t(x) \leftarrow g_t(x) + c$. El centro se actualizó con el Media Móvil Exponencial (EMA), que permite que el enfoque funcione bien en diferentes tamaños de batch, como se muestra en la sección 5.4.5:

$$c \leftarrow mc + (1 - m) \frac{1}{B} \sum_{i=1}^B g_{\theta_t}(x_i) \quad (4.4)$$

donde $m > 0$ es un parámetro de ratio y B es el tamaño de batch. La salida agudizada se obtiene usando un valor bajo para la temperatura τ_t en la normalización softmax de la red maestra.

4.1.2. Implementación y protocolos de evaluación

En esta sección se brindan detalles de implementación para entrenar con DINO y se presentan protocolos de evaluación usados en los experimentos.

4.1.2.1. Vision Transformer

Se utilizará la implementación DeiT [23], una variante de ViT [14]. Las diferentes configuraciones de la red se muestran en la tabla 4.1. La red toma como entrada una cuadrícula de parches de imágenes contiguas no superpuestas de resolución $N \times N$. Se usará normalmente $N = 16' / 16'$ o $N = 8' / 8'$. Luego, los parches se pasan a través de una capa lineal para formar un conjunto de embeddings. Se agrega un token extra aprendible a la secuencia [15, 14]. El papel de este token es agregar información de toda la secuencia y se adjunta el *head projection* \mathbf{h} a su salida. Este token se conoce como el token de clase [CLS] que se encarga de mantener la coherencia con trabajos anteriores [15, 14, 23], aunque no está adjunto a ninguna etiqueta ni supervisión en este caso. El conjunto de tokens de parche y el token [CLS] serán la entrada a una red Transformer estándar.

Modelo	bloques	dim.	heads	#tokens	#parám
ResNet-50	-	2048	-	-	23M
DeiT-S/16	12	384	6	197	21M
DeiT-S/8	12	384	6	785	21M
ResNet-50	12	768	12	197	85M
ResNet-50	12	768	12	785	85M

Cuadro 4.1: Configuración de los modelos. 'bloques' se refiere a los bloques Transformer, 'dim' es la dimensión del canal y 'heads' es el número de heads en *multi-head attention*. '#tokens' es la longitud de la secuencia de tokens cuando se considera entrada en resolución 224^2 , '#parám' es el número total de parámetros (sin contar los heads de proyección) y 'im/s' es el tiempo de inferencia en una NVIDIA 1050Ti con 128 muestras por forward.

4.1.2.2. Detalles de implementación

Se pre-entrenó en el dataset ImageNet [38] sin etiquetas. Se entrenó usando el optimizador *adamw* [39] (variante SGD), con un batchsize de 8. La tasa de aprendizaje aumenta linealmente

durante las primeras 10 épocas hasta su valor base determinado con la siguiente regla de escala lineal [40]: $lr = 0,0005 * \text{batch size} / 256$. Después de este calentamiento, decaemos la tasa de aprendizaje con una regla coseno [41]. El decaimiento del peso (conocido como *weight decay*) también sigue una regla coseno de 0,04 a 0,4. La temperatura τ_s se establece en 0,1 mientras que se usó un calentamiento lineal (variación de la temperatura) para τ_t de 0,04 a 0,07 durante las primeras 30 épocas. Se siguió la técnica de *data augmentation* de BYOL [7] (fluctuación de color, desenfoque gaussiano y solarización) y *multi-crop* [18] con una interpolación bicúbica para adaptar los embeddings de posición a las escalas [14, 23].

4.1.2.3. Protocolos de evaluación

Los protocolos estándar para el aprendizaje *self-supervised* son para aprender una clasificación lineal de características congeladas [42, 8] o para afinar las características en tareas posteriores. Para las evaluaciones lineales (Lin.), se aplicaron recortes aleatorios redimensionados y *data augmentation* mediante rotaciones horizontales durante el entrenamiento, mostrándose el accuracy en un recorte centrado. También se evaluó la calidad de las características aprendidas con un clasificador ($k - NN$) [37].

El clasificador $k - NN$ hace coincidir la característica de una imagen con las características almacenadas más cercanas de una determinada etiqueta. Se revisó un número diferente de k , descubriendo que 20 NN funciona mejor la mayoría de las veces. Este protocolo de evaluación no requiere ningún otro ajuste de hiperparámetros, ni *data augmentation* y puede ejecutarse con una sola pasada sobre el dataset posterior, lo que simplifica enormemente la evaluación de características.

En base a la metodología descrita, las actividades a desarrollar serán:

1. Entrenar modelos self-supervised basados en CNN's
2. Entrenar el modelo DeiT (variante ViT) bajo el enfoque *self-supervised* con DINO
3. Evaluar los modelos entrenados en diferentes tareas de Computer Vision
4. Analizar la explicabilidad e intepretabilidad de los modelos, considerando las métricas estudiadas.
5. Establecer cual es el modelo que brinda mejores resultados

4.2. Herramientas

4.2.1. Herramientas computacionales

Las principales herramientas de hardware y software utilizadas para la experimentación fueron:

1. Para la creación de los modelos, el framework de código abierto [Pytorch](#).
2. Para la visualización de diversas métricas durante el entrenamiento se usó [Tensorboard](#).
3. Para el entrenamiento del modelo, una tarjeta gráfica NVIDIA 1050Ti.

4.2.2. Datasets

4.2.2.1. ImageNet

Para las tareas más generales de visión se usó ImageNet, que contiene 14197,122 imágenes anotadas con 1k etiquetas. Desde 2010, el conjunto de datos se utiliza en el ImageNet Large Scale Visual Recognition Challenge (ILSVRC), un benchmark en clasificación de imágenes y detección de objetos. El conjunto de datos publicado abiertamente contiene un conjunto de imágenes de entrenamiento anotadas manualmente. También se publica un conjunto de imágenes de prueba sin etiquetas.



Figura 4.3: Algunas muestras del dataset ImageNet en baja resolución

4.2.2.2. DAVIS 2017 Video object segmentation

Para la segmentación de objetos se usó el dataset DAVIS (Densely Annotated Video Segmentation) [43, 44], el cual consiste en secuencias de video Full HD de alta calidad, que abarcan múltiples ocurrencias de desafíos comunes de segmentación de objetos de video, como occlusiones, desenfoque de movimiento y cambios de apariencia. Cada video está acompañado por una segmentación anotada, precisa de píxeles y por frame.

