Clasificación de Imágenes Mediante Transfer Learning de Redes Neuronales.

Universitat Oberta de Catalunya

Eleana Liscar Rey Quijada

MU Ingeniería Informática
Área de Inteligencia Artificial

Nombre Tutor/a de TF
Josep Mª Carmona Leyva
Profesor/a responsable de la asignatura
Joan M. Nuñez Do Rio
Fecha Entrega
14/01/2025

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Clasificación de Imágenes Mediante Transfer Learning de Redes Neuronales.*
Nombre del autor:	*Eleana Liscar Rey Quijada*
Nombre del consultor/a:	*Josep Mª Carmona Leyva*
Nombre del PRA:	*Joan M. Nuñez Do Rio*
Fecha de entrega (mm/aaaa):	*14/01/2025*
Titulación o programa:	*MU Ingeniería Informática*
Área del Trabajo Final:	*Área de Inteligencia Artificial*
Idioma del trabajo:	*Castellano*
Palabras clave	*Clasificación de imágenes, Machine Learning, Transfer Learning, Redes Neuronales*

Resumen del Trabajo

El objetivo principal de este trabajo fue aplicar redes neuronales para resolver problemas de clasificación de imágenes mediante el modelo MobileViT, optimizado para dispositivos con recursos limitados. Los experimentos se llevaron a cabo utilizando datasets públicos y un conjunto de datos balanceado creado específicamente con clases text y no_text.

A pesar de las estrategias implementadas, como el balanceo de datos, técnicas de *data augmentation* y la normalización basada en ImageNet, los resultados mostraron limitaciones significativas. El mejor desempeño del modelo (47.11% de precisión en validación) destacó problemas de generalización debido al tamaño reducido del dataset y la falta de variabilidad en los datos. Aunque se logró estabilidad en la pérdida de entrenamiento, las predicciones incorrectas, subrayan la necesidad de ampliar el conjunto de datos y explorar arquitecturas más avanzadas, como MobileViT-V2.

En un intento final de optimización, el modelo alcanzó una precisión del 100% en validación, pero mostró signos claros de sobreajuste, indicando que los datos eran insuficientes para un aprendizaje efectivo. Estos hallazgos resaltan la importancia de mejorar la calidad y cantidad de los datos de entrenamiento para futuros experimentos y de considerar modelos con mayor capacidad de generalización.

Abstract

The main objective of this work was to apply neural networks to solve image

classification problems using the MobileViT model, optimized for resource-constrained devices. The experiments were conducted using public datasets and a balanced dataset specifically created with text and no_text classes.

Despite the implemented strategies, such as data balancing, data augmentation techniques, and normalization based on ImageNet, the results revealed significant limitations. The model's best performance (47.11% validation accuracy) highlighted generalization issues due to the dataset's small size and lack of variability. While training loss stability was achieved, incorrect predictions underscored the need to expand the dataset and explore more advanced architectures, such as MobileViT-V2.

In a final optimization attempt, the model achieved 100% validation accuracy but showed clear signs of overfitting, indicating that the data was insufficient for effective learning. These findings emphasize the importance of improving the quality and quantity of training data for future experiments and considering models with greater generalization capabilities.

Dedicatoria

Quiero dedicar este trabajo a las personas que han sido fundamentales en mi camino hacia la finalización de este Máster.

A mi madre y a mi padre, quienes han sido los pilares fundamentales de lo que soy como persona. Su apoyo y enseñanzas me han guiado en cada paso de mi vida.

A mi abuela, siempre tengo y tendré presente y cuyo amor y sabiduría siempre han sido una fuente de inspiración en mi vida.

A mi abuelo Alfredo, quien me animó a iniciar mis estudios en Ingeniería, gracias por tu apoyo y confianza en mis capacidades.

A mi compañero y esposo, Javier, por estar a mi lado en cada paso de este proceso, brindándome el apoyo incondicional que necesitaba para completar este desafío.

A mi amigo Freddy, quien me motivó a dar el primer paso hacia este Máster, tu aliento ha sido invaluable.

Y, por supuesto, a mi hijo Ángel, quien con solo 2 años me ha enseñado el verdadero significado de la perseverancia. A pesar de las dificultades de compaginar la crianza y el estudio, tu sonrisa y energía me han dado la fuerza necesaria para seguir adelante.

Gracias a todos por ser mi inspiración y mi apoyo en este viaje.

Agradecimientos

Quiero expresar mi más sincero agradecimiento a mi tutor, Josep Mª Carmona Leyva, por su apoyo y orientación a lo largo de este trabajo. Su experiencia y conocimientos en el área de Inteligencia Artificial han sido fundamentales para el desarrollo de este proyecto. Agradezco su paciencia y dedicación, así como las valiosas sugerencias que me han permitido mejorar y enriquecer mi investigación. Sin su guía, este trabajo no habría sido posible.

Índice de contenido

Índice de Figuras

1. Introducción

1.1. Contexto y justificación del Trabajo

Con el aumento exponencial de la información a la que se tiene acceso en la actualidad y la gestión de contenido digital, surge la necesidad de sistemas que gestionen eficientemente estos archivos. En este trabajo, nos enfocamos en la gestión de imágenes, que a menudo se encuentran mezcladas en la galería de un ordenador personal, teléfono móvil o tablet. La capacidad de acceder a estas imágenes de forma eficiente es crucial para su consulta o eliminación en lotes, especialmente en un contexto donde la acumulación de contenido irrelevante y la desorganización de contenido importante puede dificultar la experiencia del usuario.

Según un estudio realizado en 2021 (Garg et al., 2021), la búsqueda y gestión de documentos en smartphones es una tarea ardua, ya que la mayoría de los métodos de búsqueda dependen de metadatos o del texto contenido en los documentos, lo que dificulta la organización eficiente de archivos multimèdia y las opciones de clasificación automática disponibles son limitadas y no abordan adecuadamente la calidad y relevancia del contenido multimedia.

La aportación de este trabajo busca, por lo tanto, abordar esta necesidad mediante la optimización y adaptación de un modelo de machine learning que se pueda integrar en un sistema de gestión de imágenes. El objetivo es que este modelo clasifique las imágenes en categorías temáticas, facilitando así el acceso al contenido solicitado al momento y permitiendo al usuario eliminar de manera rápida y segura archivos innecesarios o permitiendo acceder de manera rápida al archivo que se quiere consultar. La implementación de un sistema de este tipo no solo mejorará la experiencia del usuario al interactuar con su contenido digital, sino que también optimizará el rendimiento de los dispositivos móviles mediante una gestión más eficiente del almacenamiento.

La relevancia de este enfoque se ve respaldada por la creciente cantidad de datos generados y almacenados en dispositivos móviles, lo que hace que la organización y clasificación de imágenes sea una tarea cada vez más necesaria. Al utilizar técnicas avanzadas de transfer learning, como el modelo MobileViT (Mehta & Rastegari, 2022a), se espera lograr un rendimiento eficiente en la clasificación de imágenes, adaptándose a las limitaciones de recursos de los dispositivos de uso personal como dispositivos móviles y ordenadores personales, mejorando la capacidad de los usuarios para gestionar su contenido digital de manera efectiva.

1.2. Objetivos del Trabajo

Objetivo general:

Aplicar un mecanismo basado en redes neuronales sobre un conjunto de imágenes reales para resolver un problema de clasificación de imágenes.

Objetivos específicos:

- Adaptar modelos preentrenados a tareas específicas: MobileVit
- Comprobar si este modelo basado en MobileViT mejora otros resultados anteriores de clasificación en conjuntos de datos públicos, utilizando métricas como precisión y pérdida.
- Analizar los tiempos requeridos para entrenar este modelo y evaluar su viabilidad en dispositivos con recursos limitados.
- Evaluar si se necesitan menos datos de aprendizaje para alcanzar un rendimiento competitivo en comparación con otras arquitecturas, para reducir la dependencia de datos extensivos.
- Secundariamente, y si los recursos lo permiten, especificar los requisitos de hardware mínimos y probar el sistema en entornos reales a fin de conocer el impacto que tiene en la gestión de imágenes y la mejora de la experiencia del usuario.

1.3. Impacto en sostenibilidad, ético-social y de diversidad

El desarrollo de un sistema de clasificación y organización de contenido digital en dispositivos de uso doméstico como móviles, tablets y ordenadores personales, tiene impactos significativos en las dimensiones de sostenibilidad, comportamiento ético y responsabilidad social, y diversidad, alineados con la competencia de "Compromiso ético y global" (CCEG) y algunos Objetivos de Desarrollo Sostenible (ODS) de la ONU:

1. **Impacto en Sostenibilidad**

   Positivos: Al optimizar el almacenamiento en dispositivos móviles mediante la eliminación de contenido irrelevante (según los criterios de búsqueda personal del usuario), este sistema contribuye al ODS 12 (Consumo y Producción Responsables) y al ODS 11 (Ciudades y Comunidades Sostenibles), ya que prolonga la vida útil de los dispositivos al reducir la necesidad de almacenamiento adicional y mejora el rendimiento, disminuyendo el consumo energético del dispositivo. Estos beneficios ayudan a reducir el impacto ambiental asociado con la fabricación y el desecho de dispositivos tecnológicos.

Negativos: Es posible que la implementación y uso de modelos de machine learning incremente temporalmente el uso de energía en el dispositivo. Sin embargo, al optimizar el procesamiento y almacenamiento en el largo plazo, este impacto negativo puede ser minimizado.

2. **Impacto Ético-Social**

   Positivos: El sistema ayuda a mejorar la experiencia del usuario al facilitar una gestión más eficiente de sus datos, contribuyendo al ODS 16 (Paz, Justicia e Instituciones Sólidas) al eliminar y clasificar de forma más eficiente archivos relacionados a documentos o información sensible, el sistema promueve una gestión ética y segura de los datos multimedia personales.

   Negativos: La aplicación debe garantizar que el análisis de archivos se realice de manera local en el dispositivo, sin transmitir datos sensibles o personales a servidores externos, protegiendo así la privacidad del usuario. De no abordarse adecuadamente, podría haber preocupaciones sobre la privacidad y la recopilación de datos personales.

3. **Impacto en Diversidad**

   Positivos: El sistema está diseñado para ser accesible e intuitivo, lo que permite que usuarios de diferentes niveles tecnológicos y de distintas culturas puedan beneficiarse de su funcionalidad. Además, el desarrollo sigue principios de diseño inclusivo, garantizando que su uso no dependa del género, etnia, o características socioeconómicas del usuario, alineándose con el ODS 10 (Reducción de Desigualdades).

   Consideraciones de Mejora: Se ha considerado evitar cualquier tipo de sesgo en la clasificación de archivos multimedia, asegurando que la organización y categorización de contenido no favorezcan estereotipos o sesgos predefinidos.

1.4. Enfoque y método seguido

Para el desarrollo del proyecto utilizaremos el lenguaje de programación Python (versión 3), debido a su versatilidad y amplia disponibilidad de bibliotecas especializadas para aprendizaje automático y procesamiento de datos multimedia.

Entre las herramientas destacadas, emplearemos PyTorch (*PyTorch documentation — PyTorch 2.5 documentation*, s. f.) y torchvision (*torchvision — Torchvision 0.20 documentation*, s. f.), que facilitan la construcción, entrenamiento y evaluación de modelos multimodales y de aprendizaje profundo, como redes neuronales convolucionales (CNNs) y transformers.

PyTorch es ampliamente reconocido por su flexibilidad y control, permitiendo la implementación eficiente de modelos personalizados y técnicas avanzadas como transfer learning.

Entorno de desarrollo y experimentación:

El entorno donde ejecutaremos el código Python, será la plataforma en la nube "Google Colab" (*Google Colab*, s. f.).

Ventajas: La elección de esta herramienta se ha realizado tomando en cuenta los siguientes puntos clave:

- Evitar problemas relacionados con la instalación de dependencias y versiones de bibliotecas, al ser un entorno preconfigurado, minimiza los errores técnicos relacionados con versiones de bibliotecas y configuraciones complejas.

- Economía: Permite realizar entrenamientos y pruebas de modelos de manera eficiente, aprovechando la aceleración del hardware especializado (Acceso a GPUs y TPUs) a un precio accesible o incluso gratuito, en comparación con tener que comprar el hardware necesario para realizar los entrenamientos (ver imagen - *Pricing* Google Colab).

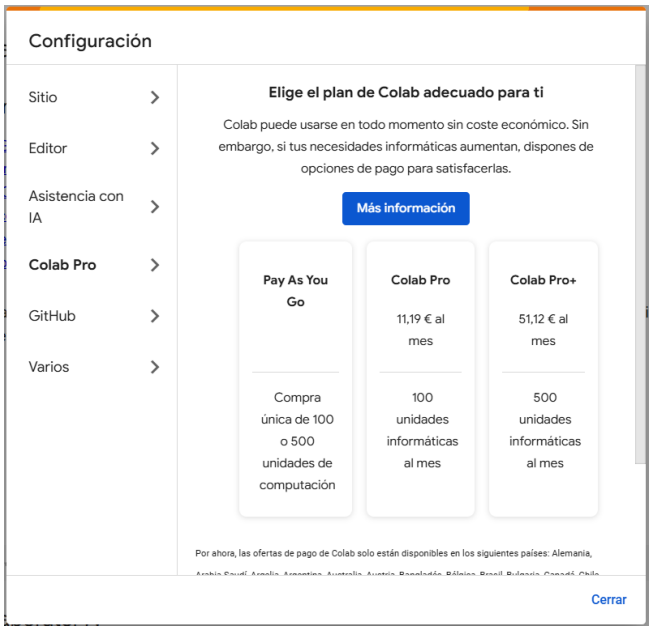


Figura 1 - Pricing Google Colab 18/11/2024 ES

- Colaboración en tiempo real: Facilita compartir y trabajar en conjunto con el código en tiempo real.

