Universitat d'Alacant
Universidad de Alicante

SESIÓN 3 DE PRÁCTICAS: COMPARACIÓN CPU VS GPU (WINDOWS)

PRÁCTICA GUIADA: OPTIMIZACIÓN DEL RECONOCIMIENTO FACIAL CON GPUS

1. INTRODUCCIÓN

En esta práctica, aprenderás a comparar el rendimiento (Benchmarking) entre una **CPU** y una **GPU** para una tarea básica de reconocimiento de imágenes. El objetivo es entender cómo la elección del hardware puede afectar el tiempo de procesamiento y cómo este conocimiento puede ser aplicado en proyectos reales.

2. OBJETIVO GENERAL

- Comparar los tiempos de ejecución en CPU y GPU para una tarea intensiva en el procesamiento de imágenes.
- Reflexionar sobre cómo esta diferencia de rendimiento puede influir en aplicaciones del mundo real, como el reconocimiento facial en situaciones críticas.

3. OBJETIVOS ESPECÍFICOS

- Implementar una tarea de reconocimiento de imágenes usando PyTorch.
- Medir los tiempos de ejecución usando CPU y GPU.
- Analizar los resultados y discutir posibles aplicaciones prácticas.

ASPECTOS DE BENCHMARKING CUBIERTOS EN LA PRÁCTICA:

- **Medición de tiempos:** Se mide el tiempo de procesamiento en CPU y GPU utilizando un modelo de aprendizaje profundo (ResNet18).
- **Comparación de resultados:** Se registran y analizan los tiempos de ejecución para distintos tamaños de imágenes.
- **Reflexión sobre el hardware:** Se evalúa cómo la elección entre CPU y GPU afecta el rendimiento y se discuten optimizaciones posibles.

RELACIÓN CON EL CONCEPTO DE BENCHMARKING:

- **Benchmarking interno:** Se compara el rendimiento entre dos configuraciones del mismo sistema (CPU y GPU).
- **Optimización:** La práctica incluye propuestas de mejora del rendimiento, como reducir el tamaño de las imágenes o usar modelos más ligeros.

Asignatura: computación de alto rendimiento
Proferor: Ricardo Moreno
Email: ricardo.moreno@ua.es

4. MATERIAL NECESARIO

- Un ordenador con **Python 3.8+** y las siguientes **bibliotecas** a instalar:
  - `torch`
  - `torchvision`
  - `pillow`
  - `matplotlib`
- Una **GPU compatible con CUDA** (opcional, la práctica puede realizarse con CPU, aunque es recomendable utilizar GPU).
- Un conjunto de imágenes simples (preparado previamente o descargado durante la práctica).

NOTA: En los anexos encontrarás información adicional sobre la función y uso de éstas bibliotecas. En caso de disponer de otra versión de Python también se puede realizar la práctica, puedes adaptarla utilizando la IA o documentación oficial del programa.

5. PASOS DE LA PRÁCTICA

NOTA: los códigos que aquí se muestran son básicos para las pruebas, pero fácilmente mejorables con ia para conseguir el propósito buscado de medición. Forma parte del aprendizaje detectar éstas modificaciones hasta conseguir que el código sea efectivo

PASO 1: PREPARAR EL ENTORNO

1. Abre la terminal o tu editor de código (el que suelas utilizar habitualmente).
2. Si estás en un entorno virtual, actívalo.

```
# En la consola terminal que prefieras actívalo:
.\venv\Scripts\activate  # Windows
source venv/bin/activate  # Linux o Mac
```

3. Crea un archivo Python, por ejemplo, `practica_cpu_vs_gpu.py`.

PASO 2: CARGAR LAS BIBLIOTECAS

En el archivo Python, importa las bibliotecas necesarias:

```
# importa las bicliotecas siguientes en Python:
import torch
import time
from torchvision import models, transforms
from PIL import Image
import matplotlib.pyplot as plt
```

NOTA: En los anexos encontrarás información adicional sobre la función y uso de éstas bibliotecas. En caso de disponer de otra versión de Python también se puede realizar la práctica, puedes adaptarla utilizando la IA o documentación oficial del programa.

PASO 3: CARGAR UNA IMAGEN

Prepara una imagen de entrada para el modelo. Puedes usar una imagen local o una de ejemplo.

```
# Cargar una imagen de prueba en Python
image = Image.open("cara_ejemplo.jpg")

# Redimensionar y convertir la imagen a tensor
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor()
```

```
])

input_tensor = transform(image).unsqueeze(0)  # Añadir dimensión batch **
```

** En los anexos puedes encontrar más información sobre porqué hay que añadir una dimensión batch

PASO 4: CARGAR UN MODELO PREENTRENADO

Utiliza un modelo de clasificación de imágenes preentrenado, como **ResNet18**:

