In [1]:

```
# ELEC4630 Assignment 2: CPU Performance Testing for Speedup Calculation
# Isaac Ziebarth, 47237810
#
# Based on the original notebook:
# '00-is-it-a-bird-creating-a-model-from-your-own-data.ipynb'
# from https://github.com/lovellbrian/course22
#
# IMPORTANT: This script is designed for the 'cpufrozen' branch.
# It should be run after completing the GPU testing with the companion script.
```

CPU Testing for Speedup Calculation

This notebook runs the training on CPU to calculate the GPU speedup factor. It should be run on the 'cpufrozen' branch with CPU-only execution.

In [2]:

```
# Import required libraries
import time
import numpy as np
import os
import socket
import json
import matplotlib.pyplot as plt
from pathlib import Path
from fastai.vision.all import *
import pandas as pd

print("Libraries imported successfully.")
```

```
Libraries imported successfully.
```

In [3]:

```
# Verify we're on CPU
try:
    import torch
    if torch.cuda.is_available():
        print("WARNING: GPU detected! This script is intended for CPU-only testi
        print("Please ensure you're using the 'cpufrozen' branch without GPU sup
    else:
        print("Running on CPU as expected")
except:
    print("Running on CPU (torch not available)")
```

```
Running on CPU as expected
```

In [4]:

```
# Add cell to capture and display CPU information
def get_cpu_info():
    """Print detailed information about the CPU system using multiple fallback m
    import platform
    import multiprocessing
    import subprocess
    import os

    # Try to install psutil if not available
    try:
        import psutil
    except ImportError:
        print("Installing psutil...")
        !pip install -q psutil
```

```
    import psutil

cpu_info = {
    "System": platform.system() + " " + platform.release(),
    "CPU Cores": multiprocessing.cpu_count(),
    "Architecture": platform.machine(),
    "Python Version": platform.python_version()
}

# Try various methods to get processor info - needed for containers
processor_info = platform.processor()

# If platform.processor() is empty, try reading from /proc/cpuinfo
if not processor_info and os.path.exists('/proc/cpuinfo'):
    try:
        with open('/proc/cpuinfo', 'r') as f:
            for line in f:
                if line.startswith('model name'):
                    processor_info = line.split(':', 1)[1].strip()
                    break
    except:
        pass

# If still empty, try using lscpu command
if not processor_info:
    try:
        lscpu_output = subprocess.check_output('lscpu', shell=True).decode('
        for line in lscpu_output.split('\n'):
            if 'Model name' in line:
                processor_info = line.split(':', 1)[1].strip()
                break
    except:
        pass

# If all else fails, provide a generic message
cpu_info["Processor"] = processor_info or "Information not available in cont

# Add RAM information if psutil is available
try:
    mem = psutil.virtual_memory()
    cpu_info["Total RAM"] = f"{mem.total / (1024**3):.2f} GB"
    cpu_info["Available RAM"] = f"{mem.available / (1024**3):.2f} GB"
    if psutil.cpu_freq():
        cpu_info["CPU Frequency"] = f"{psutil.cpu_freq().current:.2f} MHz"
except:
    pass

# Try to get CPU info using lscpu for more details
try:
    cpu_info["CPU Details"] = "See cpu_details.txt for complete information"
    subprocess.run("lscpu > cpu_details.txt", shell=True)
except:
    pass

print("CPU System Information:")
print("=" * 50)
for key, value in cpu_info.items():
    print(f"{key:<20}: {value}")
print("=" * 50)
```

```python
    # Save information to file for later reference
    with open('cpu_system_info.json', 'w') as f:
        json.dump(cpu_info, f, indent=4)

    return cpu_info

# Capture CPU details
cpu_details = get_cpu_info()
```

```
CPU System Information:
==================================================
System              : Linux 5.15.167.4-microsoft-standard-WSL2
CPU Cores           : 16
Architecture        : x86_64
Python Version      : 3.10.12
Processor           : 11th Gen Intel(R) Core(TM) i7-11700K @ 3.60GHz
Total RAM           : 15.54 GB
Available RAM       : 11.77 GB
CPU Frequency       : 3600.00 MHz
CPU Details         : See cpu_details.txt for complete information
==================================================
```