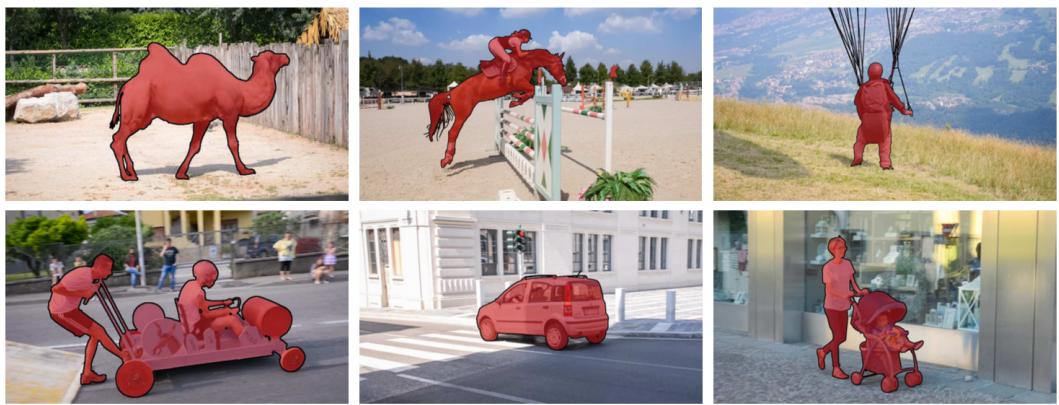


Figura 4.4: Algunas muestras del dataset DAVIS.

4.2.2.3. Pascal VOC

Para obtener métricas de validación en imágenes de segmentación, aparte del dataset mencionado en el ítem anterior, también se usó el dataset PASCAL Visual Object Classes (VOC) 2012 [45]. El dataset contiene 20 categorías de objetos que incluyen vehículos, cosas del hogar, animales y otros. Cada imagen de este conjunto de datos tiene anotaciones de segmentación a nivel de píxel, anotaciones de cuadro delimitador y anotaciones de clase del objeto. Este conjunto de datos se ha utilizado ampliamente como benchmark para la detección de objetos, la segmentación semántica y las tareas de clasificación. El dataset se divide en tres subconjuntos: 1.464 imágenes para entrenamiento, 1.449 imágenes para validación y un conjunto de pruebas privado.

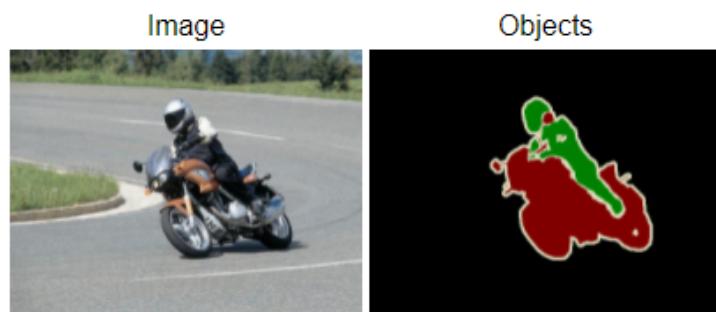


Figura 4.5: Una muestra de segmentación en Pascal VOC.

Experimentación y Resultados

En el presente capítulo se valida DINO en un benchmark estándar *self-supervised* sobre ImageNet. Se hace un estudio de las propiedades de las características resultantes.

5.1. Comparando otros enfoques SSL en ImageNet

Se consideraron dos configuraciones: comparación con la misma arquitectura y con otras arquitecturas.

5.1.1. Comparando con la misma arquitectura

En el panel superior de la tabla 5.1, se comparó DINO con otros métodos *self-supervised* con la misma arquitectura, ya sea un ResNet-50 [35] o un DeiT-small (DeiT-S) [23]. La elección de DeiT-S está motivada por su similitud con ResNet-50 en varios aspectos: número de parámetros (21M vs 23M), rendimiento *self-supervised* y rendimiento supervisado en ImageNet con el método de entrenamiento de [23] (79,2 % VS 79,5 %).

Primero, se observa que DINO funciona a la par con el SOA en ResNet-50, validando que DINO funciona en la configuración estándar. Cuando se cambia a una arquitectura ViT, DINO supera a BYOL, MoCov2 y SwAV en $\approx + 3\%$ en la clasificación lineal y en un $\approx + 7\%$ con la evaluación k-NN. Más sorprendentemente, el rendimiento con un clasificador k-NN simple está casi a la par con un clasificador lineal (74,1 % frente a 77,1 %). Esta propiedad emerge solo cuando se usa DINO con arquitecturas ViT, y no aparece con otros métodos *self-supervised* existentes ni con un ResNet-50.

5.1.2. Comparación entre arquitecturas

En el panel inferior de la Tabla 5.1, se compara el mejor desempeño obtenido entre arquitecturas. El interés de esta configuración no es comparar métodos directamente, sino para evaluar los límites de un ViT entrenado con DINO al pasar a arquitecturas más grandes. Si bien entrenar un ViT más grande con DINO mejora el rendimiento, la reducción del tamaño de los parches (variantes '/ 8') tiene un mayor impacto en el rendimiento. Si bien la reducción del tamaño del parche no agrega parámetros, aún conduce a una reducción significativa de tiempo de ejecución y mayor uso de memoria. No obstante, un ViT base con parches 8 x 8 entrenados con DINO logra un top-1 de 80,1 % en la clasificación lineal (Lin.) y 77,4 % con un clasificador k-NN con 10 veces menos parámetros y 1,4 veces más rápido que el SOA anterior [9].

Cuadro 5.1: Clasificación lineal y k -NN sobre ImageNet. Se reporta el accuracy top-1 para evaluaciones lineales y k -NN en el conjunto de validación de ImageNet para diferentes métodos *self-supervised*. Con un enfoque en las arquitecturas ResNet-50 y DeiT-S, pero además se muestran los mejores resultados obtenidos entre otras arquitecturas. El rendimiento (im/s) se calcula en una NVIDIA 1050Ti con 128 muestras por forward. Los parámetros (M) son del extractor de características.

Método	Arquit.	Parám.	im/s	Lineal	k -NN
Supervisado	RN50	23	637	79.2	79.2
SCLR [30]	RN50	23	637	69.0	60.6
MoCov2 [46]	RN50	23	637	71.2	61.8
BYOL [7]	RN50	23	637	74.2	64.4
SwAV [18]	RN50	23	637	75.1	65.2
DINO	RN50	23	637	75.1	67.4
Supervisado	DeiT-S	21	503	79.5	79.5
BYOL [7]	DeiT-S	21	503	71.5	66.7
MoCov2 [46]	DeiT-S	21	503	72.9	64.5
SwAV [18]	DeiT-S	21	503	73.6	66.5
DINO	DeiT-S	21	503	77.1	74.4
<i>Comparación entre arquitecturas</i>					
SCLR [30]	RN50w4	375	57	76.6	69.1
SwAV [18]	RN50w2	93	184	77.1	67.1
BYOL [7]	RN50w2	93	184	77.3	-
DINO	ViT-B/16	85	152	78.0	76.1
SwAV [18]	RN50w5	586	36	78.4	67.1
BYOL [7]	RN50w4	375	57	78.5	-
BYOL [7]	RN200w2	250	62	79.4	73.9
DINO	DeiT-S/8	21	88	79.8	78.1
DINO	ViT-B/8	85	32	80.1	77.4

5.2. Propiedades de ViT entrenado con SSL

5.2.1. Descubriendo el diseño semántico de escenas

Como se muestra cualitativamente en la figuras 5.1 y 5.2, los mapas de *self-attention* contienen información sobre la segmentación de una imagen. Se medirá posteriormente esta propiedad probando directamente la calidad de las máscaras generadas a partir de estos mapas de atención.

