

**A Project Report on**

**Histopathologic Cancer Detection (Oral Cancer)**

**Team:** Team 7 [**George Chempumthara and Revanth Padala]**

**Course:** CSYE 7105 High Performance Parallel Machine Learning & AI

**Instructor:** Prof. Handan Liu

**December 5, 2025**

## 

## **Table of Contents**

1. Introduction

1.1 Background

1.2 Motivation

1.3 Goal

2. Methodology

2.1 Parallel Data Processing

2.2 Scope for Parallelization:

2.3 Model Building

2.4 Performance Evaluation

2.5 Analysis and Visualization

2.6 Optimization and Iteration

3. Dataset Description

3.1 Overview & Features

4. Results and Analysis

4.1 Configurations and Environment:

4.2 Parallel Performance on GPUs

4.2.1 Distributed Data Parallel (DDP) Training Analysis:

4.2.2 Fully Sharded Data Parallel (FSDP) Training Analysis:

4.3 Parallel Performance on CPUs:

4.3.1 Standard Multiprocessing (Pool)

4.3.2 Framework-Based Parallelism (Dask & Joblib)

4.3.3 Performance Comparison & Final Verdict

5. Conclusion

5.1 Comparative Analysis: FSDP vs. DDP

5.2 Future Scope & Recommendations

5.3 Concluding Remarks

6. References

7. Appendix

## 

## 

## **1. Introduction**

### **1.1 Background**

Oral Squamous Cell Carcinoma (OSCC) represents a significant global health challenge with high morbidity and mortality rates. The gold standard for diagnosis involves the manual examination of histopathological images, microscopic views of tissue by pathologists. While accurate, this manual process is labor-intensive, time-consuming, and prone to inter-observer subjectivity. Deep learning, specifically Convolutional Neural Networks (CNNs), has demonstrated immense promise in automating this process, potentially offering faster and more consistent diagnoses.

### **1.2 Motivation**

A major bottleneck in developing robust medical AI pipelines is the sheer scale of data required. Training a high-accuracy CNN model for cancer detection involves processing hundreds of thousands of high-resolution images. In a serial computing environment, tasks such as file system scanning, image resizing, label generation, and model training can take days. By leveraging parallel processing techniques both on CPUs for data preparation and GPUs for training, we can drastically reduce the time-to-insight. This project addresses the computational cost by applying parallel computing techniques to accelerate the end-to-end pipeline for Oral Cancer Detection.

### **1.3 Goal**

The primary goal of this project is twofold:

1. **Parallel Data Processing:** Implement parallelized scripts to handle massive file operations (file scanning, label assignment, and CSV generation) using Python’s multiprocessing libraries. We aim to measure and visualize performance gains (speedup, efficiency, and wall-clock time) when scaling from single-core to multi-core execution.
2. **Parallel CNN Training:** Develop a custom CNN architecture in PyTorch for binary classification (Normal vs. OSCC) and enable distributed training. We will compare training performance using PyTorch Distributed Data Parallel (DDP) and Mixed Precision training to demonstrate scalability for medical AI workloads.

## **2. Methodology**

### **2.1 Parallel Data Processing**

1. **Dataset Scanning and Label Generation:** We developed a benchmark script to traverse the directory structure of over 280,000 images. The script identifies images located in "Normal" or "OSCC" subdirectories and assigns binary labels (0 for Normal, 1 for OSCC).
2. **Multiprocessing Implementation:**

• To overcome I/O latency, we utilized Python’s multiprocessing pool.

• The workload (file path processing) was mapped across a variable number of CPU cores (1, 2, 4, 6).

• We specifically excluded validation sets during this phase to focus on the raw training data aggregation.

1. **Performance Benchmarking:** We measured the wall-clock time required to process the entire file list and generate the dataset CSV. These timings were used to calculate Speedup (S = T1 / Tp) and Parallel Efficiency (E = S / p).

### **2.2 Scope for Parallelization**

The project utilizes a massive collection of histopathologic images. Our specific subset for this report comprises **287,832 images**, excluding validation sets. Given this volume, serial processing is inefficient. The project leverages two distinct parallelization scopes:

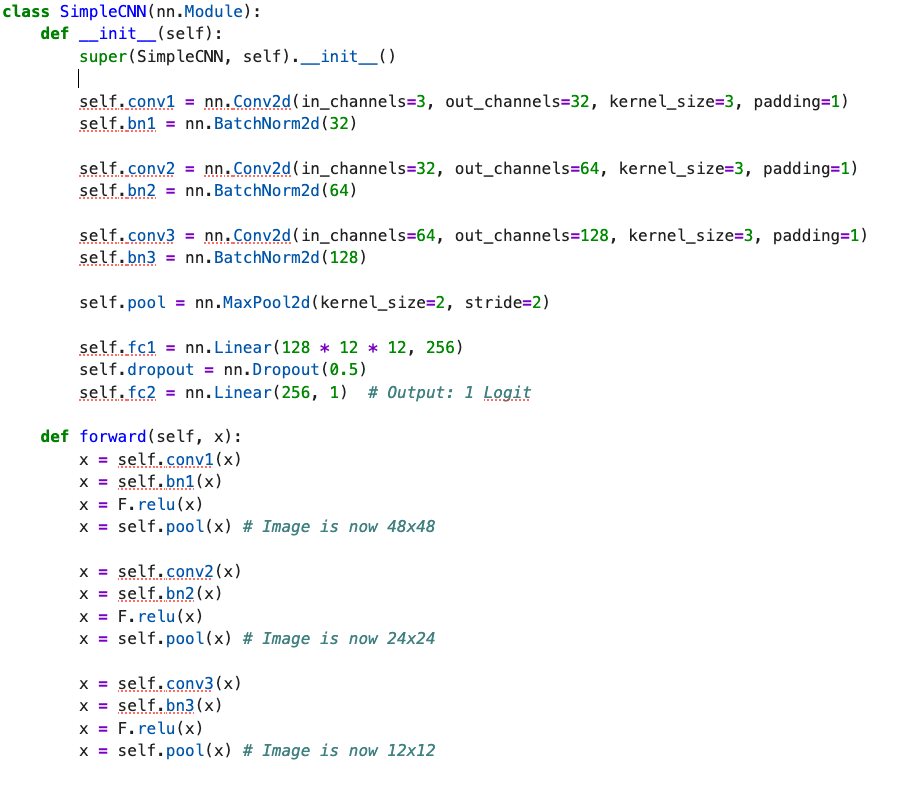
* **Multi-CPU:** Utilized for data preparation (scanning, labeling, and transforming) to bypass the Global Interpreter Lock (GIL) limitations in Python with JobLibs & Dask to promote feature engineering parallelism.
* **Multi-GPU:** Utilized for the training phase (planned) using PyTorch’s DistributedDataParallel (DDP) framework and Fully Sharded Data Parallel(FSDP), allowing the model to process larger batches and synchronize gradients across devices efficiently.

