Últimos avances en redes de cápsulas orientados a la reducción de carga computacional y uso de memoria: evaluación y aplicaciones

Máster en Ingeniería de Telecomunicación

Juan José López Conesa

Dirigido por Jorge Larrey Ruiz

Departamento de Tecnologías de la Información
ETSIT
Universidad Politécnica de Cartagena

Octubre, 2023

Universidad
Politécnica
de Cartagena

WWW.UPCT.ES

First edition, 17 de octubre de 2023

Resumen

Este estudio presenta y evalúa el rendimiento de nuevos modelos alternativos de redes de cápsulas para una tarea de clasificación de imágenes. El objetivo es determinar si estas nuevas variantes presentan una mejora significativa frente a los modelos CapsNet y MatCaps y si ponen solución a las limitaciones que hacen que no sean una opción del todo viable a día de hoy. Limitaciones las cuales son el alto coste computacional y alto consumo de memoria. Las simulaciones han sido llevadas a cabo usando el dataset CIFAR-10 y otro dataset personalizado de bandejas de carne. Los resultados serán comparados en función de la precisión, número de parámetros, uso de memoria y tiempo de ejecución. Se identifican las configuraciones y las combinaciones de algoritmos óptimas para los modelos. Los hallazgos resaltan el potencial de estos nuevos modelos en tareas de clasificación de imágenes y sus mejoras respecto a los modelos base, y brindan información sobre sus limitaciones. Se sugieren futuras direcciones de investigación para abordar estas limitaciones y mejorar el rendimiento de los modelos.

Índice general

Índice de figuras

Índice de cuadros

Glosario de Abreviaturas

GPU Graphical Processing Unit

CPU Central Processing Unit

ML Machine Learning 24

DL Deep Learning 24

VRAM Video Random Access Memory

API Application Programming Interface

PCIe Peripheral Component Interconnect Express

FLOPS Floating Point Operations per Second

OPS Operations per Second

GDDR6 Graphics Double Data Rate 6

OOM Out of Memory 39

RBA Routing by Agreement 8

SDA Scaled Distance Agreement 39

MoCapsNet Momentum Capsule Network

ResCapsNet Residual Capsule Network

CapsNet Capsule Network

DeepCaps Deep Capsule Network

MatCaps Matricial Capsule Network

1

Introducción

En los últimos años, el campo del *Deep Learning* ha experimentado avances significativos, especialmente en el área de visión por ordenador (Computer Vision). La capacidad de los ordenadores para analizar y comprender imágenes ha llevado al desarrollo de diversas técnicas y modelos que han revolucionado sectores como la medicina, la seguridad y el reconocimiento de objetos.

En este contexto, las redes de cápsulas han surgido como una nueva arquitectura de aprendizaje automático que muestra promesas en la tarea de clasificación de imágenes. A diferencia de las redes neuronales convolucionales tradicionales, las redes de cápsulas se basan en la idea de que la información visual se puede representar de manera más efectiva utilizando "cápsulas", que son conjuntos de neuronas que codifican características específicas de una entidad visual.

El objetivo principal de este trabajo es explorar y evaluar nuevas variantes de modelos de redes de cápsulas en la tarea de clasificación de imágenes. Para ello, se utilizarán dos conjuntos de datos ampliamente utilizados en la comunidad de visión por computadora: CIFAR-10 y un conjunto de imágenes de bandejas de carne.

El primer paso será realizar una revisión exhaustiva de la literatura existente sobre redes de cápsulas, centrándonos en los modelos más destacados, como MoCapsNet, ResCapsNet, CapsNet, DeepCaps y MatCaps. Analizaremos las características principales de cada modelo, incluyendo su arquitectura, enrutamiento y capacidades de generalización.

A continuación, se llevarán a cabo simulaciones utilizando estos modelos en los conjuntos de datos mencionados. Se compararán los resultados en términos de precisión, número de parámetros, consumo de memoria y tiempo de ejecución. También se investigarán los efectos de diferentes configuraciones, como el número de bloques residuales, el optimizador utilizado y el enrutamiento implementado.

Además de la evaluación comparativa de los modelos, se buscará identificar las fortalezas y debilidades de las redes de cápsulas en la tarea de clasificación de imágenes. Se analizarán los resultados obtenidos y se propondrán posibles mejoras y futuras investigaciones en esta área.

En resumen, este trabajo tiene como objetivo proporcionar una visión completa y actualizada de los modelos de redes de cápsulas en la clasificación de imágenes. Se espera que los resultados y conclusiones obtenidos contribuyan al avance de este campo y proporcionen una base sólida para investigaciones adicionales y desarrollos futuros.

2

Estado del Arte

2.1 | Introducción

Como hemos comentado en el capítulo anterior, las redes de cápsulas son una extensión de las redes neuronales convencionales, que buscan superar las limitaciones de las neuronas individuales al considerar agrupaciones de neuronas, conocidas como "cápsulas". Estas cápsulas están diseñadas para representar características específicas de una entidad, como la orientación, la posición, la escala, entre otros atributos, en lugar de simplemente activarse como las neuronas tradicionales. Esta capacidad de las cápsulas para capturar relaciones espaciales y atributos intrínsecos se considera una ventaja fundamental en comparación con las redes neuronales convencionales.

Desafortunadamente, las redes de cápsulas sufren de sus propias limitaciones. Una de ellas es que contienen un alto número de parámetros entrenables, debido a que es necesario que cada cápsula en la capa l compute un voto para cada cápsula en la capa $l + 1$, lo que da lugar a un consumo de memoria que crece de forma cuadrática con el número de cápsulas por capa. Por lo tanto, dichas redes de cápsulas consumen muchos recursos, y sus **requisitos de memoria** son un cuello de botella que evita cualquier posibilidad de entrenar arquitecturas más profundas, y como es de esperar, esto dificulta su viabilidad. Otra limitación es que las redes de cápsulas profundas son inestables, algo que se puede solucionar utilizando *conexiones atajo* (**shortcut connections**) entre las capas.

Una de las formas de solucionar el cuello de botella producido por los requisitos de memoria es sacrificar coste de computación por memoria transformando los bloques de redes residuales en funciones reversibles. Este sacrificio quiere decir que el modelo puede tardar un poco más de tiempo en entrenar a cambio de que haya un consumo menor de memoria.

En este capítulo sobre el estado del arte, exploraremos el estado actual de las redes de cápsulas, analizando los avances más recientes, los enfoques propuestos para dar solución a las limitaciones que hacen que las redes de cápsulas no sean la opción más viable a día de hoy. Además, examinaremos los desafíos técnicos y las limitaciones actuales que enfrenta este campo, así como las direcciones futuras y las tendencias emergentes que podrían impulsar aún más su desarrollo y aplicación.

2.2 | Fundamentos Teóricos

2.2.1 | Redes neuronales convencionales y limitaciones

Las redes neuronales convencionales han sido ampliamente utilizadas en el campo del aprendizaje automático debido a su capacidad para aprender y representar patrones en los datos. Sin embargo, estas redes presentan ciertas limitaciones en la forma en que procesan la información. En las redes neuronales convencionales, las neuronas individuales se activan de manera independiente y no tienen en cuenta las relaciones espaciales entre características. Además, estas redes pueden ser sensibles a cambios en la posición o la escala de los objetos, lo que puede dificultar el aprendizaje de representaciones invariantes.

Las Convolutional Neural Networks (CNNs) no son capaces de relacionar la posición de una característica con respecto a otra, simplemente detectan la presencia de dicha característica y en función de ello establecen si existe el objeto o no. Por esa razón, en la siguiente imagen la CNN detectará que hay una cara humana en ambas imágenes, ya que sabe que una cara está compuesta de dos ojos, una nariz y una boca sin tener en cuenta su posición.

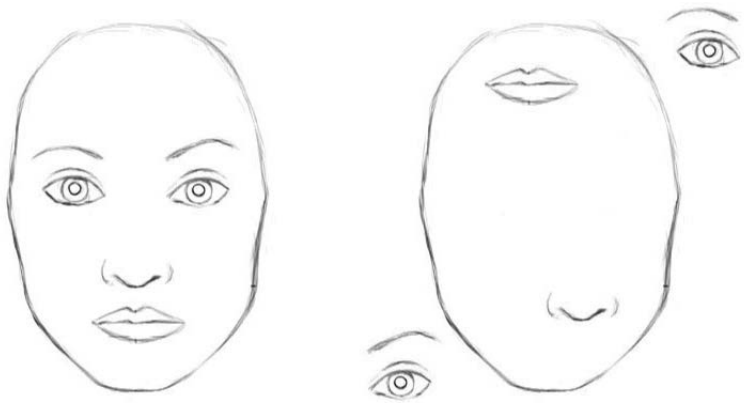


Figura 2.1: Una CNN predice que ambas son caras humanas

2.2.2 | Introducción a las redes de cápsulas

Las redes de cápsulas, propuestas en Sabour et al. (2017), buscan abordar las limitaciones de las redes neuronales convencionales al introducir el concepto de cápsulas. Una cápsula se define como un grupo de neuronas que trabajan juntas para representar características específicas de una entidad. A diferencia de las neuronas individuales, las cápsulas tienen la capacidad de codificar información espacial, atributos intrínsecos y relaciones entre entidades.

2.2.3 | Representación de cápsulas

En una red de cápsulas, cada cápsula está compuesta por un vector de activación, que representa la presencia y los atributos de una entidad. Este vector de activación no solo indica la existencia de una característica, sino también su orientación, posición relativa y escala, lo que permite una representación más rica y detallada.

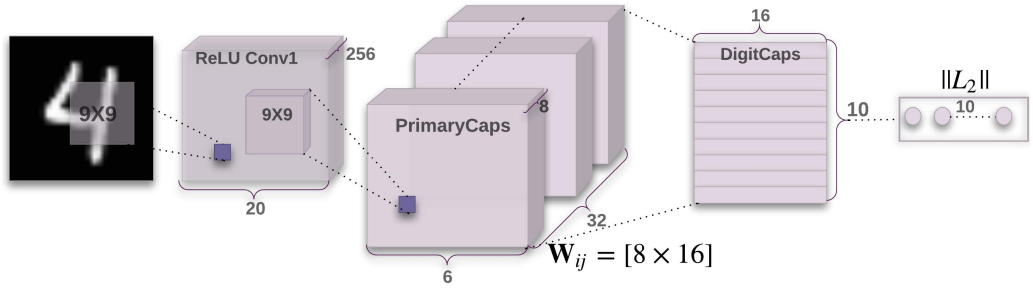


Figura 2.2: Red de cápsulas de 3 capas

2.2.4 | Aprendizaje de relaciones espaciales

Una de las principales características de las redes de cápsulas es su capacidad para aprender relaciones espaciales entre cápsulas. Esto se logra mediante la estimación de matrices de transformación que capturan las transformaciones geométricas entre las características representadas por las cápsulas. El aprendizaje de estas transformaciones permite que la red de cápsulas capture relaciones más complejas entre objetos y atributos.