NOTA: En los Anexos encontrarás más documentación sobre el modelo preentrenado **ResNet18**

```
from torchvision import models
from torchvision.models import ResNet18_Weights

# Cargar el modelo ResNet18 con los pesos actualizados
model = models.resnet18(weights=ResNet18_Weights.IMAGENET1K_V1)
model.eval()  # Modo de evaluación
```

PASO 5: MEDIR EL TIEMPO DE EJECUCIÓN EN CPU

Calcula el tiempo que toma el modelo para procesar la imagen usando la CPU:

```
import torch
import time
from torchvision import models
from torchvision.models import ResNet18_Weights

# Cargar el modelo ResNet18 preentrenado
model = models.resnet18(weights=ResNet18_Weights.IMAGENET1K_V1)
model.eval()  # Modo de evaluación

# Ejemplo de input_tensor (asegúrate de definirlo correctamente en tu entorno)
input_tensor = torch.rand(1, 3, 224, 224)  # Imagen simulada para pruebas

# Medir el tiempo en CPU
start_cpu = time.time()

with torch.no_grad():
    output_cpu = model(input_tensor)

end_cpu = time.time()
print(f"Tiempo en CPU: {end_cpu - start_cpu:.4f} segundos")
```

Paso 6: Medir el tiempo de ejecución en GPU (si está disponible)

Comprueba si hay una GPU disponible y repite la operación:

```
import torch
import time
from torchvision import models
from torchvision.models import ResNet18_Weights

# Cargar el modelo ResNet18 preentrenado
model = models.resnet18(weights=ResNet18_Weights.IMAGENET1K_V1)
```

```
model.eval()  # Modo de evaluación

# Verificar si CUDA está disponible
if torch.cuda.is_available():
    # Mover el modelo a la GPU
    model = model.cuda()

    # Crear un tensor de ejemplo en la GPU (imagen de 224x224)
    input_tensor = torch.rand(1, 3, 224, 224).cuda()  # Imagen simulada en la GPU

    # Medir el tiempo en la GPU
    start_gpu = time.time()

    with torch.no_grad():
        output_gpu = model(input_tensor)

    # Sincronizar la GPU para garantizar que termine la ejecución
    torch.cuda.synchronize()
    end_gpu = time.time()

    print(f"Tiempo en GPU: {end_gpu - start_gpu:.4f} segundos")
else:
    print("GPU no disponible. Ejecutando solo en CPU.")
```

NOTA: El código ejemplo genera una matriz aleatoria (con torch.rand), pero podrías modificarlo para que tome como muestra una de las fotos que te proporciono (u otra aportada por ti haciendo las modificaciones pertinentes), el resultado que debes entregar en la práctica es el de una matriz que represente a una imágen aleatoria)

PASO 7: COMPARAR RESULTADOS

Registra los tiempos y comapra los resultados:

Tamaño de la imagen	Tiempo CPU (s)	Tiempo GPU (s)
224 x 224		
512 x 512		
1024 x 1024		

PASO 8: VISUALIZACIÓN (OPCIONAL)

Puedes graficar los resultados usando `matplotlib`:

```
# Ejemplo de datos simulados
tiempos_cpu = [1.2, 1.4, 1.8]
tiempos_gpu = [0.4, 0.5, 0.7]

# Crear el gráfico
plt.plot(tiempos_cpu, label='CPU', marker='o')
plt.plot(tiempos_gpu, label='GPU', marker='x')
plt.legend()
plt.xlabel('Tamaño de imagen')
plt.ylabel('Tiempo (s)')
plt.title('Comparación de tiempos CPU vs GPU')
plt.show()
```

NOTA: Recuerda que el código debe depurarse

6. ANÁLISIS Y DISCUSIÓN

Ahora debes reflexionar sobre:

- ¿Por qué la GPU es más rápida en este caso?
- ¿Qué pasa cuando aumentamos el tamaño de las imágenes?
- ¿En qué tipo de situaciones reales este conocimiento sería clave?

7. TAREAS ADICIONALES

1. Cambia la imagen por otras de diferentes tamaños y mide el impacto en los tiempos.
2. Investiga cómo se podría optimizar aún más el proceso (por ejemplo, usando modelos más ligeros o GPUs más potentes).

8. INFORME DE ENTREGA

El informe debe incluir:

REFLEXIÓN SOBRE LO APRENDIDO:

- Un breve resumen de la práctica.
- El análisis y discusión sobre cómo afectó la elección del hardware (CPU vs GPU) al tiempo de ejecución.
- Una explicación de los conceptos de paralelismo y por qué las GPUs son más eficientes en ciertas tareas.

TABLA DE RESULTADOS COMPARATIVOS:

- Incluye los tiempos de ejecución obtenidos para diferentes tamaños de imagen en CPU y GPU.

Ejemplo:

Tamaño de la imagen	Tiempo CPU (s)	Tiempo GPU (s)
224 x 224		
512 x 512		
1024 x 1024		

ANÁLISIS SOBRE LAS DIFERENCIAS DE RENDIMIENTO:

- Explica por qué los tiempos de la GPU son más rápidos, o por qué no lo son.
- Discute cómo afecta el tamaño de la imagen y la cantidad de datos procesados al rendimiento.