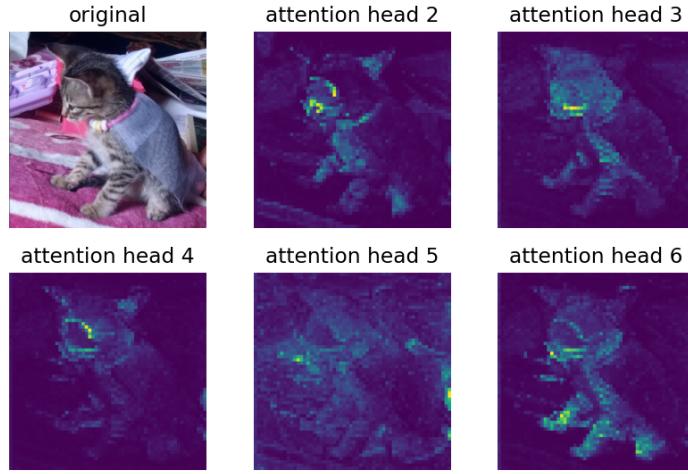


Figura 5.1: *self-attention* de un Vision Transformer con parches 8×8 entrenados de forma *self-supervised* en una imagen de un gato con una capa.

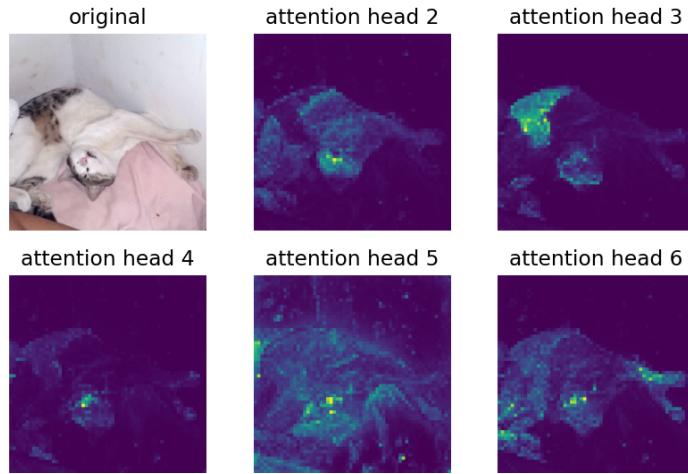


Figura 5.2: *self-attention* de un Vision Transformer con parches 8×8 entrenados de forma *self-supervised* en una imagen de un gato durmiendo de forma extraña.

5.2.1.1. Explorando el mapa *self-attention*

En la figura 5.3, se demuestra que diferentes *heads* atienden diferentes regiones semánticas de una imagen, incluso cuando están ocultas o son muy pequeñas (la bandera en la segunda fila). Las visualizaciones se obtienen con imágenes de 480p, lo que da como resultado secuencias de 3601 tokens para DeiT-S/8. En la figura 5.4, se muestra que un ViT supervisado no atiende bien a los objetos en presencia de desorden tanto cualitativa como cuantitativamente.

Se muestra además la similitud de Jaccard entre la segmentación target y las máscaras de segmentación predichas. Note que los mapas de *self-attention* son suaves y no están optimizados para producir una máscara óptima. No obstante, vemos una clara diferencia entre los modelos supervisados y los modelos con DINO con una brecha significativa en términos de similitudes Jaccard.

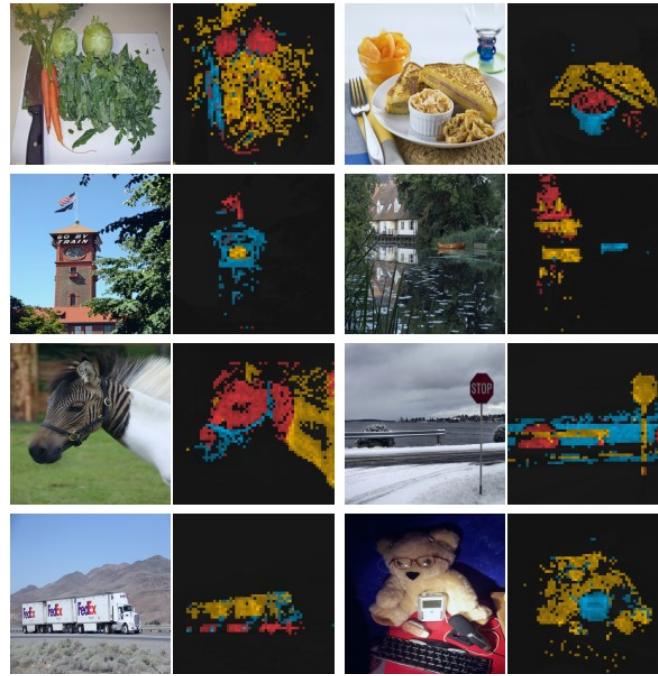


Figura 5.3: Mapas de atención de múltiples *heads*. Se han considerado los *heads* de la última capa de un DeiT-S/8 entrenado con DINO y se muestra el *self-attention* para la consulta del token *[CLS]*. Diferentes *heads*, materializados por diferentes colores, se enfocan en diferentes ubicaciones que representan diferentes objetos o partes.

Supervised



DINO



Figura 5.4: Segmentación de forma supervisada versus DINO. En la parte superior, se muestran las máscaras resultantes de un DeiT-S/8 entrenado de forma supervisada y con DINO. Se muestra el mejor *head* para ambos modelos. La tabla en la parte inferior compara la similitud Jacard entre la segmentación target (usando PASCAL VOC12 [45]) y la segmentación predicha.

	Random	Supervisado	DINO
DeiT-S/16	19.8	26.9	45.8
DeiT-S/8	20.7	23.6	44.6

5.2.1.2. Segmentación de instancias

A continuación se muestran algunos ejemplos de segmentación en DAVIS [43, 44]. La imagen superior representa la imagen original, la imagen intermedia representa la segmentación target y la imagen inferior representa la segmentación predicha.