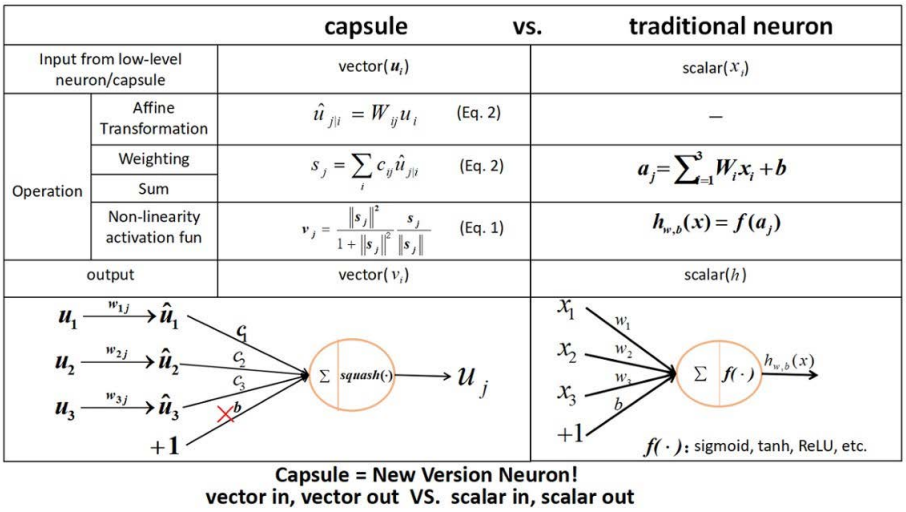


Figura 2.3: Cápsula vs Neurona

2.2.5 | Dinamic Routing by Agreement

Un componente clave en las redes de cápsulas es el algoritmo de Routing by Agreement (RBA). Este algoritmo se encarga de asignar la actividad de las cápsulas en una capa a las cápsulas en la capa siguiente, basándose en la compatibilidad y la coherencia entre las características representadas. El enrutamiento por acuerdo permite que las cápsulas en capas superiores se activen en respuesta a características específicas y mejora la estabilidad y la interpretabilidad de la red.

Procedure 1 Routing algorithm.

procedure ROUTING($\hat{\boldsymbol{u}}_{j|i}, r, l$)
  for all capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow 0$.
  for r iterations **do**
    for all capsule i in layer l: $\mathbf{c}_i \leftarrow \texttt{softmax}(\mathbf{b}_i)$ ▷ `softmax` computes Eq. 3
    for all capsule j in layer $(l+1)$: $\mathbf{s}_j \leftarrow \sum_i c_{ij}\hat{\mathbf{u}}_{j|i}$
    for all capsule j in layer $(l+1)$: $\mathbf{v}_j \leftarrow \texttt{squash}(\mathbf{s}_j)$ ▷ `squash` computes Eq. 1
    for all capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow b_{ij} + \hat{\mathbf{u}}_{j|i}.\mathbf{v}_j$
  return $\mathbf{v}_j$

Figura 2.4: Enrutamiento dinámico

En resumen, las redes de cápsulas introducen una arquitectura innovadora que supera las limitaciones de las redes neuronales convencionales. Al utilizar cápsulas para representar características y aprender relaciones espaciales, estas redes tienen el potencial de capturar información más rica y detallada sobre los objetos y las entidades en los datos. En el siguiente punto, exploraremos los trabajos relacionados y los avances más destacados en el campo de las redes de cápsulas.

2.2.6 | MatCaps

Un año después de la publicación del paper original (Sabour et al. (2017)), se publicó el paper `Matrix Capsules with EM Routing` (Hinton et al. (2018)). La contribución clave de dicho paper es la propuesta de un mejorado algoritmo de enrutamiento para las cápsulas. El enrutamiento es el proceso de determinar el flujo de información entre las cápsulas en diferentes capas. Los autores introducen un mecanismo iterativo de enrutamiento por acuerdo que emplea el algoritmo EM (Expectation maximization).

procedure EM ROUTING($\boldsymbol{a}, V$)
  $\forall i \in \Omega_L, j \in \Omega_{L+1}$: $R_{ij} \leftarrow 1/|\Omega_{L+1}|$
  for t iterations **do**
    $\forall j \in \Omega_{L+1}$: M-STEP($\boldsymbol{a}, R, V, j$)
    $\forall i \in \Omega_L$: E-STEP($\mu, \sigma, \boldsymbol{a}, V, i$)
  return $\boldsymbol{a}, M$

Figura 2.5: Enrutamiento EM

El enrutamiento EM ayuda a establecer conexiones entre las cápsulas basándose en el acuerdo de sus predicciones y la entrada real. Durante el entrenamiento, el algoritmo ajusta los *coeficientes de emparejamiento* entre las cápsulas, permitiéndoles llegar a un consenso sobre la instanciación de parámetros de las entidades (objetos/características) que representan.

El método introducido en este paper demostró resultados prometedores en diversos *benchmarks* de *datasets*, superando en rendimiento a CNNs tradicionales en tareas como el reconocimiento de objetos. También mostró capacidades superiores de generalización al tratar casos de variación de posición y oclusiones.

En este paper también se introduce el concepto de *cápsula matricial*, la cual es un tipo de cápsula que representa la entidad u objeto. Consiste de una matriz de neuronas en lugar de una neurona individual como en el primer paper. Cada fila de la matriz representa una instanciación del objeto, codificando sus propie-

dades como la posición, escala, deformación, presencia, etc.

Las dimensiones de la matriz corresponden a los diferentes aspectos o atributos del objeto. Por ejemplo, en el contexto de reconocimiento de imágenes, una cápsula matricial puede codificar la posición, escala, orientación y otras propiedades relevantes de un objeto.

La estructura matricial de las cápsulas les permite capturar información valiosa sobre las propiedades de un objeto de manera jerárquica y estructurada. El uso de matrices facilita la transformación y manipulación de las propiedades durante el enrutamiento, permitiendo a las cápsulas preservar las relaciones espaciales y manejar variaciones posicionales de manera más efectiva que en las redes neuronales tradicionales.

En resumen, las cápsulas matriciales son un esquema de representación usado en las redes de cápsulas para modelar entidades u objetos. Consisten en matrices de neuronas que codifican diferentes propiedades de los objetos, permitiendo una representación más estructurada y flexible de las jerarquías y variaciones espaciales.

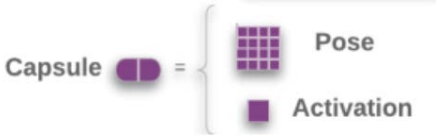


Figura 2.6: Cápsula matricial

2.3 | Trabajos relacionados

Una de las maneras de aumentar el rendimiento en una red neuronal es incrementando la profundidad de dicha red. Sin embargo, uno de los problemas a los que nos enfrentamos al aumentar la profundidad de una red es el gradiente desvanecedor. Cuando la señal de error se transmite a través de muchas capas, ésta puede desvanecerse y desaparecer antes de que llegue al principio de la red, lo que dificulta la convergencia del modelo. El modelo ResNet (He et al. (2015)) se propone resolver este problema. La estructura básica de ResNet es la unidad de aprendizaje residual, la cual utiliza múltiples capas de redes paramétricas para aprender la relación entre la entrada y la salida. La unidad residual puede transmitir directamente la información de entrada a la salida a través de otro canal, lo cual reduce la pérdida de información durante la transmisión, y mantiene la integridad de la información. Al mismo tiempo, ResNet puede reducir la dificultad del entrenamiento y el aprendizaje. Usando las ResNet como modelo, propuestas como DeepCaps (Rajasegaran et al. (2019)) y Res-CapsNet (Jia et al. (2022)) intentan aplicar métodos como las *skip connections* entre otros a las redes de cápsulas con el propósito de hacerlas más profundas y reducir su dificultad de entrenamiento.

En DenseNets (Huang et al. (2017)) aseguran el flujo máximo de información entre las capas de una red, conectando todas las capas directamente con el resto. Para preservar la naturaleza de la *propagación hacia adelante* (feed-forward), cada capa obtiene entradas adicionales de todas las capas que le preceden y pasa sus propios mapas de características a las capas siguientes. Crean pequeños caminos desde las primeras capas hasta las últimas.

La idea de agrupar las neuronas en capas fue propuesta en Hinton et al. (2011). Y como extensión a ello, Sabour et al. (2017) propusieron el algoritmo de enrutamiento dinámico entre cápsulas como hemos visto en el apartado anterior. Como añadido al enrutamiento dinámico, Hinton et al. (2018) usaron el enrutamiento EM para cápsulas matriciales representando cada entidad mediante una matriz de posición. Han habido muchas contribuciones a esto: HitNet (Deliège

et al. (2018)) usa una capa de "acierto o fallo" (hit-or-miss layer) para aumentado de datos. Wang and Liu (2018) resuelve el enrutamiento dinámico como un problema de optimización, y consigue mejor rendimiento introduciendo divergencia KL entre la distribución de acople. CapsGan (Jaiswal et al. (2018)) utiliza una red de cápsulas como el discriminador en el proceso GAN, para obtener mejores resultados visuales que las GANs convolucionales.

SegCaps (LaLonde and Bagci (2018)) usa capsulas para segmentación de imágenes y obtienen resultados de estado del arte en el dataset LUNA16. Este modelo utiliza convoluciones 2D para el proceso de votación. Al utilizar convoluciones 2D, se toman todas las cápsulas a lo largo de la profundidad como entradas para la transformación, mezclando así la información contenida en las cápsulas.

Otros trabajos han ido más enfocados a resolver el problema principal que presentan las redes de cápsulas el cual es el alto coste computacional que suponen. Una forma de reducir el coste computacional y de memoria es usando arquitecturas reversibles, como en el modelo RevNet (Gomez et al. (2017)). Una red es reversible si se pueden recalcular todas sus activaciones en el *backward pass*. Una de las ventajas de las redes invertibles es que se puede realizar el *backpropagation* sin tener que guardar las activaciones del *forward pass*, lo que disminuye el uso de memoria en gran medida. Con el modelo Momentum ResNets, Sander et al. (2021) introdujeron una forma de transformar cualquier modelo ResNet existente en un Momentum ResNet - sin tener que cambiar la red - haciendo que los bloques residuales de la red fuesen reversibles. Momentum ResNet cambia el modo del *forward pass* de una red residual clásica introduciendo un término de momento (momentum term) γ. Los modelos Momentum ResNet son una generalización de las ResNet clásicas (para $\gamma = 0$) y RevNets (para $\gamma = 1$). Estas ideas se aplican a las redes de cápsulas en el modelo Momentum CapsNet (Gugglberger et al. (2022)) con el objetivo de reducir el número de parámetros, el uso de memoria y el coste computacional en general.

2.4 | Modelos destacados

2.4.1 | DeepCaps

En 2019 se publicó el paper `DeepCaps: Going Deeper with Capsule Networks`, Rajasegaran et al. (2019), surgió bajo la premisa de que las redes de cápsulas tienen mucho potencial pero no es aprovechado y su rendimiento es pobre en *benchmarks* de *datasets* más complejos que los habituales como *MNIST*. Inspirándose en el éxito de las CNNs al pasar a arquitecturas más profundas, presentan *DeepCaps*, una arquitectura de red de cápsulas profunda que utiliza la novedosa convolución 3D basada en el algoritmo de enrutamiento dinámico.

Un intento de hacer las redes de cápsulas más profundas sería apilar capas de cápsulas completamente conectadas, pero esto tiene varias limitaciones. Primero, el enrutamiento dinámico usado en redes de cápsulas es un procedimiento extremadamente costoso computacionalmente hablando, y tener múltiples capas de enrutamiento incurriría en altos costes en los tiempos de entrenamiento e inferencia. Segundo, se ha demostrado que apilar capas de cápsulas completamente conectadas resulta en un entrenamiento pobre en las capas intermedias, debido a que cuando hay demasiadas cápsulas, los coeficientes de emparejamiento tienden a ser demasiado pequeños, disminuyendo el flujo del gradiente e inhabilitando el aprendizaje. Tercero, también se ha demostrado que , especialmente en las capas inferiores, las unidades correladas tienden a concentrarse en regiones locales. Aunque, el enrutamiento localizado pude aprovechar esta observación, dicho enrutamiento localizado no puede ser implementado en cápsulas completamente conectadas.

