Computación de Altas Prestaciones (2025-2026)

Análisis comparativo del entrenamiento de modelos de Machine Learning en CPU y GPU

Realizado por: Javier Agüero y Marouane Chahbar

1. Introducción

En esta práctica se analiza el uso de CPU y GPU para el entrenamiento de modelos de Machine Learning, con el objetivo de estudiar en qué situaciones el uso de una GPU proporciona una mejora real de rendimiento frente a la ejecución en CPU. Para ello, se han realizado dos experimentos diferenciados, uno con un problema de tamaño reducido y otro con un problema de mayor complejidad, comparando en ambos casos tiempos de ejecución y resultados obtenidos.

La parte de CPU se ha ejecutado en el cluster de la UAM, mientras que la parte de GPU se ha llevado a cabo utilizando Google Colab, que proporciona acceso a una GPU NVIDIA Tesla T4. De esta forma se reproduce un escenario realista de computación de altas prestaciones, donde se combinan recursos de computación tradicionales y aceleradores especializados.

2. Entornos de ejecución

2.1 Entorno CPU (cluster UAM)

Los experimentos en CPU se han ejecutado en el cluster proporcionado por la UAM. En este entorno se ha utilizado Python con PyTorch en modo CPU, creando un entorno virtual específico para garantizar la reproducibilidad de los resultados.

- Python 3.9
- PyTorch (versión CPU)
- Ejecución sin aceleración por GPU

Este entorno permite evaluar el rendimiento de los modelos utilizando únicamente recursos de cómputo generalistas.

2.2 Entorno GPU (Google Colab)

Para la ejecución en GPU se ha utilizado Google Colab, conectándose a un entorno con GPU NVIDIA Tesla T4.

- GPU: NVIDIA Tesla T4 (16 GB)
- CUDA disponible en el entorno
- PyTorch con soporte CUDA

```
Python 3.12.12
Mon Dec 15 17:26:04 2025
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 550.54.15              Driver Version: 550.54.15      CUDA Version: 12.4     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  Tesla T4                       Off |   00000000:00:04.0 Off |                    0 |
| N/A   47C    P0             26W /   70W |     158MiB /  15360MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=======================================================================================|
+---------------------------------------------------------------------------------------+
Device: cuda
```

Este entorno permite ejecutar exactamente el mismo código que en CPU, pero aprovechando el paralelismo masivo de la GPU.

3. Experimento 1: Clasificación con Digits y MLP

3.1 Descripción del experimento

En el primer experimento se ha utilizado el dataset Digits, compuesto por imágenes de 8x8 píxeles correspondientes a dígitos manuscritos. Se ha entrenado una red neuronal MLP sencilla con las siguientes características:

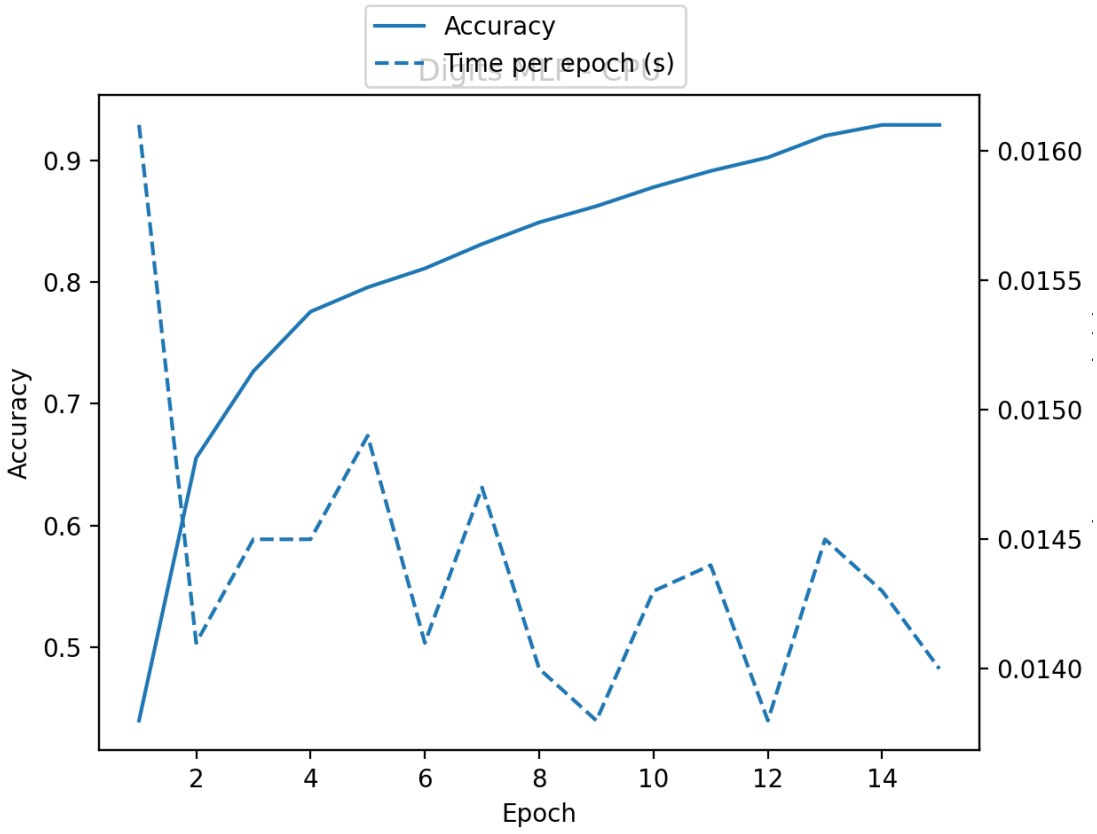
- Entrada: 64 características
- Capa oculta: 128 neuronas con activación ReLU
- Salida: 10 clases
- Optimizador: Adam
- Función de pérdida: CrossEntropyLoss
- Número de épocas: 15

Este experimento representa un problema pequeño, tanto en tamaño de datos como en complejidad del modelo.

3.2 Resultados en CPU

En el cluster UAM, el entrenamiento en CPU obtuvo los siguientes resultados:

- Tiempo medio por época: 0.014392 s
- Mejor accuracy en test: 0.928889



La gráfica muestra la evolución de la accuracy y del tiempo por época durante el entrenamiento del modelo MLP en CPU. Desde el punto de vista teórico, este comportamiento es normal en un problema de tamaño reducido, tanto en volumen de datos como en complejidad computacional del modelo.

El tiempo por época se mantiene prácticamente constante a lo largo del entrenamiento, lo que indica que:

- El coste computacional por época está dominado por operaciones matriciales pequeñas.
- Estas operaciones encajan bien en la jerarquía de memoria y en la caché de la CPU.
- No existen fases del entrenamiento que introduzcan sobrecostes significativos.