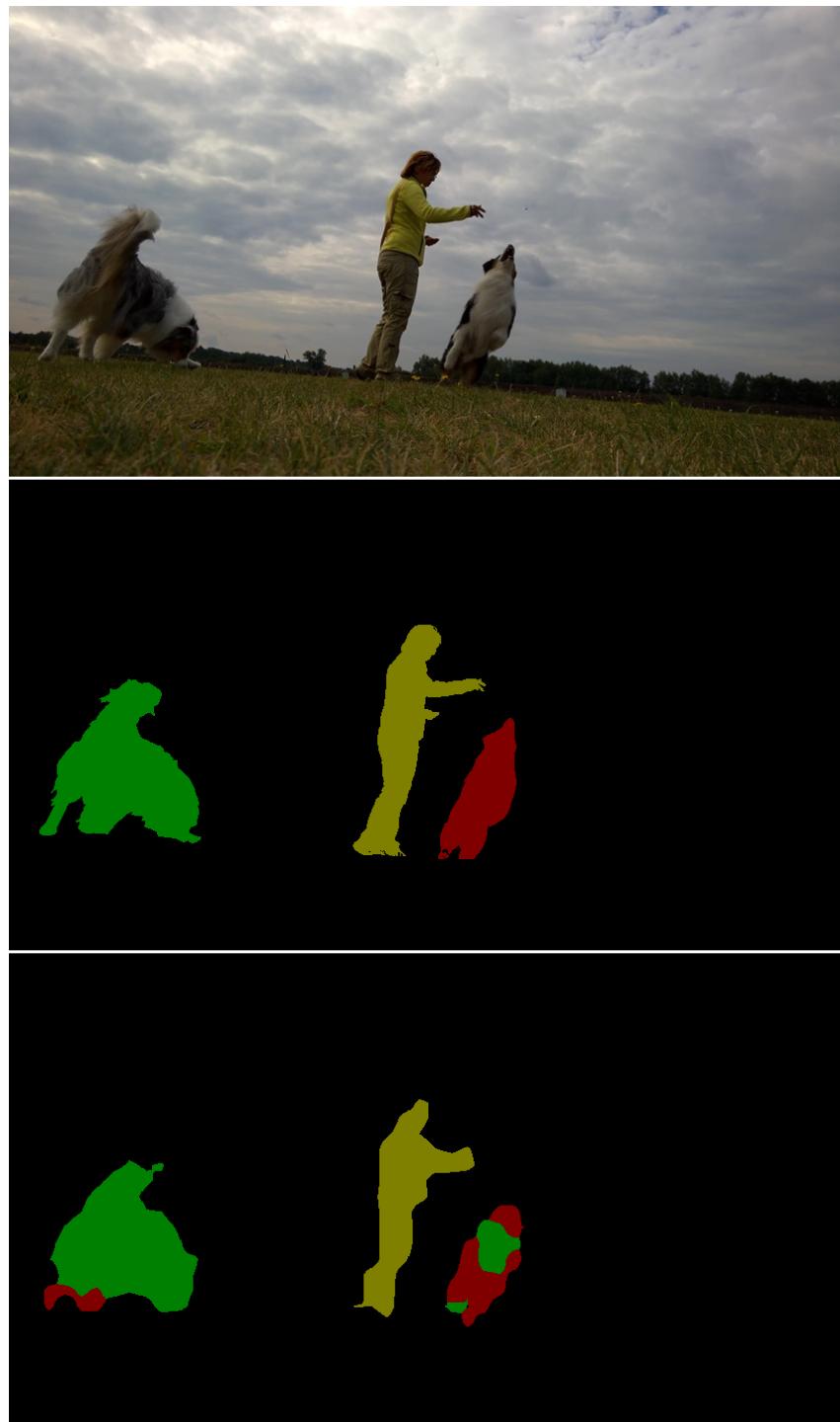


Figura 5.5: Ejemplo de segmentación de unos perros jugando.