PROPUESTAS DE OPTIMIZACIÓN (NO ES NECESARIO IMPLEMENTARLAS):

- Plantea posibles mejoras en el tiempo de ejecución.
- Considera opciones como:
  - Usar modelos más ligeros (ej. MobileNet en lugar de ResNet).
  - Implementar paralelismo adicional (por ejemplo, procesamiento por lotes en GPU).
  - Optimizar la gestión de memoria en la GPU.
  - Etc..

Universitat d'Alacant
Universidad de Alicante

Rúbrica de Evaluación y Relación con Competencias y Resultados de Aprendizaje

Aspecto Evaluado	Ponderación	Criterios de Evaluación	Relación con Competencias y Resultados de Aprendizaje
1. Estructura del informe	10%	- El informe presenta una estructura clara y ordenada, con secciones diferenciadas. - La redacción es coherente y bien organizada.	- **CT02:** Comunicación escrita clara y organizada. - **CB2:** Demostración de competencias profesionales a través de la presentación de resultados.
2. Resultados experimentales	20%	- La tabla de resultados incluye tiempos de ejecución para diferentes tamaños de imagen en CPU y GPU (o CPU vs tiempos de referencia). - Resultados claros y precisos.	- **CE15:** Seleccionar y aplicar sistemas computacionales de altas prestaciones para acelerar la ejecución de métodos. - **Objetivo específico:** Conocer el impacto del uso de aceleradores (GPU).
3. Análisis de los resultados	30%	- Explicación detallada de las diferencias entre CPU y GPU. - Reflexión sobre cómo el paralelismo afecta al rendimiento. - Justificación técnica correcta.	- **CG1:** Análisis y diseño de soluciones eficientes. - **CE15:** Comprensión del rol de las GPUs como aceleradores. - **Resultado de aprendizaje 1:** Selección de arquitecturas paralelas adecuadas.
4. Propuestas de optimización	20%	- Propuestas originales y bien fundamentadas para optimizar el tiempo de ejecución. - Consideración de modelos ligeros, paralelismo adicional y gestión eficiente de la GPU.	- **CG4:** Obtención de soluciones óptimas aplicando principios de ingeniería. - **CG3:** Investigación de nuevas soluciones basadas en fuentes documentales. - **CE15:** Aplicación de sistemas computacionales avanzados.
5. Reflexión y aplicación práctica	20%	- Reflexión profunda sobre el aprendizaje adquirido y su posible aplicación en otros escenarios. - Relación con casos prácticos del mundo real (inteligencia artificial en situaciones críticas).	- **CT01:** Uso habitual de herramientas informáticas y tecnologías de la información. - **CG8:** Consideración del uso eficiente de recursos computacionales. - **Resultado de aprendizaje 5:** Conocimiento práctico del uso de GPUs.

Justificación del Sistema de Evaluación en Relación con Competencias y Objetivos

Asignatura: computación de alto rendimiento
Proferor: Ricardo Moreno
Email: ricardo.moreno@ua.es

Universitat d'Alacant
Universidad de Alicante

COMPETENCIAS TRABAJADAS A LO LARGO DE LA PRÁCTICA:

- **CT01 (Uso de herramientas informáticas):**
  La práctica exige el manejo de tecnologías específicas como PyTorch, GPU y transformaciones de imágenes, lo que fomenta el uso habitual de herramientas tecnológicas avanzadas.
- **CT02 (Comunicación oral y escrita):**
  Los alumnos deben presentar los resultados de la práctica en un informe bien estructurado, demostrando su capacidad para explicar conceptos técnicos de forma clara y comprensible.
- **CG1 (Análisis, diseño e implementación):**
  La práctica integra la selección de hardware adecuado (CPU vs GPU) y el diseño de posibles soluciones de optimización, que son habilidades clave para diseñar sistemas completos de inteligencia artificial.
- **CG4 (Obtener soluciones eficientes):**
  A través de la propuesta de optimización, los alumnos deben demostrar su capacidad para plantear soluciones probables y óptimas, evaluando diferentes factores de rendimiento.
- **CE15 (Sistemas computacionales de altas prestaciones):**
  Este aspecto es central en la práctica, ya que los alumnos trabajan directamente con aceleradores (GPUs) y analizan cómo estos mejoran la eficiencia computacional en tareas específicas.

RESULTADOS DE APRENDIZAJE TRABAJADOS:

- **Resultado 1 (Seleccionar la mejor arquitectura paralela):**
  Los alumnos comparan el rendimiento de la CPU y la GPU y justifican qué arquitectura es más adecuada para tareas de procesamiento de imágenes.
- **Resultado 5 (Conocer arquitecturas específicas basadas en GPUs):**
  La práctica pone el foco en las GPUs como aceleradores y su capacidad para manejar tareas intensivas, como el reconocimiento de imágenes.