Limitaciones:

Por otro lado, también es importante mencionar algunas limitaciones de este entorno:

- Conexiones intermitentes: Colab requiere la confirmación periódica de la actividad del usuario mediante mecanismos como CAPTCHA, lo que puede interrumpir ejecuciones prolongadas.
- Desconexiones automáticas: A pesar de la supervisión constante, el entorno de ejecución tiene un tiempo limitado de actividad que puede provocar desconexiones inesperadas.

Enfoque metodológico

El proyecto utiliza estrategias de ***transfer learning*** con modelos preentrenados, priorizando la adaptación de arquitecturas existentes a las necesidades del problema. En particular, se ha seleccionado el modelo MobileViT, que destaca porque está diseñado para tener eficiencia en dispositivos con recursos limitados, es ideal para aplicaciones móviles.

Transfer Learning

La metodología aplicada en este trabajo se basa en el uso de *transfer learning* con modelo preentrenado, específicamente se han hecho experimentos con el modelo MobileViT, del repositorio: https://github.com/quic/ai-hub-models.git

Pasos clave de la metodología

En el capítulo 3. “Materiales y metodología”, veremos en detalle el desarrollo de estos pasos, y en este apartado podemos destacar los siguientes:

1. **Carga y preprocesamiento de datos:**
    - Se creo a partir de los data set públicos MSRA-TD500 (Kwon & Lee, 2012a) y una selección aleatoria de Flickr-8k (Hodosh et al., 2013); en Flick se seleccionó un numero de muestras similar al que contiene MSRA-TD500, para mantener un balanceo de las clases.
    - El conjunto de datos adaptado se trata un conjunto de datos dividido en dos clases: text y no_text.
    - Las imágenes fueron redimensionadas a 224x224 píxeles, esto asegura que la entrada sea compatible con las dimensiones del modelo
    - Las imágenes fueron normalizadas para facilitar el procesamiento por los modelos.

2. **División del conjunto de datos:**
    - Se separaron los datos en un 80% para entrenamiento y un 20% para validación, asegurando una distribución representativa de ambas clases.
3. **Configuración de los modelos:**
    - Se cargaron los modelos preentrenados.
    - Se ajustaron las últimas capas de los modelos para adaptarlos a la clasificación binaria.

- Se configuró el optimizador Adam con una tasa de aprendizaje baja de valor 0.0001, adecuada para el reentrenamiento, ya que reduce el riesgo de destruir la información adquirida durante el preentrenamiento. Por el contrario, una tasa de aprendizaje alta (como 0.001 o mayor) podría sobrescribir los pesos preentrenados rápidamente, causando pérdida de generalización.

4. **Entrenamiento y evaluación:**
   - Se realizaron entrenamientos durante 10 épocas y se monitoreó la pérdida y la precisión en el conjunto de validación.
   - Se utilizó una matriz de confusión para analizar los errores de clasificación y mejorar la interpretación de los resultados.
5. **Análisis de resultados:**
   - Se logró una precisión promedio del 47,11% en el conjunto de validación utilizando MobileViT, destacando que el tamaño de nuestro conjunto de datos no es suficiente para proporcionar un avance positivo.

Ventajas de utilizar transfer learning:

1. **Menor necesidad de datos**: *Transfer learning* permite utilizar modelos preentrenados que han sido entrenados en grandes conjuntos de datos, lo que significa que se necesita menos cantidad de datos específicos para la tarea en cuestión. Esto es especialmente útil en situaciones donde los datos son escasos o difíciles de obtener.

2. **Ahorro de tiempo y recursos**: Entrenar un modelo desde cero puede ser muy costoso en términos de tiempo y recursos computacionales. *Transfer learning* reduce significativamente el tiempo de entrenamiento, ya que se parte de un modelo que ya ha aprendido características útiles.

3. **Mejor rendimiento inicial**: Los modelos preentrenados suelen tener un rendimiento superior desde el inicio en comparación con modelos entrenados desde cero, especialmente en tareas donde los datos son limitados. Esto se debe a que han aprendido representaciones generales que pueden ser aplicadas a nuevas tareas.

4. **Facilidad de ajuste**: Ajustar un modelo preentrenado a una nueva tarea puede ser más sencillo, ya que solo se necesita modificar las últimas capas del modelo y realizar un *fine-tuning*, en lugar de entrenar toda la red desde cero.

Desventajas de utilizar *transfer learning*:

1. **Dependencia del modelo preentrenado**: La efectividad del *transfer learning* depende en gran medida de la calidad y relevancia del modelo preentrenado. Si el modelo no es adecuado para la nueva tarea, el rendimiento puede ser subóptimo.

2. **Posibilidad de sobreajuste**: Si el conjunto de datos específico es muy pequeño, existe el riesgo de que el modelo se sobreajuste a los datos de entrenamiento, especialmente si se realizan demasiadas modificaciones en las capas finales.

3. **Limitaciones en la personalización**: Aunque se pueden ajustar las capas finales, puede ser difícil personalizar completamente el modelo para tareas muy específicas, ya que las características aprendidas en el preentrenamiento pueden no ser completamente relevantes.

4. **Requerimientos computacionales**: Aunque *transfer learning* puede ser más eficiente que entrenar desde cero, aún puede requerir recursos computacionales significativos, especialmente si se utilizan modelos grandes y complejos.

1.5. Planificación del Trabajo

Las fechas de entrega de las actividades PAC están detalladas en el plan docente de la asignatura y se resumen en la Figura 2.

Nombre de la actividad	Fecha de inicio	Fecha de entrega
Entrega de la actividad P1	25/09/2024	14/10/2024
Entrega de la actividad P2	15/10/2024	18/11/2024
Entrega de la actividad P3	19/11/2024	23/12/2024
Entrega de la actividad P4	24/12/2024	14/01/2025
Entrega de la actividad P5	20/01/2025	31/01/2025
Entrega de la actividad P4 - Vídeo		14/01/2025

Figura 2 - Fecha de entrega de las PAC

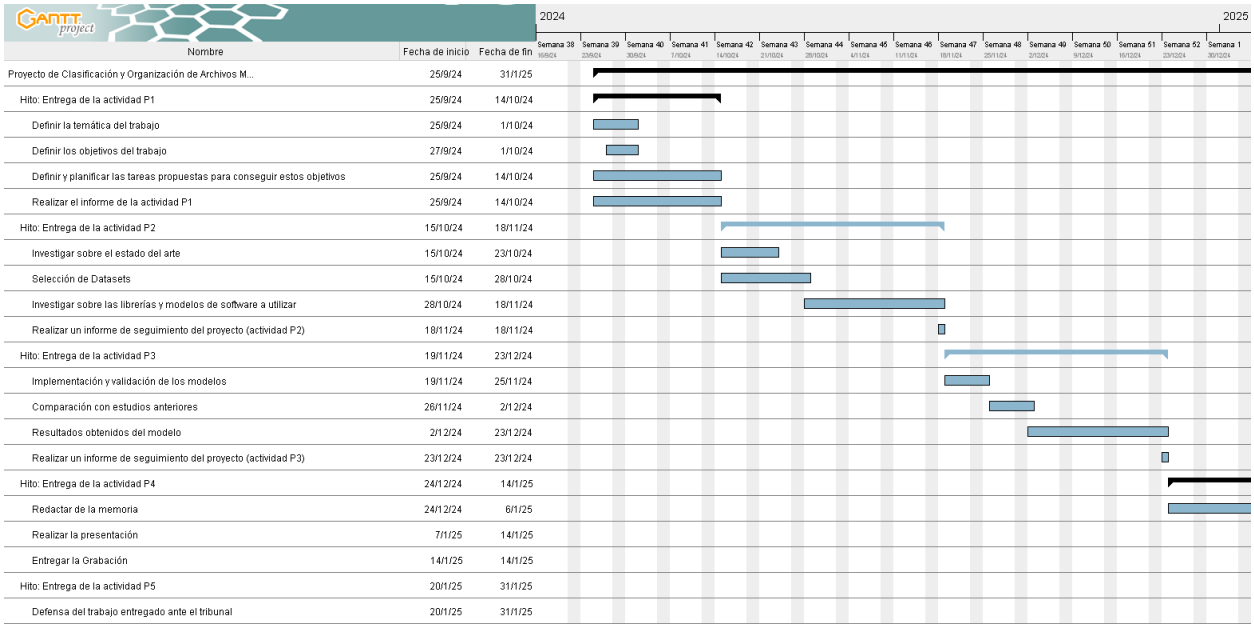


Figura 3 - Diagrama de GANTT V1

2. Estado del arte

2.1 - Evolución de las Arquitecturas de Redes Neuronales

La clasificación de imágenes ha sido impulsada por el desarrollo de diversas arquitecturas de redes neuronales, comenzando con las redes neuronales convolucionales (CNN) que establecieron un estándar en el campo. Modelos como ImageNet (Krizhevsky et al., 2012) revolucionaron la visión por computadora al demostrar que las CNN podían superar a los métodos tradicionales en tareas de clasificación de imágenes. Estas redes utilizan capas convolucionales para extraer características espaciales de las imágenes, lo que les permite aprender representaciones jerárquicas de los datos visuales. Sin embargo, a medida que la complejidad de las tareas aumentó, también lo hizo la necesidad de arquitecturas más eficientes y ligeras, especialmente para aplicaciones en dispositivos con recursos de procesamiento y memoria limitados.

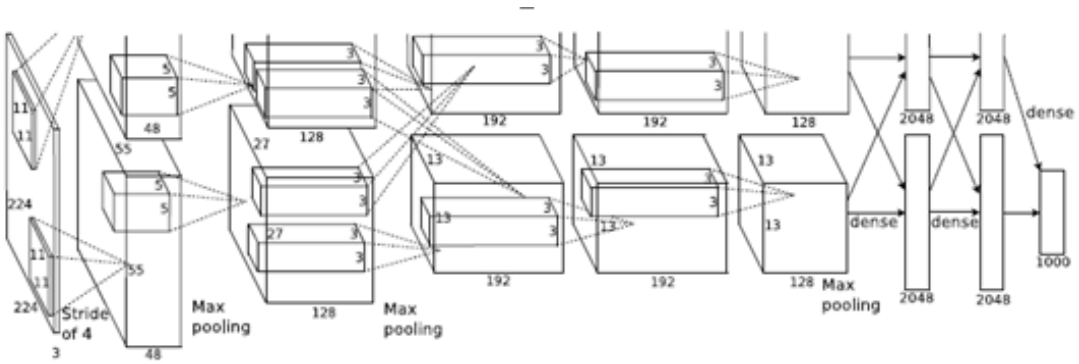


Figura 4 – ImageNet CNN (Krizhevsky et al., 2012)

La Figura de arriba, describe la arquitectura de la red neuronal convolucional (CNN) presentada por Krizhevsky et al. (2012), utilizada en su modelo AlexNet para clasificación de imágenes. Y quiere explicar gráficamente los puntos descritos a continuación:

1. **División del trabajo entre GPUs**:
    - La arquitectura de la CNN está diseñada para ejecutarse en dos GPUs.
    - Una GPU procesa las capas superiores (las partes de la figura en la parte superior), mientras que la otra GPU procesa las capas inferiores (las partes en la parte inferior de la figura).
    - Las GPUs solo se comunican en ciertas capas de la red, lo que optimiza el uso de recursos.

2. **Dimensión de entrada**:
    - La entrada a la red tiene una dimensión de 150,528, lo que corresponde al tamaño de las imágenes procesadas.

3. **Capas de la red**:

- Se enumeran las dimensiones y el número de neuronas en cada capa de la red.
- Las dimensiones de salida de las capas de convolución y agrupamiento (*max pooling)* se reducen gradualmente.
- Las capas totalmente conectadas al final tienen tamaños grandes (4096 neuronas por capa), lo que permite capturar patrones de alto nivel.

4. **Estructura jerárquica**:
   - El diseño de la red incluye capas convolucionales, de agrupamiento (pooling) y densamente conectadas (*fully connected).*

   - Cada capa extrae características más abstractas a medida que se avanza hacia las capas finales, lo que facilita la clasificación de imágenes.

En resumen, esta arquitectura está optimizada para grandes volúmenes de datos y aprovecha la computación distribuida en GPUs para acelerar el entrenamiento y la inferencia. Esto fue un avance clave en 2012 para mejorar la eficiencia en tareas de visión por computadora.

La clasificación de imágenes ha evolucionado significativamente en los últimos años, impulsada por el desarrollo de arquitecturas de redes neuronales profundas (Deep Neural Network, DNN). En este contexto, el ***transfer learning*** ha emergido como una técnica poderosa que permite aprovechar modelos preentrenados en grandes conjuntos de datos para tareas específicas, especialmente en dispositivos con recursos limitados. Este enfoque es particularmente relevante para la clasificación de imágenes, donde la eficiencia computacional y la precisión son cruciales.