Con el objetivo de abordar estas limitaciones se proponen las siguientes soluciones en el paper:

- Para **reducir la complejidad computacional** introducida por el hecho de que múltiples capas requieran enrutamiento dinámico:
  - **Reducir el número de iteraciones de enrutamiento** en las capas ini-

ciales que son más grandes en tamaño reduce la complejidad a la vez que no afecta a las características ya que no son características complejas.

- Usando un **enrutamiento** inspirado en la **convolución 3D** en las capas intermedias -debido a la compartición de parámetros- reduce el número de parámetros.

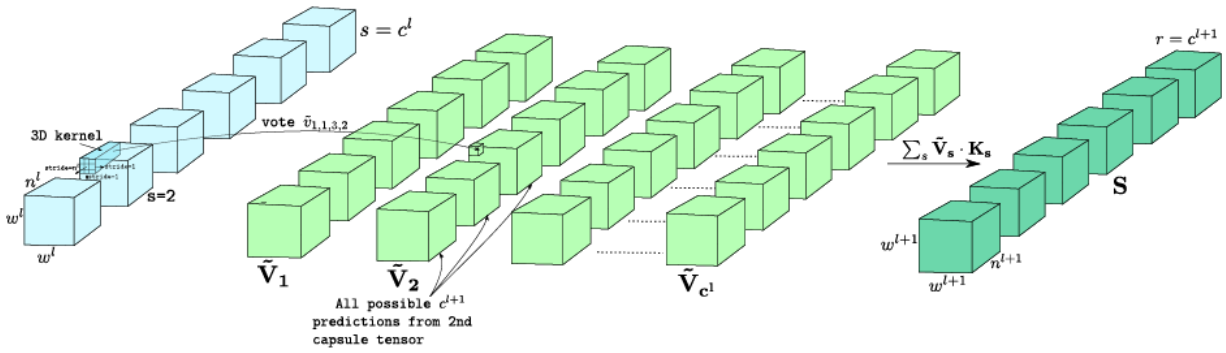


Figura 2.7: Enrutamiento dinámico usando convoluciones 3D

Algorithm 1 Dynamic Routing using 3D convolution

procedure ROUTING
Require: $\Phi^l \in \mathbb{R}^{(w^l, w^l, c^l, n^l)}$, r and c^{l+1}, n^{l+1}
  $\tilde{\Phi}^l \leftarrow \texttt{Reshape}(\Phi_l) \in \mathbb{R}^{(w^l, w^l, c^l \times n^l, 1)}$
  $\mathbf{V} \leftarrow \texttt{Conv3D}(\tilde{\Phi}^l) \in \mathbb{R}^{(w^{l+1}, w^{l+1}, c^l, c^{l+1} \times n^{l+1})}$
  $\tilde{\mathbf{V}} \leftarrow \texttt{Reshape}(\mathbf{V}) \in \mathbb{R}^{(w^{l+1}, w^{l+1}, n^{l+1}, c^{l+1}, c^l)}$
  $\mathbf{B} \leftarrow \mathbf{0} \in \mathbb{R}^{(w^{l+1}, w^{l+1}, c^{l+1}, c^l)}$
  Let $p \in w^{l+1}, q \in w^{l+1}, r \in c^{l+1}$ and $s \in c^l$
  for i iterations **do**
    for all p, q, r, $k_{pqrs} \leftarrow \texttt{softmax\_3D}(b_{pqrs})$
    for all s, $S_{pqr} \leftarrow \sum_s k_{pqrs} \cdot \tilde{V}_{pqrs}$
    for all s, $\hat{S}_{pqr} \leftarrow \texttt{squash\_3D}(S_{pqr})$
    for all s, $b_{pqrs} \leftarrow b_{pqrs} + \hat{S}_{pqr} \cdot \tilde{V}_{pqrs}$
  return $\Phi^{l+1} = \hat{\mathbf{S}}$

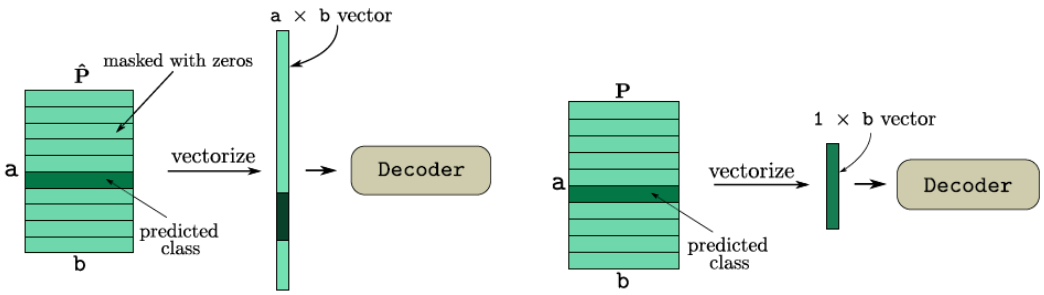
Figura 2.8: Algoritmo del enrutamiento dinámico mediante convoluciones 3D

- Para abordar el problema del pobre aprendizaje en las capas intermedias debido al apilamiento de capas se **mejora el flujo del gradiente** mediante

skip connections (conexiones de salto).

- **Enrutamiento localizado** que sea capaz de capturar con más eficacia información de alto nivel en enrutamiento completamente conectado.

En el paper de 2017, se utilizaba regularización a través de la incorporación del error de reconstrucción para reducir el *overfitting*. Sin embargo, es necesaria una regularización más fuerte cuando se desarrollan redes profundas, debido al inherente aumento de la complejidad del modelo conforme aumenta la profundidad. Por lo tanto se propone el *decodificador independiente de clase* (class-independent decoder). En el decodificador actual no es posible garantizar que la propiedad física representada por un parámetro de instanciación dado sea igual en todas las clases. En el decodificador propuesto, se garantiza que la propiedad representada será la misma para cada parámetro de instanciación dado en todas las clases, proporcionando una mayor controlabilidad, la cual supone una ventaja inmensa en aplicaciones prácticas y estudios teóricos.



(a) Decodificador usado en Sabour et al. (2017) (b) Decodifcador propuesto

Figura 2.9: Diferencia de decodificadores

2.4.2 | ResCapsNet

En 2022 fue publicado el paper `Res-CapsNet: Residual Capsule Network for Data Classification`, Jia et al. (2022), y al igual que en el caso anterior se centra en resolver los problemas principales de las redes de cápsulas.

Con el objetivo de resolver la contradicción entre la profundidad de la red y la extracción de información profunda, y mejorar la precisión de clasificación en imágenes complejas, el model Res-CapsNet basado en el modelo ResNet, He et al. (2015), es propuesto. Entre las contribuciones están,

- Se utiliza un pequeño kernel de convolución y *identity mapping* para extraer diferentes características, y se utiliza una red residual para la fusión de características para asegurar la completa extracción de características de una imagen para la codificación en la red de cápsulas. Esta estructura resuelve el problema de múltiples parámetros y degradación del modelo causado por la simple superposición de las capas (la apilación de capas que mencionamos en el apartado anterior). Mientras tanto, es beneficioso extraer características profundas.
- Antes de codificar la red de cápsulas, se usa la función de activación betamish (2.1) para incrementar la no linealidad del modelo de clasificación, permitiéndole obtener mejor flujo del gradiente cuando tiene ligeros valores negativos, por lo tanto asegurando que la red de cápsulas puede activar más neuronas.

$$f(x) = x \tanh(softplus(x)) = x \tanh\left(\ln\left(1 + e^{x}\right)\right) \tag{2.1}$$

- Se obtiene el módulo de reconstrucción usando una deconvolución en cascada de 4 capas, la cual puede capturar más relaciones espaciales en la imagen de entrada para garantizar la autenticidad de la imagen reconstruida a la vez que se reduce el número de parámetros.

La estructura propuesta de la red de cápsulas residual se muestra en la figura (2.10), la cual incluye tres partes conectadas en secuencia, que son, el *residual extraction module* (REM), el *capsule module* (CM) y el *deconvolution reconstruction module* (DRM). El propósito del REM es extraer información profunda de la imagen, como las características de las texturas de los objetos. El CM se utiliza para clasificar, lo cual convierte neuronas escalares a vectores de neuronas a través de la capa de cápsulas principal, y usa el algoritmo de enrutamiento para actualizar iterativamente los parámetros entre las cápsulas, y finalmente obtiene los resultados de reconocimiento de la *digital capsule layer*. El DRM es responsable de la reconstrucción de los resultados de reconocimiento de la *digital capsule layer*, y de reducir el error de reconstrucción usando la deconvolución de 4 capas.

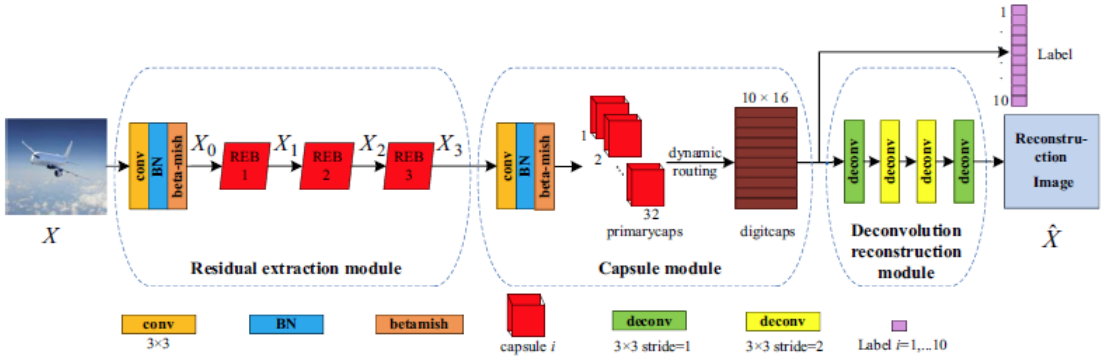


Figura 2.10: El modelo Res-CapsNet

2.4.3 | MoCapsNet

En el paper `Momentum Capsule Networks`, Gugglberger et al. (2022), se presenta otro intento de dar solución a los problemas principales que presentan las redes de cápsulas. Uno de esos problemas es común en la mayoría de CNNs, el cual es el gradiente desvanecedor que ocurre en redes muy profundas. Para abordarlo se utilizan bloques residuales con conexiones de salto, dando a lugar a Res-CapsNet (el modelo del apartado anterior).

Otro problema que tenemos en CapsNet es el cuello de botella de memoria. Estas redes requieren de una gran cantidad de memoria por lo que su entrenamiento puede ser problemático. Una manera de superar este inconveniente es hacer un *trade-off* entre memoria y computación convitiendo los bloques de redes residuales en funciones reversibles.

Bajo esta premisa se propone el modelo *Momentum Capsule Network* o *MoCapsNet*. Con el que intentan demostrar que se pueden diseñar redes de cápsulas de manera que una parte de la red pueda ser invertida.