OBJETIVOS ESPECÍFICOS DEL CURSO TRABAJADOS:

- **Entender el uso de la computación de altas prestaciones:**
  La comparación de tiempos de ejecución entre CPU y GPU permite a los alumnos comprender cómo la computación de altas prestaciones impacta en el rendimiento de aplicaciones prácticas.
- **Conocer el uso de aceleradores (p.e., GPUs):**
  Los alumnos experimentan directamente con GPUs, analizan sus resultados y reflexionan sobre cómo aprovechar mejor este recurso.

Asignatura: computación de alto rendimiento
Proferor: Ricardo Moreno
Email: ricardo.moreno@ua.es

Universitat d'Alacant
Universidad de Alicante

ANEXOS DE CONFIGURACIÓN Y DOCUMENTACIÓN EXTRA

Asignatura: computación de alto rendimiento
Proferor: Ricardo Moreno
Email: ricardo.moreno@ua.es

Universitat d'Alacant
Universidad de Alicante

GUÍA PARA CONFIGURAR PYTHON 3.13.2 Y PROBAR LA PRÁCTICA EN TU PC

1. DESCARGAR E INSTALAR PYTHON 3.13.2

- Descarga la versión **python-3.13.2-amd64** desde la web oficial de Python.
- Durante la instalación:
  - **Marca la opción "Add Python to PATH"** (esto te permitirá ejecutarlo desde la terminal).
  - Elige la opción **"Install for all users"** si estás en un PC personal.

2. VERIFICAR LA INSTALACIÓN

Una vez completada la instalación:

1. Abre la terminal (Command Prompt o PowerShell).
2. Escribe:

```
python --version
```

Deberías ver algo como:

```
Python 3.13.2
```

3. CREAR UN ENTORNO VIRTUAL (OPCIONAL PERO RECOMENDADO)

Esto te permitirá aislar esta práctica de otras configuraciones de Python en tu PC.

1. En la terminal, navega hasta la carpeta donde quieras crear el entorno:

```
cd C:\ruta\a\tu\proyecto
```

Crea un entorno virtual llamado `venv`:

```
python -m venv venv
```

Activa el entorno:

- **En Windows:**

```
.\venv\Scripts\activate
```

Deberías ver el nombre del entorno `(venv)` al inicio de la línea de comandos.

4. INSTALAR PYTORCH Y BIBLIOTECAS ADICIONALES

Con el entorno activado, ejecuta los siguientes comandos:

- **Si tienes una GPU compatible con CUDA (versión 11.8):**

```
pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu118
```

- **Si no quieres usar la GPU o no tienes soporte CUDA:**

```
pip install torch torchvision torchaudio
```

- **Instala también otras bibliotecas necesarias:**

```
pip install pillow matplotlib
```

5. VERIFICAR QUE PYTORCH DETECTA LA GPU

Abre Python desde la terminal:

```
python
```

Escribe el siguiente código para comprobar si la GPU está disponible:

```
import torch
print("¿CUDA disponible?:", torch.cuda.is_available())
```

- **Si devuelve `True`:** ¡Todo está listo para usar la GPU! 🎉

Asignatura: computación de alto rendimiento
Proferor: Ricardo Moreno
Email: ricardo.moreno@ua.es

- **Si devuelve `False`:** Verifica los drivers de la GPU y asegúrate de que CUDA esté instalado.

6. PRUEBA RÁPIDA DEL ENTORNO

Prueba este código básico de multiplicación de matrices, tanto en la CPU como en la GPU:

```
import torch
import time

# Crear una matriz grande
size = 10000
matrix_cpu = torch.randn(size, size)
matrix_gpu = matrix_cpu.cuda()  # Mover la matriz a la GPU

# Medir el tiempo en CPU
start_cpu = time.time()
result_cpu = torch.matmul(matrix_cpu, matrix_cpu)
end_cpu = time.time()
print(f"Tiempo en CPU: {end_cpu - start_cpu:.4f} segundos")

# Medir el tiempo en GPU
start_gpu = time.time()
result_gpu = torch.matmul(matrix_gpu, matrix_gpu)
torch.cuda.synchronize()  # Asegurar que la operación ha terminado
end_gpu = time.time()
print(f"Tiempo en GPU: {end_gpu - start_gpu:.4f} segundos")
```

Deberías ver una diferencia significativa en el tiempo de ejecución entre CPU y GPU.

7. DIFERENCIAS CON EL AULA

Cuando vuelvas al aula, si los ordenadores tienen una versión diferente de Python, solo asegúrate de:

- Ejecutar **PyTorch y el código base de la práctica** con la versión instalada.
- Si hay problemas de compatibilidad, chatGPT o la documentación del programa pueden ayudarte a solucionarlo (PyTorch es bastante flexible).