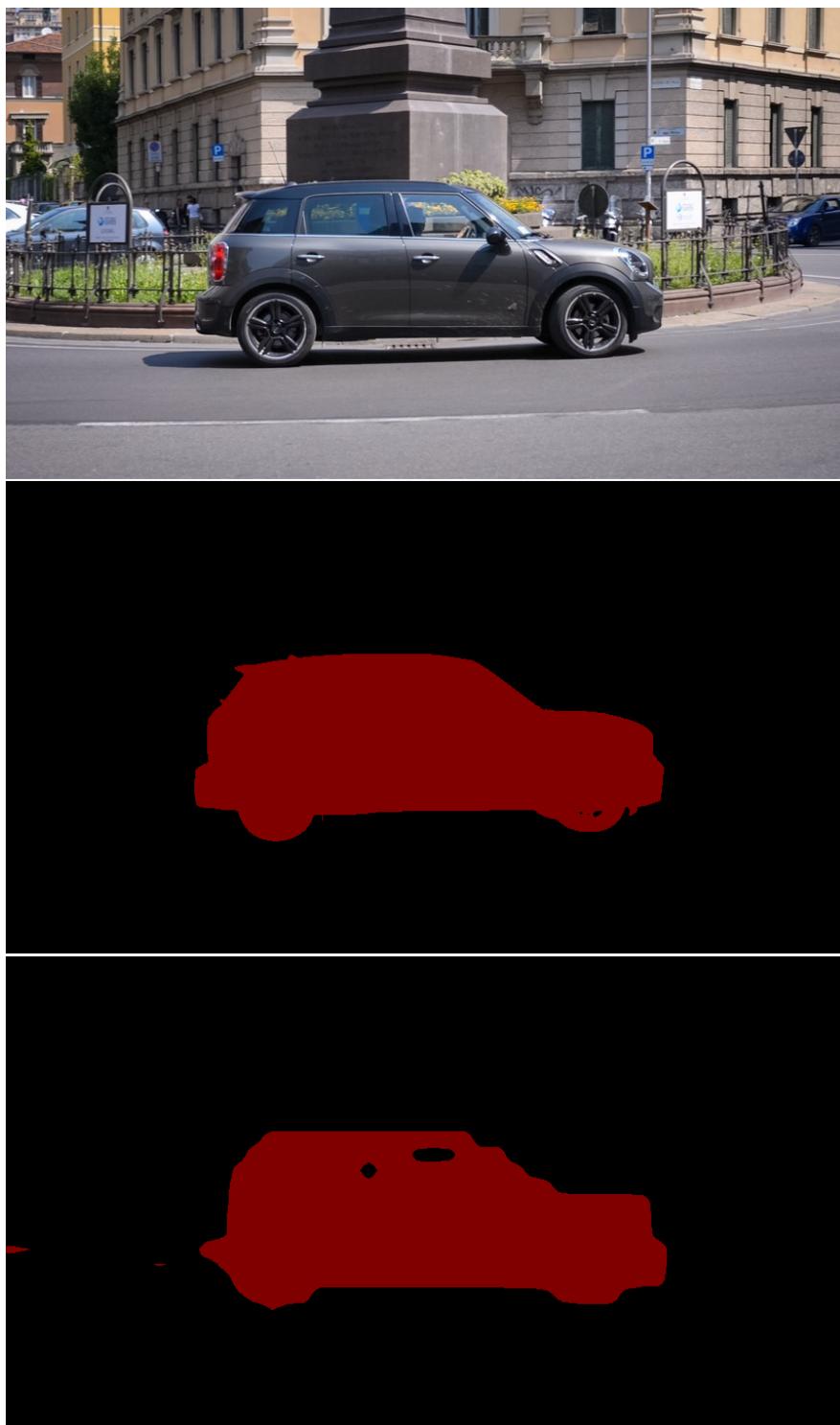


Figura 5.6: Ejemplo de segmentación de un coche en la ciudad

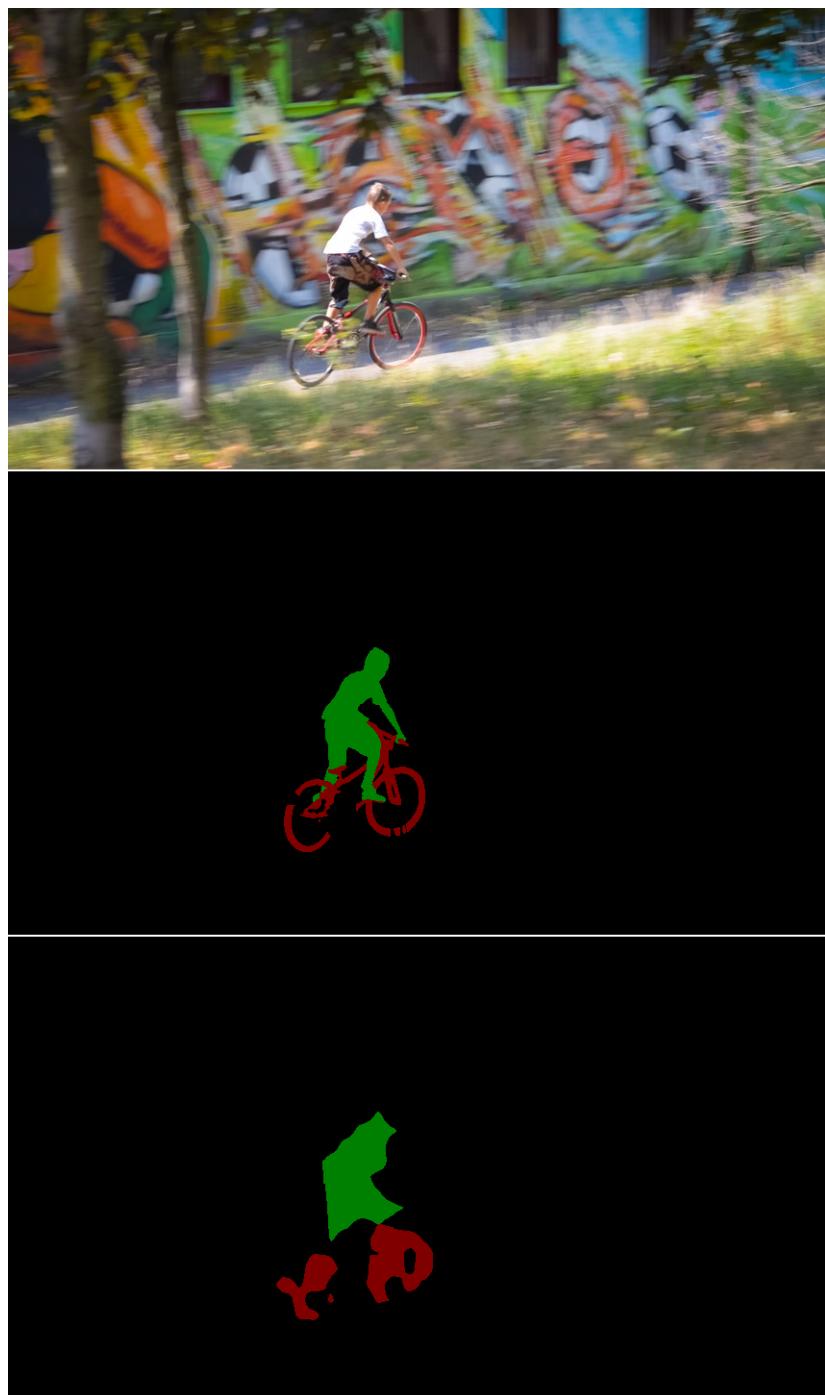


Figura 5.7: Ejemplo de segmentación de un ciclista.

5.3. Despliegue

Se ha implementado una página web¹ para poder visualizar los mapas de atención en una reducida muestra de ImageNet. Se usó Streamlit y Heroku para la construcción y despliegue

¹<https://vision-transformer-t1.herokuapp.com/>

de la página. Puede encontrar el código de la implementación en [mi repositorio](#).

5.4. Estudio de ablación de DINO

En esta sección se estudia empíricamente DINO aplicado para ViT. El modelo considerado para este estudio es DeiT-S.

5.4.1. Importancia de los diferentes componentes

Se demostrará el impacto de agregar diferentes componentes del aprendizaje *self-supervised* en un ViT entrenado con DINO.

En la tabla 5.2, se muestran los resultados de diferentes variantes de modelo a medida que se agregan o eliminan componentes. Primero, se observa que en ausencia de momentum, DINO no funciona (fila 2). En segundo lugar, en las filas 3 y 4, se observa que el entrenamiento multi-crop y la pérdida de entropía cruzada (Cross Entropy) en DINO son componentes importantes para obtener buenas características. También se observa que agregar un predictor a la red estudiante tiene poco impacto (fila 5) mientras que es crítico en BYOL para evitar colapsos [7, 29].

Cuadro 5.2: Importancia de las componentes para el pre-entrenamiento de ViT self-supervised. Los modelos fueron entrenados durante 50 épocas con DeiT-S/16. Se ha estudiado los diferentes componentes que importan para las evaluaciones k-NN y lineales ('Lin.'). Para las diferentes variantes, se resaltan las diferencias de la configuración predeterminada de DINO. La mejor combinación es el *momentum encoder* con *multicrop augmentation* y la pérdida de entropía cruzada. También se muestran resultados con BYOL [7], MoCo-v2 [46] y SwAV [18]. MC: Multi-Crop, Pred.: Predictor, CE: Cross-Entropy, MSE: Mean Square Error, INCE: InfoNCE (Noise-Contrastive Estimation).