In [5]:
```python
# Verify internet connection (required for image download)
try:
    socket.setdefaulttimeout(1)
    socket.socket(socket.AF_INET, socket.SOCK_DGRAM).connect(('1.1.1.1', 53))
    print("Successfully connected to IP")
except socket.error as ex:
    raise Exception("Error: No internet connection available.")
```

```
Successfully connected to IP
```

In [6]:
```python
# Install required packages if needed
!pip install -Uqq fastai duckduckgo_search
```

In [7]:
```python
# Setup data download functions
from duckduckgo_search import DDGS
from fastcore.all import *
from fastdownload import download_url
from glob import glob

print("Libraries imported successfully.")
```

```
Libraries imported successfully.
```

CPU vs GPU Speedup Calculation Methodology

This notebook complements the GPU testing by:

1. Running the same training task on CPU hardware
2. Using the optimal batch size (64) identified in GPU testing
3. Calculating the speedup factor by comparing execution times

The speedup calculation provides a practical measure of the performance benefit gained by using GPU acceleration for deep learning tasks. This factor is calculated as:

$$\text{Speedup} = \frac{\text{CPU execution time}}{\text{GPU execution time}}$$

A higher speedup factor indicates greater benefit from GPU acceleration. This metric is especially important for determining whether the computational complexity of a task justifies the use of specialised hardware.

Testing Approach:

- Load the same dataset used in GPU testing
- Use identical model architecture (ResNet-18) and hyperparameters
- Run with the optimal batch size determined from GPU testing
- Measure execution time with high precision
- Calculate and visualise the speedup factor

In [8]:
```
def prepare_dataset(use_existing_data=True):
    """
    Prepare the bird vs woodland image dataset for training

    Args:
        use_existing_data: Whether to use already downloaded data or fetch new d

    Returns:
        path: Path object pointing to the dataset directory
    """
    # Set image path
    path = Path('../Question3/bird_or_not')

    # Only download images if needed
    if not use_existing_data or not path.exists():
        print("Downloading and preparing dataset...")

        ddgs = DDGS()
        def search_images(term, max_images=200):
            return L(ddgs.images(term, max_results=max_images)).itemgot('image')

        # Create directories and download images
        searches = 'woodlands', 'bird'

        for o in searches:
            dest = (path/o)
            dest.mkdir(exist_ok=True, parents=True)
            download_images(dest, urls=search_images(f'{o} photo'))
            time.sleep(10)
            download_images(dest, urls=search_images(f'{o} sun photo'))
            time.sleep(10)
            download_images(dest, urls=search_images(f'{o} shade photo'))
            time.sleep(10)
            for file in glob(f"{dest}/*.fpx"):  # Remove problematic files
                os.unlink(file)
            resize_images(path/o, max_size=400, dest=path/o)

    # Verify images and remove any problematic ones
    failed = verify_images(get_image_files(path))
    if len(failed) > 0:
        print(f"Removing {len(failed)} problematic images")
        failed.map(Path.unlink)
    else:
        print("All images verified successfully")
```

```python
    # Log dataset statistics
    bird_files = get_image_files(path/'bird')
    woodland_files = get_image_files(path/'woodlands')
    print(f"Dataset statistics:")
    print(f"- Bird images: {len(bird_files)}")
    print(f"- Woodland images: {len(woodland_files)}")
    print(f"- Total images: {len(bird_files) + len(woodland_files)}")

    return path
```