Universitat d'Alacant
Universidad de Alicante

ANEXO 1: ¿QUÉ HACER SI TU PC NO DISPONE DE GPU? (OPCIÓN B)

INTRODUCCIÓN:

En esta práctica, es posible que algunos alumnos no tengan acceso a un ordenador con **GPU compatible con CUDA**. Esto no supone un obstáculo, ya que la práctica puede realizarse usando **solo la CPU** y complementarse con otras estrategias de análisis colaborativo o simulación. Este anexo te guía paso a paso sobre cómo proceder si tu ordenador no tiene GPU (Opción B).
En caso de que optes por realizar la práctica de ésta forma debes indicar claramente al principio de la memoria de entrega que se ha realizado con un PC que no dispone de GPU, pero en todo caso recomiendo que aproveches la ocasión de disponer un PC con GPU como el que disponemos en el Aula para comprobar la efectividad real de su uso y si no te diera tiempo a terminarla valoraría la opción B

PASO 1: EJECUTAR LA PRÁCTICA USANDO SOLO CPU

Si tu PC no tiene GPU, puedes ejecutar el código en **modo CPU** sin problemas. El código ya está preparado para detectar si hay una GPU disponible y, en caso contrario, ejecutará el modelo automáticamente en la CPU.
Repite los pasos de la práctica principal, pero ignora la parte del código relacionada con la GPU. Aquí te dejo el fragmento que puedes ejecutar:

```
import torch
import time
from torchvision import models, transforms
from PIL import Image

# Cargar el modelo ResNet18 preentrenado
model = models.resnet18(pretrained=True)
model.eval()

# Cargar la imagen de prueba y aplicarle transformaciones
image = Image.open("cara_ejemplo.jpg")
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor()
])
input_tensor = transform(image).unsqueeze(0)  # Añadir dimensión batch

# Medir el tiempo de ejecución en CPU
start_cpu = time.time()
with torch.no_grad():
    output_cpu = model(input_tensor)
end_cpu = time.time()

print(f"Tiempo en CPU: {end_cpu - start_cpu:.4f} segundos")
```

Asignatura: computación de alto rendimiento
Proferor: Ricardo Moreno
Email: ricardo.moreno@ua.es

Nota importante: Si trabajas con imágenes más grandes (por ejemplo, 512x512 o 1024x1024), los tiempos de ejecución en CPU serán más lentos. Asegúrate de probar con imágenes de distintos tamaños para comparar los resultados.

PASO 2: COMPARAR TUS RESULTADOS CON LOS OBTENIDOS POR COMPAÑEROS CON GPU

- Aunque estés trabajando solo con CPU, otros grupos o compañeros que sí tengan GPU pueden proporcionarte sus tiempos de ejecución.
- Completa la tabla de resultados comparativos con los tiempos de CPU y GPU:

Tamaño de la imagen	**Tiempo CPU (s)**	**Tiempo GPU (s) (obtenido de otro grupo)**
224 x 224		
512 x 512		
1024 x 1024		

- **Análisis colaborativo:** Compara y reflexiona sobre la diferencia entre los tiempos de CPU y GPU. Discute con tus compañeros por qué la GPU es más eficiente en este tipo de tareas.

PASO 3: OPTIMIZACIÓN Y ANÁLISIS ADICIONAL

Si no tienes GPU, investiga **cómo mejorar el tiempo de ejecución en CPU** utilizando las siguientes estrategias:

1. USAR UN MODELO MÁS LIGERO

- En lugar de `ResNet18`, puedes utilizar un modelo más ligero como `MobileNetV2`:

```
# En Phyton
model = models.mobilenet_v2(pretrained=True)
model.eval()
```

Por qué funciona: MobileNet está **diseñado para dispositivos móviles** y tiene menos parámetros que **ResNet**, lo que reduce el tiempo de procesamiento en CPU

2. REDUCIR EL TAMAÑO DE LA IMAGEN

- Trabaja con imágenes más pequeñas, por ejemplo, 128x128 o 256x256, en lugar de 512x512 o más grandes:

```
# En Phyton
transform = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.ToTensor()
])
```

Por qué funciona: Las imágenes más pequeñas requieren menos operaciones matemáticas y, por lo tanto, reducen el tiempo de ejecución.

Asignatura: computación de alto rendimiento
Proferor: Ricardo Moreno
Email: ricardo.moreno@ua.es

3. PARALELISMO A NIVEL DE CPU

- Investiga cómo podrías utilizar múltiples núcleos de la CPU para procesar varias imágenes en paralelo.
- Puedes probar la biblioteca **concurrent.futures** en Python para paralelizar el proceso.

PASO 4: AMPLIA TUS CONOCIMIENTOS USANDO SERVICIOS EN LA NUBE (OPCIONAL)

Ésta es una tarea simplemente de investigación y ampliación que no debes realizar de forma obligatoria, pero la recomiendo:
Si tienes conexión a internet y quieres probar cómo funciona la GPU, puedes ejecutar la práctica en **Google Colab** o **Kaggle Notebooks**, que ofrecen acceso gratuito a GPUs.