Método	Mom.	MC	Loss	Pred.	k-NN	Lin.
DeiT-S/16	✓	✓	CE	X	72.7	76.0
	X	✓	CE	X	0.1	0.1
	✓	X	CE	X	67.8	72.4
	✓	✓	MSE	X	52.5	62.3
	✓	✓	CE	✓	71.7	75.5
	✓	X	MSE	✓	66.5	71.3
BYOL	✓	X	INCE	X	62.0	71.5
MoCov2	✓	X	CE	X	64.6	71.7
SwAV	X	✓	CE	X		

5.4.1.1. Importancia del tamaño del parche

En la figura 5.8, se compara el rendimiento de clasificación k-NN de modelos DeiT-S entrenados con diferentes tamaños de parche, 16 x 16, 8 x 8 y 5 x 5. Se comparó también con ViT-B con parches de 16 x 16 y 8 x 8. El modelo ha sido entrenado durante 50 épocas. Se observa que el rendimiento mejora enormemente a medida que se disminuye el tamaño del parche. Es interesante ver que el rendimiento puede ser enormemente mejorado sin agregar parámetros adicionales.

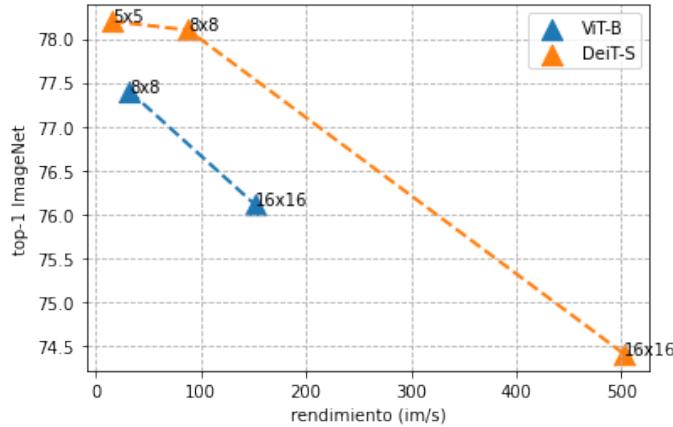


Figura 5.8: Efecto del tamaño del parche. Evaluación de k -NN en función del rendimiento para diferentes tamaños de parches de entrada con ViT-B y DeiT-S.

5.4.2. Impacto de la elección de la red maestra

En esta ablación, se experimentó con diferentes redes maestras para comprender su papel en DINO. Se comparan los modelos entrenados durante 50 épocas utilizando el protocolo k -NN.

5.4.2.1. Construyendo maestros distintos al estudiante

En la figura 5.9, se comparan diferentes estrategias para construir el maestro a partir de instancias anteriores del estudiante además de la red maestra momentum. Primero, se consideró utilizar la red de estudiantes de una época anterior como profesor. Esta estrategia se ha utilizado en el banco de memoria de Wu et al. [37] y como de forma de destilación dura en Caron et al. [47] y Asano et Alabama. [48]. En segundo lugar, se consideró utilizar la red de estudiantes de la iteración anterior, así como una copia del estudiante para el maestro. En el entorno actual, utilizar un profesor basado en una versión reciente del estudiante no converge. Esta configuración requiere más normalizaciones para trabajar. Curiosamente, se observó que usar un maestro de la época anterior no colapsa, proporcionando un desempeño en la evaluación k-NN competitiva con marcos existentes como MoCo-v2 o BYOL. Mientras que al usar un *momentum encoder* se tiene claramente un rendimiento superior a este maestro ingenuo (o naive), este hallazgo sugiere que hay un espacio para investigar alternativas para la red maestra.

Maestro	Top-1
Estudiante copia	0.10
Iteración previa	0.12
Época previa	66.5
Momentum	72.7

5.4.2.2. Analizando la dinámica del entrenamiento

Para comprender mejor las razones por las que un maestro momentum funciona bien en DINO, se estudió su dinámica durante el entrenamiento de un ViT en la figura 5.9. Una observación clave es que este maestro supera constantemente al estudiante durante el entrenamiento. Este comportamiento no ha sido observado en otros trabajos que utilizan momentum [8, 7], ni cuando la red maestra se construye a partir de la época anterior.

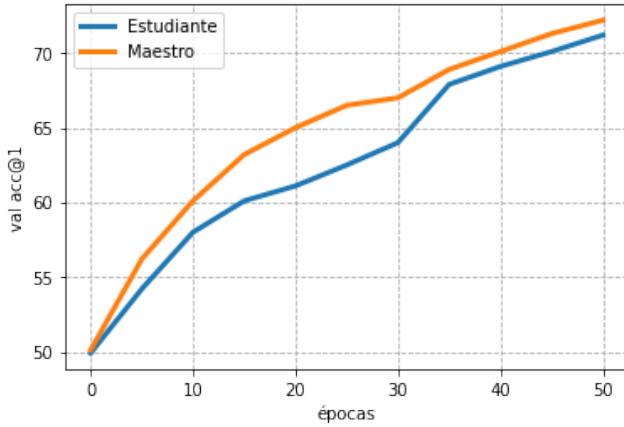


Figura 5.9: Accuracy Top-1 en el conjunto de validación de ImageNet con el clasificador k -NN. Comparación entre el desempeño de la red maestra momentum y el estudiante durante el entrenamiento. En la tabla inferior se hace una comparación entre las variantes de la red maestra. Usar *momentum encoder* conduce al mejor rendimiento, pero no es la única opción viable.

Podríamos interpretar al maestro momentum en DINO como una forma de promedio Polyak-Ruppert [16, 32] con un decaimiento exponencial. El promedio Polyak-Ruppert se usa a menudo para simular el ensamblaje del modelo para mejorar el rendimiento de una red al final del entrenamiento [33]. DINO puede ser interpretado como la aplicación del promedio Polyak-Ruppert durante el entrenamiento para construir constantemente un modelo ensamblado que da rendimientos superiores. Este modelo ensamblado luego guía el entrenamiento de la red estudiante [49].

5.4.3. Evitando el colapso

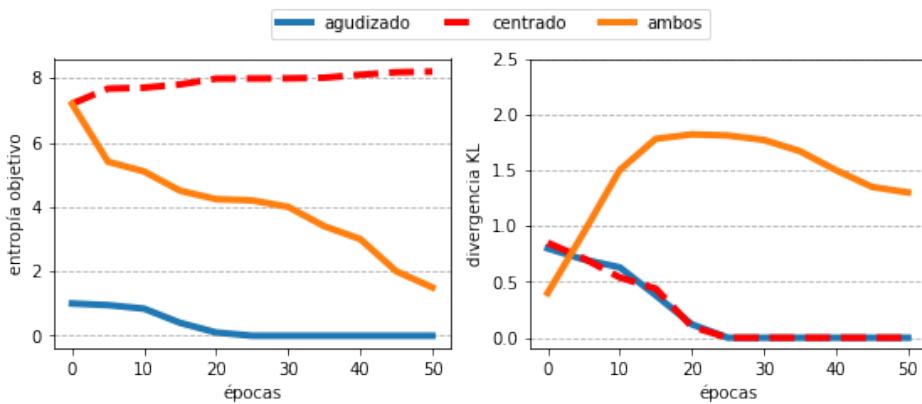


Figura 5.10: Estudio del colapso. Izquierda: Evolución de la entropía objetivo de la red maestra. Derecha: Evolución de la divergencia KL entre las salidas de la red maestra y el estudiante.