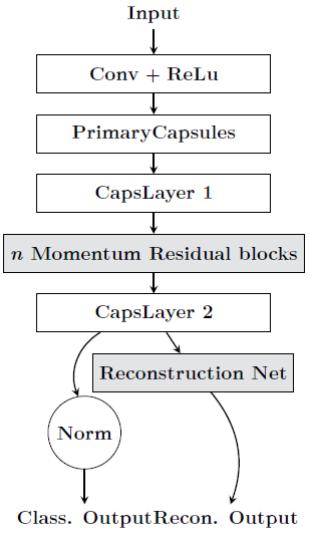
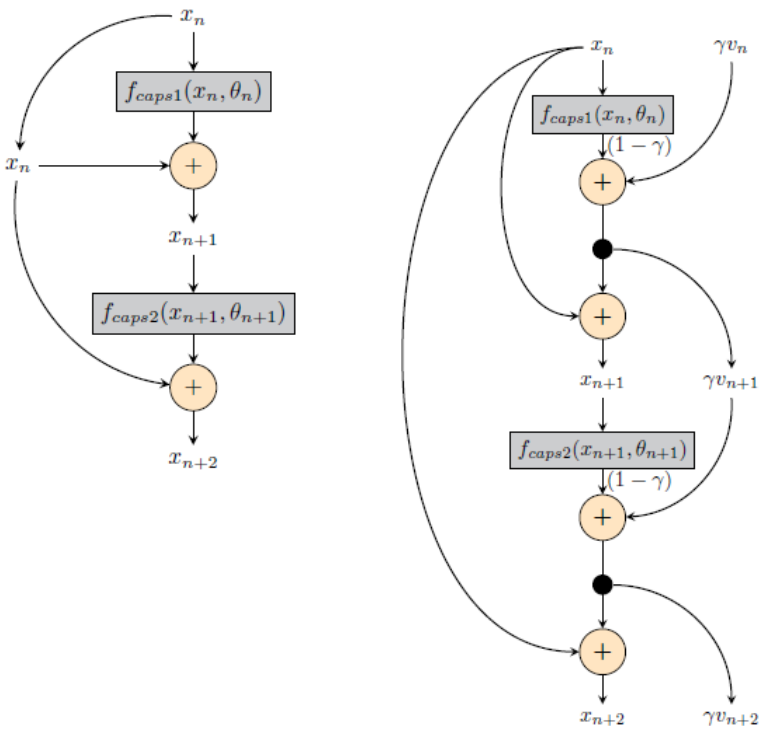


Figura 2.11: Arquitectura de MoCapsNet

Una red se considera reversible si se pueden recalcular todas sus activaciones en el *backward pass*. Una ventaja de las redes invertibles es que se puede realizar el *backpropagation* sin tener que guardar las activaciones en el *forward pass*, lo que disminuye el uso de memoria de forma significativa. Varias arquitecturas reversibles o parcialemente reversibles han sido propuestas, como RevNet. Momentum ResNets cambia la *forward rule* de la clásica red residual insertando un término de *momentum* γ. Momentun ResNets sería por tanto una generalización de la clásica ResNet (para $\gamma = 0$) y RevNet (para $\gamma = 1$). Adicionalmente, Momentum ResNets tiene un mayor grado de expresividad que las redes residuales clásicas.



(a) Bloque residual en ResCapsNet (b) Bloque residual en MoCapsNet

Figura 2.12: Bloques residuales

2.5 | Resultados

En esta sección vamos a evaluar cuales son los resultados que reportan en cada uno de los papers y como se comparan entre ellos.

A continuación vemos los resultados mostrados en el paper de *DeepCaps*. En los cuales se puede observar que el modelo DeepCaps de 7 agrupaciones supera al resto de variantes de redes de cápsulas del experimento en los datasets CIFAR10, SVHN y Fashion MNIST mientras que el modelo base CapsNet de 2017 es el que mejor rendimiento presenta en el dataset MNIST. Sin embargo, es necesario resaltar que como se ve en la parte superior, otros modelos de CNNs son superiores en todos los datasets.

Model	CIFAR10	SVHN	F-MNIST	MNIST
DenseNet [9]	96,40 %	98,41 %	95,40 %	-
ResNet [6]	93,57 %	-	-	99,59 %
DPN [1]	96,35 %	-	95,70 %	-
Wan et al. [22]	-	-	-	99,79 %
Zhong et al. [27]	96,92 %	-	96,35 %	-
Sabour et al. [19]	89,40 %	95,70 %	93,60 %	**99,75** %
Nair et al. [17]	67,53 %	91,06 %	89,80 %	99,50 %
HitNet [3]	73,30 %	94,50 %	92,30 %	99,68 %
DeepCaps	91,01 %	97,16 %	94,46 %	99,72 %
DeepCaps (7-ensembles)	**92,74** %	**97,56** %	**94,73** %	-

Cuadro 2.1: Resultados del paper *DeepCaps*

Pasando a los resultados que se muestran en el paper de MoCapsNet podemos ver la siguiente tabla en la que se muestran los resultados de MoCapsNet y ResCapsNet para MNIST y SVHN en función de los bloques residuales que se utilizan. Y se puede observar que para MNIST el modelo ganador es MoCapsNet con 7 bloques residuales y para SVHN MoCapsNet con 3 bloques residuales.

	MNIST		SVHN	
Blocks	ResCapsNet	MoCapsNet	ResCapsNet	MoCapsNet
1	$99{,}41 \pm 0{,}01$	$99{,}42 \pm 0{,}04$	$92{,}32 \pm 0{,}52$	$92{,}68 \pm 0{,}23$
2	$99{,}28 \pm 0{,}05$	$99{,}25 \pm 0{,}11$	$92{,}57 \pm 0{,}12$	$92{,}54 \pm 0{,}43$
3	$99{,}30 \pm 0{,}08$	$99{,}31 \pm 0{,}04$	$92{,}13 \pm 0{,}59$	$\mathbf{93{,}00 \pm 0{,}65}$
4	$99{,}38 \pm 0{,}03$	$99{,}42 \pm 0{,}05$	$91{,}87 \pm 0{,}95$	$92{,}03 \pm 1{,}12$
5	$99{,}30 \pm 0{,}03$	$99{,}27 \pm 0{,}06$	$92{,}58 \pm 0{,}42$	$92{,}78 \pm 0{,}81$
6	$99{,}35 \pm 0{,}02$	$99{,}38 \pm 0{,}02$	$92{,}37 \pm 0{,}16$	$92{,}50 \pm 1{,}60$
7	$99{,}36 \pm 0{,}02$	$\mathbf{99{,}54 \pm 0{,}23}$	$92{,}52 \pm 0{,}23$	$91{,}35 \pm 1{,}60$
8	$99{,}34 \pm 0{,}06$	$99{,}37 \pm 0{,}06$	$92{,}63 \pm 0{,}34$	$91{,}20 \pm 1{,}58$

Cuadro 2.2: MoCapsNet vs ResCapsNet (MNIST y SVHN)

En la siguiente tabla podemos ver la siguiente comparativa pero para CIFAR-10 y CIFAR-100. Como podemos observar, los resultados son bastante peores en general debido a que son datasets más complejos. En este caso el ganador en CIFAR-10 es MoCapsNet con 1 bloque residual y en CIFAR-100 es MoCapsNet con 8 bloques residuales, ya que ResCapsNet apenas supera el 1 % de precisión. Parece ser que en el caso de un dataset complejo como CIFAR-100 el modelo se beneficia notoriamente de un mayor número de bloques, mientras que en CIFAR-10 la diferencia no es tan significativo.

	CIFAR-10		CIFAR-100	
Blocks	ResCapsNet	MoCapsNet	ResCapsNet	MoCapsNet
1	$71{,}49 \pm 0{,}39$	$\mathbf{72{,}18 \pm 0{,}62}$	$\approx 1{,}00$	$9{,}53 \pm 0{,}04$
2	$70{,}59 \pm 0{,}90$	$71{,}65 \pm 0{,}62$	$\approx 1{,}00$	$26{,}05 \pm 0{,}12$
3	$71{,}09 \pm 0{,}86$	$71{,}08 \pm 0{,}71$	$\approx 1{,}00$	$33{,}24 \pm 0{,}29$
4	$71{,}50 \pm 0{,}57$	$70{,}29 \pm 0{,}06$	$\approx 1{,}00$	$39{,}57 \pm 0{,}11$
5	$71{,}94 \pm 0{,}34$	$71{,}74 \pm 0{,}74$	$\approx 1{,}00$	$42{,}54 \pm 0{,}16$
6	$71{,}22 \pm 0{,}70$	$71{,}17 \pm 0{,}32$	$\approx 1{,}00$	$42{,}79 \pm 0{,}13$
7	$71{,}85 \pm 0{,}91$	$71{,}50 \pm 0{,}93$	$\approx 1{,}00$	$43{,}22 \pm 0{,}17$
8	$70{,}90 \pm 1{,}02$	$70{,}48 \pm 0{,}69$	$\approx 1{,}00$	$\mathbf{43{,}48 \pm 0{,}06}$

Cuadro 2.3: MoCapsNet vs ResCapsNet (CIFAR-10 y CIFAR-100)

Si comparamos el modelo MoCapsNet, el cual podemos ver que supera a ResCapsNet en todos los casos, y DeepCapsNet vemos que este último afirma ser el mejor modelo de todos los mencionados en este artículo con un 92,74 % de precisión en un dataset relativamente complejo como CIFAR-10.

3

Materiales y Métodos

En esta sección describiremos en detalle el equipo y las herramientas utilizadas para llevar a cabo las simulaciones y el entrenamiento de los distintos modelos que se han llevado a cabo.

3.1 | Entorno de desarrollo

Para el desarrollo y ejecución de las simulaciones y entrenamiento de los modelos, hemos utilizado Google Colab, una plataforma en línea de Google que proporciona acceso a recursos informáticos de gran potencia y herramientas de desarrollo de código abierto. Google Colab ofrece una interfaz de Jupyter Notebook que permite escribir y ejecutar código Python de manera interactiva, lo que facilita el proceso de desarrollo, la experimentación y la visualización de gráficas y datos. Además, al ser una plataforma basada en la nube, elimina la necesidad de configurar y mantener una infraestructura local costosa, lo que brinda una solución conveniente y escalable para realizar las simulaciones.

Figura 3.1: Logo de Google Colab

3.2 | Unidad de Procesamiento Gráfico (GPU)

Para entrenar modelos de Deep Learning (DL) o Machine Learning (ML) se suelen emplear GPUs en lugar de CPUs debido a que las primeras son más eficientes y presentan un mayor rendimiento en tareas que se benefician de cálculos y operaciones en paralelo. Lo que da lugar a unos tiempos de entrenamiento mucho menores. En la plataforma Google Colab se ofrecen tres GPUs de alto rendimiento:

- NVIDIA Tesla T4
- NVIDIA V100
- NVIDIA P100

En nuestro caso hemos tenido acceso a la **NVIDIA Tesla T4**, una potente tarjeta gráfica diseñada específicamente para cargas de trabajo de aprendizaje automático (Machine Learning) y análisis de datos de alto rendimiento. La Nvidia Tesla T4 ofrece un rendimiento excepcional gracias a su arquitectura de GPU acelerada por tensor, que permite realizar cálculos matemáticos complejos de manera simultánea y en paralelo. Esta capacidad de procesamiento paralelo resultó fundamental para acelerar el entrenamiento y la evaluación de los modelos basados en redes de cápsulas utilizados en este trabajo.