GOOGLE COLAB (OPCIONAL)

1. Accede a Google Colab (https://colab.research.google.com/)
2. Crea un nuevo cuaderno y selecciona **GPU**:
   - Ve a **Entorno de ejecución** > **Cambiar tipo de entorno de ejecución** > Selecciona **GPU**.
3. Copia el código de la práctica principal y ejecútalo allí.

🔑 **Nota:** Aunque no sea necesario para aprobar la práctica, esta es una excelente opción para experimentar cómo cambia el rendimiento al usar GPU.

PASO 5: INFORME DE ENTREGA

El informe debe incluir:

REFLEXIÓN SOBRE LO APRENDIDO:

- Un breve resumen de la práctica.
- El análisis y discusión sobre la ejecución en CPU y cómo podría mejorarse el rendimiento.
- Reflexiona sobre los conceptos de paralelismo y qué beneficios podría ofrecer una GPU si estuviera disponible.

TABLA DE RESULTADOS DE EJECUCIÓN EN CPU:

- Incluye los tiempos de ejecución obtenidos para diferentes tamaños de imagen.

Ejemplo:

Tamaño de la imagen	**Tiempo CPU (s)**
224 x 224	
512 x 512	
1024 x 1024	

COMPARACIÓN CON TIEMPOS DE REFERENCIA DE GPU (SI ESTÁN DISPONIBLES):

- Utiliza los tiempos obtenidos por otros compañeros con GPU o tiempos de referencia proporcionados por el docente.
- Reflexiona sobre las diferencias observadas.

Ejemplo de comparación:

Tamaño de la imagen	Tiempo CPU (s)	Tiempo de referencia (GPU) (s)
224 x 224		
512 x 512		
1024 x 1024		

ANÁLISIS SOBRE LAS DIFERENCIAS DE RENDIMIENTO:

- Explica por qué los tiempos de CPU son más lentos y qué factores influyen en el rendimiento.
- Reflexiona sobre cómo cambiaría el rendimiento si tuvieras una GPU disponible.

PROPUESTAS DE OPTIMIZACIÓN (NO ES NECESARIO IMPLEMENTARLAS):

- Plantea posibles mejoras en el tiempo de ejecución en CPU.
- Considera opciones como:
  - Usar modelos más ligeros.
  - Reducir el tamaño de las imágenes para optimizar el procesamiento.
  - Explorar técnicas de paralelismo en CPU, como dividir las tareas en múltiples núcleos.

ANEXO 2: BIBLIOTECAS UTILIZADAS EN LA PRÁCTICA

Asignatura: computación de alto rendimiento
Proferor: Ricardo Moreno
Email: ricardo.moreno@ua.es

1. BIBLIOTECA `TORCH` (PYTORCH)

DESCRIPCIÓN:

PyTorch es una biblioteca de código abierto desarrollada por Facebook's AI Research lab (FAIR), diseñada para realizar cálculo numérico y aprendizaje automático con un enfoque en el entrenamiento de modelos de redes neuronales profundas. Es una de las herramientas más populares en el mundo de la inteligencia artificial (IA) y la investigación.
PyTorch está optimizada para el cálculo numérico y el aprendizaje automático. Su principal estructura de datos es el **tensor**, una matriz generalizada que permite realizar operaciones matemáticas complejas de manera eficiente, incluso utilizando **GPU**.
PyTorch permite mover los cálculos entre la CPU y la GPU de manera sencilla, aprovechando CUDA para paralelizar los cálculos y acelerar el entrenamiento de redes neuronales.
PyTorch proporciona un módulo llamado **torch.nn**, que facilita la construcción de redes neuronales utilizando capas predefinidas como convolucionales, recurrentes y totalmente conectadas.

EJEMPLO DE UNA RED NEURONAL SIMPLE:

```
import torch
import torch.nn as nn
# Definir una red neuronal simple
class SimpleNet(nn.Module):
    def __init__(self):
        super(SimpleNet, self).__init__()
        self.fc = nn.Linear(10, 2)  # Capa totalmente conectada (10 entradas, 2 salidas)
    def forward(self, x):
        return self.fc(x)
model = SimpleNet()
```

MODELOS PREENTRENADOS:

- PyTorch incluye la biblioteca **torchvision**, que permite usar modelos preentrenados (ResNet, MobileNet, etc.) listos para tareas como clasificación de imágenes.

EJEMPLO DE USO DE UN MODELO PREENTRENADO (RESNET18):

```
From torchvision import models
# cargar el modelo resnet18 preentrenado
Model = models.resnet18(pretrained=true)
Model.eval()  # modo de evaluación
```

Asignatura: computación de alto rendimiento
Proferor: Ricardo Moreno
Email: ricardo.moreno@ua.es

¿POR QUÉ LA USAMOS EN ESTA PRÁCTICA?