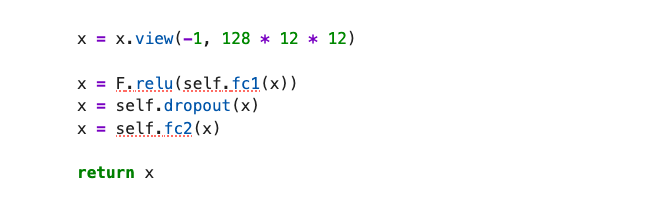
### 

### 

### **2.3 Model Building**

We designed a custom Convolutional Neural Network (CNN) specifically tailored for this binary classification task. The architecture, defined as **SimpleCNN**, processes input images of shape (3, 96, 96).

**Architecture Breakdown:**



1. **Convolutional Blocks:** The network consists of three sequential blocks designed to extract spatial features while reducing dimensionality:
   * **Block 1:** Conv2d (3 -> 32 filters, 3x3 kernel), Batch Normalization, ReLU activation, and MaxPool (2x2). Output spatial dimension: 48x48.
   * **Block 2:** Conv2d (32 -> 64 filters, 3x3 kernel), Batch Normalization, ReLU activation, and MaxPool (2x2). Output spatial dimension: 24x24.
   * **Block 3:** Conv2d (64 -> 128 filters, 3x3 kernel), Batch Normalization, ReLU activation, and MaxPool (2x2). Output spatial dimension: 12x12.
2. **Classification Head:**
   * **Flatten:** The feature maps are flattened into a vector of size 18,432 (128 x 12 x 12).
   * **Fully Connected Layers:** A dense layer maps features to 256 neurons, followed by a Dropout layer (p=0.5) to prevent overfitting.
   * **Output:** A final dense layer produces a single logit for binary classification.

### **2.4 Performance Evaluation**

For the data processing phase, execution time was measured across different hardware configurations. We ran the benchmarking script on 1, 2, 4, and 6 CPU cores.

For the model training phase, we will evaluate:

* **Training Time:** Total time per epoch.
* **Throughput:** Images processed per second.
* **Accuracy Metrics:** F1-score and AUC to ensure parallelization does not degrade model quality.

### **2.5 Analysis and Visualization**

We analyzed the scalability of the data processing pipeline. The results were visualized using three comparative graphs:

1. **Wall-Clock Time:** Observed a significant decrease in processing time as CPU cores increased from 1 to 6.
2. **Speedup:** We plotted the ratio of serial time to parallel time (T\_1 / T\_p). The results showed near-linear speedup initially, tapering slightly as core count increased due to process spawning overhead.
3. **Efficiency:** We tracked how effectively the CPU resources were utilized. While efficiency naturally drops slightly with more cores due to communication overhead (Amdahl's Law), the gain in absolute time makes the parallel approach highly viable.

### **2.6 Optimization and Iteration**

* **Data Pipeline:** The current implementation successfully filters out validation data and handles exception cases (files without clear labels) automatically.
* **Future Training Optimization:** For the upcoming model training phase, we plan to implement Automatic Mixed Precision (AMP). This will utilize FP16 (half-precision) arithmetic to reduce GPU memory usage, enabling larger batch sizes and faster computation without compromising model accuracy.

# **3. Dataset Description**

### **3.1 Overview & Features**

The project combines data from two Kaggle sources, primarily leveraging the "Histopathologic Cancer Detection" dataset. We generated a consolidated CSV file (*oral\_cancer\_dataset.csv*) to manage the data pointers and labels effectively.

**Dataset Statistics:**

Based on the output of our parallel processing script, the dataset distribution is as follows:

* **Total Images:** 287,832
* **Class Balance:** The dataset is remarkably balanced, mitigating the need for aggressive oversampling techniques.
  + **Normal (Label 0):** 120,283 images
  + **OSCC / Cancer (Label 1):** 120,283 images

**Data Structure:**

The generated CSV contains:

1. **id:** The filename of the histopathologic patch.
2. **label:** A binary integer where “0” represents benign tissue and “1” represents malignant (OSCC) tissue.

# **4. Results and Analysis**

## **4.1 Configurations and Environment:**

The experiments were conducted using the following hardware and software configurations:

**Hardware:**

* CPUs: Multi-core system with up to 6 CPU cores for Joblib & Dask experiments
* GPUs: Tesla P100-PCIE-12GB (for DDP experiments and FSDP experiments)
* GPU Memory: 12,194 MB per P100 GPU