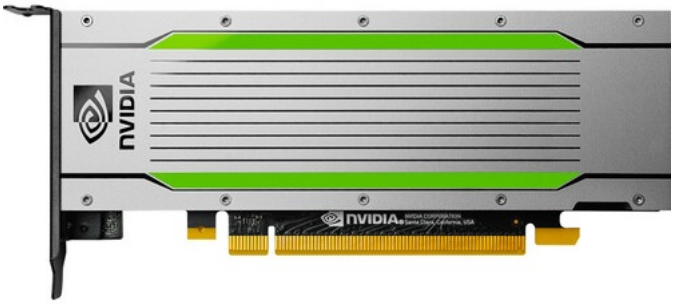


Figura 3.2: Nvidia Tesla T4

La Nvidia Tesla T4 también cuenta con una gran cantidad de memoria de vídeo (VRAM), 16 GB para ser exactos, y ancho de banda de memoria, lo que permite trabajar con conjuntos de datos más grandes y acelerar aún más el rendimiento del entrenamiento. Esto resulta especialmente beneficioso cuando se aplican técnicas de Deep Learning (DL), que suelen requerir un alto consumo de recursos computacionales.

GPU Architecture	**NVIDIA TURING**
NVIDIA Turing Tensor Cores	**320**
NVIDIA CUDA Cores	**8.1 TFLOPS**
Mixed-Precision (FP16/FP32)	**65 TFLOPS**
INT8	**130 TOPS**
INT4	**260 TOPS**
GPU Memory	**16 GB GDDR6 300 GB/sec**
ECC	**Yes**
Interconnect Bandwidth	**32 GB/sec**
System Interface	**x16 PCIe Gen3**
Form Factor	**Low-Profile PCIe**
Thermal Solution	**Passive**
Compute APIs	**CUDA, NVIDIA TensorRT, ONNX**

Cuadro 3.1: Especificaciones de la Nvidia Tesla T4

3.3 | Framework utilizada: Pytorch

La elección de la biblioteca de aprendizaje automático PyTorch para implementar los modelos de redes de cápsulas se basó en su popularidad y versatilidad en la comunidad de investigación y desarrollo de aprendizaje automático. PyTorch es una biblioteca de código abierto ampliamente utilizada y respaldada por una comunidad activa, lo que garantiza una amplia documentación, recursos y soporte disponibles.

PyTorch proporciona una interfaz intuitiva y flexible para construir y entrenar redes neuronales, lo que facilitó la implementación y experimentación con los modelos de redes de cápsulas en este trabajo. Además, PyTorch ofrece un enfoque dinámico y computacionalmente eficiente para la construcción de gráficos computacionales, lo que permite realizar ajustes y modificaciones en los modelos con facilidad. Esta flexibilidad resultó invaluable a la hora de explorar diferentes arquitecturas y parámetros de las redes de cápsulas, lo que a su vez contribuyó al avance de la investigación en este campo.

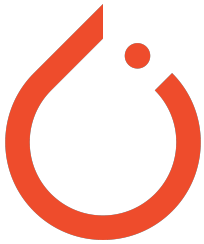


Figura 3.3: Logo de Pytorch

3.4 | Datasets utilizados

En esta sección describiremos los conjuntos de datos utilizados para entrenar los modelos empleados.

3.4.1 | CIFAR-10

El primer conjunto de datos que hemos utilizado es CIFAR-10, un popular conjunto de datos ampliamente utilizado en la comunidad de visión artificial. CIFAR-10 consta de 60.000 imágenes a color (3 canales) en 10 categorías diferentes, con 6.000 imágenes por categoría. Cada imagen tiene una resolución de 32x32 píxeles. Este conjunto de datos es comúnmente utilizado en la evaluación

de algoritmos de clasificación de imágenes debido a la diversidad y a la complejidad de las imágenes en cada categoría.

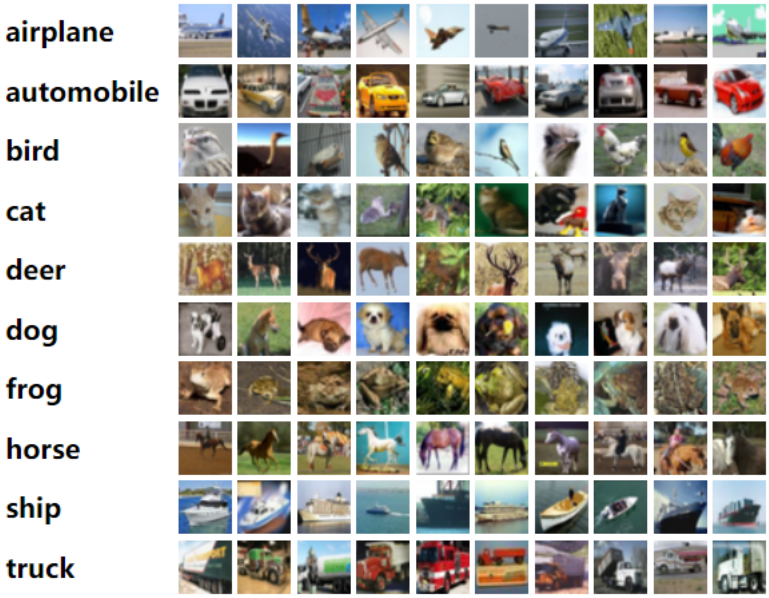


Figura 3.4: CIFAR-10

La elección de CIFAR-10 se basó en su amplia disponibilidad y en su facilidad para integrarlo de manera automática en cualquier modelo utilizando tanto Tensorflow como Pytorch. El conjunto de datos se división en un conjunto de entrenamiento y un conjunto de prueba, siguiendo el estándar de la comunidad de Deep Learning.

3.4.2 | Bandejas de carne

Además de CIFAR-10, hemos utilizado un conjunto de datos personalizado que consiste en imágenes de bandejas de carne. Este conjunto de datos fue elegido para este trabajo con el objetivo de explorar el rendimiento de los modelos de redes de cápsulas en un escenario con aplicación *real*. A diferencia de otros datasets como MNIST o CIFAR que son utilizados como un *benchmark*.

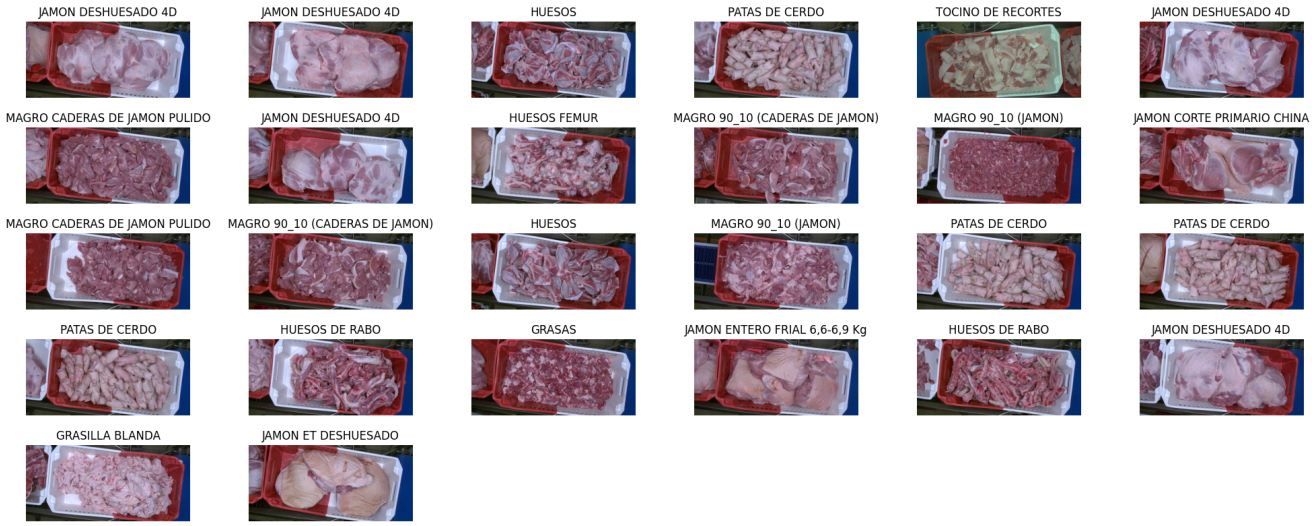


Figura 3.5: Dataset de bandejas de carne

El conjunto de datos de bandejas de carne contiene imágenes de diferentes tipos de carne dispuestas en bandejas como se puede observar en la figura 3.5. Cada imagen fue etiquetada con la categoría correspondiente (por ejemplo, jamón deshuesado, huesos, tocino, patas de cerdo etc.) para tareas de clasificación.

3.5 | Preprocesamiento de los Datasets y aumentado de datos

En esta sección se describe el preprocesado realizado en los conjuntos de datos utilizados para entrenar los modelos de redes de cápsulas utilizados en el trabajo. Se incluyen las transformaciones aplicadas, así como la normalización de los datos.

Se utilizaron dos conjuntos de datos: CIFAR-10 y un conjunto personalizado de bandejas de carne. Para ambos conjuntos de datos, se aplicaron transformaciones de aumento de datos para enriquecer la diversidad de las imágenes y mejorar la capacidad del modelo para generalizar.

Las transformaciones aplicadas al conjunto de datos incluyen:

- **Redimensionamiento:** Se redimensionaron las imágenes al tamaño deseado utilizando `transforms.Resize(size)`, donde `size` representa las dimensiones requeridas.
- **Volteo horizontal aleatorio**: Se aplicó un volteo horizontal aleatorio utilizando `transforms.RandomHorizontalFlip()` para introducir variaciones en la orientación de las imágenes.
- **Ajuste de color:** Se aplicó el ajuste de color utilizando `transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1)` para modificar el brillo, el contraste, la saturación y el tono de las imágenes.
- **Escala de grises aleatoria:** Se aplicó una escala de grises aleatoria con una probabilidad del 10 % utilizando `transforms.RandomGrayscale(p=0.1)` para convertir algunas imágenes a escala de grises.
- **Transformación afín aleatoria**: Se aplicó una transformación afín aleatoria utilizando `transforms.RandomAffine(degrees=10, translate=(0.1, 0.1), scale=(0.8, 1.2), shear=10)` para realizar rotaciones, traslaciones, escalado y cizallamiento aleatorios en las imágenes.
- **Perspectiva aleatoria:** Se aplicó una perspectiva aleatoria con una probabilidad del 50 % utilizando `transforms.RandomPerspective(distortion_scale=0.1, p=0.5)` para simular deformaciones de perspectiva en las imágenes.
- **Desenfoque gaussiano:** Se aplicó un desenfoque gaussiano utilizando `transforms.GaussianBlur(kernel_size=3, sigma=(0.1, 2.0))` para suavizar las imágenes y agregar variaciones sutiles.
- **Rotación aleatoria:** Se aplicó una rotación aleatoria de hasta 10 grados utilizando `transforms.RandomRotation(10)` para introducir variaciones en la orientación de las imágenes.

- **Recorte aleatorio:** Se aplicó un recorte aleatorio con relleno de ceros utilizando `transforms.RandomCrop(size, padding=2)` para extraer una región aleatoria de las imágenes con el tamaño deseado.
- **Conversión a tensor y normalización:** Finalmente, se convirtieron las imágenes en tensores utilizando `transforms.ToTensor()`. Además, se normalizaron los tensores utilizando `transforms.Normalize(mean, std)`, donde `mean` y `std` son los valores de media y desviación estándar respectivamente.

Estas transformaciones se aplicaron secuencialmente utilizando `transforms.Compose()` para obtener un pipeline de transformación completo.

El procesamiento y aumentado de datos realizados en los datasets ayudó a mejorar la capacidad de generalización y robustez de los modelos de redes de cápsulas entrenados.

Figura 3.6: Ejemplo de las transformaciones aplicadas

3.6 | Procedimiento de Simulación

En esta sección, se describe el procedimiento seguido para llevar a cabo las simulaciones de los modelos de redes de cápsulas.