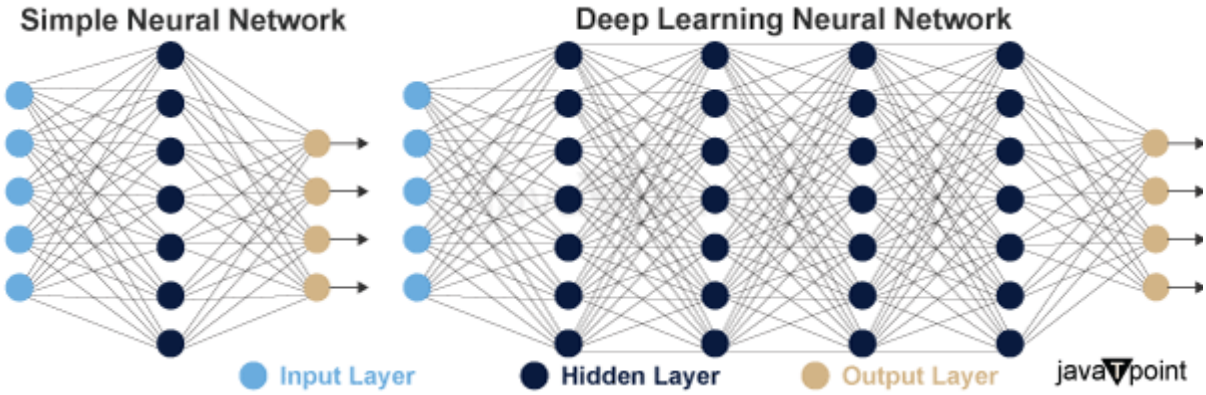


Figura 5 - Comparación entre SNN y DNN.(*DNN Machine Learning - Javatpoint*, s. f.) Fuente: https://www.javatpoint.com/dnn-machine-learning (Visitado el 11/12/2024)

Con el surgimiento de MobileNets (Howard et al., 2017) y MobileNetV2 (Sandler et al., 2019), se introdujeron modelos diseñados específicamente para ser eficientes en términos de recursos. Estas arquitecturas utilizan convoluciones separables en profundidad, lo que reduce significativamente la cantidad de parámetros y la carga computacional, permitiendo su

implementación en dispositivos con limitaciones de hardware. MobileNets han sido ampliamente adoptados en aplicaciones de visión por computadora, demostrando su eficacia en la clasificación de imágenes y otras tareas relacionadas, como la detección de objetos y la segmentación semántica.

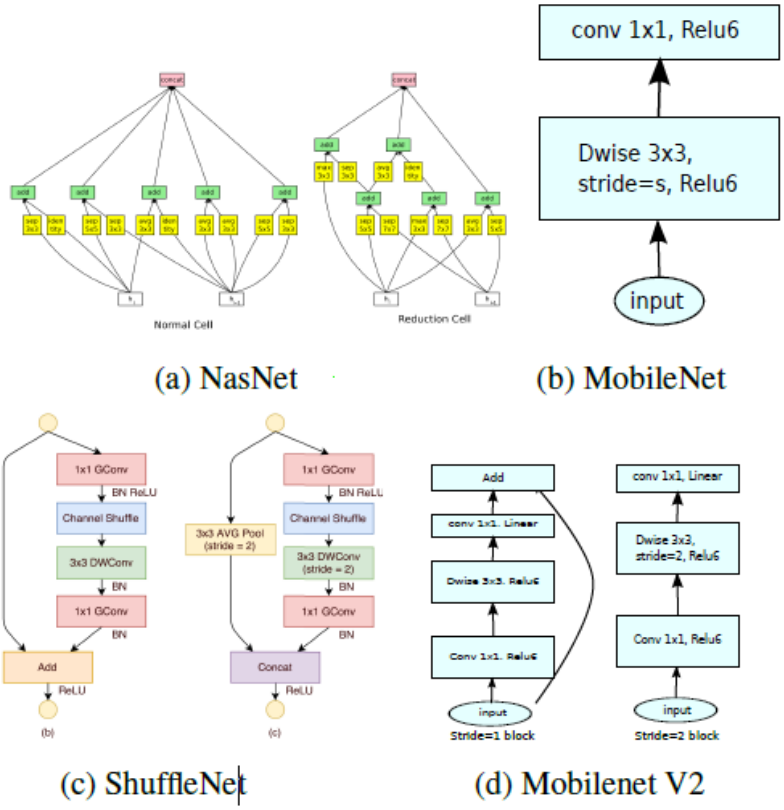


Figura 6 - Comparación de Bloques convolucionales de diferentes Arquitecturas. (Sandler et al., 2019)

2.2 - Transformers en la Clasificación de Imágenes

Los Transformers han revolucionado el campo del aprendizaje profundo, especialmente en tareas de procesamiento de lenguaje natural y, más recientemente, en visión por computadora. Introducidos por (Vaswani et al., 2017) en su trabajo *"Attention Is All You Need*", los Transformers utilizan mecanismos de atención para modelar relaciones entre diferentes partes de los datos de entrada, lo que les permite capturar dependencias a largo plazo de manera más efectiva que las arquitecturas tradicionales. Esta capacidad ha llevado a su adopción en modelos de visión, como el Vision Transformer (ViT), que divide las imágenes en parches y aplica atención *multi-head* para aprender representaciones *inter-patch*. Este enfoque ha demostrado ser efectivo en tareas de clasificación de imágenes a gran escala, logrando resultados competitivos en comparación con las redes neuronales convolucionales (CNN).

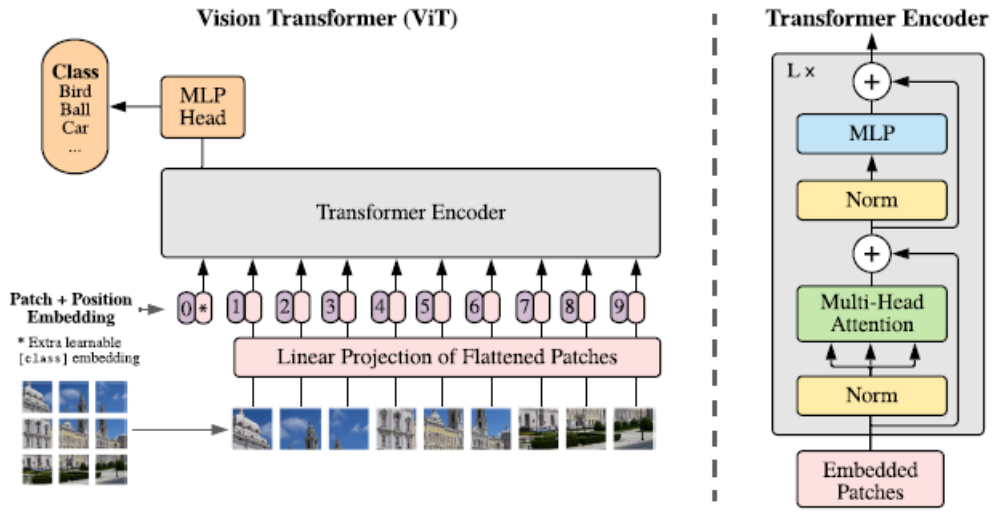


Figura 7 - Arquitecturas Transformers y Vision Transformers.(Dosovitskiy et al., 2021)

En la Figura 7 se explica el esquema del modelo Vision Transformer (ViT) y su funcionamiento:

- División en parches: La imagen de entrada se divide en parches de tamaño fijo.
- Embeddings: Cada parche se convierte linealmente en un vector y se le agrega información posicional mediante "position embeddings".
- Secuencia de vectores: Los vectores resultantes se combinan en una secuencia y se pasan al codificador Transformer estándar.
- Clasificación: Para realizar la clasificación, se agrega un token adicional que aprende una representación específica para la tarea de clasificación (llamado "*classification* token").
- Estructura del codificador Transformer: El codificador está compuesto por varias capas que incluyen atención multi-cabeza (*multi-head attention*), normalización (norm) y capas MLP (perceptrón multicapa).
- Inspiración del diseño: Este diseño del codificador Transformer se inspiró en el trabajo original de Vaswani et al. (2017), que introdujo los Transformers en el procesamiento de lenguaje natural.

Sin embargo, a pesar de su éxito, los Transformers presentan desafíos significativos cuando se implementan en dispositivos con recursos limitados. Los modelos basados en Transformers, como ViT, tienden a ser pesados en términos de parámetros y requieren grandes cantidades de datos para entrenarse adecuadamente. Esto se debe a que carecen de la inducción

espacial inherente a las CNN, lo que les dificulta aprender representaciones visuales de manera eficiente. Por ejemplo, en comparación con MobileNets, que están diseñados específicamente para ser ligeros y eficientes, los Transformers requieren más capacidad de procesamiento y memoria, lo que limita su aplicabilidad en dispositivos móviles. Esta limitación es crítica en aplicaciones donde la latencia y el uso de recursos son preocupaciones primordiales.

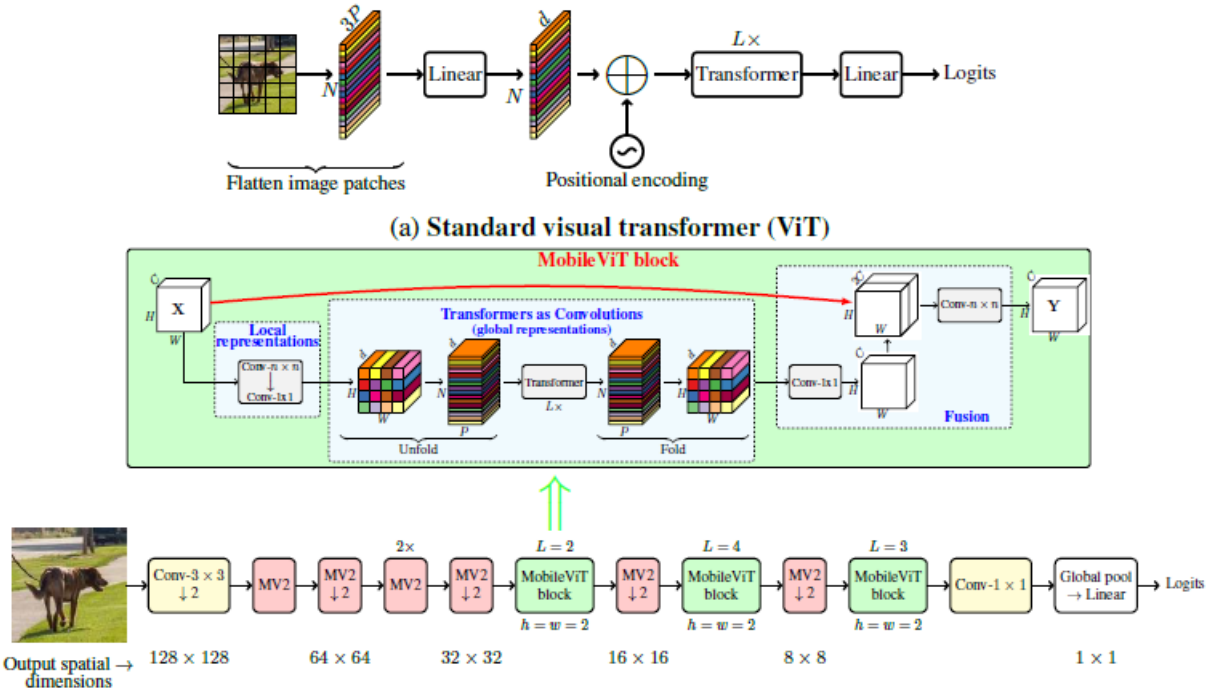


Figura 8 - MobileViT.(Mehta & Rastegari, 2022a)

En Figura 8, se describen las características y propiedades del MobileViT en comparación con modelos CNN ligeros. A continuación, describimos la imagen en detalle:

MobileViT Block:

- El término Conv-n x n en los bloques de MobileViT se refiere a una convolución estándar de tamaño n x n.
- MV2 hace referencia a los bloques de MobileNetV2, utilizados para realizar operaciones de "down-sampling" (reducción de la dimensión espacial). Estos bloques están marcados con una flecha hacia abajo (↓2), indicando la reducción en el tamaño espacial de las características procesadas.

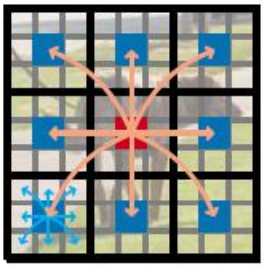
Generalización del MobileViT:

- Capacidad de generalización: MobileViT demuestra mejores propiedades de generalización a nivel de tarea en comparación con

modelos ligeros de CNN. Esto significa que es más capaz de adaptarse a diferentes tareas manteniendo una alta precisión.

- Parámetros de red: Se enumeran los parámetros de la red para diferentes extractores de características (como MobileNetV1, MobileNetV2, y MobileNetV3, entre otros) utilizados en el SSDLite, que es una red de detección de objetos.
- Datasets y resultados: La comparación se realiza sobre el dataset MS-COCO, destacando que MobileViT logra resultados superiores.