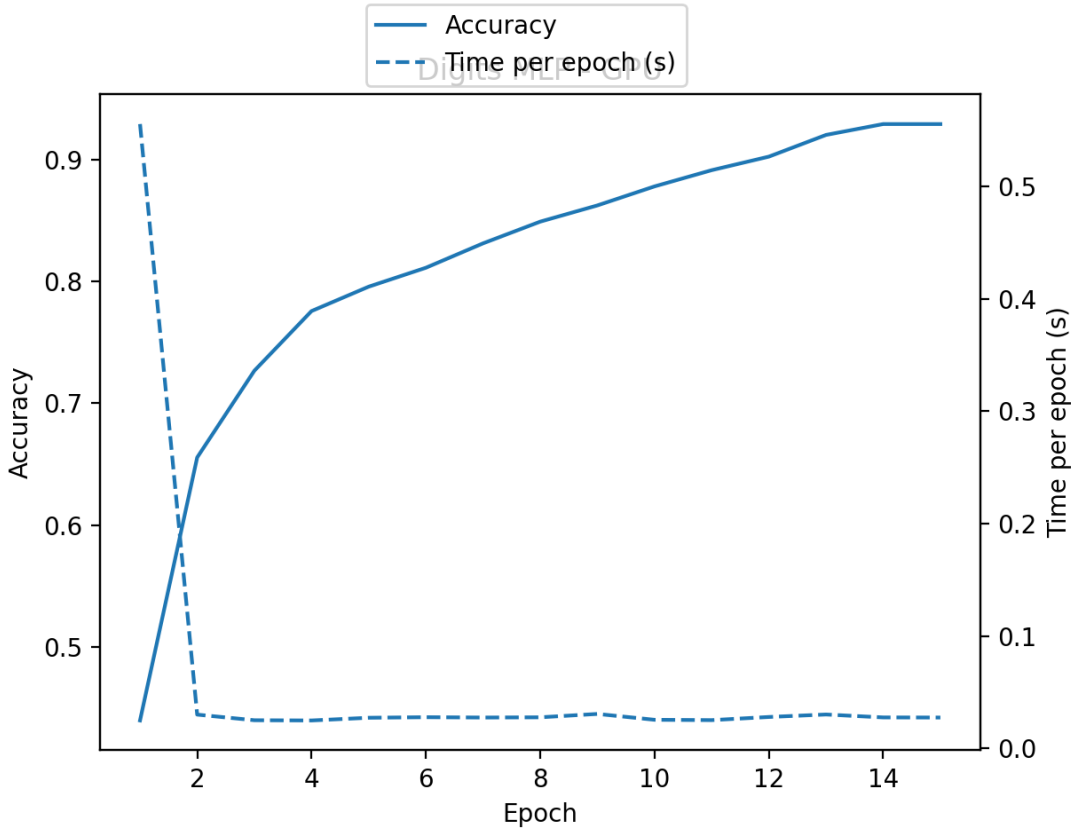
En cuanto a la accuracy, se observa una mejora progresiva que se estabiliza tras varias épocas. Este comportamiento es típico de modelos sencillos entrenados sobre datasets pequeños, donde la convergencia se alcanza rápidamente y no se requiere un gran número de iteraciones.

Esta gráfica ilustra que, para cargas pequeñas, una CPU moderna puede ofrecer un rendimiento muy elevado sin necesidad de paralelismo masivo.

3.3 Resultados en GPU

En Google Colab, ejecutando el mismo modelo en GPU, se obtuvieron los siguientes resultados:

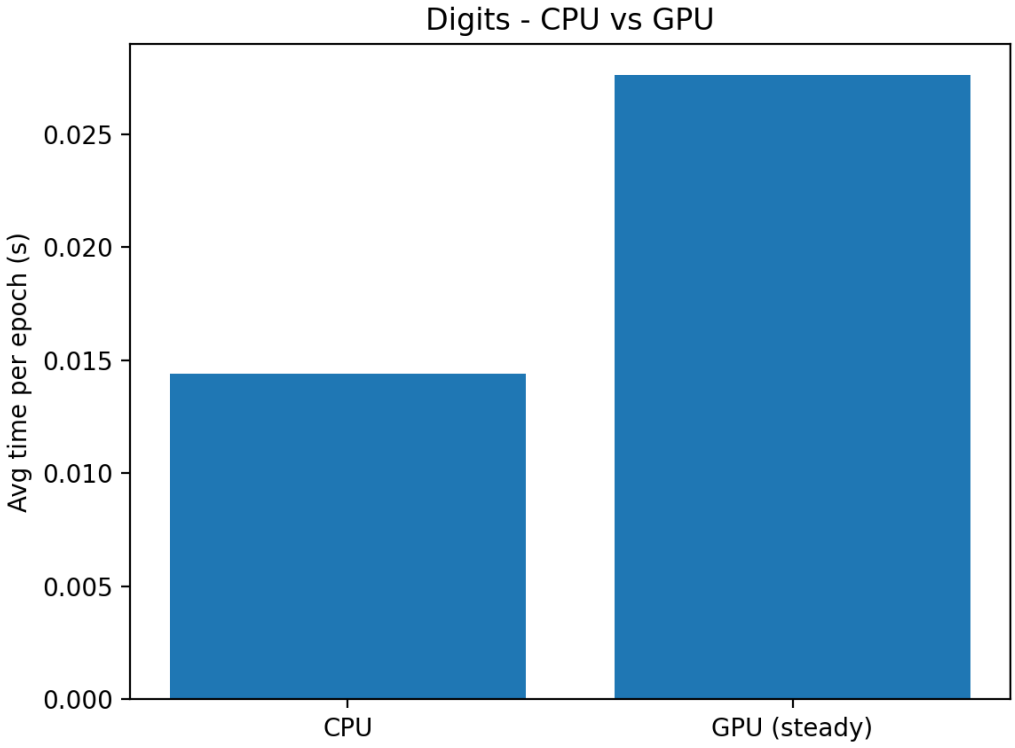
- Tiempo medio por época (incluyendo warmup): 0.062787 s
- Tiempo medio por época en régimen estable (épocas 2 a 15): 0.027607 s
- Mejor accuracy en test: 0.928889