Se ha estudiado el papel de complementariedad del centrado y el target (objetivo) agudizado para evitar el colapso. Hay dos formas de colapso: independientemente de la entrada, la salida del modelo es uniforme a lo largo de todas las dimensiones o es dominado por una dimensión. El centrado evita el colapso inducido por una dimensión dominante, pero fomenta una

salida uniforme. El agudizado induce el efecto contrario. Se muestra esta complementariedad descomponiendo la entropía cruzada H en una entropía h y la divergencia Kullback-Leibler ('KL') D_{KL} :

$$H(P_t, P_s) = h(P_t) + D_{KL}(P_t | P_s) \quad (5.1)$$

Un KL igual a cero indica una salida constante, y por lo tanto un colapso. En la figura 5.10, se muestra la entropía y KL durante entrenamiento con y sin centrado y agudizado. Si una operación falta, KL converge a cero, lo que indica un colapso. Sin embargo, la entropía h converge a diferentes valores: 0 sin centrado y $-\log(1/K)$ sin agudizado, lo que indica que ambas operaciones inducen una forma diferente de colapso.

5.4.4. Requerimientos de cómputo

multi-crop	20 épocas		50 épocas		mem.
	top-1	tiempo	top-1	tiempo	
2×224^2	67.8	15.2h	72.5	45.9h	9.3G
$2 \times 224^2 + 2 \times 96^2$	71.5	17.0h	74.5	51.0h	10.5G
$2 \times 224^2 + 6 \times 96^2$	73.8	20.3h	75.9	60.7h	12.9G
$2 \times 224^2 + 10 \times 96^2$	74.6	24.2h	76.1	65.1h	15.4G

Cuadro 5.3: Requerimientos de tiempo y memoria. Se muestra el tiempo total de entrenamiento y la memoria máxima ocupada en la GPU ('mem.'). Se reporta el acc val top-1 en ImageNet con evaluación lineal para varias variantes de multi-crop, cada uno con un nivel diferente de requisitos de cálculo.

En tabla 5.3, se detallan los requisitos de tiempo y memoria de la GPU cuando se ejecutan los modelos DeiT-S/16 DINO en una GPU NVIDIA 1050Ti. Se muestran los resultados con variantes multi-crop, cada uno con un nivel diferente de requisito computacional. Se observa que el uso de multi-crop mejora la compensación entre el accuracy y el tiempo de entrenamiento.

Por ejemplo, el rendimiento es del 72,5 % después de 46 horas de entrenamiento sin multi-crop (por ejemplo, 2×224^2) mientras que con multi-crop $2 \times 224^2 + 10 \times 96^2$ alcanza el 74,6 % en solo 24 horas. Esta es una mejora de + 2 % mientras que requiere 2 veces menos tiempo, aunque el uso de memoria es mayor (15,4G frente a 9,3G). Se observa que el aumento de rendimiento que trae consigo multi-crop no se puede mejorar con más entrenamiento usando la configuración 2×224^2 , lo que demuestra el valor del aumento 'local a global'. En general, el entrenamiento de DINO con Vision Transformers logra una accuracy Top-1 de 76.1 entrenado durante 2 días. Este resultado supera a los SOA *self-supervised* basados en redes convolucionales de tamaños comparables con una reducción significativa de los requisitos computacionales [7, 18].

5.4.5. Entrenando con pequeños batchs

En la tabla 5.4, se estudió el impacto del *batchsize* en las características obtenidas con DINO. Se ha escalado la tasa de aprendizaje linealmente con el *batchsize* [40]: $lr = 0,0005 * \text{batchsize} / 256$. La tabla 5.4 confirma que se puede entrenar modelos para un alto rendimiento con con pequeños batchs. Los resultados con tamaños de batchs más pequeños (bs = 128) están ligeramente por debajo de la configuración de entrenamiento predeterminada de bs = 512, y ciertamente requerirían reajustar hiperparámetros como el *momentum rate*, por ejemplo.

Se ha explorado el entrenamiento de un modelo con un tamaño de lote de 8, alcanzando el 35,2 % después de 10 épocas, mostrando el potencial para entrenar modelos grandes en una sola GPU.

bs	128	256	512
top-1	57.6	59.0	59.8

Cuadro 5.4: Efecto del tamaño de los batchs. Top-1 con k -NN para modelos entrenados por 50 épocas sin multi-crop.

Conclusiones y Trabajos Futuros

6.1. Conclusiones

En este trabajo, se demuestra el potencial del pre-entrenamiento *self-supervised* de un modelo ViT estándar, logrando un rendimiento que es comparable con los mejores convnets diseñados específicamente para esta configuración. También se observó que emergen 3 propiedades que pueden aprovecharse en aplicaciones futuras:

- La calidad de las características son muy útiles para una clasificación *k-NN*.
- La presencia de información sobre el diseño de la escena también puede beneficiar a la segmentación de imágenes débilmente supervisada.
- Hay una clara sinergia entre DINO y ViT, comparado con otros métodos *self-supervised / contrastive*.

Sin embargo, el principal resultado de este trabajo es que se evidencia de que el aprendizaje *self-supervised* podría ser la clave para desarrollar un modelo similar a BERT basado en ViT.

6.2. Trabajos Futuros

Este tema de investigación me ha parecido muy interesante y de cara a la continuación de este trabajo de tesis propongo algunas alternativas a elegir:

- Explorar si el pre-entrenamiento de un gran modelo ViT con DINO en imágenes aleatorias sin tratar podría empujar los límites de las características visuales [50].
- Explorar el enfoque self-supervised en imágenes de análisis médico, buscando producir un impacto más directo y tangible como resultado de usar estos novedosos métodos en casos reales.