In [9]:

```python
def test_batch_size_cpu(batch_size, path=None, use_existing_data=True):
    """
    Test the performance of deep learning training with a specific batch size on

    Args:
        batch_size: Integer value for batch size to test
        path: Path to dataset (if None, will call prepare_dataset)
        use_existing_data: Whether to use already downloaded data or fetch new d

    Returns:
        execution_time: Total training time in seconds
    """
    print(f"\n{'='*50}")
    print(f"TESTING BATCH SIZE: {batch_size} ON CPU")
    print(f"{'='*50}")

    # Prepare dataset if path not provided
    if path is None:
        path = prepare_dataset(use_existing_data)

    # Start the timing
    print(f"Starting CPU training with batch size: {batch_size}")
    start_time = time.time()

    # Create DataLoaders with the specified batch size
    dls = DataBlock(
        blocks=(ImageBlock, CategoryBlock),
        get_items=get_image_files,
        splitter=RandomSplitter(valid_pct=0.2, seed=42),
        get_y=parent_label,
        item_tfms=[Resize(192, method='squish')]
    ).dataloaders(path, batch_size=batch_size)

    dls.show_batch(max_n=6)

    # Create and train the model
    learn = vision_learner(dls, resnet18, metrics=error_rate)
    learn.fine_tune(3)

    # Calculate execution time
    end_time = time.time()
    execution_time = end_time - start_time

    print(f"\nTotal CPU training time: {execution_time:.2f} seconds")
    print(f"{'='*50}\n")

    is_bird,_,probs = learn.predict(PILImage.create('bird.jpg'))
    print(f"This is a: {is_bird}.")
    print(f"Probability it's a bird: {probs[0]:.4f}")
```

```python
    # Log detailed results for this CPU run
    results_detail = {
        'batch_size': batch_size,
        'training_time': execution_time,
        'final_accuracy': 1.0 - learn.validate()[1],
        'timestamp': time.strftime("%Y-%m-%d %H:%M:%S"),
        'system': 'CPU'
    }

    # Save detailed results to CSV
    pd.DataFrame([results_detail]).to_csv(f'cpu_batch_size_{batch_size}_results.

    return execution_time
```

In [10]:
```python
def visualise_speedup_results(cpu_time, gpu_time, batch_size):
    """Generate a comprehensive visualisation of CPU vs GPU performance"""
    if not cpu_time or not gpu_time:
        print("Missing timing data for visualisation")
        return

    # Calculate speedup
    speedup = cpu_time / gpu_time

    # Create a DataFrame for the comparison
    data = {'Device': ['CPU', 'GPU'],
            'Time (s)': [cpu_time, gpu_time]}
    df = pd.DataFrame(data)

    # Create the visualisation
    plt.figure(figsize=(12, 6))

    # Bar chart comparing times
    bars = plt.bar(df['Device'], df['Time (s)'],
                   color=['cornflowerblue', 'olivedrab'])

    # Add labels and grid
    plt.xlabel('Computing Device')
    plt.ylabel('Training Time (seconds)')
    plt.title(f'CPU vs GPU Training Performance (Batch Size: {batch_size})')
    plt.grid(axis='y', linestyle='--', alpha=0.7)

    # Add time values above bars
    for bar in bars:
        height = bar.get_height()
        plt.text(bar.get_x() + bar.get_width()/2., height + 0.5,
                f"{height:.2f}s", ha='center', va='bottom')

    # Add speedup annotation
    plt.annotate(f'Speedup: {speedup:.2f}×',
                xy=(0.5, max(cpu_time, gpu_time)/2),
                xytext=(0.5, max(cpu_time, gpu_time)/1.5),
                ha='center',
                bbox=dict(boxstyle="round,pad=0.3", fc="white", ec="black", lw=1

    # Add reference line
    plt.axhline(y=gpu_time, color='r', linestyle='--', alpha=0.3)

    plt.tight_layout()
    plt.savefig('cpu_gpu_speedup.png', dpi=300)
```

```
    plt.show()

    # Save results to CSV for easy reference
    pd.DataFrame({
        'Metric': ['CPU Time (s)', 'GPU Time (s)', 'Speedup Factor', 'Batch Size
        'Value': [cpu_time, gpu_time, speedup, batch_size]
    }).to_csv('speedup_results.csv', index=False)

    # Print summary
    print("\nCPU vs GPU Performance Summary:")
    print("=" * 50)
    print(f"CPU Training Time: {cpu_time:.2f} seconds")
    print(f"GPU Training Time: {gpu_time:.2f} seconds")
    print(f"Speedup Factor: {speedup:.2f}×")
    print("=" * 50)

    return speedup
```