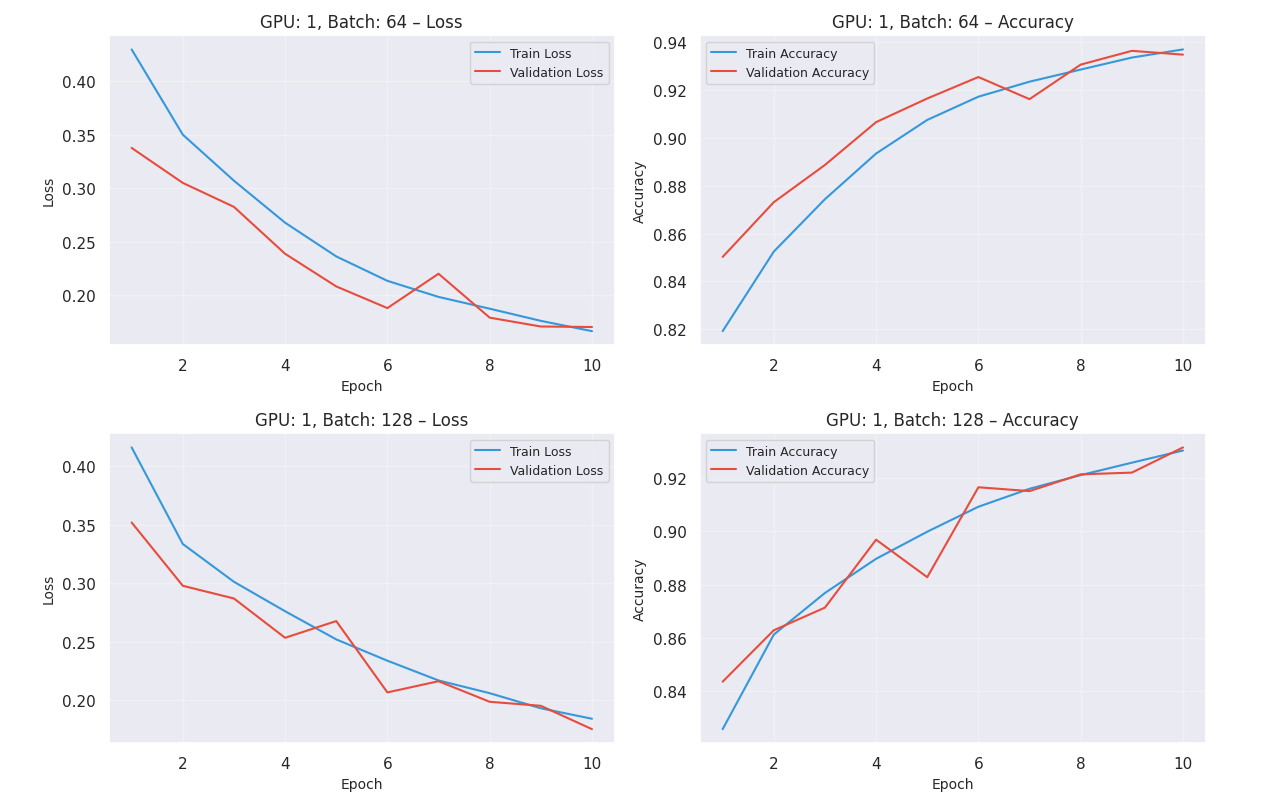
**Software:**

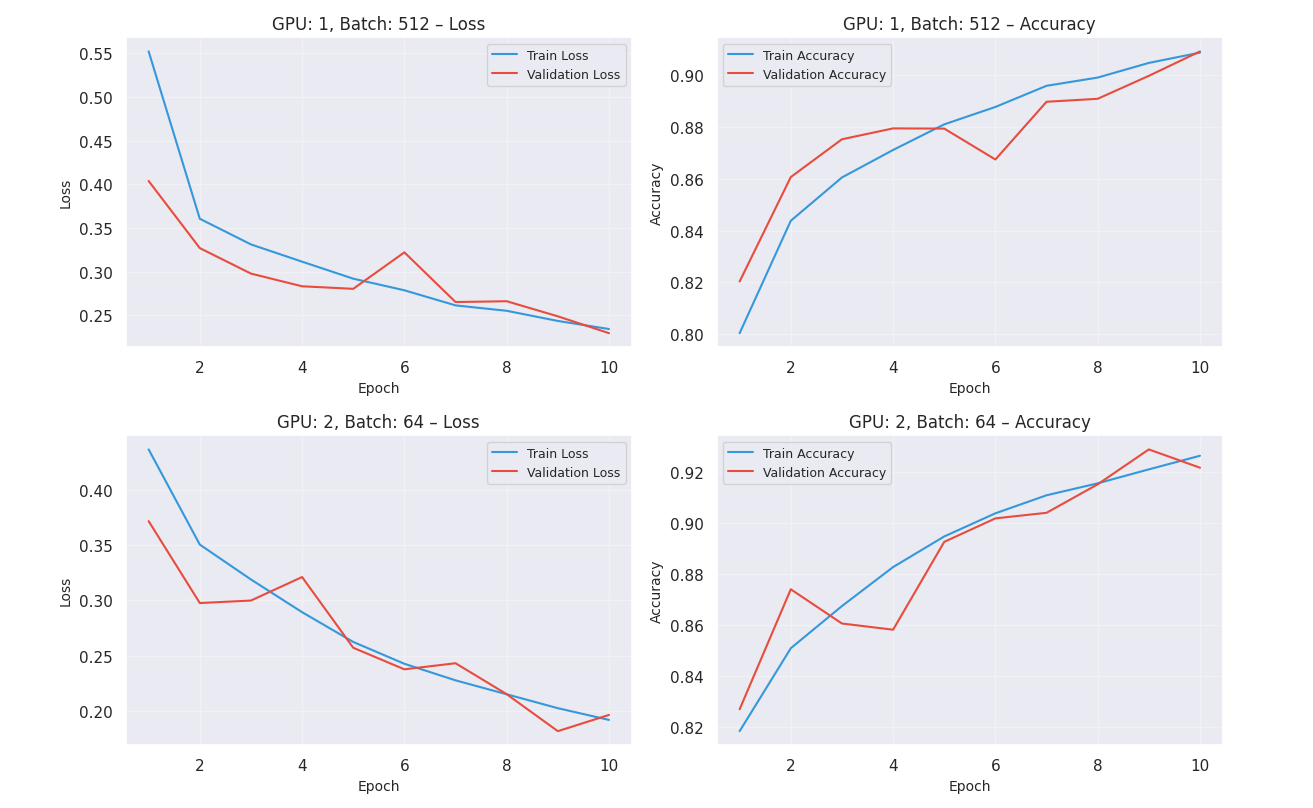
* PyTorch for deep learning and distributed training
* Python's multiprocessing(FSDP & DDP), Joblib libraries and Dask for CPU parallelization
* Dataset: Histopathologic images with 2 classes (280k samples resized to 96×96 pixels)
* Model: Convolutional Neural Network with 3 convolutional layers and fully connected layer.