Si observamos la gráfica del entrenamiento ejecutado en GPU, lo más llamativo desde una perspectiva teórica es el comportamiento del tiempo, sobre todo durante la primera iteración. Es completamente normal notar que esa primera época tarda bastante más, ya que es el momento en el que el sistema tiene que realizar todo el trabajo de preparación: desde inicializar el contexto CUDA y reservar la memoria gráfica, hasta transferir los datos iniciales y compilar los kernels necesarios.

Una vez superada esa fase de arranque, los tiempos se estabilizan reflejando el ritmo natural de la GPU, aunque es interesante notar que sigue siendo más lenta que la CPU. Esto ocurre simplemente porque el volumen de trabajo es demasiado pequeño; los costes fijos de gestión acaban pesando más que la potencia de cálculo, por lo que no se llega a aprovechar realmente el paralelismo masivo de la tarjeta. Afortunadamente, la evolución de la precisión es idéntica en ambos casos, lo que nos confirma que el modelo está aprendiendo exactamente igual y que las diferencias son puramente una cuestión de rendimiento computacional.

3.4 Análisis comparativo



Esta gráfica resume el comportamiento de la CPU y la GPU para el experimento Digits y resulta especialmente ilustrativa ya que el hecho de que la GPU no logre mejorar los tiempos de la CPU pone de manifiesto un principio fundamental: la aceleración gráfica solo es realmente efectiva cuando nos enfrentamos a problemas que presentan suficiente carga computacional y paralelismo.

En este caso concreto, nos encontramos con un dataset reducido y un modelo que requiere pocas operaciones por muestra, lo que permite que la CPU resuelva todo el cálculo de forma muy eficiente y sin incurrir en costes adicionales. Por tanto, estos datos confirman que el uso de la GPU no es beneficioso en todos los escenarios y nos recuerdan que una elección incorrecta del hardware puede, de hecho, degradar el rendimiento en lugar de mejorarlo.

4. Experimento 2: Clasificación con MNIST y CNN

4.1 Descripción del experimento

En el segundo experimento se ha utilizado el dataset MNIST, compuesto por imágenes de 28x28 píxeles. En este caso se ha entrenado una red neuronal convolucional sencilla (SmallCNN) con la siguiente estructura:

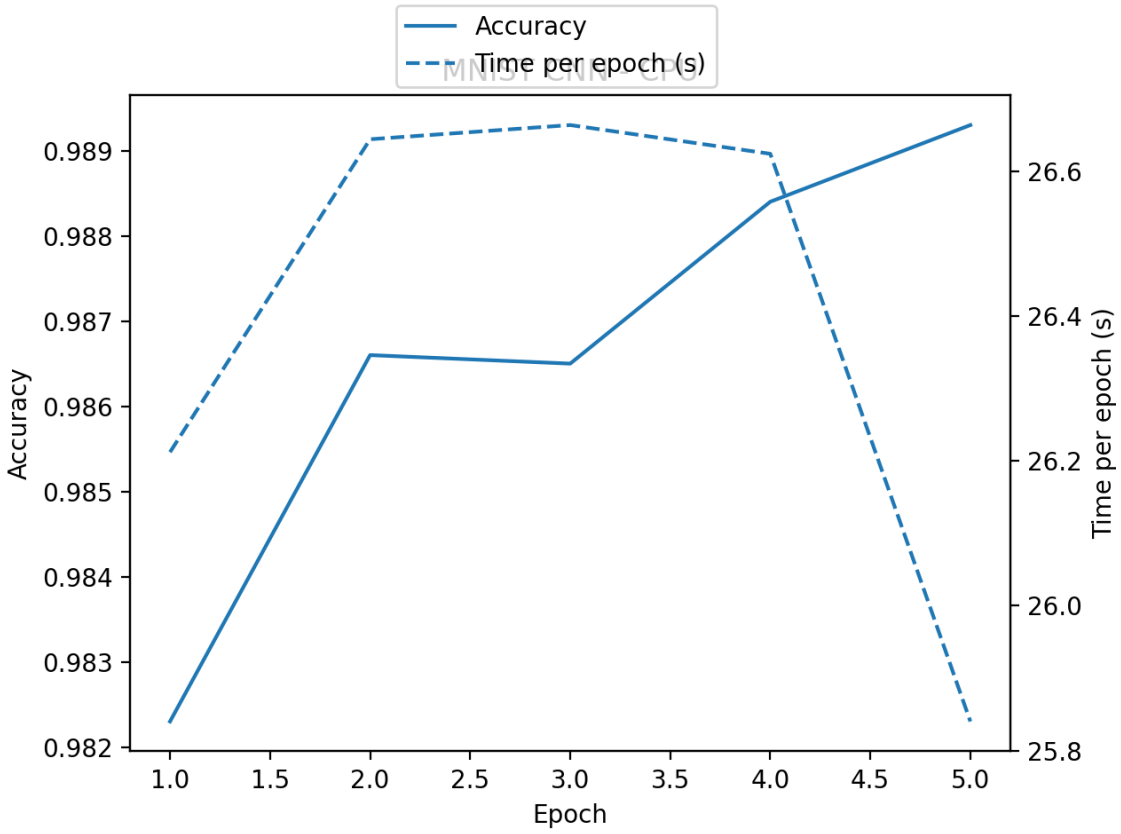
- Dos capas convolucionales
- Capa de pooling
- Dos capas completamente conectadas
- Optimizador: Adam
- Función de pérdida: CrossEntropyLoss
- Número de épocas: 5

Este experimento representa un problema significativamente más grande y computacionalmente más costoso que el anterior.

4.2 Resultados en CPU

En el cluster, el entrenamiento en CPU produjo los siguientes resultados:

- Tiempo medio por época: 26.397161 s
- Mejor accuracy en test: 0.9893



Al analizar esta gráfica, se percibe un cambio drástico respecto al experimento anterior. El tiempo por época ha crecido considerablemente, algo lógico si tenemos en cuenta que ahora la carga computacional es mucho mayor debido al tamaño del dataset y, sobre