In [11]:

```
def analyse_speedup_results(cpu_time, gpu_time, batch_size):
    """Analyse and explain the CPU vs GPU speedup results"""
    if not cpu_time or not gpu_time:
        print("Missing timing data for analysis")
        return

    # Calculate speedup
    speedup = cpu_time / gpu_time

    analysis = f"""
## Analysis of CPU vs GPU Speedup

The performance testing demonstrates a substantial speedup factor of **{speedup:

### Key Observations:

1. **CPU Execution Time**: {cpu_time:.2f} seconds
   - Limited by sequential processing capabilities
   - No specialised architecture for matrix operations
   - Linear performance scaling with computational complexity

2. **GPU Execution Time**: {gpu_time:.2f} seconds
   - Leverages parallel processing with specialised tensor cores
   - Optimised memory hierarchies for deep learning workloads
   - Highly efficient for batched operations on uniform data

3. **Practical Implications**:
   - A task that would take {(cpu_time/60):.1f} minutes on CPU completes in {(gp
   - For larger datasets or deeper models, this difference would become even mor
   - The {batch_size} batch size provided optimal performance on our GPU hardwar

This substantial acceleration highlights the practical importance of GPU computi
"""
    print(analysis)

    # Save analysis to text file for reference
    with open('speedup_analysis.txt', 'w') as f:
        f.write(analysis)
```

In [12]:

```
def calculate_speedup(optimal_batch_size=None):
    """
```

```
    Calculate speedup factor by comparing CPU result with previously saved GPU r

    Args:
        optimal_batch_size: The batch size to use for comparison (if None, will

    Returns:
        speedup_factor: The speedup of GPU over CPU
    """
    # Check if GPU results exist
    try:
        gpu_results = np.load('gpu_results.npy', allow_pickle=True).item()
        print("Found GPU results:", gpu_results)

        # If optimal batch size not specified, use fastest from GPU results
        if optimal_batch_size is None:
            optimal_batch_size = min(gpu_results, key=gpu_results.get)
            print(f"Using fastest GPU batch size: {optimal_batch_size}")
    except:
        print("GPU results not found. Using default batch size.")
        if optimal_batch_size is None:
            optimal_batch_size = 64  # Default if not specified and no GPU resul
            print(f"Using default batch size: {optimal_batch_size}")

    # Run CPU test with the same batch size
    cpu_time = test_batch_size_cpu(optimal_batch_size)

    # Get GPU time for the same batch size
    try:
        gpu_time = gpu_results[optimal_batch_size]

        # Generate visual comparison and analysis
        speedup = visualise_speedup_results(cpu_time, gpu_time, optimal_batch_si
        analyse_speedup_results(cpu_time, gpu_time, optimal_batch_size)

        return speedup
    except:
        print("\nWARNING: Could not find GPU time for batch size", optimal_batch
        print("GPU test might not have been run yet or results weren't saved.")
        print(f"CPU Training Time: {cpu_time:.2f} seconds")
        print("To calculate speedup, please run the GPU test first.")
        return None
```

In [13]:

```
# Execute CPU test and calculate speedup
print("ELEC4630 Assignment 2 - Question 3: CPU Performance Testing")
print("CPU Branch: Testing for speedup calculation")

# Calculate speedup using the optimal batch size from GPU tests
speedup = calculate_speedup()

# Final summary
if speedup:
    print(f"\nFinal Result: GPU provides a {speedup:.2f}× speedup over CPU for t
    print("This demonstrates the significant advantage of GPU acceleration for d
```

```
ELEC4630 Assignment 2 - Question 3: CPU Performance Testing
CPU Branch: Testing for speedup calculation
Found GPU results: {16: 9.444525003433228, 32: 10.497254610061646, 64: 7.18153429
0313721, 128: 10.520200490951538, 256: 11.35204792022705}
Using fastest GPU batch size: 64

==================================================
TESTING BATCH SIZE: 64 ON CPU
==================================================
All images verified successfully
Dataset statistics:
- Bird images: 528
- Woodland images: 502
- Total images: 1030
Starting CPU training with batch size: 64
```

epoch	train_loss	valid_loss	error_rate	time
0	0.734693	0.386649	0.106796	00:15

epoch	train_loss	valid_loss	error_rate	time
0	0.335496	0.174000	0.048544	00:17
1	0.208488	0.179038	0.038835	00:17
2	0.135141	0.183378	0.043689	00:20