Para abordar estas limitaciones, se han desarrollado modelos híbridos que combinan las ventajas de las CNN y los Transformers. Uno de los modelos más destacados en este ámbito es **MobileViT**, propuesto por (Mehta & Rastegari, 2022a). MobileViT combina las ventajas de las redes neuronales convolucionales (CNN) y los transformadores, ofreciendo una arquitectura ligera y optimizada para dispositivos móviles. Este modelo ha demostrado ser efectivo en la clasificación de imágenes, logrando un equilibrio entre rendimiento y eficiencia, lo que lo convierte en una opción ideal para aplicaciones en entornos con recursos limitados. La capacidad de MobileViT para adaptarse a diferentes tareas de visión por computadora se basa en su diseño que integra características de ambas arquitecturas, permitiendo un procesamiento más eficiente de las imágenes. En comparación con otros modelos, MobileViT requiere conjuntos de datos más grandes y diversos para alcanzar su máximo potencial, lo que puede ser un desafío en aplicaciones de transfer learning.



Every pixel sees every other pixel in the MobileViT block.

Figura 9 - Arquitectura MobileViT. (Mehta & Rastegari, 2022a)

La Figura 9, describe cómo funciona el bloque MobileViT utilizando transformadores para permitir que cada píxel en una imagen "vea" y utilice información de todos los demás píxeles. Podemos tener una explicación más detallada a continuación:

1. **Conexión entre píxeles**:
    - El *píxel rojo* (marcado en la imagen) atiende a los *píxeles azules*, que corresponden a ubicaciones similares en otros parches.
    - Esta atención es realizada por los transformadores, que permiten que los píxeles intercambien información más allá de sus vecindades inmediatas.

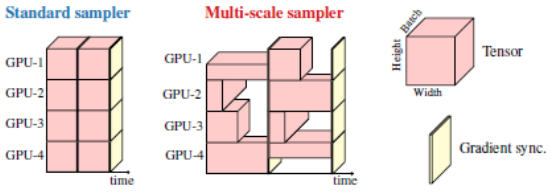
2. **Codificación previa con convoluciones**:
    - Los *píxeles azules* ya han codificado información sobre sus píxeles vecinos utilizando convoluciones.
    - Esto significa que antes de que los transformadores procesen la información, las convoluciones han capturado relaciones locales.

3. **Ventaja de los transformadores**:
    - Debido a que los *píxeles azules* ya contienen información local, el *píxel rojo* puede utilizar esta información para construir una representación que integre datos de toda la imagen.
    - Esto le permite al *píxel rojo* "ver" indirectamente todos los demás píxeles en la imagen, proporcionando un contexto global.

4. **Representación visual**:
    - En la figura, las celdas en *negro* representan un parche de la imagen, mientras que las celdas en *gris* representan píxeles individuales dentro del parche.
    - El esquema muestra cómo se logra la conexión entre píxeles utilizando tanto convoluciones como transformadores.

Con esta explicación podemos ver cómo MobileViT combina las capacidades de las convoluciones (para capturar relaciones locales) y los transformadores (para capturar relaciones globales), lo que mejora la capacidad del modelo para procesar imágenes completas.

La publicación **Mobilevit: light-weight, general-purpose, and mobile-friendly vision transformer** (Mehta & Rastegari, 2022a) muestra en los resultados de sus experimentos que MobileViT supera significativamente a las redes basadas en CNN y ViT en diferentes tareas y conjuntos de datos. En el conjunto de datos ImageNet-1k, MobileViT logra una precisión top-1 del 78.4% con aproximadamente 6 millones de parámetros, lo que es un 3.2% y un 6.2% más preciso que **MobileNetv3** (basado en CNN) y **DeIT** (basado en ViT), respectivamente, para un número similar de parámetros. En la tarea de detección de objetos de MS-COCO, MobileViT es un 5.7% más preciso que **MobileNetv3** para un número similar de parámetros.



(a) Standard vs. multi-scale sampler illustration

Sampler	# Updates	Epoch time
Standard	375 k	380 sec
Multi-scale (ours)	**232 k**	**270 sec**

(b) Training efficiency. Here, standard sampler refers to PyTorch's Distributed-DataParallel sampler.

Figura 10 - MobileViT Architecture. (Mehta & Rastegari, 2022a)

La Figura 10, ilustra la comparación entre el uso de un muestreador estándar y un muestreador de múltiples escalas, destacando cómo este último optimiza la eficiencia en el entrenamiento distribuido. Mientras que el muestreador

estándar asigna datos de forma uniforme a través de las GPUs, el enfoque de múltiples escalas implementa una estrategia que permite el uso más efectivo de los recursos computacionales, reduciendo significativamente el tiempo por época. Este enfoque fue desarrollado como parte de la investigación publicación Mobilevit: light-weight, general-purpose, and mobile-friendly vision transformer (Mehta & Rastegari, 2022a), donde se detallan los beneficios del muestreador de múltiples escalas en términos de actualizaciones necesarias y tiempo de entrenamiento, resultando en una mejora de la eficiencia general del modelo.

Además, el modelo CLIP (Radford et al., 2021) ha demostrado ser eficaz en la clasificación de imágenes mediante el aprendizaje multimodal, utilizando datos de texto e imágenes. Este enfoque permite que CLIP generalice bien incluso con conjuntos de datos limitados, lo que es una ventaja significativa en escenarios donde la cantidad de datos es un factor limitante. Pero no está optimizado para ejecutarse en dispositivos con limitaciones de hardware.

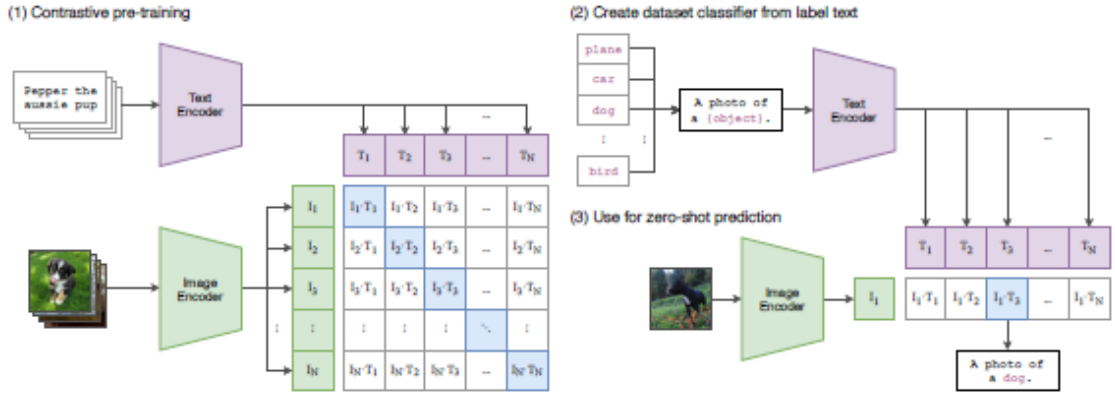


Figura 11 - Arquitectura CLIP Transformer. (Radford et al., 2021)

La Figura 11, presenta el enfoque utilizado por el modelo CLIP, donde se combina el preentrenamiento contrastivo con supervisión del lenguaje natural para lograr un aprendizaje transferible. A diferencia de los modelos estándar que entrenan únicamente un extractor de características de imágenes junto con un clasificador lineal, CLIP entrena de manera conjunta un codificador de imágenes (*image encoder*) y un codificador de texto (*text encoder*). Esta estrategia permite al modelo predecir las relaciones correctas entre pares de datos de imágenes y texto, basándose en descripciones semánticas de las clases objetivo.(Radford et al., 2021)
Una de las características distintivas de CLIP es su capacidad para generar clasificadores lineales de cero-shot al incrustar los nombres o descripciones de las clases del dataset objetivo, eliminando la necesidad de entrenamiento específico en dichas clases. Este enfoque innovador es detallado en el trabajo de Radford et al. (2021), proporcionando una solución altamente generalizable para tareas de clasificación.

En resumen, la combinación de arquitecturas ligeras como MobileViT con técnicas de transfer learning representa un avance significativo en la clasificación de imágenes optimizada para dispositivos con recursos limitados.

Este enfoque no solo mejora la eficiencia computacional, sino que también permite una mayor adaptabilidad a diversas tareas de visión por computadora, lo que es esencial en el desarrollo de aplicaciones móviles efectivas. Por este anàlisis se ha seleccionado el modelo MobileViT para desarrollar este Trabajo.

2.3 Justificación de la elección entre la arquitectura CNN y Transformers

El avance de las redes convolucionales (CNN), como las arquitecturas MobileNetV2 o ResNet, ha permitido grandes logros en tareas de clasificación y detección. Sin embargo, las CNN presentan limitaciones inherentes debido a su enfoque local: cada convolución analiza solo una porción reducida de la imagen, lo que puede dificultar la captura de patrones globales. Los Transformers, introducidos por Vaswani et al. (2017), abordan esta limitación mediante mecanismos de atención que permiten modelar relaciones globales dentro de una imagen completa, facilitando el aprendizaje de dependencias complejas entre regiones de la imagen. Por ejemplo, los Transformers pueden identificar correlaciones entre elementos distantes de una imagen, lo que resulta crucial en escenarios como la clasificación de escenas o la detección de objetos en contextos complejos. Esta capacidad se traduce en una ventaja significativa en tareas que requieren una comprensión más holística del contenido visual. En los documentos sobre aprendizaje multimodal con Transformers que hemos revisado, hemos concluido que el preentrenamiento en conjuntos de datos grandes permite a los modelos capturar representaciones generales, facilitando su transferencia *(transfer learning*) a aplicaciones especializadas.

3. Materiales y metodología

3.1. Introducción

Este capítulo describe los materiales utilizados y la metodología aplicada para aplicar varios mecanismos basados en redes neuronales sobre varios conjuntos de imágenes para resolver un problema de clasificación de imágenes y la optimización modelos para entornos de recursos limitados. Se realizaron experimentos utilizando los datasets públicos MSRA-TD500, flickr8k y tiny-imagenet-200.

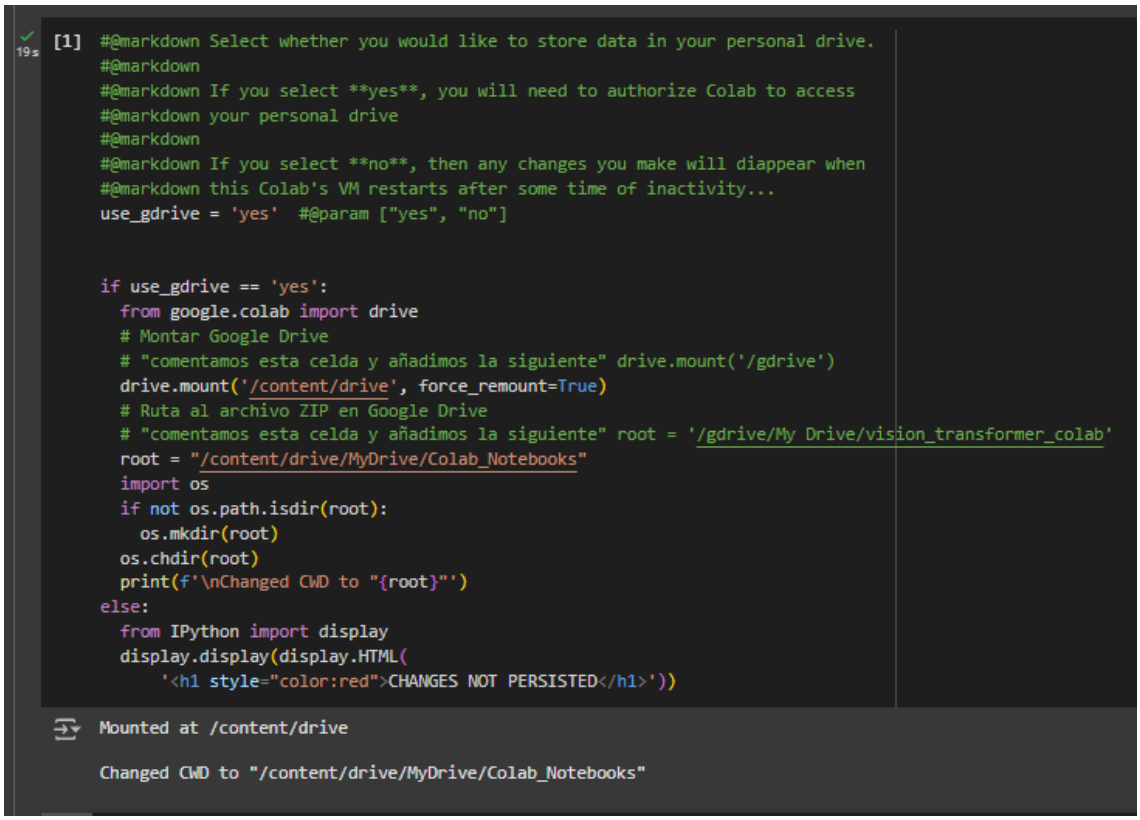
3.2. Entorno de trabajo

Para llevar a cabo los experimentos, se utilizó el siguiente entorno de trabajo:

- Hardware:
  - Portátil personal:
    - Procesador: Intel Core i5-1135G7 @ 2.4GHz (cuatro núcleos, ocho hilos).
    - Memoria RAM: 16 GB DDR4.
    - Almacenamiento: 512 GB SSD NVMe.
    - Tarjeta gráfica: Intel Iris Xe Graphics integrada.
    - Sistema operativo: Windows 11 Pro de 64 bits.
  - Entorno en la nube (Google Colab Pro):
    - Procesador: Servidores de Google con acceso a GPUs NVIDIA Tesla T4 o TPUs, dependiendo de la configuración.
    - Memoria: Hasta 334 GB de RAM asignada en el entorno de ejecución.
    - Almacenamiento: Integración con Google Drive 100 GB.