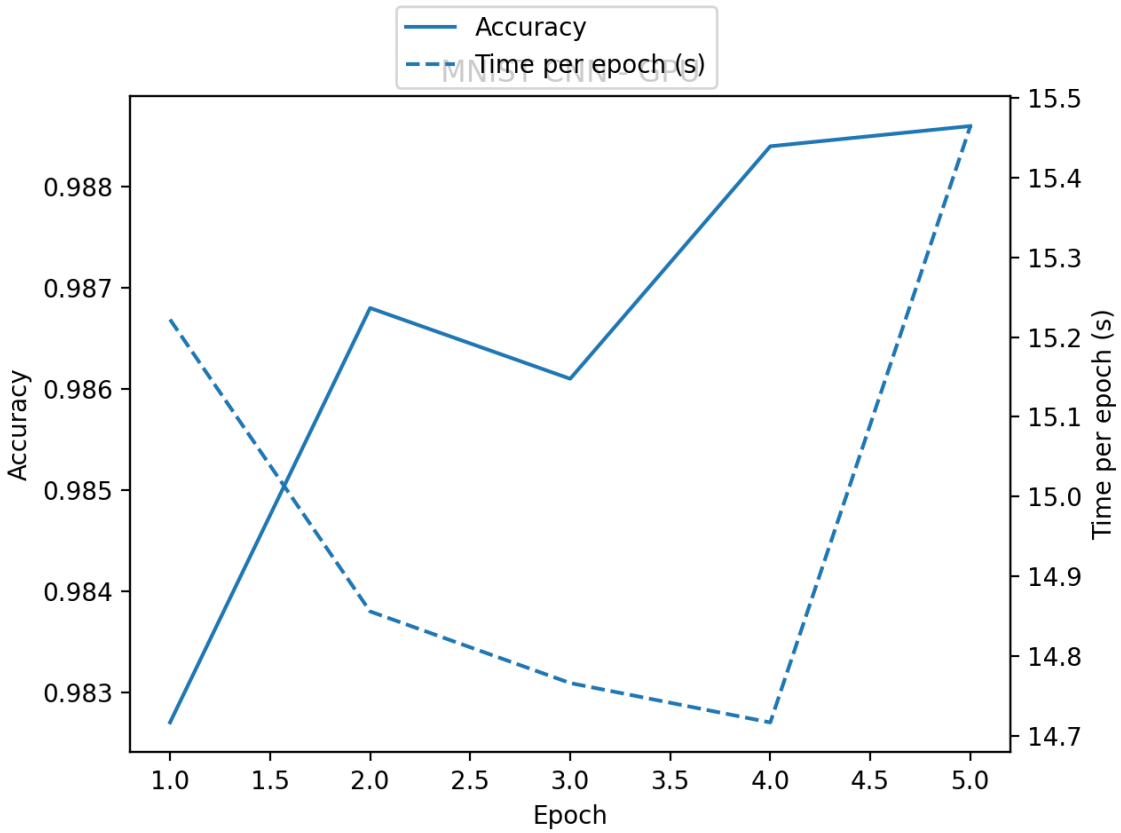
todo, a la introducción de capas convolucionales que disparan el número de operaciones aritméticas por muestra.

El problema reside en que las convoluciones requieren un volumen masivo de operaciones repetitivas. Si bien la CPU es perfectamente capaz de ejecutarlas, su diseño orientado más a procesos secuenciales o con un paralelismo limitado hace que el coste total del entrenamiento aumente de forma notable. Afortunadamente, la evolución de la precisión muestra una convergencia rápida hacia valores altos, lo que confirma que el modelo es el adecuado para el problema; sin embargo, esta gráfica nos sirve para ilustrar claramente dónde está el límite práctico de utilizar exclusivamente una CPU cuando nos enfrentamos a modelos de mayor complejidad.

4.3 Resultados en GPU

En Google Colab, ejecutando el mismo modelo sobre GPU, se obtuvieron los siguientes resultados:

- Tiempo medio por época: 15.005147 s
- Mejor accuracy en test: 0.9886

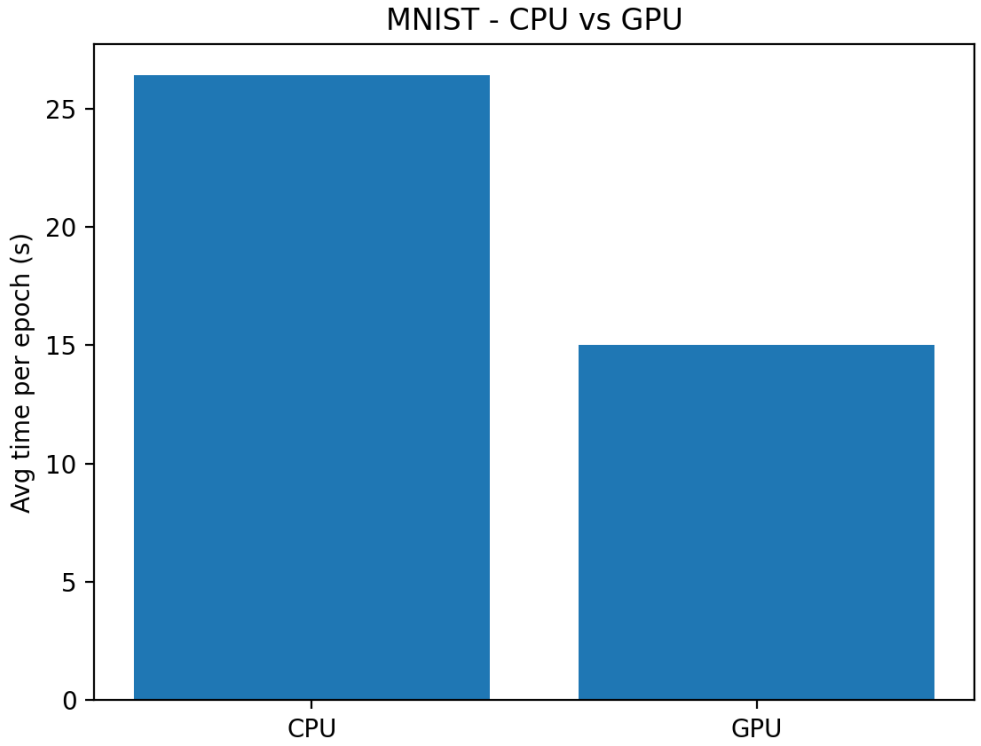


En este escenario, la ejecución en GPU marca una diferencia clara, reduciendo significativamente el tiempo por época en comparación con la CPU. Desde un punto de vista teórico, este resultado es totalmente esperable, ya que las operaciones

convolucionales son ideales para ser paralelizadas y permiten que la GPU despliegue toda su potencia ejecutando miles de hilos de forma simultánea.

Lo crucial aquí, a diferencia de lo que vimos en el experimento Digits, es que el tamaño del problema es lo suficientemente grande como para amortizar los costes fijos de la tarjeta gráfica. En este contexto, los tiempos de inicialización y transferencia de datos se diluyen y pasan a un segundo plano frente al enorme volumen total de cómputo, permitiendo que el paralelismo masivo se traduzca, esta vez sí, en una mejora real del rendimiento. Además, la precisión final es comparable a la obtenida en CPU, lo que nos confirma que haber acelerado el proceso no ha tenido ningún impacto negativo en la calidad del aprendizaje del modelo.

4.4 Análisis comparativo



El speedup obtenido al usar GPU frente a CPU es aproximadamente: 1.759x

Esta comparación final deja patente el claro beneficio de utilizar la GPU en este experimento. El speedup obtenido no es casualidad, refleja que el problema tiene por fin el tamaño y la complejidad suficientes para que el paralelismo de la GPU se explote de manera efectiva, logrando que la aceleración en el propio cálculo compense con creces los costes fijos de gestión.

Esto nos ayuda a representar el escenario ideal para el uso de aceleradores: nos encontramos ante problemas intensivos en cómputo, con operaciones repetitivas y

altamente paralelizables. Además, la similitud en la precisión final entre la CPU y la GPU refuerza la validez de todo el análisis, demostrando que la mejora observada se debe estrictamente a un aumento del rendimiento y no a diferencias en el aprendizaje del modelo.

5. Discusión global

Al poner en perspectiva los resultados de ambos experimentos, llegamos a una conclusión fundamental: el uso de una GPU no es una garantía automática de mejora en el rendimiento. En escenarios más sencillos, como el caso de Digits con una red MLP, la CPU demuestra ser más eficiente precisamente porque se libra de los costes de gestión o overheads que penalizan a la gráfica en tareas ligeras.

Sin embargo, la situación cambia radicalmente cuando aumentamos la escala y la complejidad del problema, como vimos con la red CNN para MNIST. Es ahí donde la GPU logra amortizar sus costes iniciales y ofrecer una aceleración clara. Dado que la precisión se mantiene estable en ambos hardware, validando la comparativa de tiempos, estos resultados confirman un principio básico de la computación de altas prestaciones: el paralelismo masivo de la GPU solo resulta verdaderamente ventajoso cuando existe una carga de trabajo suficiente que lo justifique.

6. Conclusiones

En este proyecto se ha demostrado de forma experimental que la elección entre CPU y GPU debe hacerse en función del tamaño y la complejidad del problema. Mientras que la CPU es adecuada para tareas pequeñas y ligeras, la GPU se convierte en la opción óptima cuando el volumen de cálculo es elevado.

El uso combinado del cluster de la UAM y Google Colab ha permitido reproducir un entorno realista de computación de altas prestaciones, analizando de forma crítica cuándo y por qué el uso de GPU aporta ventajas reales.