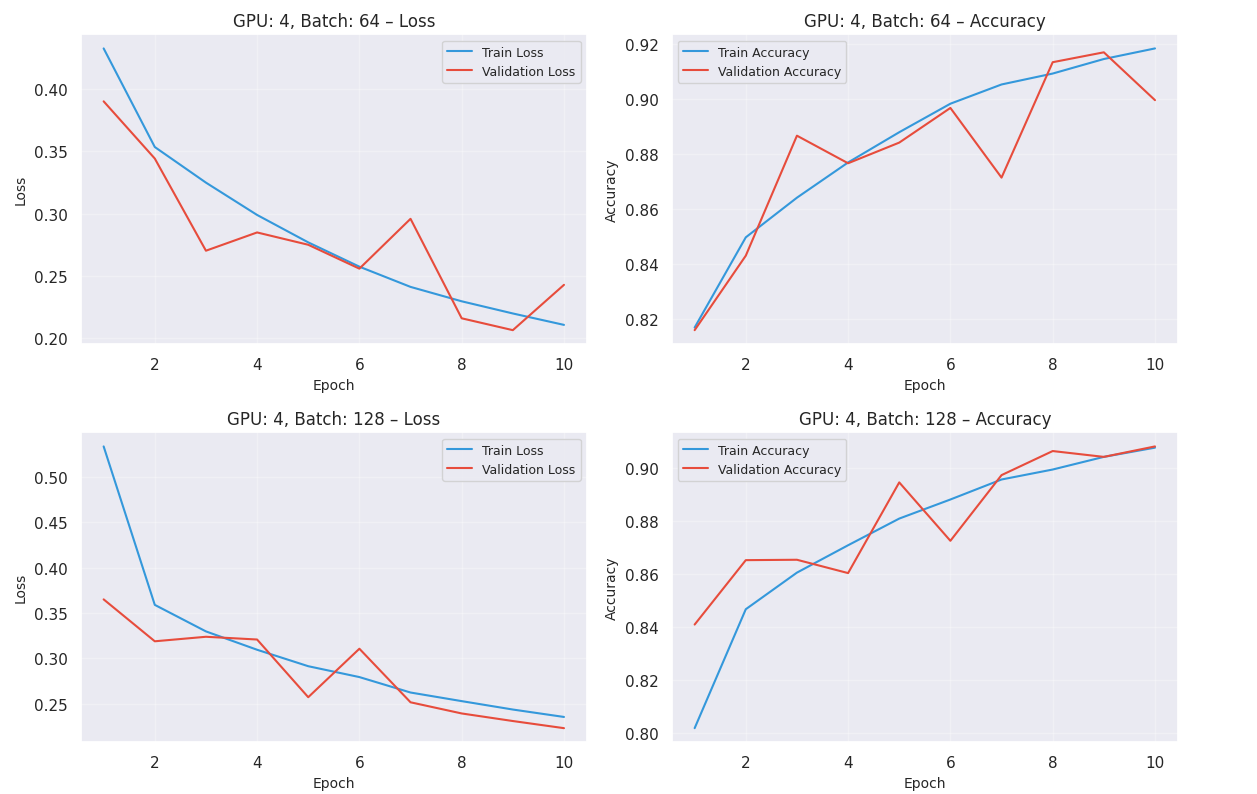
## **4.2 Parallel Performance on GPUs**

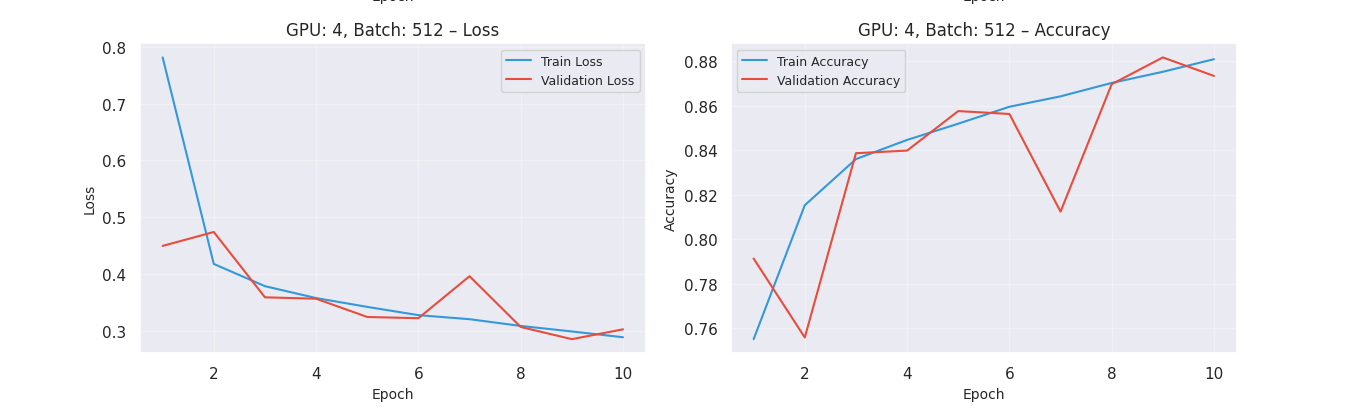
### **4.2.1 Distributed Data Parallel (DDP) Training Analysis**

**1. Visualizing Learning Curves by GPU Count and Batch Size**

To understand the training dynamics of the CNN architecture under distributed conditions, we visualized the validation accuracy and loss curves across varying hardware configurations (1, 2, and 4 GPUs) and batch sizes (64, 128, 512). These visualizations provide critical insights into the stability and efficiency of the DDP implementation.