- PyTorch nos permite cargar el modelo de reconocimiento de imágenes, realizar la inferencia y comparar los tiempos de ejecución entre CPU y GPU.

EJEMPLO PRÁCTICO:

```
# En Phyton
import torch

# Crear un tensor de ejemplo (matriz 2x2)
tensor = torch.tensor([[1, 2], [3, 4]])
print(tensor)
```

CONCEPTO CLAVE:

Un tensor es similar a un array en NumPy, pero tiene la ventaja de poder ejecutarse en GPU para operaciones paralelas.

Asignatura: computación de alto rendimiento
Proferor: Ricardo Moreno
Email: ricardo.moreno@ua.es

2. TORCHVISION

DESCRIPCIÓN:

- `torchvision` es una biblioteca complementaria de PyTorch, especializada en el procesamiento de imágenes. Incluye funciones para cargar datasets populares, aplicar transformaciones a imágenes y usar modelos preentrenados para tareas de visión por computadora.

¿POR QUÉ LA USAMOS EN ESTA PRÁCTICA?

- La utilizamos para:
  - Cargar el modelo **ResNet18** preentrenado.
  - Aplicar transformaciones a las imágenes antes de enviarlas al modelo.

EJEMPLO PRÁCTICO:

```
from torchvision import models, transforms

# Cargar el modelo ResNet18 preentrenado
model = models.resnet18(pretrained=True)

# Definir transformaciones: redimensionar y convertir la imagen a tensor
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor()
])
```

CONCEPTO CLAVE:

Los modelos preentrenados nos permiten realizar tareas de clasificación de imágenes sin necesidad de entrenar un modelo desde cero.

3. TORCHAUDIO

DESCRIPCIÓN:

Torchaudio es otra biblioteca complementaria de PyTorch, enfocada en el **procesamiento de datos de audio**. Aunque **no la usamos en esta práctica**, es muy útil para futuros proyectos relacionados con **reconocimiento de voz o clasificación de sonidos**.

¿QUÉ PODRÍAS HACER CON TORCHAUDIO?

- Cargar archivos de audio.
- Extraer características como espectrogramas.
- Aplicar modelos de clasificación de audio.

EJEMPLO PRÁCTICO:

```
import torchaudio

# Cargar un archivo de audio
waveform, sample_rate = torchaudio.load('mi_audio.wav')
print(f"Forma del waveform: {waveform.shape}")
```

CONCEPTO CLAVE:

Torchaudio es ideal si te interesa explorar cómo funciona el aprendizaje automático en datos de audio, como reconocimiento de voz o detección de eventos sonoros.

4. PILLOW (PIL)

DESCRIPCIÓN:

pillow es una biblioteca de procesamiento de imágenes. Permite abrir, modificar y guardar imágenes en diversos formatos, como PNG, JPEG o BMP.

¿POR QUÉ LA USAMOS EN ESTA PRÁCTICA?

Para cargar la imagen de entrada, redimensionarla y prepararla antes de enviarla al modelo de PyTorch.

EJEMPLO PRÁCTICO:

```
from PIL import Image

# Cargar una imagen y redimensionarla
image = Image.open('cara_ejemplo.jpg')
image_resized = image.resize((224, 224))
image_resized.show()  # Mostrar la imagen redimensionada
```

CONCEPTO CLAVE:

La imagen debe transformarse en un formato adecuado (tensor) para ser procesada por el modelo de PyTorch.

Nota: ¿Qué es un tensor y por qué lo necesitamos en esta práctica?

Un **tensor** es una estructura de datos similar a un **array multidimensional** que generaliza vectores y matrices. En el contexto del procesamiento de imágenes, un tensor puede representar una imagen en 3 dimensiones:

(C, H, W) → (Canales, Altura, Ancho)

Por ejemplo, una imagen RGB de 224x224 píxeles tiene un tensor con las dimensiones **(3, 224, 224)**, donde:

- **C:** Número de canales (rojo, verde, azul).
- **H:** Altura de la imagen.
- **W:** Ancho de la imagen.

En esta práctica, necesitamos transformar la imagen en un tensor porque los modelos de **PyTorch** no pueden procesar directamente imágenes en formatos como **JPEG** o **PNG**. La transformación permite que el modelo realice operaciones matemáticas eficientes, como multiplicaciones de matrices y convoluciones, utilizando **CPU o GPU** para extraer características y clasificar la imagen.
*Te animo a investigar más sobre el manejo de tensores para el tratamiento de imágenes

5. MATPLOTLIB

DESCRIPCIÓN:

matplotlib es una biblioteca de Python para crear gráficos y visualizaciones de datos. Es muy útil para analizar resultados y presentar comparaciones.

¿POR QUÉ LA USAMOS EN ESTA PRÁCTICA?

Para graficar los tiempos de ejecución de CPU y GPU y analizar visualmente las diferencias.