Bibliografía

- [1] Alexander Kolesnikov y col. "Big transfer (bit): General visual representation learning". En: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part V* 16. Springer. 2020, págs. 491-507.
- [2] Huimin Huang y col. "Unet 3+: A full-scale connected unet for medical image segmentation". En: *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2020, págs. 1055-1059.
- [3] Mingxing Tan y Quoc Le. "Efficientnet: Rethinking model scaling for convolutional neural networks". En: *International Conference on Machine Learning*. PMLR. 2019, págs. 6105-6114.
- [4] Andrew G Howard y col. "Mobilenets: Efficient convolutional neural networks for mobile vision applications". En: *arXiv preprint arXiv:1704.04861* (2017).
- [5] Chris Olah, Alexander Mordvintsev y Ludwig Schubert. *Feature visualization: How neural networks build up their understanding of images*. Distill. 2018.
- [6] Y LeCun e I Misra. *Self-supervised Learning: The Dark Matter of Intelligence*. 2021.
- [7] Jean-Bastien Grill y col. "Bootstrap your own latent: A new approach to self-supervised learning". En: *arXiv preprint arXiv:2006.07733* (2020).
- [8] Kaiming He y col. "Momentum contrast for unsupervised visual representation learning". En: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, págs. 9729-9738.
- [9] Ting Chen y col. "Big self-supervised models are strong semi-supervised learners". En: *arXiv preprint arXiv:2006.10029* (2020).
- [10] Ashish Vaswani y col. "Attention is all you need". En: *Advances in neural information processing systems*. 2017, págs. 5998-6008.
- [11] Kwonjoon Lee y col. "ViTGAN: Training GANs with Vision Transformers". En: *arXiv preprint arXiv:2107.04589* (2021).
- [12] Nicolas Carion y col. "End-to-end object detection with transformers". En: *European Conference on Computer Vision*. Springer. 2020, págs. 213-229.
- [13] Hu Cao y col. "Swin-UNet: Unet-like Pure Transformer for Medical Image Segmentation". En: *arXiv preprint arXiv:2105.05537* (2021).
- [14] Alexey Dosovitskiy y col. "An image is worth 16x16 words: Transformers for image recognition at scale". En: *arXiv preprint arXiv:2010.11929* (2020).
- [15] Jacob Devlin y col. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: [1810.04805 \[cs.CL\]](https://arxiv.org/abs/1810.04805).
- [16] Boris T Polyak y Anatoli B Juditsky. "Acceleration of stochastic approximation by averaging". En: *SIAM journal on control and optimization* 30.4 (1992), págs. 838-855.
- [17] Kaiming He y col. "Deep Residual Learning for Image Recognition". En: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), págs. 770-778.
- [18] Mathilde Caron y col. "Unsupervised learning of visual features by contrasting cluster assignments". En: *arXiv preprint arXiv:2006.09882* (2020).
- [19] Kaiming He y col. "Momentum Contrast for Unsupervised Visual Representation Learning". En: *arXiv preprint arXiv:1911.05722* (2019).

- [20] Kelvin Xu y col. "Show, attend and tell: Neural image caption generation with visual attention". En: *International conference on machine learning*. PMLR. 2015, págs. 2048-2057.
- [21] Ramprasaath R Selvaraju y col. "Grad-cam: Visual explanations from deep networks via gradient-based localization". En: *Proceedings of the IEEE international conference on computer vision*. 2017, págs. 618-626.
- [22] Shikhar Tuli y col. "Are Convolutional Neural Networks or Transformers more like human vision?" En: *arXiv preprint arXiv:2105.07197* (2021).
- [23] Hugo Touvron y col. "Training data-efficient image transformers & distillation through attention". En: *International Conference on Machine Learning*. PMLR. 2021, págs. 10347-10357.
- [24] Mathilde Caron y col. "Emerging properties in self-supervised vision transformers". En: *arXiv preprint arXiv:2104.14294* (2021).
- [25] Alaaeldin El-Nouby y col. "XCiT: Cross-Covariance Image Transformers". En: *arXiv preprint arXiv:2106.09681* (2021).
- [26] Ze Liu y col. "Swin transformer: Hierarchical vision transformer using shifted windows". En: *arXiv preprint arXiv:2103.14030* (2021).
- [27] Chunyuan Li y col. "Efficient Self-supervised Vision Transformers for Representation Learning". En: *arXiv preprint arXiv:2106.09785* (2021).
- [28] Jinghao Zhou y col. "iBOT: Image BERT Pre-Training with Online Tokenizer". En: *CoRR* abs/2111.07832 (2021). arXiv: 2111.07832. URL: <https://arxiv.org/abs/2111.07832>.
- [29] Xinlei Chen y Kaiming He. "Exploring simple siamese representation learning". En: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, págs. 15750-15758.
- [30] Ting Chen y col. "A simple framework for contrastive learning of visual representations". En: *International conference on machine learning*. PMLR. 2020, págs. 1597-1607.
- [31] Geoffrey Hinton, Oriol Vinyals y Jeff Dean. "Distilling the knowledge in a neural network". En: *arXiv preprint arXiv:1503.02531* (2015).
- [32] David Ruppert. *Efficient estimations from a slowly convergent Robbins-Monro process*. Inf. téc. Cornell University Operations Research e Industrial Engineering, 1988.
- [33] Sébastien Jean y col. "On using very large target vocabulary for neural machine translation". En: *arXiv preprint arXiv:1412.2007* (2014).
- [34] Pierre H Richemond y col. "BYOL works even without batch statistics". En: *arXiv preprint arXiv:2010.10241* (2020).
- [35] Kaiming He y col. "Deep residual learning for image recognition". En: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, págs. 770-778.
- [36] Tim Salimans y Durk P Kingma. "Weight normalization: A simple reparameterization to accelerate training of deep neural networks". En: *Advances in neural information processing systems* 29 (2016), págs. 901-909.
- [37] Zhirong Wu y col. "Unsupervised feature learning via non-parametric instance discrimination". En: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, págs. 3733-3742.
- [38] Olga Russakovsky y col. "Imagenet large scale visual recognition challenge". En: *International journal of computer vision* 115.3 (2015), págs. 211-252.
- [39] Ilya Loshchilov y Frank Hutter. "Fixing weight decay regularization in adam". En: (2018).
- [40] Priya Goyal y col. "Accurate, large minibatch sgd: Training imagenet in 1 hour". En: *arXiv preprint arXiv:1706.02677* (2017).
- [41] Ilya Loshchilov y Frank Hutter. "Sgdr: Stochastic gradient descent with warm restarts". En: *arXiv preprint arXiv:1608.03983* (2016).

- [42] Richard Zhang, Phillip Isola y Alexei A Efros. "Colorful image colorization". En: *European conference on computer vision*. Springer. 2016, págs. 649-666.
- [43] Federico Perazzi y col. "A Benchmark Dataset and Evaluation Methodology for Video Object Segmentation". En: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.
- [44] Jordi Pont-Tuset y col. "The 2017 DAVIS Challenge on Video Object Segmentation". En: *arXiv:1704.00675* (2017).
- [45] M. Everingham y col. *The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results*. <http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html>.
- [46] Xinlei Chen y col. "Improved baselines with momentum contrastive learning". En: *arXiv preprint arXiv:2003.04297* (2020).
- [47] Mathilde Caron y col. "Deep clustering for unsupervised learning of visual features". En: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, págs. 132-149.
- [48] Yuki Markus Asano, Christian Rupprecht y Andrea Vedaldi. "Self-labelling via simultaneous clustering and representation learning". En: *arXiv preprint arXiv:1911.05371* (2019).
- [49] Antti Tarvainen y Harri Valpola. "Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results". En: *arXiv preprint arXiv:1703.01780* (2017).
- [50] Priya Goyal y col. "Self-supervised pretraining of visual features in the wild". En: *arXiv preprint arXiv:2103.01988* (2021).