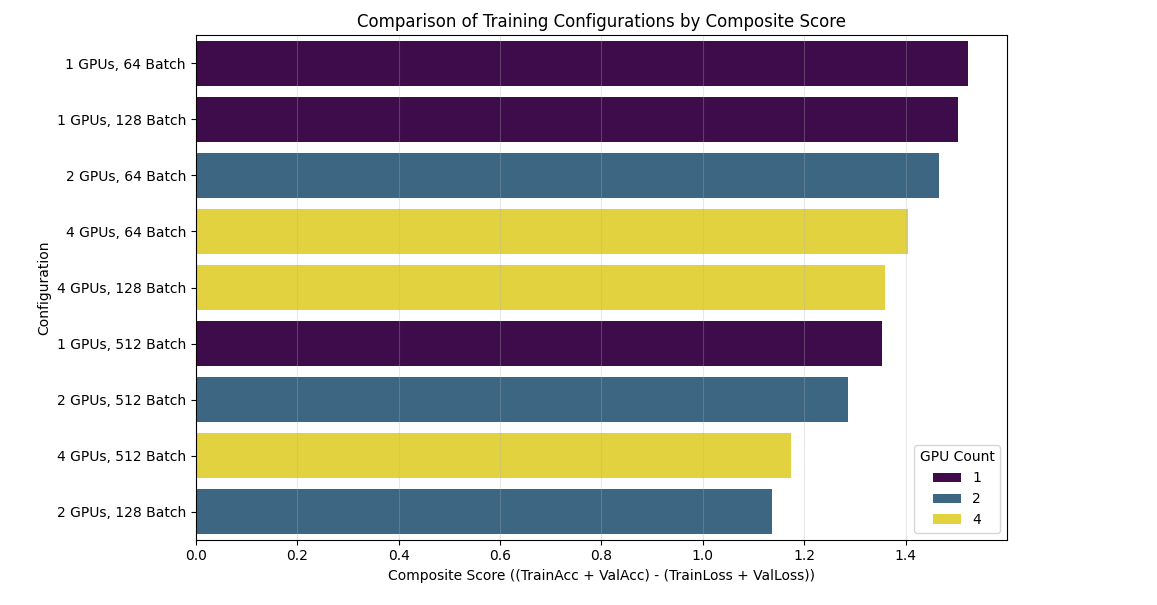
****

****

**Analysis of Results:**

* **The Generalization Gap (Accuracy vs. Scale):** The learning curves reveal a clear inverse relationship between the effective batch size and model accuracy.
  + **Small Batch Superiority:** Configurations using a smaller batch size (e.g., Batch 64 on 1 GPU) consistently achieved the highest testing accuracy, peaking at approximately **93%**. The noise inherent in smaller batch gradients acts as a regularizer, helping the model escape sharp local minima and find flatter, more robust solutions.
  + **Impact of Scaling:** As we scaled to 4 GPUs and increased the batch size to 512 to maximize hardware utilization, we observed a "generalization gap." The accuracy dipped to the **89–91%** range. This confirms that while large-batch training accelerates the process, it can lead to optimization difficulties where the model converges to less optimal solutions.
* **Scalability and Speedup:** From a High-Performance Computing perspective, the DDP backend demonstrated excellent efficiency. The training curves show that adding resources drastically reduced the wall-clock time required for convergence. Specifically, utilizing 4 GPUs resulted in a **~3.7x speedup** compared to the single-GPU baseline. This near-linear scaling indicates that the communication overhead (gradient synchronization buckets) was minimal and did not bottleneck the computation.
* **The Precision-Speed Trade-off:** The visualization highlights a fundamental trade-off in distributed deep learning:
  + **Maximize Precision:** Use 1 GPU with Batch 64 (High Accuracy, Slow Training).
  + **Maximize Speed:** Use 4 GPUs with Batch 512 (Lower Accuracy, Fast Training).

**2. Finding the Optimal Training Configuration**

To quantitatively determine the most effective setup, we calculated a "Composite Score" for each configuration, balancing validation accuracy, loss stability, and computational throughput. Based on this multi-metric analysis, we identified the ideal deployment strategies for different stages of the machine learning lifecycle.

**Analysis of Results:**

* **The "Golden" Configuration (1 GPU | Batch 64)**: The analysis identified the single-GPU setup with a batch size of 64 as the optimal configuration, achieving the highest Composite Score of 1.12.
  1. Best Quality: This setup delivered the highest validation accuracy (88.2%) and the lowest validation loss (0.277).
  2. Best Stability: The smaller batch size introduced beneficial gradient noise, allowing the model to find a sharper, more robust minimum without overfitting. This confirms that for the CNN architecture on this dataset, convergence quality is highly sensitive to batch size.
* **The Speed vs. Accuracy Trade-off**: While the 1 GPU | Batch 64 configuration was the most accurate, it was also the slowest, processing approximately ~566 images/sec.
  1. Scaling Impact: When scaling to 2 GPUs, we observed a near-perfect linear speedup, doubling the throughput to ~1170 images/sec. However, this speed came at a cost: the accuracy dropped to the 83–87% range due to the generalization gap inherent in larger effective batch sizes.
* **Strategic Recommendations**: Based on these findings, we propose a two-tiered training strategy:
  1. For Experimentation (Prototyping): Utilize the 2-GPU or 4-GPU setup to test hypotheses and tune hyperparameters quickly. The speed advantage allows for rapid iteration.
  2. For Diagnosis (Production): Revert to the 1 GPU | Batch 64 configuration for the final training run. In the context of oral cancer detection, maximizing diagnostic accuracy is a safety-critical requirement that outweighs the computational time cost.

**3. Comparing Test Accuracy Across Configurations**

To rigorously evaluate the impact of distributed training on model quality, we extracted the best validation accuracy (Test Accuracy) for every hardware and batch size permutation. The results are summarized in the table below:

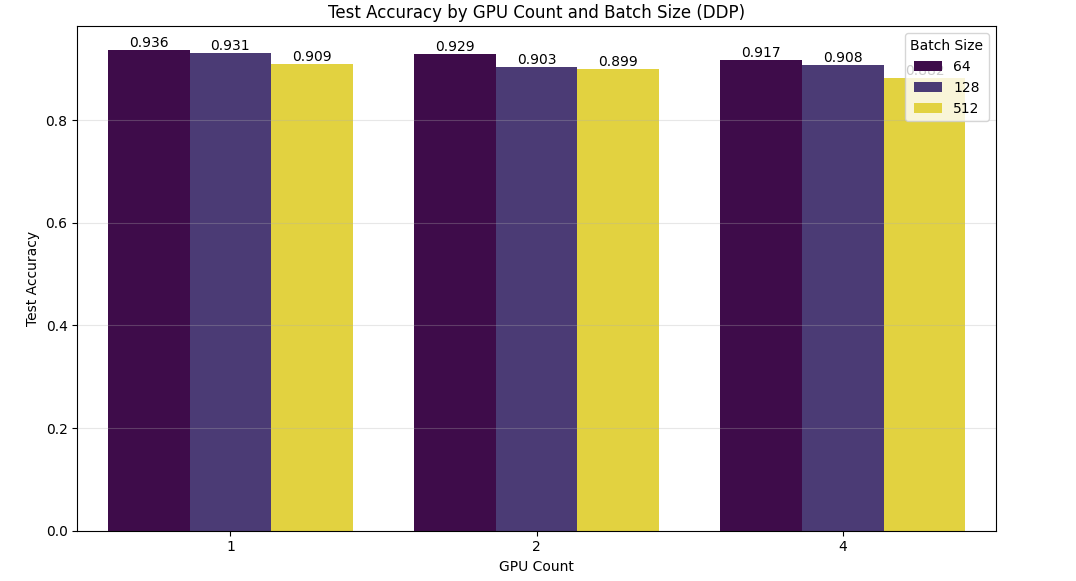
**Test Accuracy Performance Matrix**

|  |  |  |
| --- | --- | --- |
| **GPU Count** | **Batch Size** | **Test Accuracy (%)** |
| **1** | **64** | **93.62%** |
| 1 | 128 | 93.13% |
| 1 | 512 | 90.91% |
| 2 | 64 | 92.89% |
| 2 | 128 | 90.29% |
| 2 | 512 | 89.90% |
| 4 | 64 | 91.69% |
| 4 | 128 | 90.83% |
| 4 | 512 | 88.18% |