3.6.1 | Obtención de la implementación del modelo

El primer paso consistió en buscar implementaciones existentes de los modelos de redes de cápsulas requeridos para este trabajo en plataformas de desarrollo colaborativo como GitHub. A través de una búsqueda exhaustiva, se localizaron repositorios que ofrecían implementaciones de los modelos deseados.

Una vez identificados los repositorios adecuados, se realizó un *fork* de cada uno de ellos en GitHub. Esto permitió tener una copia personalizada del repositorio en la propia cuenta de GitHub, lo que facilitó la realización de mejoras y correcciones de errores necesarios para que los modelos funcionaran correctamente con los datasets específicos utilizados en este trabajo.

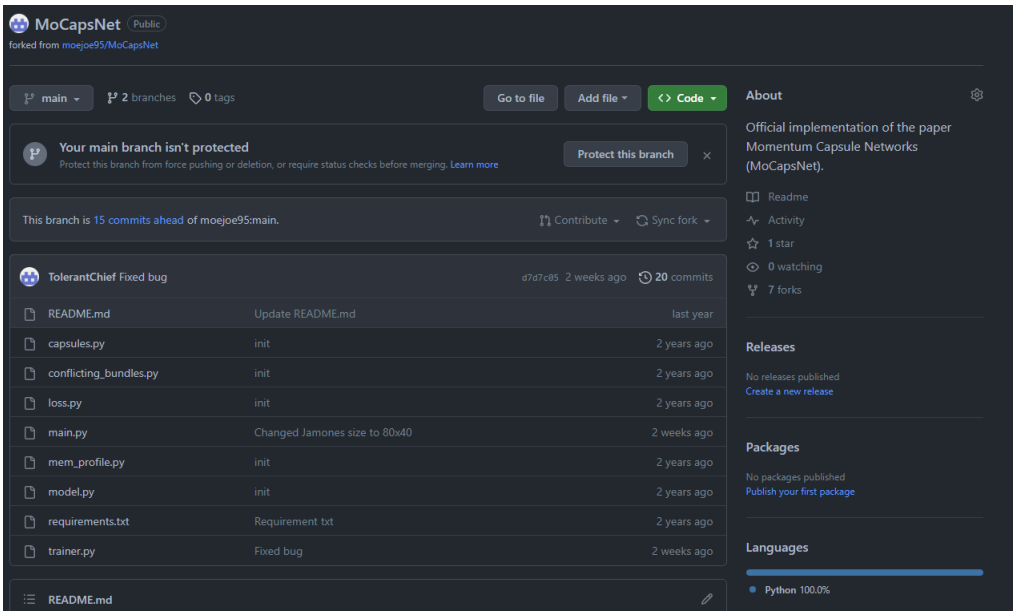


Figura 3.7: Repositorio de GitHub

3.6.2 | Entrenamiento de los modelos

Una vez realizadas las modificaciones en los modelos necesarias para ser utilizados, se procedió al entrenamiento de los modelos. Utilizando la implementación personalizada del modelo, se definieron los hiperparámetros relevantes, como la tasa de aprendizaje, el número de épocas y el tamaño del lote (*batch size*).

Durante el entrenamiento, se emplearon técnicas adicionales para mejorar el rendimiento y la estabilidad de los modelos. Una de estas técnicas fue la implementación del *Early Stopping* mediante la librería propuesta en Harris et al. (2018), que permitió detener el entrenamiento prematuramente si no se observaba una mejora significativa en la métrica de desempeño elegida (por ejemplo, la precisión).

Asimismo, se agregaron funciones para guardar el modelo con la mejor precisión alcanzada durante el entrenamiento y para poder exportar gráficas que representaran la precisión y las pérdidas a lo largo del entrenamiento. Esto facilitó el análisis y la visualización de los resultados obtenidos.

3.6.3 | Evaluación de los modelos

Una vez completado el entrenamiento, se evaluaron los modelos utilizando los conjuntos de prueba previamente separados. Se calcularon diversas métricas de evaluación, como la precisión, las pérdidas, el número de parámetros del modelo, la cantidad de memoria utilizada y el tiempo de ejecución, para analizar el desempeño y la capacidad de generalización de los modelos.

Adicionalmente, se llevaron a cabo análisis adicionales, como la visualización de las predicciones realizadas por los modelos en un conjunto de imágenes de prueba, con el objetivo de comprender mejor el comportamiento de los modelos y detectar posibles áreas de mejora.

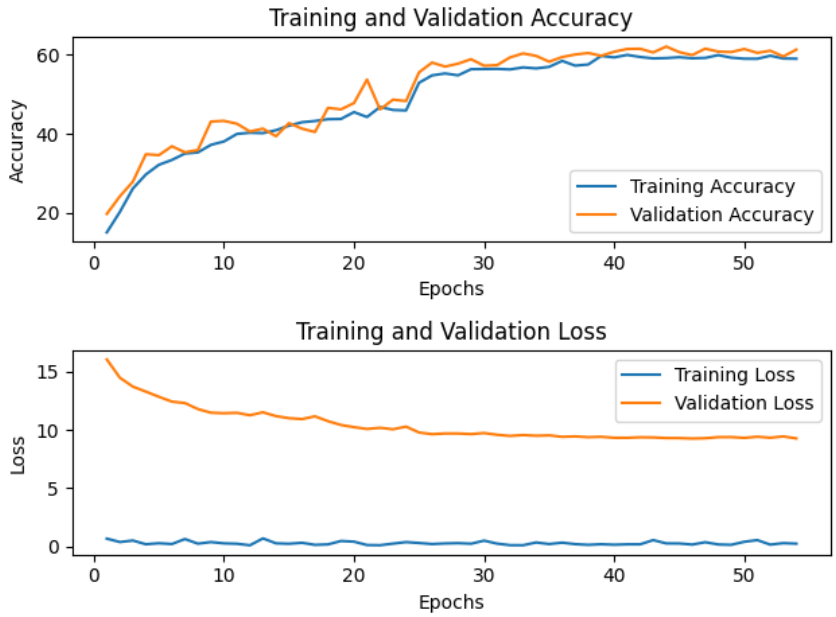


Figura 3.8: Ejemplo de gráfica de precisión y pérdidas

En resumen, el procedimiento de simulación involucró la obtención de implementaciones de modelos de redes de cápsulas, la personalización de los modelos y a través de *forks* en GitHub, el entrenamiento de los modelos con técnicas de mejora y monitoreo del rendimiento, y finalmente la evaluación de los modelos y el análisis de los resultados obtenidos.

4

Resultados

En esta sección, se presentan los resultados obtenidos de las simulaciones realizadas utilizando varios modelos de redes de cápsulas, incluyendo MoCapsNet, ResCapsNet, CapsNet, DeepCaps y MatCaps. Se compararán los modelos en términos de su rendimiento, número de parámetros, consumo de memoria y tiempo de ejecución. Además, se analizarán los efectos de diferentes configuraciones, como el número de bloques residuales, el optimizador utilizado y el enrutamiento implementado.

Nota: En la mayoría de simulaciones, a no ser que se indique lo contrario, se ha empleado un learning rate de 0.001, un tamaño de lote de 128, optimizador Adam, enrutamiento RBA, imágenes 50x50 para las bandejas de carne y 32x32 para CIFAR-10.

4.1 | MoCapsNet y ResCapsNet

4.1.1 | Número de bloques residuales

Se llevaron a cabo simulaciones utilizando MoCapsNet y ResCapsNet con diferentes números de bloques residuales para determinar su impacto en el rendimiento de los modelos. Se realizaron pruebas con 1, 5 y 8 bloques residuales en cada modelo. A continuación, se presentan los resultados obtenidos.

MoCapsNet:

Bloques residuales	Precisión(%)	Parámetros	Memoria (MB)	Segundos/época
1	71,93 ± 0,41	13494016	7107	120
5	72,9 ± 0,17	14018304	7327	126
8	**73,66 ± 0,67**	14411520	7335	142

Cuadro 4.1: Resultados de MoCapsNet (CIFAR-10)

Bloques residuales	Precisión(%)	Parámetros	Memoria (MB)	Segundos/época
1	73,92 ± 1,33	32974924	7853	101
5	**77,19 ± 2,53**	33499212	7861	105.23
8	71,55 ± 0,84	33892428	7871	107.81

Cuadro 4.2: Resultados de MoCapsNet (Bandejas de carne)

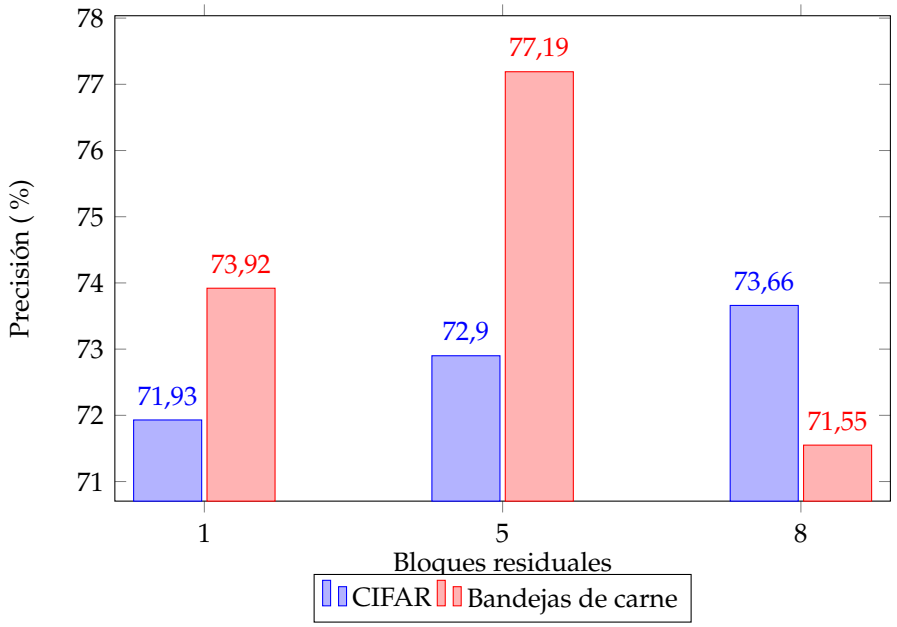


Figura 4.1: Rendimiento de MoCapsNet en función de los bloques residuales

ResCapsNet:

Bloques residuales	Precisión(%)	Parámetros	Memoria (MB)	Segundos/época
1	$71{,}96 \pm 0{,}37$	13494016	7037	120
5	$71{,}55 \pm 0{,}21$	14018304	7903	123
8	$\mathbf{72{,}09 \pm 0{,}24}$	14411520	8575	128

Cuadro 4.3: Resultados de ResCapsNet (CIFAR-10)

Bloques residuales	Precisión(%)	Parámetros	Memoria (MB)	Segundos/época
1	$\mathbf{71{,}85 \pm 1{,}63}$	32974924	7829	98.2
5	$67{,}42 \pm 2{,}5$	33499212	9587	101.4
8	$71{,}28 \pm 0{,}53$	33892428	9607	103.21

Cuadro 4.4: Resultados de ResCapsNet (Bandejas de carne)