- Software:
  - Entorno de desarrollo: Google Colab, un entorno en la nube preconfigurado que permite la ejecución de código Python sin necesidad de configuraciones locales complejas.
  - Lenguaje de programación Python 3.
  - Framework PyTorch, que facilita la implementación y personalización de modelos de aprendizaje profundo.
  - Librerías auxiliares: torchvision, matplotlib, numpy, y scikit-learn.

  - Preparación del entorno de trabajo:
    - Abrimos un cuaderno en la plataforma Google Colab:
    - En una celda de código configuramos la ruta para montar nuestra unidad de Drive:



```
[1] #@markdown Select whether you would like to store data in your personal drive.
    #@markdown
    #@markdown If you select **yes**, you will need to authorize Colab to access
    #@markdown your personal drive
    #@markdown
    #@markdown If you select **no**, then any changes you make will diappear when
    #@markdown this Colab's VM restarts after some time of inactivity...
    use_gdrive = 'yes'  #@param ["yes", "no"]


    if use_gdrive == 'yes':
      from google.colab import drive
      # Montar Google Drive
      # "comentamos esta celda y añadimos la siguiente" drive.mount('/gdrive')
      drive.mount('/content/drive', force_remount=True)
      # Ruta al archivo ZIP en Google Drive
      # "comentamos esta celda y añadimos la siguiente" root = '/gdrive/My Drive/vision_transformer_colab'
      root = "/content/drive/MyDrive/Colab_Notebooks"
      import os
      if not os.path.isdir(root):
        os.mkdir(root)
      os.chdir(root)
      print(f'\nChanged CWD to "{root}"')
    else:
      from IPython import display
      display.display(display.HTML(
          '<h1 style="color:red">CHANGES NOT PERSISTED</h1>'))

    Mounted at /content/drive

    Changed CWD to "/content/drive/MyDrive/Colab_Notebooks"
```

Figura 12 - Montar unidad Drive en Colab

3.3. Carga de datos

Se utilizó *torchvision.datasets.ImageFolder* para organizar las imágenes en carpetas por clase, facilitando su integración con el pipeline de entrenamiento. El dataset se dividió en un 80% para entrenamiento y un 20% para validación, asegurando que los modelos se entrenen y validen de manera adecuada y evitar sobreajuste.

Para los experimentos, se utilizaron los siguientes datasets:

- **Tiny-ImageNet-200** (Le, Y., & Yang, X., 2015)
  - **Descripción:** Este dataset se creó como parte de un proyecto para el curso de visión por computadora CS231N de Stanford y es una versión reducida de ImageNet diseñada para investigación académica y educativa. Contiene 200 clases con 500 imágenes de entrenamiento por clase, 50 de validación y 50 de prueba. Las imágenes son de baja resolución (64x64 píxeles), lo que lo hace útil para entrenar modelos con recursos computacionales limitados.
  - **Aplicación:** Este dataset se evaluó inicialmente para probar el modelo en tareas de clasificación general. Sin embargo, se descartó porque no contenía las etiquetas específicas necesarias para la detección de texto en imágenes.

- **Dataset propio de clasificación binaria:**

- Este dataset lo vamos a llamar **“text_recognition”** y a lo largo de los experimentos lo cargaremos desde la ruta: "/content/drive/MyDrive/Colab_Notebooks/dataset/text_recognition_balanceado"

- **Descripción:** Para la creación de este dataset propio se utilizaron los dataset públicos MSRA-TD500 y Flickr8k. En concreto se extrajeron 200 muestras aleatorias del conjunto train de MSRA-TD500 y 200 muestras aleatorias del conjunto Flickr8k.
- **Metodología:** Se utilizo la metodología de **combinación de datasets** fusionando 2 fuentes públicas existentes, además de hacerles **transformación de dataset** modificando sus clases y también se les aplicó ***Data Augmentation***, para generar nuevas muestras a partir de transformaciones como rotación.
- **Aplicación:** Se generó un conjunto de datos con las clases text y no_text, que fue preprocesado mediante las siguientes transformaciones:
  - **Redimensionamiento:** Imágenes escaladas a 224x224 píxeles para garantizar la compatibilidad con los modelos.
  - **Normalización:** Ajuste de los valores de píxeles para una convergencia más rápida durante el entrenamiento.
  - **Balanceo de Clases:** Para descartar que esta fuera la causa de una precisión engañosa y a un rendimiento deficiente en la clasificación de clases menos representadas.

- **MSRA-TD500** (Kwon & Lee, 2012b; *MSRA Text Detection 500 Database (MSRA-TD500) - TC11*, s. f.)
  - **Descripción:** Contiene 500 imágenes, de escenas naturales con texto orientado en diferentes direcciones. Las imágenes se distribuyen en 300 imágenes que se utilizan para el conjunto de entrenamiento y 200 imágenes se reservan para pruebas. Las imágenes incluyen texto tanto en chino como en inglés.
  - **Aplicación:** Fue diseñado específicamente para evaluar algoritmos de detección de texto en escenarios complejos y realistas con orientaciones arbitrarias.

- **Flickr8k** (Hodosh et al., 2013)
  - **Descripción:** Un dataset compuesto por 8,000 imágenes acompañadas de descripciones textuales generadas por humanos. Estas imágenes son diversas y capturan una amplia gama de actividades, objetos y escenas.
  - **Aplicación:** Este dataset permitió complementar el experimento al extraer de este conjunto de imagenes un numero de muestras similar al contenido en **MSRA-TD500** para utilizarlo como clase de imágenes que no contienen texto.

3.4 Desarrollo de los experimentos y resultados obtenidos

3.4.1. Primer experimento con el modelo MobileViT y el dataset tiny-imagenet-200

- Pasos de este experimento:
  - Se ejecuta cuaderno Jupiter en Google Colaboratory.
  - Se monta la unidad Drive
  - Clonamos en repositorio del modelo preentrenado:

```
!git clone https://github.com/quic/ai-hub-models.git
%cd ai-hub-models/qai_hub_models/models/mobile_vit
```

  - Redimentsionamos las imágenes del dataset:

```
# Directorio de entrada y salida
input_directory = "/content/drive/MyDrive/Colab_Notebooks/dataset/images/tiny-imagenet-200/tiny-imagenet-200-64x64"
output_directory = "/content/drive/MyDrive/Colab_Notebooks/dataset/images/tiny-imagenet-200/tiny-imagenet-200-224x224"
```

  - Realizamos la carga del dataset tiny-imagenet-200:
    - Datasets y DataLoaders creados exitosamente.
    - Tamaño del conjunto de entrenamiento: 80000
    - Tamaño del conjunto de validación: 20000
  - Comprobamos que el modelo previamente predice el contenido con el Dataset creado para esta investigación, en concreto la selección aleatoria de 200 imágenes del dataset publico Flickr-8k (Hodosh et al., 2013)

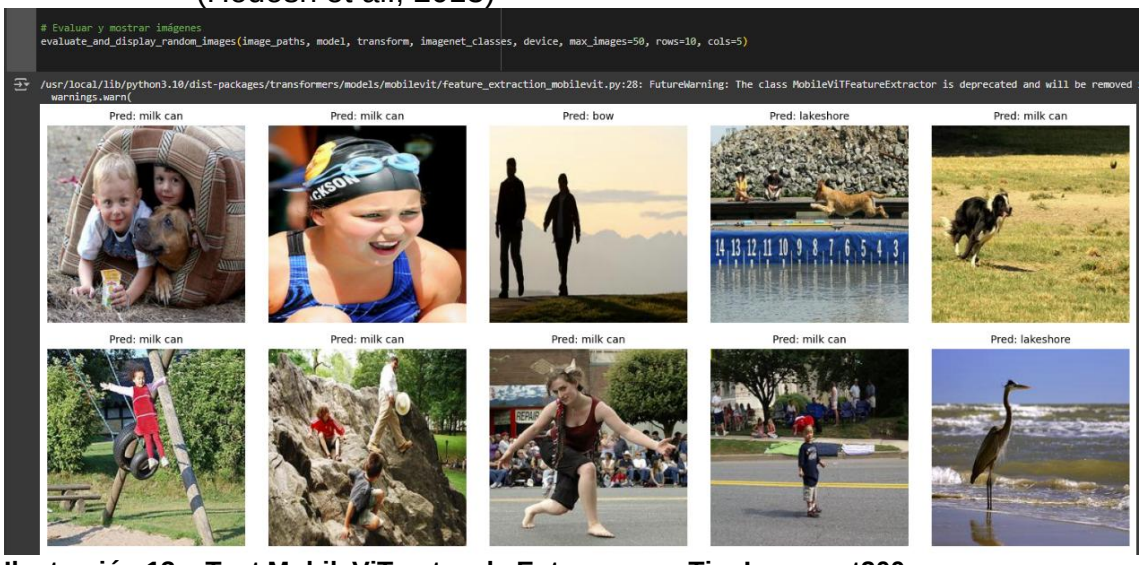


Ilustración 13 – Test MobileViT antes de Entrenar con TinyImagenet200

  - Ejecutamos la configuración del modelo:

```
# 2. Configurar el modelo MobileViT
# Importar el modelo MobileViT desde el repositorio
from qai_hub_models.models.mobile_vit.model import MobileVIT
import torch.nn as nn
import torch.optim as optim

# Cargar el modelo MobileViT preentrenado en ImageNet
model = MobileVIT.from_pretrained()

# Ajustar la última capa para 200 clases de Tiny-ImageNet
num_classes = 200
model.net.classifier =
nn.Linear(model.net.classifier.in_features, num_classes)

# 3. Configurar el optimizador y la función de pérdida
criterion = nn.CrossEntropyLoss()  # Pérdida de clasificación
estándar
# Se configuró el optimizador Adam con una tasa de
aprendizaje baja de valor 0.0001, adecuada para el
reentrenamiento, ya que reduce el riesgo de destruir la
información adquirida durante el preentrenamiento.
optimizer = optim.Adam(model.parameters(), lr=0.0001)
```

- Se entrena el modelo para 2 etapas como prueba para medir el tiempo de entrenamiento, y vemos que hay una mejora en la segunda etapa.
  - Epoch 1/2, Loss: 1.7959, Validation Accuracy: 68.93%
  - Epoch 2/2, Loss: 1.1366, Validation Accuracy: 73.48%

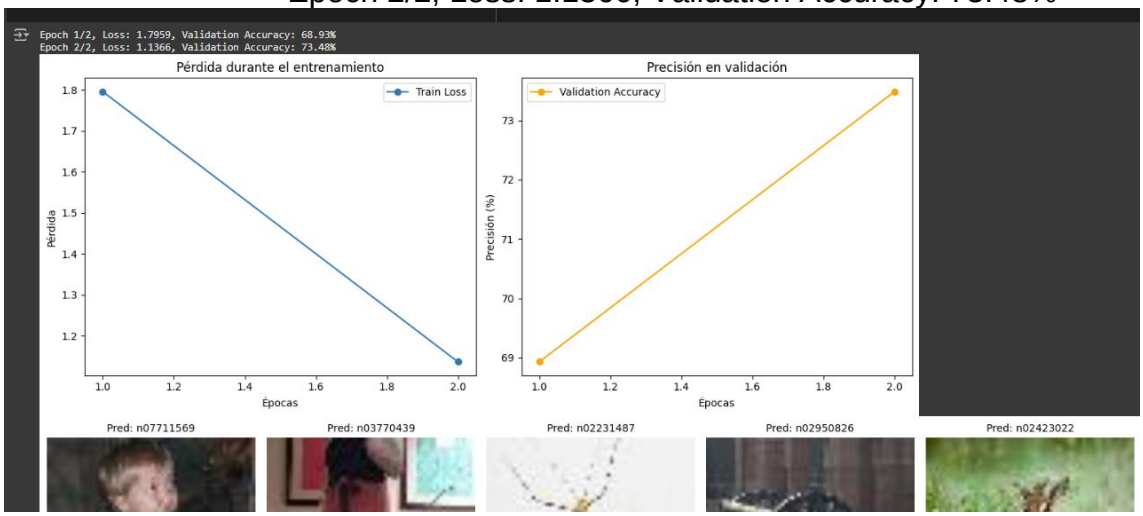


Figura 14 - Resultados Mobilevit con TinyImagenet200

- Intentamos entrenar en 10 épocas y finalizamos manualmente ejecución tras 9 horas en proceso y un consumo de mas de 40 unidades informáticas

```
# Entrenamiento y evaluación
num_epochs = 10
for epoch in range(num_epochs):
```

```
    # Entrenamiento
    model.train()
    running_loss = 0.0
    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(inputs)  # Forward
        loss = criterion(outputs, labels)  # Calcular pérdida
        loss.backward()
        optimizer.step()

        running_loss += loss.item()

    train_losses.append(running_loss / len(train_loader))  # Pérdida promedio de la época
```