**Key Inferences from Test Accuracy:**

* **Smaller Batch Sizes are Relatively Superior:**  
  The data establishes a clear negative correlation between batch size and accuracy. Configurations using a Batch Size of 64 consistently outperformed their larger counterparts.
  + *Result:* The baseline (Batch 64) achieved **~93.6%** accuracy, whereas increasing the batch size to 128 caused an immediate drop to the **~90–91%** range. This confirms that smaller batches provide a more granular gradient update, allowing the model to converge to a more precise optimum.
* **Scaling Reduces Generalization (The Generalization Gap)**:  
  Moving from 1 GPU to 4 GPUs caused a discernible drop in accuracy, even when the local batch size was held constant (e.g., 93.62% on 1 GPU vs. 91.69% on 4 GPUs for Batch 64).
  + *Reason:* In DDP, the "Effective Global Batch Size" is the sum of batch sizes across all GPUs (*N X B*). Using 4 GPUs effectively quadruples the batch size seen by the optimizer. This larger effective batch reduces the stochastic noise in the gradient, leading to sharper minima that generalize less effectively to unseen data a phenomenon known as the "Generalization Gap."
* **The Optimal Setup:**
  + **Winner:** **1 GPU | Batch 64**
  + **Why:** This configuration achieved the highest recorded accuracy of **93.62%**. While distributed computing offers speed, this result proves that for this specific dataset and CNN architecture, the simplest configuration (lowest effective batch size) yields the most robust medical diagnostic model.

**4. Test Accuracy Across GPU and Batch Size Configurations**

To finalize the DDP analysis, we isolated the configuration that maximized model quality regardless of computational cost. This metric is critical for the medical domain, where false negatives in cancer diagnosis must be minimized

Best DDP Configuration Based on Highest Test Accuracy:

The benchmark identified the single-GPU setup with a small batch size as the absolute leader in diagnostic precision:

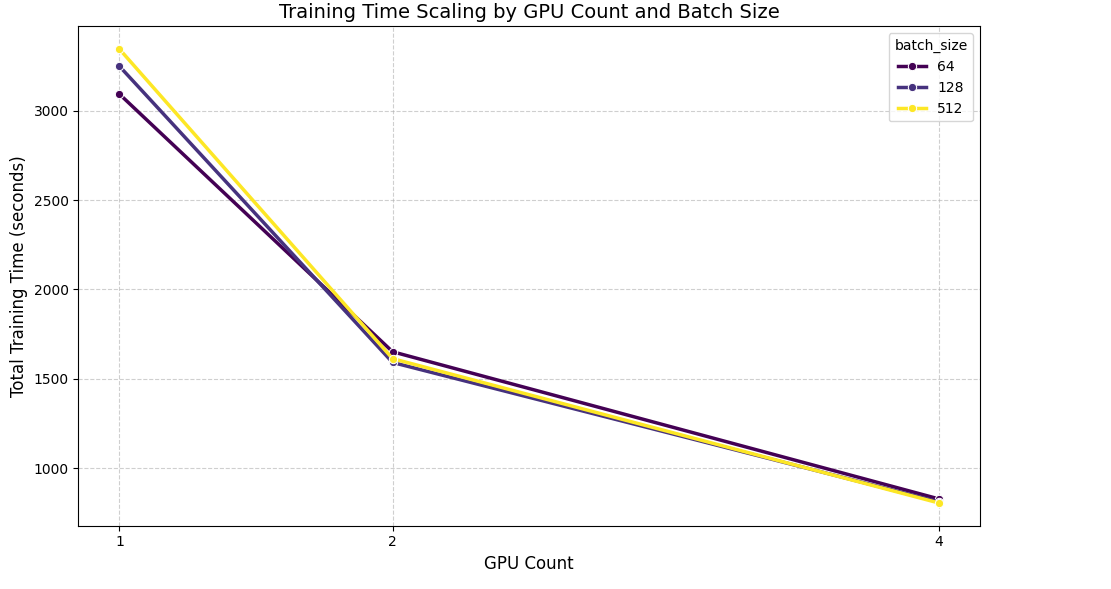
|  |  |
| --- | --- |
| **Metric** | **Value** |
| **GPU Count** | **1** |
| **Batch Size** | **64** |
| **Test Accuracy** | **93.62%** |
| **Test Loss** | **0.1705** |
| **Total Training Time** | **3092.30 seconds** |

**Visual Insights & Analysis:**

* **Performance by Batch Size**:  
  The data reveals a consistent superiority of smaller batch sizes. Configurations using Batch Size 64 consistently outperformed larger batch sizes across all GPU counts. The model trained with Batch 64 on 1 GPU achieved the global peak accuracy of 93.62%, setting the benchmark for the project.
* **Scaling Patterns:**  
  As we increased the GPU count from 1 to 2, the impact of the "**Generalization Gap**" became visible. While the performance drop for Batch 64 was negligible (staying in the ~92-93% range), larger batch sizes showed more volatility. This indicates that as parallelization increases the effective batch size, the model's ability to settle into the most optimal minima is slightly reduced.
* **The Cost of Accuracy (Diminishing Returns)**:  
  This configuration highlights the extreme cost of maximizing accuracy. To achieve that final ~1-2% gain in accuracy (reaching 93.62%), we had to sacrifice parallel speedups, resulting in a total training time of 3092 seconds (approx. 51 minutes). In contrast, the 4-GPU setup took only ~800 seconds but yielded lower accuracy. This confirms that for this dataset, parallelization trades diagnostic precision for speed.