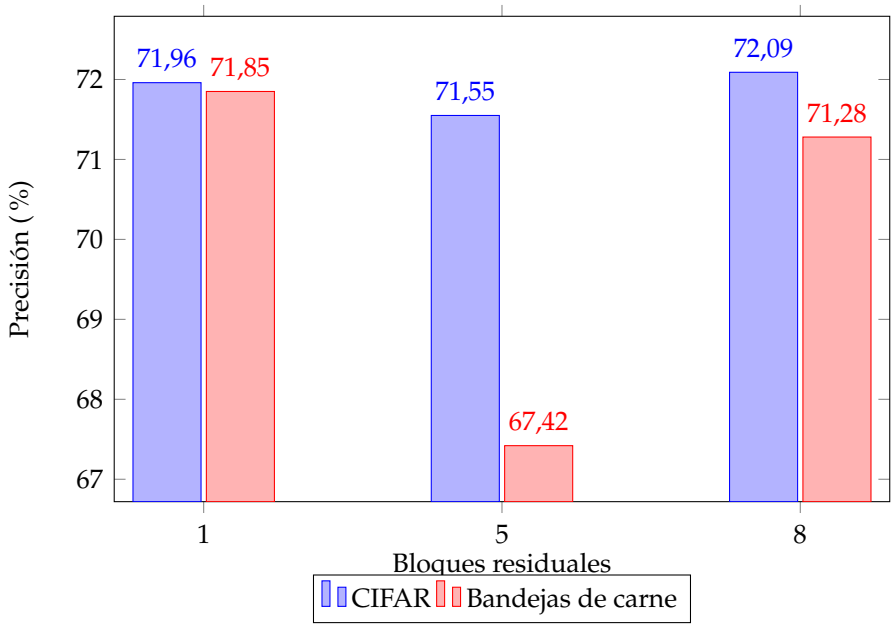


Figura 4.2: Rendimiento de ResCapsNet en función de los bloques residuales

4.1.2 | Optimizador

Se realizaron simulaciones adicionales con diferentes optimizadores para evaluar su impacto en el rendimiento de los modelos. Se compararon los resultados obtenidos utilizando el optimizador Adam y el optimizador Ranger21 (Wright and Demeure (2021)) utilizando el modelo MoCapsNet de 5 bloques residuales. A continuación, se presentan los resultados obtenidos: **Nota: Debido a problemas con el optimizador Ranger21 no se pudieron realizar las simulaciones con CIFAR-10.**

Optimizador	Precisión(%)	Parámetros	Memoria (MB)	Segundos/época
Ranger21	$74{,}27 \pm 0{,}5$	33499212	9605	88.92
Adam[1]	$\mathbf{78{,}9 \pm 1{,}72}$	41877632	10031	120.26

Cuadro 4.5: Resultados de los optimizadores Ranger21 y Adam (Bandejas de carne)

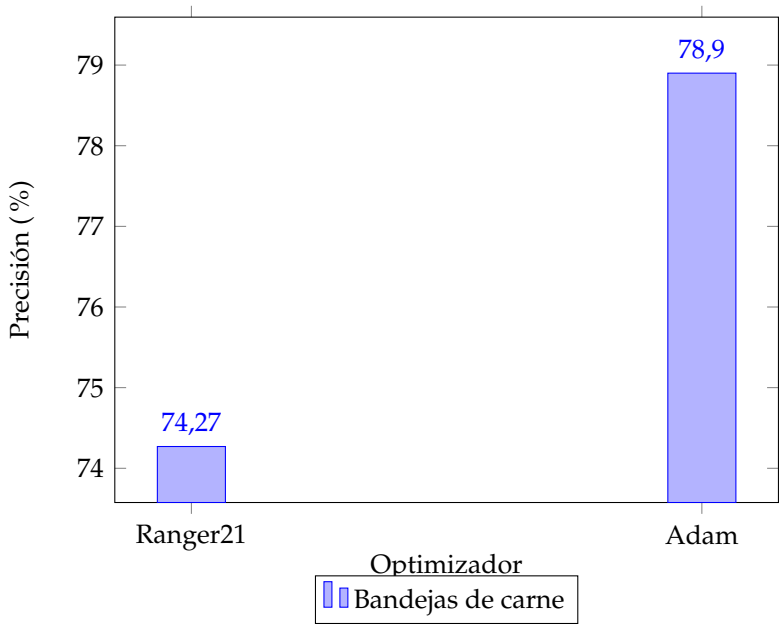


Figura 4.3: Rendimiento de los optimizadores Ranger21 y Adam

[1]Para esta simulación se utilizaron imágenes 80x40, respetando la ratio original, y un tamaño de lote de 32 para que cupiese en la memoria de la GPU.

4.1.3 | Enrutamiento

Además, se realizaron simulaciones comparativas con diferentes algoritmos de enrutamiento para evaluar su efecto en el rendimiento de los modelos. Se compararon los resultados obtenidos utilizando el enrutamiento Routing by Agreement (RBA) y el enrutamiento Scaled Distance Agreement (SDA).Para estas simulaciones se ha utilizado el modelo MoCapsNet de 5 bloques residuales. A continuación, se presentan los resultados obtenidos.

Enrutamiento	Precisión(%)	Parámetros	Memoria (MB)	Segundos/época
SDA	$61{,}17 \pm 1{,}4$	14021280	OOM[2]	274.42
RBA	$\mathbf{72{,}9 \pm 0{,}17}$	14018304	7327	126

Cuadro 4.6: CIFAR-10

Enrutamiento	Precisión(%)	Parámetros	Memoria (MB)	Segundos/época
SDA[3]	$69{,}35 \pm 0{,}47$	14218144	4887	65.94
RBA	$\mathbf{77{,}19 \pm 2{,}53}$	33499212	7861	105.23

Cuadro 4.7: Bandejas de carne

[2]Al final de la simulación cuando se imprime por pantalla la cantidad de memoria utilizada salta el error Out of Memory (OOM). Por lo que se puede intuir que se han utilizado los 16GB de memoria de la GPU. Reduciendo el tamaño del lote a 50 la memoria máxima utilizada es de 6955MB.

[3]En esta simulación se redujo el tamaño de las imágenes a 32x32 para no tener problemas de memoria.

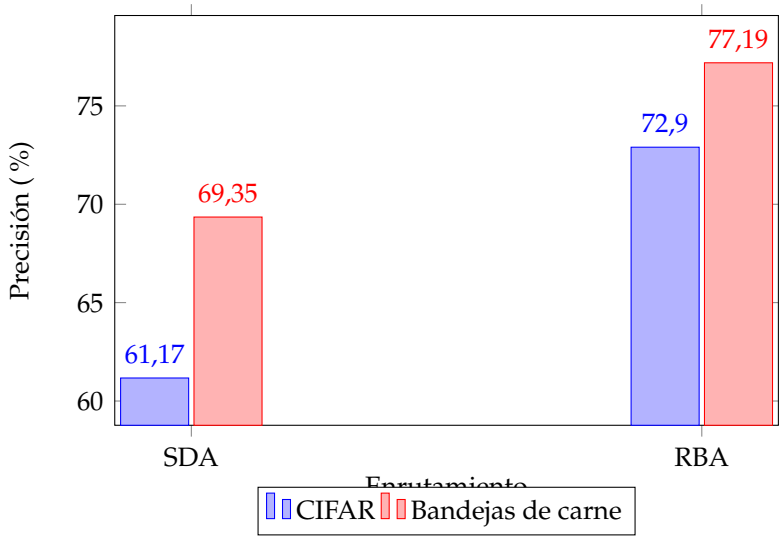


Figura 4.4: Rendimiento del enrutamiento

4.1.4 | MatCaps

Se realizaron simulaciones con diferentes configuraciones de MatCaps para evaluar como afectan en su rendimiento. Las configuraciones utilizadas incluyen:

- A, B, C, D = 64, 8, 16, 16
- A, B, C, D = 32, 4, 4, 4
- A, B, C, D = 32, 32, 32, 32

A continuación, se presentan los resultados obtenidos:

Configuración	Precisión(%)	Parámetros	Memoria (MB)	Segundos/época
A, B, C, D = 64, 8, 16, 16	**66,03 ± 0,95**	70044	11567	554
A, B, C, D = 32, 4, 4, 4	62,93 ± 0,96	40996	7109	310
A, B, C, D = 32, 32, 32, 32	-	-	-	-

Cuadro 4.8: Resultados de MatCaps (CIFAR-10)

Configuración	Precisión(%)	Parámetros	Memoria (MB)	Segundos/época
A, B, C, D = 64, 8, 16, 16	**59,92 ± 3,8**	43452	4639	176.24
A, B, C, D = 32, 4, 4, 4	54,46 ± 3,48	7656	3179	101.55
A, B, C, D = 32, 32, 32, 32	57,41 ± 5,25	164148	6577	244.52

Cuadro 4.9: Resultados de MatCaps (Bandejas de carne)[4]

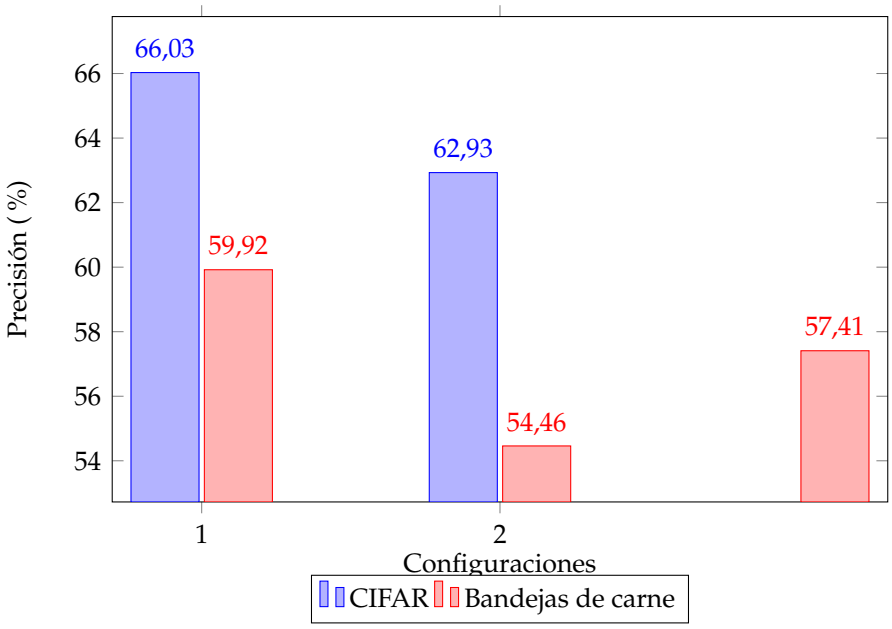


Figura 4.5: Rendimiento según la configuración de MatCaps

[4]En este caso se utilizó un tamaño de lote de 20, de ahí las cantidades de memoria más bajas.

4.2 | Comparación general

Finalmente, se presentan los resultados y comparaciones generales de todos los modelos y configuraciones evaluados en los datasets CIFAR-10 y bandejas de carne. Se muestra una tabla comparativa con las métricas clave, como la precisión, el número de parámetros, el consumo máximo de memoria y el tiempo de ejecución. Además, se incluyen gráficas que ilustran las comparaciones de rendimiento entre los modelos y las configuraciones.

Nota: Se ha escogido el mejor modelo y configuración de los apartados anteriores para elaborar las tablas siguientes.