EJEMPLO PRÁCTICO:

```
# En phyton
import matplotlib.pyplot as plt

# Datos de ejemplo
tiempos_cpu = [1.2, 1.4, 1.8]
tiempos_gpu = [0.4, 0.5, 0.7]

# Crear un gráfico de comparación
plt.plot(tiempos_cpu, label='CPU', marker='o')
plt.plot(tiempos_gpu, label='GPU', marker='x')
plt.legend()
plt.xlabel('Tamaño de la imagen')
plt.ylabel('Tiempo de ejecución (s)')
plt.title('Comparación de tiempos CPU vs GPU')
plt.show()
```

CONCEPTO CLAVE:

Los gráficos permiten observar rápidamente cómo varía el rendimiento según el tamaño de la imagen y el hardware utilizado.

Universitat d'Alacant
Universidad de Alicante

ANEXO 3: DIMENSIONES BATCH EN PROCESAMIENTO DE DATOS PARA REDES NEURONALES

Asignatura: computación de alto rendimiento
Proferor: Ricardo Moreno
Email: ricardo.moreno@ua.es

INTRODUCCIÓN

En el contexto del aprendizaje profundo (deep learning), las dimensiones de los datos son fundamentales para que los modelos de redes neuronales funcionen de manera correcta y eficiente.

Una de las más importantes es la dimensión batch, que permite procesar múltiples datos (como imágenes, volúmenes 3D o nubes de puntos) en paralelo y aprovechar al máximo el hardware (especialmente las GPUs).

Este anexo explica qué es la dimensión batch, cómo se representa en tensores, y cómo se aplica a datos 2D y 3D.

2. ¿QUÉ ES LA DIMENSIÓN BATCH?

La dimensión batch agrupa un conjunto de datos para ser procesados simultáneamente por el modelo de red neuronal. Este enfoque es esencial para aprovechar el paralelismo de los procesadores modernos, como las GPUs.

Por ejemplo, si se procesa una sola imagen a la vez, se tendría un tensor con forma [C, H, W], donde:

- **C:** Número de canales (por ejemplo, 3 para imágenes RGB).
- **H:** Altura de la imagen.
- **W:** Ancho de la imagen.

Si se quieren procesar varias imágenes simultáneamente (por ejemplo, un lote de 32 imágenes), es necesario añadir una dimensión batch:

- **[batch_size, C, H, W]:** Representa 32 imágenes de 3 canales cada una, con altura y ancho de 224x224 píxeles.

3. DIMENSIÓN BATCH EN TENSORES 2D: IMÁGENES

Cuando se trabaja con imágenes 2D en **PyTorch**, el flujo típico es:

1. Cargar la imagen.
2. Convertirla a tensor.
3. Añadir la dimensión batch (si es necesario).

Ejemplo de código en PyTorch:

```
from PIL import Image
from torchvision import transforms
import torch

# Cargar una imagen
image = Image.open("imagen_ejemplo.jpg")

# Convertir a tensor y redimensionar
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor()
])

input_tensor = transform(image)  # Forma: [C, H, W]
input_tensor = input_tensor.unsqueeze(0)  # Añadir dimensión batch: [1, C, H, W]

print(input_tensor.shape)  # Salida esperada: [1, 3, 224, 224]
```

4. DIMENSIÓN BATCH EN DATOS 3D

Tipos comunes de datos 3D y su representación en tensores:

- **Volúmenes 3D:** Representados por tensores de la forma **[batch_size, C, D, H, W]**.
- **Nubes de puntos:** Utilizan la forma **[batch_size, num_puntos, num_características]**.
- **Mallas 3D:** Se representan mediante **[batch_size, num_vértices, 3]**, donde las coordenadas (x, y, z) representan cada vértice.

5. EJEMPLOS PRÁCTICOS DE TRANSFORMACIONES

VOLÚMENES 3D:

```
import torch

# Crear un volumen 3D simulado (sin batch)
volume = torch.rand(1, 32, 128, 128)

# Añadir la dimensión batch
volume_batch = volume.unsqueeze(0)  # [1, 1, 32, 128, 128]
print(volume_batch.shape)
```

NUBES DE PUNTOS:

```
# Crear una nube de puntos simulada
point_cloud = torch.rand(1024, 3)

# Añadir la dimensión batch
point_cloud_batch = point_cloud.unsqueeze(0)  # [1, 1024, 3]
print(point_cloud_batch.shape)
```

. RESUMEN

La dimensión batch permite procesar múltiples datos simultáneamente y es fundamental para optimizar el procesamiento en GPUs. Puedes aplicarla tanto en datos 2D como en datos 3D. En **PyTorch**, se puede usar la función `unsqueeze(0)` para transformar datos individuales en lotes listos para ser procesados.

Asignatura: computación de alto rendimiento
Proferor: Ricardo Moreno
Email: ricardo.moreno@ua.es