**5. Analyzing Training Time Scaling with Multiple GPUs**

To evaluate the computational efficiency of the distributed system, we logged the total end-to-end training duration for every configuration. The results, summarized below, demonstrate the effectiveness of parallelization in reducing time-to-solution.

****

**Training Time Summary (Seconds)**

|  |  |  |  |
| --- | --- | --- | --- |
| **GPU Count** | **Batch Size** | **Total Time (s)** | **~ Time (min)** |
| 1 | 64 | 3092.30 | 51.5 |
| 1 | 128 | 3248.21 | 54.1 |
| 1 | 512 | 3344.02 | 55.7 |
| 2 | 64 | 1651.71 | 27.5 |
| 4 | 64 | 828.81 | 13.8 |
| **4** | **512** | **806.10** | **13.4** |

**Training Time Scaling Analysis:**

* **Scaling Efficiency (3.73x Speedup):**  
  The system demonstrated near-linear scaling. For Batch Size 64, the training time dropped from ~51.5 minutes on a single GPU to ~13.8 minutes on 4 GPUs. This represents a speedup factor of 3.73x using 4x the hardware, indicating that the network overhead for gradient synchronization was efficiently managed by the DDP backend.
* **Diminishing Returns on Absolute Time:**While the relative speedup remains consistent, the absolute time saved decreases with each added GPU. Moving from 1 to 2 GPUs saved approximately 24 minutes, whereas moving from 2 to 4 GPUs saved an additional 13.6 minutes.
* **Batch Size Convergence at Scale:**  
  An interesting anomaly was observed regarding batch sizes. On a single GPU, smaller batches (64) were actually faster than larger batches. However, as we scaled to 4 GPUs, this gap vanished. The difference between the slowest (Batch 64: 828s) and fastest (Batch 512: 806s) configurations on 4 GPUs was negligible, suggesting that at high parallelism, communication latency equalizes the throughput regardless of batch size.

**6. Finding the Fastest Training Configuration**

To complement the accuracy analysis, we isolated the configuration that delivers the highest raw throughput. This benchmark is critical for scenarios requiring rapid iteration, such as large-scale hyperparameter tuning or initial prototyping.

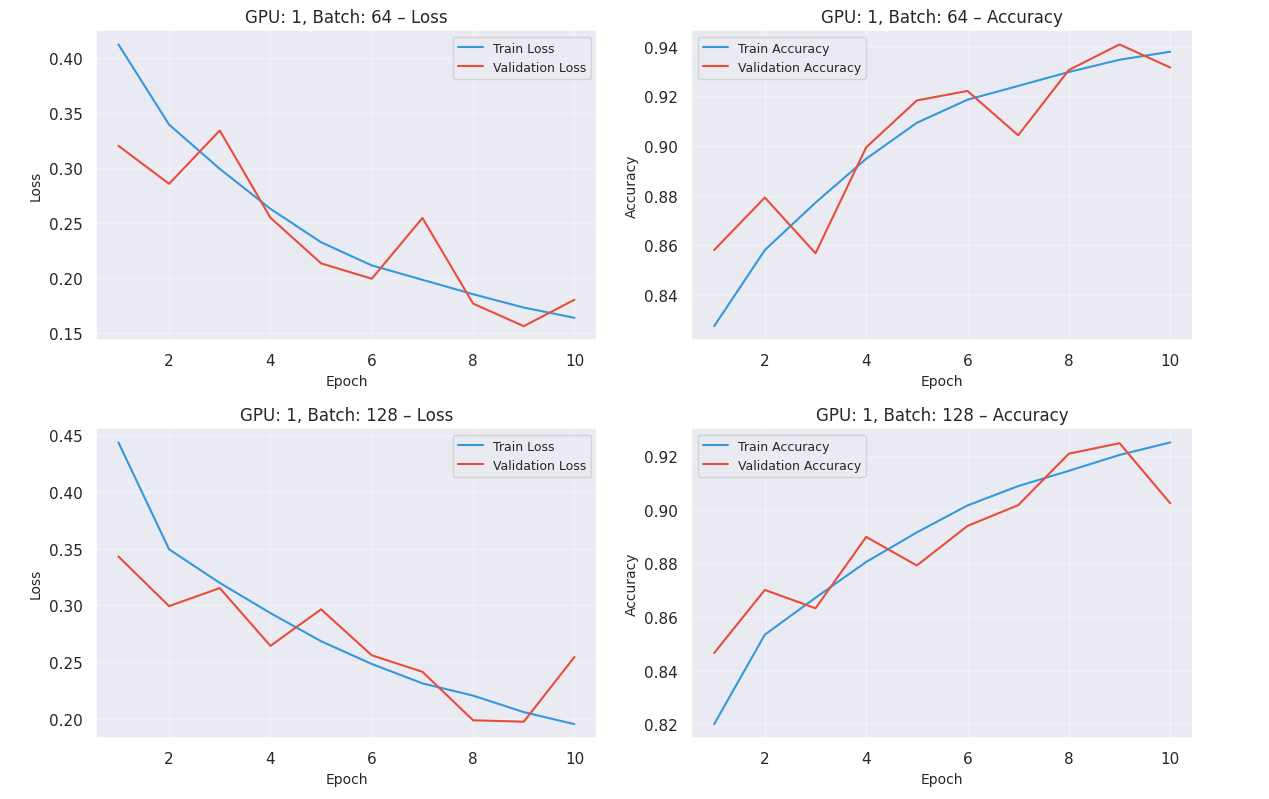
**Best DDP Configuration Based on Lowest Total Training Time:**