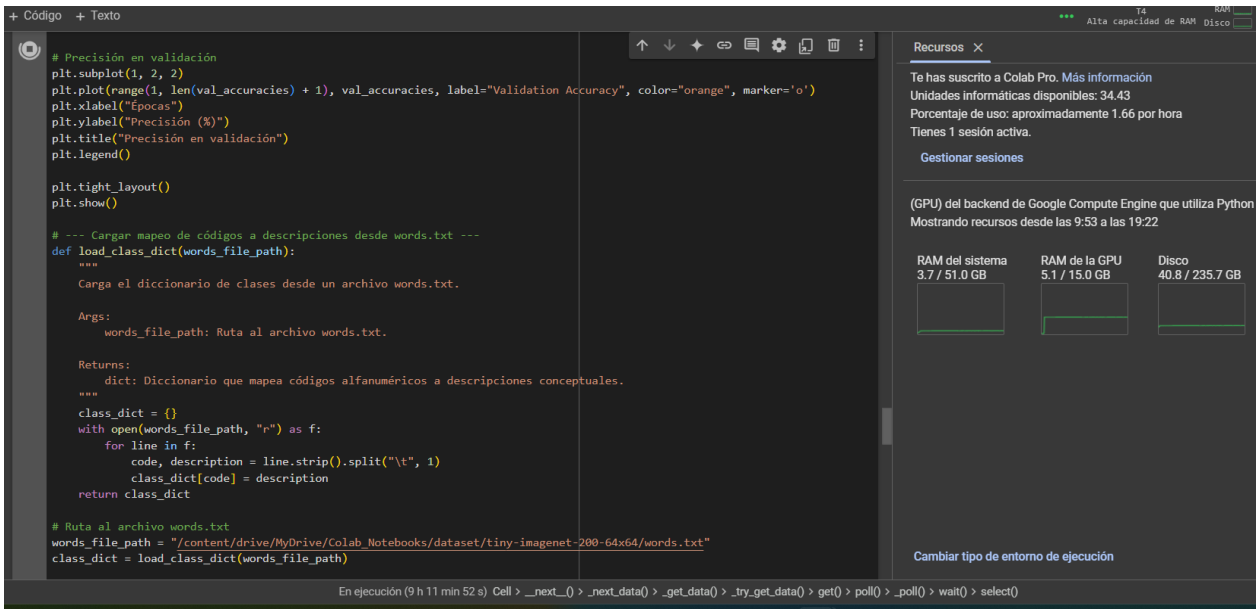


Figura 15 - Finalizamos ejecución con alto consumo

- Código para ver todas las clases del dataset en el fichero “1er_experimento_MobileViT_y_tiny-imagenet-200.ipynb”

```
# Diccionario de classes: Los títulos de las imágenes mostrarán la descripción semántica en lugar del código alfanumérico
# Cargar el archivo words.txt para mapear códigos a descripciones
words_path = "/content/drive/MyDrive/Colab_Notebooks/dataset/images/tiny-imagenet-200/tiny-imagenet-200-64x64/words.txt"
def load_class_descriptions(words_path):
    class_descriptions = {}
```

```
    with open(words_path, "r") as f:
        for line in f:
            code, description = line.strip().split("\t", 1)
            class_descriptions[code] =
description.split(",")[0]  # Usar solo la primera descripción
    return class_descriptions
```

Clases en Tiny-ImageNet con sus descripciones semánticas: n01443537: goldfish; n01629819: European fire salamander; n01641577: bullfrog; n01644900: tailed frog; n01698640: American alligator; n01742172: boa constrictor; n01768244: trilobite; n01770393: scorpion; n01774384: black widow; n01774750: tarantula; n01784675: centipede; n01855672: goose; n01882714: koala; n01910747: jellyfish; n01917289: brain coral; n01944390: snail; n01945685: slug; n01950731: sea slug; n01983481: American lobster; n01984695: spiny lobster; n02002724: black stork; n02056570: king penguin; n02058221: albatross; n02074367: dugong; n02085620: Chihuahua; n02094433: Yorkshire terrier; n02099601: golden retriever; n02099712: Labrador retriever; n02106662: German shepherd; n02113799: standard poodle; n02123045: tabby; n02123394: Persian cat; n02124075: Egyptian cat; n02125311: cougar; n02129165: lion; n02132136: brown bear; n02165456: ladybug; n02190166: fly; n02206856: bee; n02226429: grasshopper; n02231487: walking stick; n02233338: cockroach; n02236044: mantis; n02268443: dragonfly; n02279972: monarch; n02281406: sulphur butterfly; n02321529: sea cucumber; n02364673: guinea pig; n02395406: hog; n02403003: ox; n02410509: bison; n02415577: bighorn; n02423022: gazelle; n02437312: Arabian camel; n02480495: orangutan; n02481823: chimpanzee; n02486410: baboon; n02504458: African elephant; n02509815: lesser panda; n02666196: abacus; n02669723: academic gown; n02699494: altar; n02730930: apron; n02769748: backpack; n02788148: bannister; n02791270: barbershop; n02793495: barn; n02795169: barrel; n02802426: basketball; n02808440: bathtub; n02814533: beach wagon; n02814860: beacon; n02815834: beaker; n02823428: beer bottle; n02837789: bikini; n02841315: binoculars; n02843684: birdhouse; n02883205: bow tie; n02892201: brass; n02906734: broom; n02909870: bucket; n02917067: bullet train; n02927161: butcher shop; n02948072: candle; n02950826: cannon; n02963159: cardigan; n02977058: cash machine; n02988304: CD player; n02999410: chain; n03014705: chest; n03026506: Christmas stocking; n03042490: cliff dwelling; n03085013: computer keyboard; n03089624: confectionery; n03100240: convertible; n03126707: crane; n03160309: dam; n03179701: desk; n03201208: dining table; n03250847: drumstick; n03255030: dumbbell; n03355925: flagpole; n03388043: fountain; n03393912: freight car; n03400231: frying pan; n03404251: fur coat; n03424325: gasmask; n03444034: go-kart; n03447447: gondola; n03544143: hourglass; n03584254: iPod; n03599486: jinrikisha; n03617480: kimono; n03637318: lampshade; n03649909: lawn mower; n03662601: lifeboat; n03670208: limousine; n03706229: magnetic compass; n03733131: maypole; n03763968: military uniform; n03770439: miniskirt; n03796401: moving van; n03804744: nail; n03814639: neck brace; n03837869: obelisk; n03838899: oboe; n03854065: organ; n03891332: parking meter; n03902125: pay-phone; n03930313: picket fence; n03937543: pill bottle; n03970156: plunger; n03976657: pole; n03977966: police van; n03980874: poncho; n03983396: pop bottle; n03992509: potter's wheel; n04008634: projectile; n04023962: punching bag; n04067472: reel; n04070727: refrigerator; n04074963: remote control; n04099969: rocking chair; n04118538: rugby ball; n04133789: sandal; n04146614: school bus; n04149813: scoreboard; n04179913: sewing machine; n04251144: snorkel; n04254777: sock; n04259630: sombrero; n04265275: space heater; n04275548: spider web; n04285008: sports car; n04311004: steel arch bridge; n04328186: stopwatch; n04356056: sunglasses; n04366367: suspension bridge; n04371430: swimming trunks; n04376876: syringe; n04398044: teapot; n04399382: teddy; n04417672: thatch; n04456115: torch; n04465501: tractor; n04486054: triumphal arch; n04487081: trolleybus; n04501370: turnstile; n04507155: umbrella; n04532106: vestment; n04532670: viaduct; n04540053: volleyball; n04560804: water jug; n04562935: water tower; n04596742: wok; n04597913: wooden spoon; n06596364: comic book; n07579787: plate; n07583066: guacamole; n07614500: ice cream; n07615774: ice lolly; n07695742: pretzel; n07711569: mashed potato; n07715103: cauliflower; n07720875: bell pepper; n07734744: mushroom; n07747607: orange; n07749582: lemon; n07753592: banana; n07768694: pomegranate; n07871810: meat loaf; n07873807: pizza; n07875152: potpie; n07920052: espresso; n09193705: alp; n09246464: cliff; n09256479: coral reef; n09332890: lakeside; n09428293: seashore; n12267677: acorn

- Conclusión de este experimento: Además del consumo de unidades informáticas que supuso el uso del dataset tiny-imagenet-200, realizamos una consulta y vemos que las etiquetas de las clases son limitadas, sobre todo tomando en cuenta que una de las clasificaciones que queremos tomar en cuenta a parte de la clasificación por personas, objetos o animales, es la clasificación por imágenes que contengan etiquetas de tipo: texto, documento, recibo.

3.4.2. Segundo experimento con el modelo MobileViT y el dataset text_recognition:

- Se realiza la carga de datos, con el objetivo de optimizar el preprocesamiento según las recomendaciones del modelo preentrenado, para obtener mayor compatibilidad con MobileViT:
    - Transformaciones de las imágenes:
        - Resize: Asegura que las imágenes sean compatibles con MobileViT, que espera entradas de 224x224 píxeles.
        - Normalize: **((0.5,), (0.5,)):** Normalización estándar, este rango normaliza los valores de los píxeles a [-1, 1]
    - División en entrenamiento y validación: Proporción del 80%-20%, estándar en tareas de aprendizaje supervisado. Garantiza suficiente cantidad de datos para entrenamiento y evaluación.
    - Tamaño del batch (32): Se recomienda usar tamaños de lote moderados con valor entre 16 y 64.
    - Se utilizó torchvision.datasets.ImageFolder, compatible con estructuras jerárquicas de carpetas (clase/imágenes).
    - batch_size = 32: Es un tamaño común para GPUs con memoria moderada (16-64 GB entre este rango estarían los valores de consumo que se configuran generalmente).
    - criterion = nn.CrossEntropyLoss(): Para clasificación binaria
    - optimizer = optim.Adam(model.parameters(), lr=0.001): Se configuró el optimizador Adam con una tasa de aprendizaje baja de valor 0.0001, adecuada para el reentrenamiento, ya que reduce el riesgo de destruir la información adquirida durante el preentrenamiento. Por el contrario, una tasa de aprendizaje alta (como 0.001 o mayor) podría sobreescribir los pesos preentrenados rápidamente, causando pérdida de generalización.
    - num_epochs = 10: Entrenamos por 10 epocas

- Se guarda el modelo que acabamos de entrenar con el nombre: mobilevit_text_classification_v1.pth

- Obtenemos una precisión del 81,27 % en un primer entrenamiento.

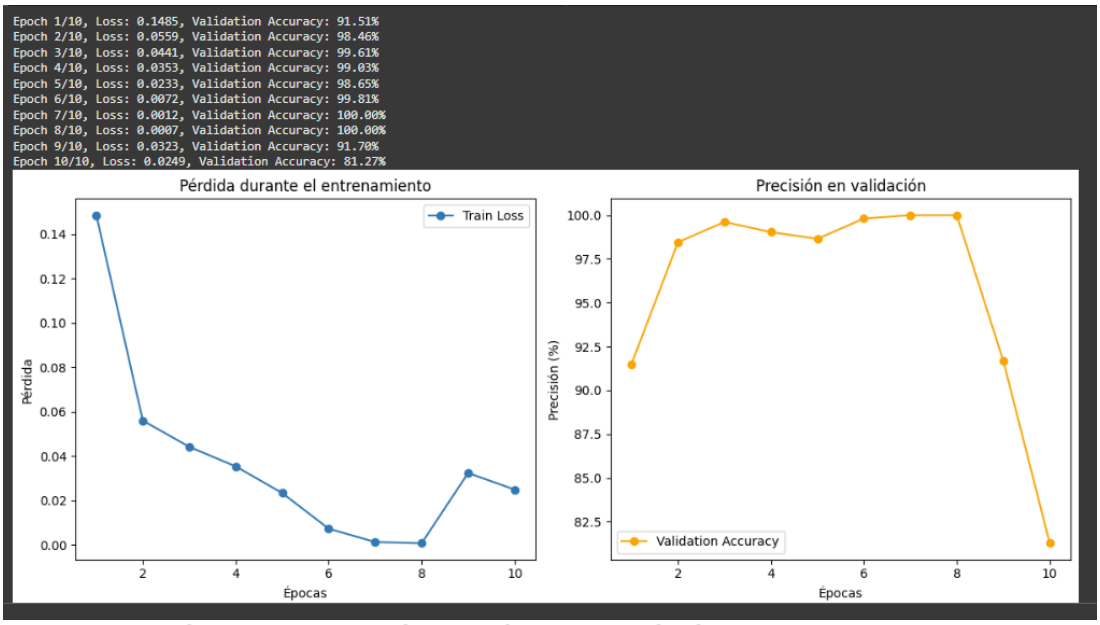


Figura 16 - Segundo experimento resultados 1.1

- Se hace una prueba, con una imagen propia de un set de fotos personales y no acierta la predicción el modelo:

Figura 17 - Segundo experimento prueba

- Se carga el modelo que acabamos de entrenar y guardar con el nombre: mobilevit_text_classification_v1.pth y obtenemos resultados de Pérdidas

de entrenamiento de 0% y precisión en validación de 99.80% lo que sugiere que:

- El modelo está sobre ajustado: En este caso es pequeño el set y puede ser que el modelo se ajustó demasiado rápido a los datos de entrenamiento.
- La métrica constante desde el inicio también podría indicar que no hay suficiente variación en los datos para desafiar al modelo.
- Problemas con el conjunto de validación: En este caso se ha utilizado el mismo set.
- Además, **estaban desbalanceadas las clases, las balanceamos y volvemos a entrenar, pero los resultados no mejoran.**
  - Distribución de clases en entrenamiento: Counter({0: 1913, 1: 157})
  - Distribución de clases en validación: Counter({0: 475, 1: 43})