Modelo	Precisión(%)	Parámetros	Memoria (MB)	Segundos/época
CapsNet	$74{,}4 \pm 0{,}12$	11749120	5643	73.63
DeepCaps	$\mathbf{78{,}95 \pm 0{,}72}$	7660241	3641	59.94
ResCapsNet_8	$72{,}09 \pm 0{,}24$	14411520	8575	128
MoCapsNet_8	$73{,}66 \pm 0{,}67$	14411520	7335	142
MatCaps	$66{,}03 \pm 0{,}95$	70044	11567	554

Cuadro 4.10: Comparación general (CIFAR-10)

Modelo	Precisión(%)	Parámetros	Memoria (MB)	Segundos/época
CapsNet	$63{,}99 \pm 2{,}18$	16074496	9737	89
DeepCaps[5]	$52{,}65 \pm 6{,}63$	8971217	5449	35.16
ResCapsNet_1	$71{,}85 \pm 1{,}63$	32974924	7829	98.2
MoCapsNet_5	$\mathbf{77{,}19 \pm 2{,}53}$	33499212	7861	105.23
MatCaps	$59{,}92 \pm 3{,}8$	43452	4639	176.24

Cuadro 4.11: Comparación general (Bandejas de carne)

[5]En este caso se utilizaron imágenes 28x28 debido a limitaciones del modelo.

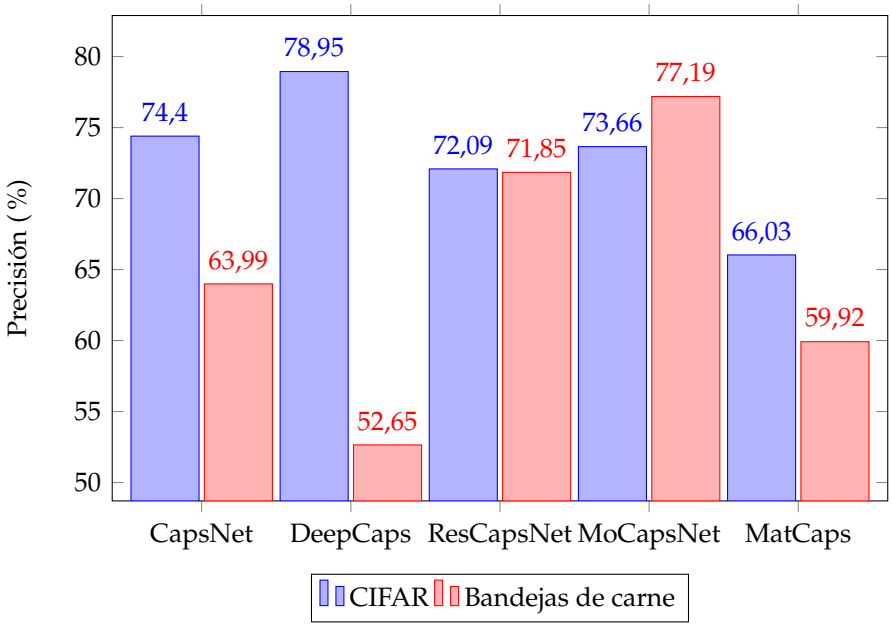


Figura 4.6: Comparación general

4.3 | Resumen

En este trabajo, se realizaron simulaciones utilizando varios modelos de redes de cápsulas, como MoCapsNet, ResCapsNet, CapsNet, DeepCaps y MatCaps, con el objetivo de encontrar modelos que logren una mayor precisión, menor consumo de memoria y menor tiempo de ejecución en el contexto de la clasificación de imágenes.

En el caso de MoCapsNet y ResCapsNet, se llevaron a cabo simulaciones con diferentes números de bloques residuales para evaluar su impacto en el rendimiento de los modelos. Los resultados mostraron que tanto MoCapsNet como ResCapsNet alcanzan su mejor rendimiento con 8 bloques residuales en el dataset CIFAR-10. Sin embargo, en el caso de las bandejas de carne, MoCapsNet obtuvo una mayor precisión con 5 bloques residuales.

Además, se compararon los resultados utilizando los optimizadores Adam y Ranger21. En el dataset de bandejas de carne, Adam mostró un rendimiento superior en comparación con Ranger21 para el modelo probado el cual fue MoCapsNet de 5 bloques residuales.

En relación al enrutamiento, se compararon los algoritmos RBA y SDA. En el dataset CIFAR-10, el enrutamiento RBA demostró un mejor rendimiento en el modelo probado, MoCapsNet de 5 bloques residuales. En las simulaciones con las bandejas de carne, nuevamente se observó que el enrutamiento RBA superó al enrutamiento SDA en términos de precisión.

En cuanto a MatCaps, se realizaron simulaciones con diferentes configuraciones. Tanto para el dataset CIFAR-10 como el de bandejas de carne, la configuración A, B, C, D = 64, 8, 16, 16 alcanzó el mejor rendimiento en términos de precisión.

En la comparación general de todos los modelos y configuraciones evaluadas, se observó que DeepCaps obtuvo la mayor precisión en el dataset CIFAR-10, seguido por MoCapsNet. Por otro lado, MoCapsNet lideró en las simulaciones con las bandejas de carne, seguido por ResCapsNet. MatCaps presentó un rendimiento inferior en comparación con los otros modelos en ambos conjuntos de datos.

5

Conclusiones

En este capítulo, se presentan las conclusiones del estudio realizado sobre varios modelos de redes de cápsulas, así como los objetivos alcanzados, las críticas y limitaciones identificadas, las posibles direcciones para futuras investigaciones y algunas observaciones finales.

5.1 | Objetivos cumplidos

Durante el desarrollo de este trabajo, se lograron alcanzar los siguientes objetivos establecidos:

1. Se consiguieron encontrar modelos alternativos a las redes de cápsulas que prometen mejorar en rendimiento a los modelos base de CapsNet y MatCaps.

2. Se simularon dichos modelos de redes profundas de cápsulas manteniendo el consumo de memoria menor o igual al de los modelos base y obteniendo una mayor precisión (con algunos de ellos).

3. Se analizaron los efectos de diferentes configuraciones, como el número de bloques residuales, el optimizador utilizado y el enrutamiento implementado. De esta manera se observó como afectan estos elementos al rendimiento de los modelos.

4. Se presentaron tablas comparativas y gráficas que ilustran las diferencias de rendimiento entre los modelos y configuraciones evaluadas, determinando de esta manera cuál es el mejor modelo para nuestro caso concreto.

5.2 | Críticas y limitaciones

A pesar de los resultados obtenidos, es importante tener en cuenta algunas críticas y limitaciones del estudio:

- La comparación de los modelos se realizó en los conjuntos de datos CIFAR-10 y bandejas de carne, lo cual limita la generalización de los resultados a otros conjuntos de datos.

- Se utilizaron configuraciones y parámetros específicos en las simulaciones, lo que puede haber influido en los resultados. Otras configuraciones y ajustes podrían haber llevado a resultados diferentes.

- El tiempo de ejecución es superior al modelo base CapsNet, puede ser debido a que en las arquitecturas reversibles activaciones se vuelven a calcular en el *backward pass* para ahorrar memoria. Aún así sigue siendo muy inferior al de MatCaps y ofrecen mejor precisión.

- El rendimiento de los modelos de redes de cápsulas sigue siendo inferior a otros modelos del estado del arte como los Visual Transformers (ViTs).

- Las simulaciones se llevaron a cabo en un entorno de desarrollo específico, utilizando recursos y herramientas disponibles en ese entorno y una GPU concreta, lo cual afecta aspectos como el tiempo de ejecución y el tamaño de las imágenes que se pueden tratar. Cuanto más potente sea la GPU utilizada menores serán los tiempos de ejecución y cuanta más memoria disponga mayor será la red que se podrá entrenar y de mayor dimensión serán las imágenes que se puedan utilizar.

5.3 | Líneas de investigación futuras

Basándonos en los resultados y las limitaciones identificadas, se proponen las siguientes direcciones para futuras investigaciones:

- Realizar comparaciones y evaluaciones de los modelos en una variedad más amplia de conjuntos de datos para obtener una visión más completa de su rendimiento y generalización.
- Explorar y ajustar diferentes configuraciones y parámetros en los modelos de redes de cápsulas para optimizar aún más su rendimiento en diferentes tareas de clasificación de imágenes.
- Investigar nuevas variantes de los modelos de redes de cápsulas que incorporen mejoras y adaptaciones específicas para abordar las limitaciones identificadas en este estudio.
- Examinar la viabilidad de implementar los modelos en entornos de producción y evaluar su rendimiento en escenarios del mundo real.
- Investigar el uso de técnicas de transferencia de aprendizaje (*transfer learning*) para mejorar el rendimiento de los modelos en conjuntos de datos más pequeños o especializados.

5.4 | Observaciones finales

En conclusión, este estudio proporcionó una evaluación comparativa de varios modelos de redes de cápsulas en los conjuntos de datos CIFAR-10 y bandejas de carne. Los resultados obtenidos indican que MoCapsNet y ResCapsNet son modelos prometedores en términos de precisión y rendimiento, especialmente en el contexto de las bandejas de carne. Sin embargo, se debe tener en cuenta que los resultados pueden variar según el conjunto de datos y las configuraciones específicas utilizadas.

El trabajo realizado ha contribuido a ampliar el conocimiento sobre las redes de cápsulas y su aplicabilidad en la clasificación de imágenes. Se han identificado áreas de mejora y posibles direcciones para futuras investigaciones. Se espera que estos hallazgos sirvan como punto de partida para investigaciones adicionales y el desarrollo de modelos de redes de cápsulas más avanzados y efectivos.

Bibliografía

A. Deliège, A. Cioppa, and M. Van Droogenbroeck. Hitnet: a neural network with capsules embedded in a hit-or-miss layer, extended with hybrid data augmentation and ghost capsules. *CoRR*, 2018.

Aidan N. Gomez, Mengye Ren, Raquen Urtasun, and Roger B. Grosse. The reversible residual network: Backpropagation without storing activations. 2017.

Josef Gugglberger, David Peer, and Antonio Rodríguez-Sánchez. Momentum capsule networks. *Transactions on Machine Learning Research*, 2022.

Ethan Harris, Matthew Painter, and Jonathon Hare. Torchbearer: A model fitting library for pytorch. *arXiv preprint arXiv:1809.03363*, 2018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *LSVRC*, 2015.

G.E. Hinton, A. Krizhevsky, and J. Sun. Transforming auto-encoders. *ICANN*, 2011.

Geoffrey Hinton, S. Sabour, and Nicholas Frost. Matrix capsules with em routing. *International conference on learning representations*, 2018.

G. Huang, Z.Liu, L. Van Der Maaten, and K. Q. Weinberger. Denseley connected convolutional networks. *CVPR*, 2017.

A. Jaiswal, W. AbdAlmageed, Y. Wu, and P. Natarajan. Capsulegan: Generative adversarial capsule network. *ECCV*, 2018.

Xiaofen Jia, Baiting Zhao, Yongcun Guo, and Yourui Huang. Res-capsnet: Residual capsule network for data classification. *Neural Processsing Letters*, 2022.

R. LaLonde and U. Bagci. Capsules for object segmentation. *arXiv preprint*, 2018.

Jathushan Rajasegaran, Vinoj, Sandaru Jayasekara, Hirunima Jayasekara, Suranga Seneviratne, and Ranga Rodrigo. Deepcaps: Going deeper with capsule networks. *CVPR*, 2019.

Sara Sabour, Nicholas Frosst, and Geoffrey Hinton. Dynamic routing between capsules. *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017.

Michael E Sander, Pierre Ablin, Mathieu Blondel, and Gabriel Peyré. Momentum residual neural networks. *arXiv preprint arXiv:2102.07870*, 2021.

D. Wang and Q. Liu. An optimization view on dynamic routing between capsules. 2018.

Less Wright and Nestor Demeure. Ranger21: a synergistic deep learning optimizer. *arXiv preprint arXiv:2106.13731*, 2021.