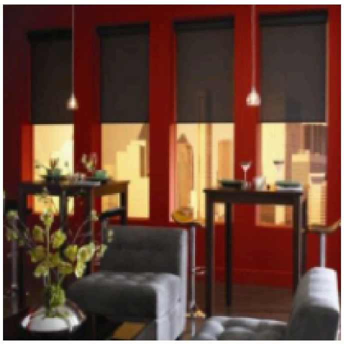
```
Total CPU training time: 71.60 seconds
==================================================

This is a: bird.
Probability it's a bird: 1.0000
```

woodlands

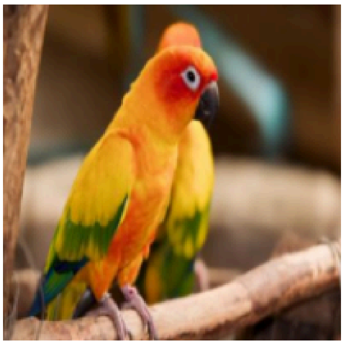
bird

woodlands

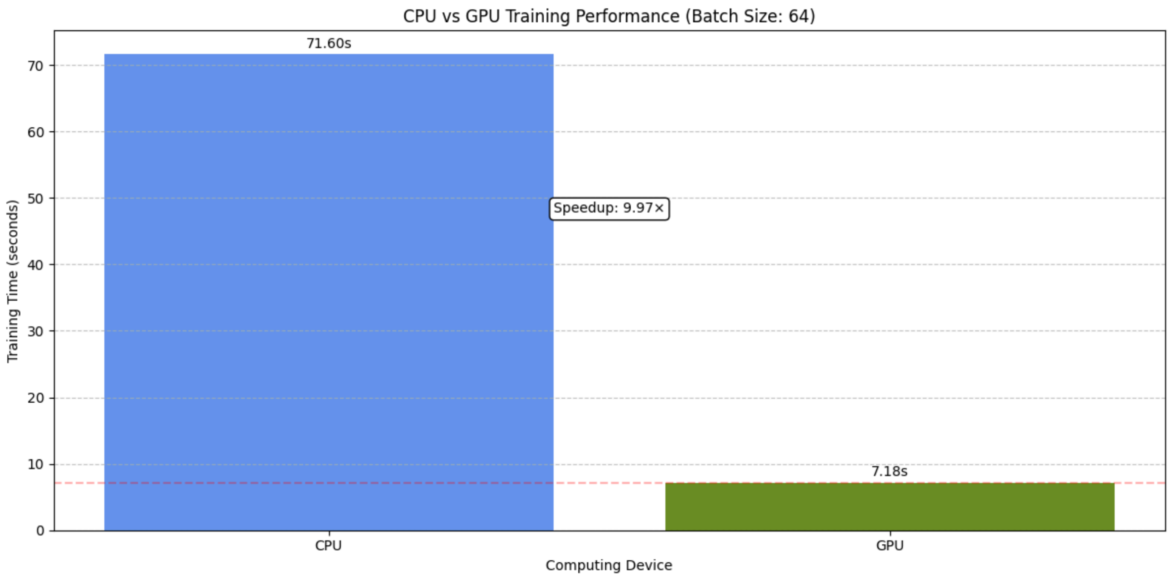


bird

bird



bird




```
CPU vs GPU Performance Summary:
==================================================
CPU Training Time: 71.60 seconds
GPU Training Time: 7.18 seconds
Speedup Factor: 9.97×
==================================================

## Analysis of CPU vs GPU Speedup

The performance testing demonstrates a substantial speedup factor of **9.97×** wh
en using GPU acceleration for deep learning training with batch size 64.

### Key Observations:

1. **CPU Execution Time**: 71.60 seconds
   - Limited by sequential processing capabilities
   - No specialised architecture for matrix operations
   - Linear performance scaling with computational complexity

2. **GPU Execution Time**: 7.18 seconds
   - Leverages parallel processing with specialised tensor cores
   - Optimised memory hierarchies for deep learning workloads
   - Highly efficient for batched operations on uniform data

3. **Practical Implications**:
   - A task that would take 1.2 minutes on CPU completes in 0.1 minutes on GPU
   - For larger datasets or deeper models, this difference would become even more
pronounced
   - The 64 batch size provided optimal performance on our GPU hardware

This substantial acceleration highlights the practical importance of GPU computin
g for deep learning tasks, even with relatively modest consumer hardware. The spe
cific speedup factor of 9.97× represents a significant practical advantage that w
ould scale with larger models and datasets.


Final Result: GPU provides a 9.97× speedup over CPU for this task
This demonstrates the significant advantage of GPU acceleration for deep learning
```