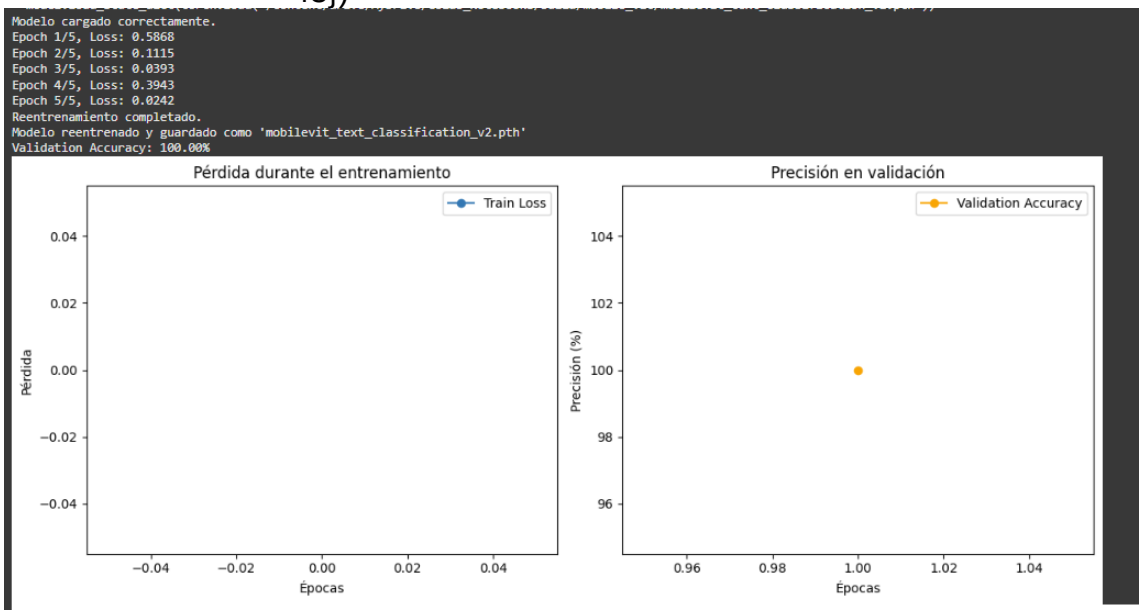


Figura 18 – Segundo experimento resultados 1.2 Overfitting

3.4.3 Tercer experimento con el modelo con el modelo MobileViT y el dataset text_recognition tras balancear las clases y aplicar auentacion de datos:

- Se vuelve a entrenar el modelo y obtenemos resultados de Pérdidas de entrenamiento de 0.7130 y precisión en validación de 51.24%.

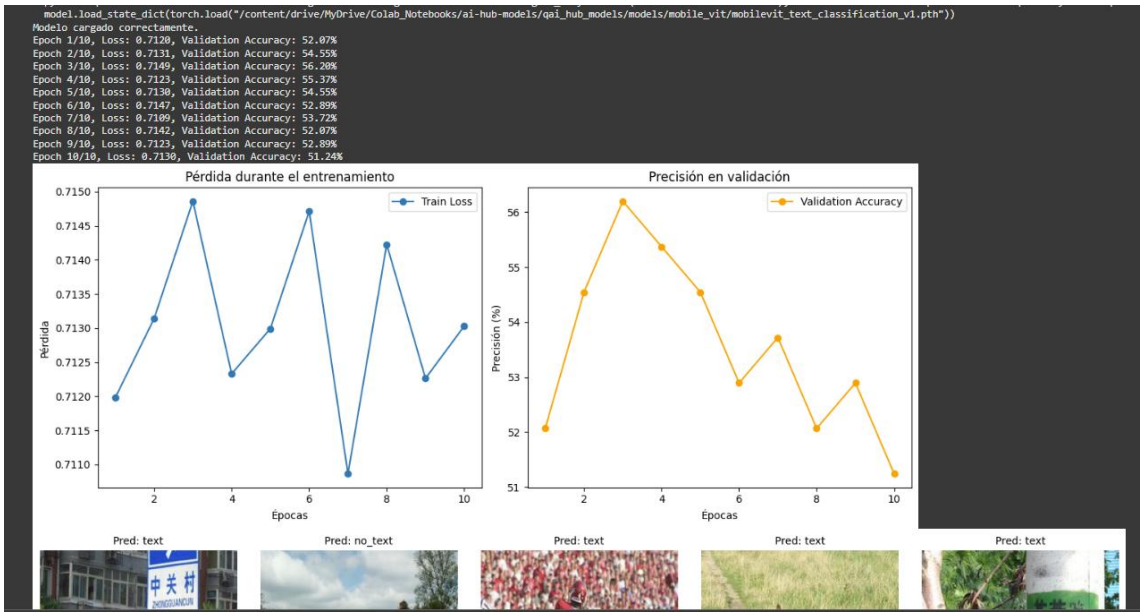


Figura 19 - Tercer experimento resultados 1.3

- Se carga el modelo que acabamos de entrenar y guardar con el nombre: mobilevit_text_classification_v3.pth y obtenemos resultados de Pérdidas de entrenamiento de 0.7118 y precisión en validación de 47.11% lo que sugiere que ya no estamos presentamos problemas de overfitting pero seguimos teniendo malos resultados en cuanto a la precisión.

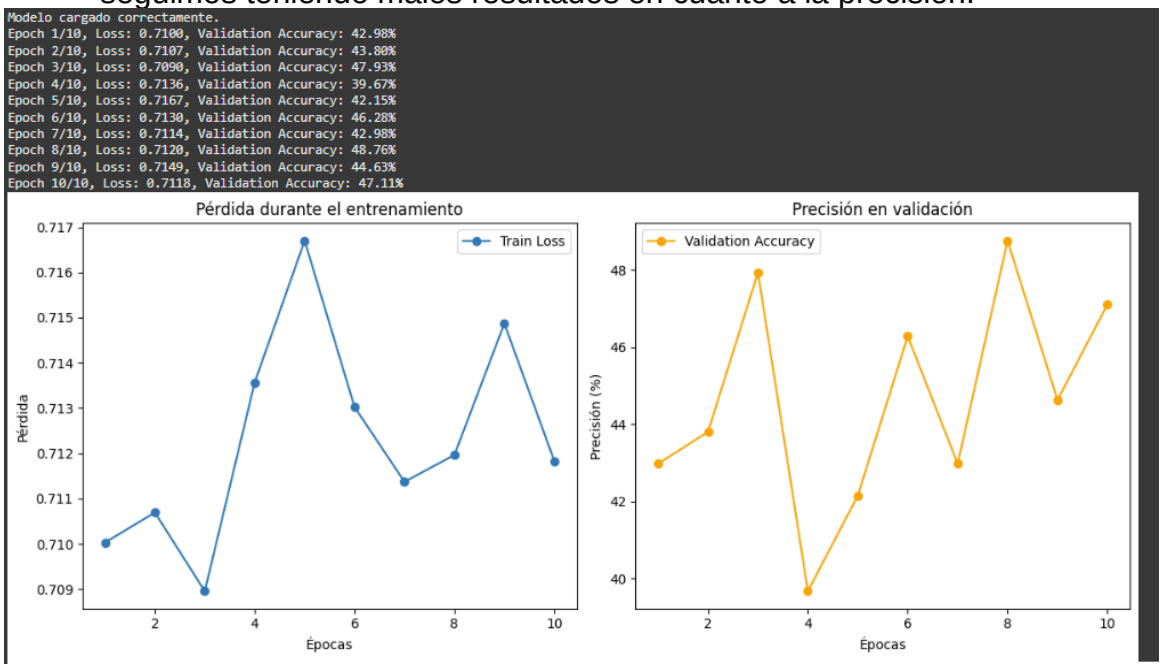


Figura 20 - Tercer experimento resultados 1.4

Resultados de precisión y pérdida:

Luego de balancear los datos y realizar la aumentación de datos, el entrenamiento del modelo se llevó a cabo durante 10 épocas y los resultados

obtenidos se presentan en términos de la pérdida de entrenamiento y la precisión en validación.

Pérdida durante el entrenamiento: La pérdida de entrenamiento osciló entre 0.7100 y 0.7149 a lo largo de las 10 épocas. Se observa cierta fluctuación, lo que podría ser indicativo de la necesidad de un ajuste más fino en los hiperparámetros o un dataset más representativo. Sin embargo, el valor de la pérdida se mantuvo relativamente estable, mostrando que el modelo logró cierta convergencia.

Precisión en validación: La precisión en el conjunto de validación inició en un 42.98% y mostró una tendencia variable, alcanzando su valor máximo de 47.11% en la última época. Aunque el aumento en la precisión no es significativo, sí refleja una mejora gradual en la capacidad del modelo para generalizar los datos no vistos.

Interpretación de los resultados

1. **Estabilidad del entrenamiento:** La estabilidad en la pérdida sugiere que el modelo no experimentó problemas graves de divergencia.
2. **Variabilidad en validación:** Las oscilaciones en la precisión de validación indican que el modelo podría beneficiarse de un conjunto de datos más amplio o de técnicas avanzadas de data augmentation para mejorar su capacidad de generalización.
3. **Precisión limitada:** La precisión obtenida está por debajo de lo esperado para tareas similares. Esto resalta la importancia de explorar modelos más avanzados, como MobileViT V2, que podrían manejar mejor las características del dataset utilizado.

La matriz de confusión: Esta proporciona un análisis detallado del rendimiento del modelo en términos de clasificaciones correctas e incorrectas para las clases ‘no_text’ y ‘text’. Aquí está el desglose de los resultados mostrados:

```python
# Se usa una matriz de confusión para visualizar cómo el modelo
# clasifica las imágenes

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Matriz de confusión
cm = confusion_matrix(all_labels, all_predictions)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=classes)
disp.plot(cmap=plt.cm.Blues)
plt.title("Matriz de Confusión")
plt.show()
```

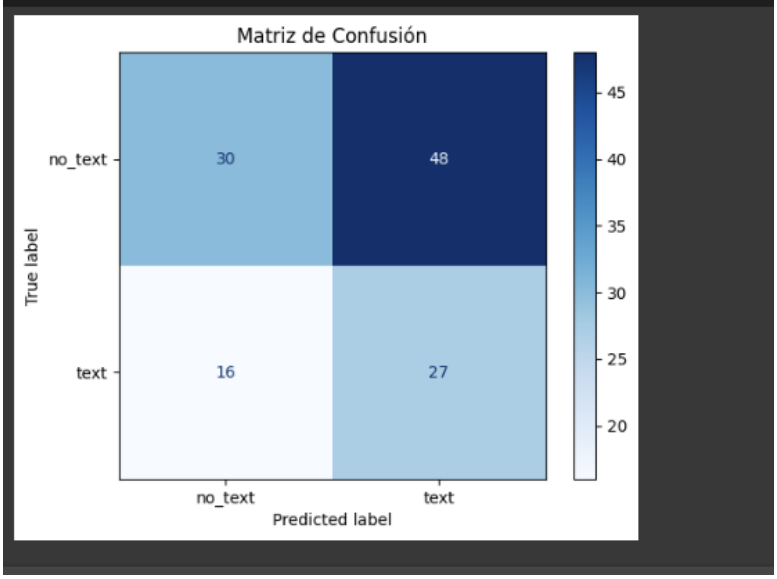


Figura 21 - Tercer experimento Matriz de confusión

Interpretación de la Matriz de Confusión
Ejes de la matriz:
El eje vertical representa las etiquetas reales del conjunto de validación.
El eje horizontal representa las predicciones del modelo.

Valores en la matriz:
(30): Cantidad de imágenes correctamente clasificadas como no_text.
(27): Cantidad de imágenes correctamente clasificadas como text.
(48): Cantidad de imágenes clasificadas incorrectamente como text cuando eran no_text.
(16): Cantidad de imágenes clasificadas incorrectamente como no_text cuando eran text.

Análisis del Modelo: El modelo muestra un bajo rendimiento, ya que hay muchas clasificaciones incorrectas, especialmente para la clase no_text.

Errores predominantes: El modelo confunde frecuentemente imágenes de no_text como text (48 casos). Esto podría deberse a similitudes visuales entre las clases y falta de información distintiva en las características de la clase no_text.

Clasificaciones correctas, **57 imágenes clasificadas correctamente**: no_text 30 correctas y text 27 correctas.

Tasa de error:

- **Errores totales**: 48 (falsos positivos) + 16 (falsos negativos) = **64 errores**.
- **Precisión del modelo**: correctas/total = 57/121 ≈ 47.1%
- Esto sugiere que el modelo tiene dificultades para generalizar.

3.4.4. Cuarto experimento con el modelo Mobile ViT y el dataset text_recognition, optimizando la normalizacion:

En este experimento, se optimizó el preprocesamiento según las recomendaciones de normalización del modelo preentrenado:

- Se realiza la carga de datos, con el objetivo de optimizar el preprocesamiento según las recomendaciones del modelo preentrenado, para obtener mayor compatibilidad con MobileViT:
    - Transformaciones de las imágenes:
        - Normalize: MobileViT fue preentrenado en ImageNet, con lo que usar las estadísticas de este conjunto mejora la compatibilidad y el rendimiento del modelo.
            - Media: [0.485, 0.456, 0.406] (promedio de valores RGB).
            - Desviación estándar: [0.229, 0.224, 0.225].

El modelo ya ha aprendido representaciones relevantes utilizando imágenes normalizadas con las estadísticas de ImageNet y aplicar estas mismas estadísticas asegura que las nuevas imágenes procesadas tengan distribuciones similares a las vistas durante el preentrenamiento, lo que reduce discrepancias en las representaciones internas del modelo.

Antes no se utilizó la normalización basada en ImageNet que estamos utilizando en esta fase y en cambió se optó por (0.5,0.5) para normalizar al rango [-1, 1] y por este motivo pudieron surgir los siguientes problemas:

- Desempeño degradado por no estar optimizados los pesos preentrenados para los datos de entrada.
- Necesidad de más tiempo para ajustar las capas iniciales para adaptarse a la nueva distribución de datos.
- Las métricas obtenidas pueden no ser comparables con estudios anteriores debido a las diferencias en la normalización, obteniendo resultados inconsistentes.

En este cuarto experimento se ajusta la normalizacion de los datos, utilizando los de Imagenet, ejecutamos el entrenamiento por 10 epocas y obtenemos los siguientes resultados:

- Perdida: 0.0029
- Precisión: 100.00%

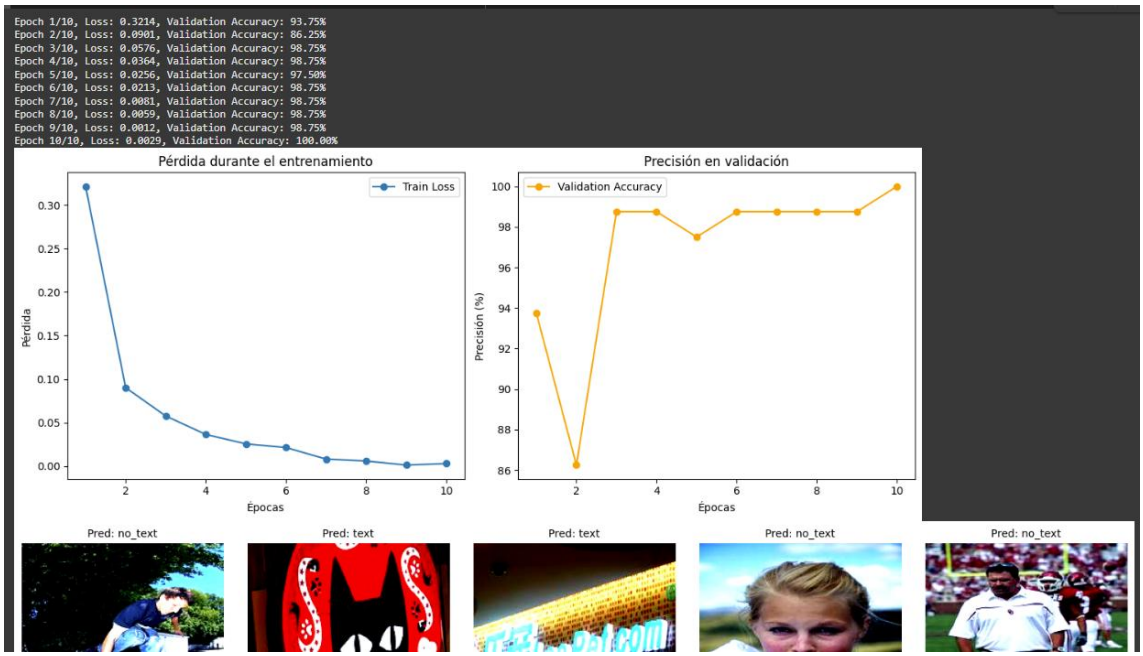


Figura 22 - Cuarto experimento resultados 1

- Se carga el modelo que acabamos de entrenar y guardar con el nombre: mobilevit_4to_exp_v1.pth y obtenemos resultados de Pérdidas de entrenamiento de 0% y precisión en validación de 100.00% lo que nos sigue dando pésimos resultados y nuevamente sugiere que:
    - El modelo está sobre ajustado.
    - No hay suficiente variación y estamos teniendo grandes limitaciones con el conjunto de datos.
    - En este caso ya, estaban balanceadas las clases, pero sin suficientes datos, no podemos obtener buenos resultados en estas redes neuronales Transformers.

4. Resultados

Los resultados obtenidos en esta investigación demuestran que la estrategia de preprocesamiento, como el balanceo de datos, la normalización con base en ImageNet y el data augmentation, ayudaron a estabilizar la pérdida del entrenamiento del modelo, aunque la precisión en validación alcanzada (47.11%) arrojó que el modelo no pudo generalizar correctamente, así como los tamaños reducidos y la baja variabilidad del dataset utilizado.

Otro hallazgo importante fue la sensibilidad del modelo al tamaño del dataset, resultado en un sobreajuste en escenarios de validación controlada, sin importar que los datos fueran balanceados.

5. Conclusiones y trabajos futuros

5.1. Conclusiones

El modelo ajustado de MobileViT ha mostrado un bajo rendimiento, con un número significativo de errores en las predicciones en el conjunto de validación, aunque se trabajó en el balance del dataset, se aplicaron técnicas de aumento de datos y se ajustaron hiperparámetros como la tasa de aprendizaje con el objetivo de mejorar la capacidad del modelo para generalizar y reducir estos errores, pero no conseguimos buenos resultados

Personalmente también puedo concluir que, este trabajo no solo ha contribuido a mi desarrollo profesional, sino que también ha establecido una base sólida para la clasificación de imágenes mediante mecanismos de Inteligencia Artificial y estoy emocionada por las posibilidades futuras y por seguir explorando este fascinante campo.

5.2. Trabajos Futuros

Para las futuras iteraciones del proyecto, sería interesante probar otros modelos preentrenados, en este caso sugiero el modelo MobileViT_V2, propuesto por Mehta & Rastegari (2022b) en "Separable Self-Attention for Mobile Vision Transformers" y que se puede extraer directamente del repositorio original de la publicacion de Mehta & Rastegari: apple/mlcvnets(**mobilevit_v2.py**).

En esta investigación hemos realizado una implementación de un modelo derivado (qai_hub_models[mobile-vit]) del original del repositorio de apple/ml-cvnets (**mobilevit.py**), por lo que propongo utilizar en futuras iteraciones el original con la finalidad de aprovechar las 21K clases del dataset con el que ha sido entrenado y a partir de allí realizar la adaptación al modelo deseado, según los resultados de precisión y eficiencia al implementarlo. En este sentido supone una ventaja sobre MobileViT_V1 que ha sido entrenado en ImagenNet-1K.

En la documentación oficial del modelo MobileViTv2(Mehta & Rastegari, 2022b), se destacan las siguientes mejoras:

- **Precisión mejorada:** MobileViT_V2 logra métricas superiores en clasificación de imágenes, superando a su versión anterior (MobileViT) en datasets como ImageNet, con una mejora en precision del 1%(Mehta & Rastegari, 2022b).

- **Eficiencia computacional:** Diseñado con atención separable, reduce significativamente el costo computacional y el tamaño del

modelo, haciéndolo ideal para implementaciones en dispositivos móviles con recursos limitados. Ejecutándose 3,2 veces más rápido en dispositivos móviles (Mehta & Rastegari, 2022b).

- **Base de entrenamiento en ImageNet-21k:** Al estar preentrenado en ImageNet-21k, MobileViT_V2 aprovecha un conjunto de datos mucho más amplio y diverso, lo que le permite capturar representaciones más generales y transferibles a tareas específicas como la detección de texto en imágenes.

- **Compatibilidad para transfer learning:** La arquitectura de MobileViT V2 es altamente adaptable a nuevas tareas mediante el ajuste de sus capas finales, permitiendo una implementación eficiente de sistemas personalizados.

5. Glosario

- Clasificación de Imágenes: Proceso de asignar etiquetas o categorías a imágenes basándose en su contenido visual. Es una tarea fundamental en la visión por computadora.

- Transfer Learning: Técnica de aprendizaje automático que permite utilizar un modelo preentrenado en una tarea diferente, adaptándolo a un nuevo conjunto de datos con menos recursos computacionales.

- MobileViT: Modelo de visión por computadora que combina arquitecturas de redes neuronales convolucionales (CNN) y transformadores, diseñado para ser eficiente en dispositivos móviles con recursos limitados.

- Fine Tuning: Proceso de ajustar un modelo preentrenado para mejorar su rendimiento en una tarea específica, modificando sus capas finales y reentrenándolo con un nuevo conjunto de datos.

- Dataset: Conjunto de datos utilizado para entrenar y evaluar modelos de aprendizaje automático. Puede incluir imágenes, texto, o cualquier otro tipo de información.

- Matriz de Confusión: Herramienta utilizada para evaluar el rendimiento de un modelo de clasificación, mostrando el número de clasificaciones correctas e incorrectas para cada clase.

- Aumento de Datos (Data Augmentation): Técnicas utilizadas para aumentar la diversidad del conjunto de datos de entrenamiento mediante transformaciones como rotaciones, recortes y cambios de color.

- Preprocesamiento de Datos: Conjunto de técnicas aplicadas a los datos antes de ser utilizados en el entrenamiento de un modelo, que incluye normalización, redimensionamiento y organización de datos.

- CNN (Redes Neuronales Convolucionales): Tipo de red neuronal diseñada para procesar datos con una estructura de cuadrícula, como imágenes, utilizando capas convolucionales para extraer características.

- DNN (Redes Neuronales Profundas): Redes neuronales con múltiples capas ocultas que permiten aprender representaciones complejas de los datos.

- Google Colab: Plataforma en la nube que permite ejecutar código Python y realizar experimentos de aprendizaje automático sin necesidad de configuraciones locales complejas.

- Precisión: Métrica que indica la proporción de clasificaciones correctas realizadas por un modelo en relación con el total de clasificaciones realizadas.

6. Bibliografía

DNN Machine Learning—Javatpoint. (s. f.). Www.Javatpoint.Com. Recuperado 11 de enero de 2025, de https://www.javatpoint.com/dnn-machine-learning

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. (2021). *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale* (No. arXiv:2010.11929). arXiv. https://doi.org/10.48550/arXiv.2010.11929

Garg, S., Harichandana, & Kumar, S. (2021, enero 6). *On-Device Document Classification using multimodal features*. arXiv.Org. https://doi.org/10.1145/3430984.3431030

Google Colab. (s. f.). Recuperado 18 de noviembre de 2024, de https://colab.research.google.com/

Hodosh, M., Young, P., & Hockenmaier, J. (2013). Framing Image Description as a Ranking Task: Data, Models and Evaluation Metrics. *Journal of Artificial Intelligence Research*, *47*, 853-899. https://doi.org/10.1613/jair.3994

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications* (No. arXiv:1704.04861). arXiv. https://doi.org/10.48550/arXiv.1704.04861

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural*

Information Processing Systems, *25*. https://proceedings.neurips.cc/paper_files/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html

Kwon, J., & Lee, K. M. (2012a). A unified framework for event summarization and rare event detection. *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 1266-1273. https://doi.org/10.1109/CVPR.2012.6247810

Kwon, J., & Lee, K. M. (2012b). A unified framework for event summarization and rare event detection. *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 1266-1273. https://doi.org/10.1109/CVPR.2012.6247810

Le, Y., & Yang, X. (2015). *Stanford University CS231n: Deep Learning for Computer Vision*. Tiny ImageNet Visual Recognition Challenge. CS231N Course Project, Stanford University. https://cs231n.stanford.edu/

Mehta, S., & Rastegari, M. (2022a). *MobileViT: Light-weight, General-purpose, and Mobile-friendly Vision Transformer* (No. arXiv:2110.02178). arXiv. https://doi.org/10.48550/arXiv.2110.02178

Mehta, S., & Rastegari, M. (2022b). *Separable Self-attention for Mobile Vision Transformers* (No. arXiv:2206.02680). arXiv. https://doi.org/10.48550/arXiv.2206.02680

MSRA Text Detection 500 Database (MSRA-TD500)—TC11. (s. f.). Recuperado 9 de enero de 2025, de http://www.iapr-tc11.org/mediawiki/index.php?title=MSRA_Text_Detection_500_Database_(MSRA-TD500)

PyTorch documentation—PyTorch 2.5 documentation. (s. f.). Recuperado 12 de enero de 2025, de https://pytorch.org/docs/stable/index.html

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., & Sutskever, I. (2021). *Learning Transferable Visual Models From Natural Language Supervision* (No. arXiv:2103.00020). arXiv. https://doi.org/10.48550/arXiv.2103.00020

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2019). *MobileNetV2: Inverted Residuals and Linear Bottlenecks* (No. arXiv:1801.04381). arXiv. https://doi.org/10.48550/arXiv.1801.04381

torchvision—Torchvision 0.20 documentation. (s. f.). Recuperado 12 de enero de 2025, de https://pytorch.org/vision/stable/index.html

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2023). *Attention Is All You Need* (No. arXiv:1706.03762). arXiv. http://arxiv.org/abs/1706.03762

7. Anexos

Enlaces al Repositorio de Github donde estará publicado el código fuente y la documentación de este proyecto:

https://github.com/eleanarey/TFM-MUII-UOC