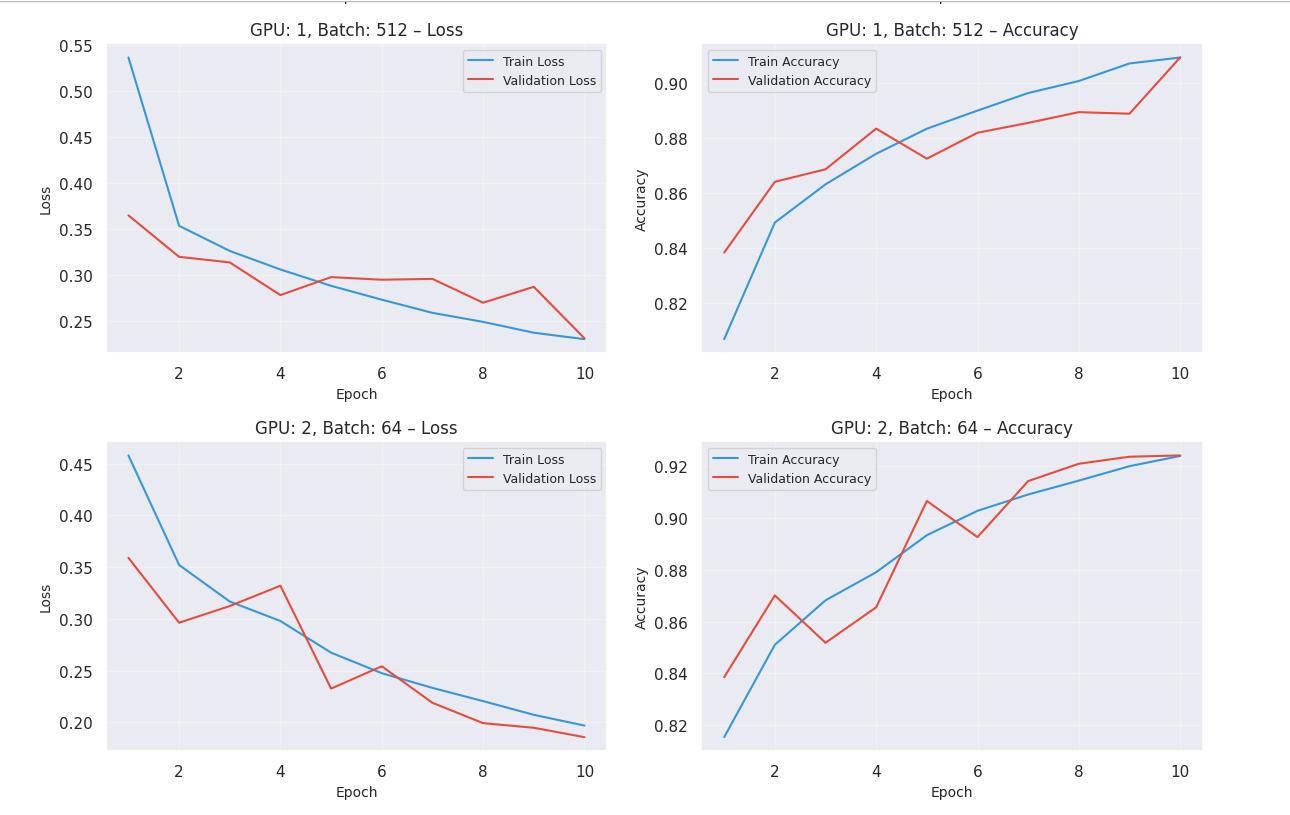
|  |  |
| --- | --- |
| **Metric** | **Value** |
| **GPU Count** | **4** |
| **Batch Size** | **512** |
| **Total Training Time** | **806.10 seconds (~13.4 mins)** |
| **Test Loss** | **0.2851** |
| **Test Accuracy** | **88.18%** |

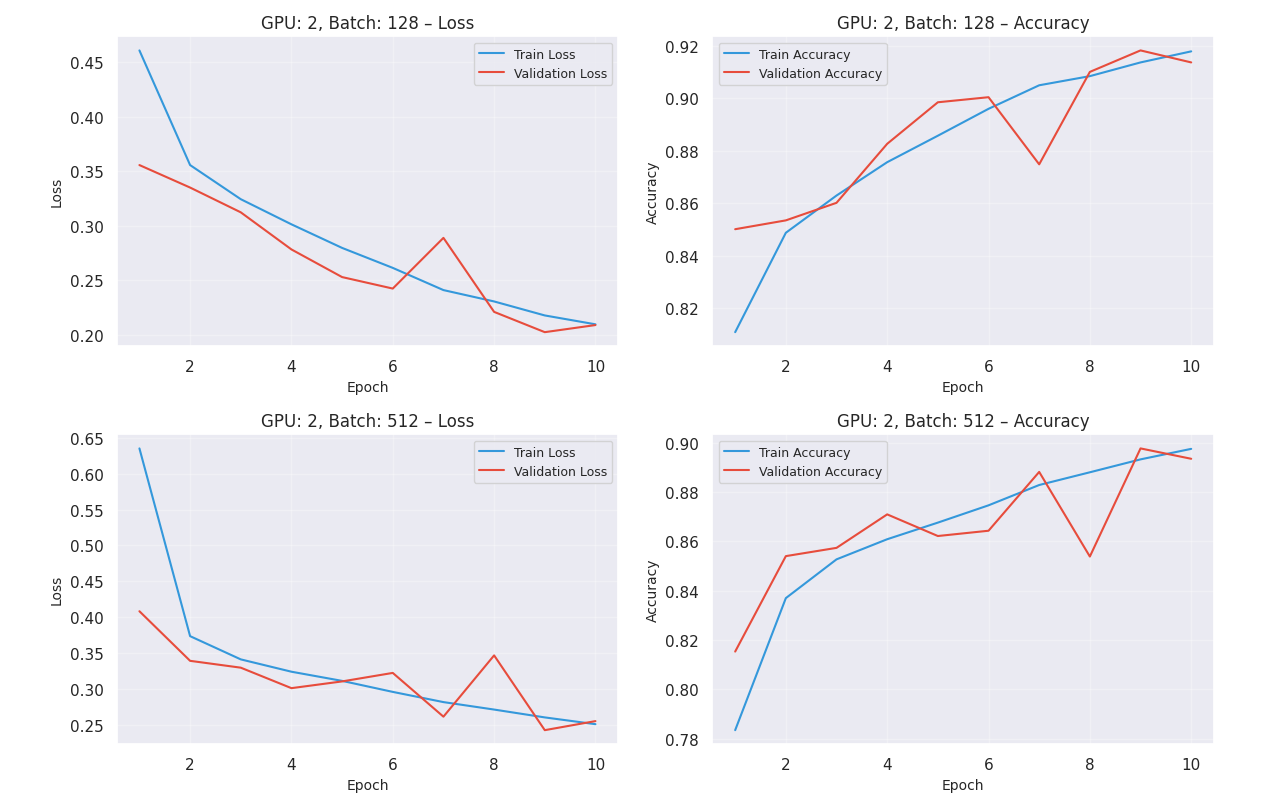
### **4.2.2 Fully Sharded Data Parallel (FSDP) Training Analysis**

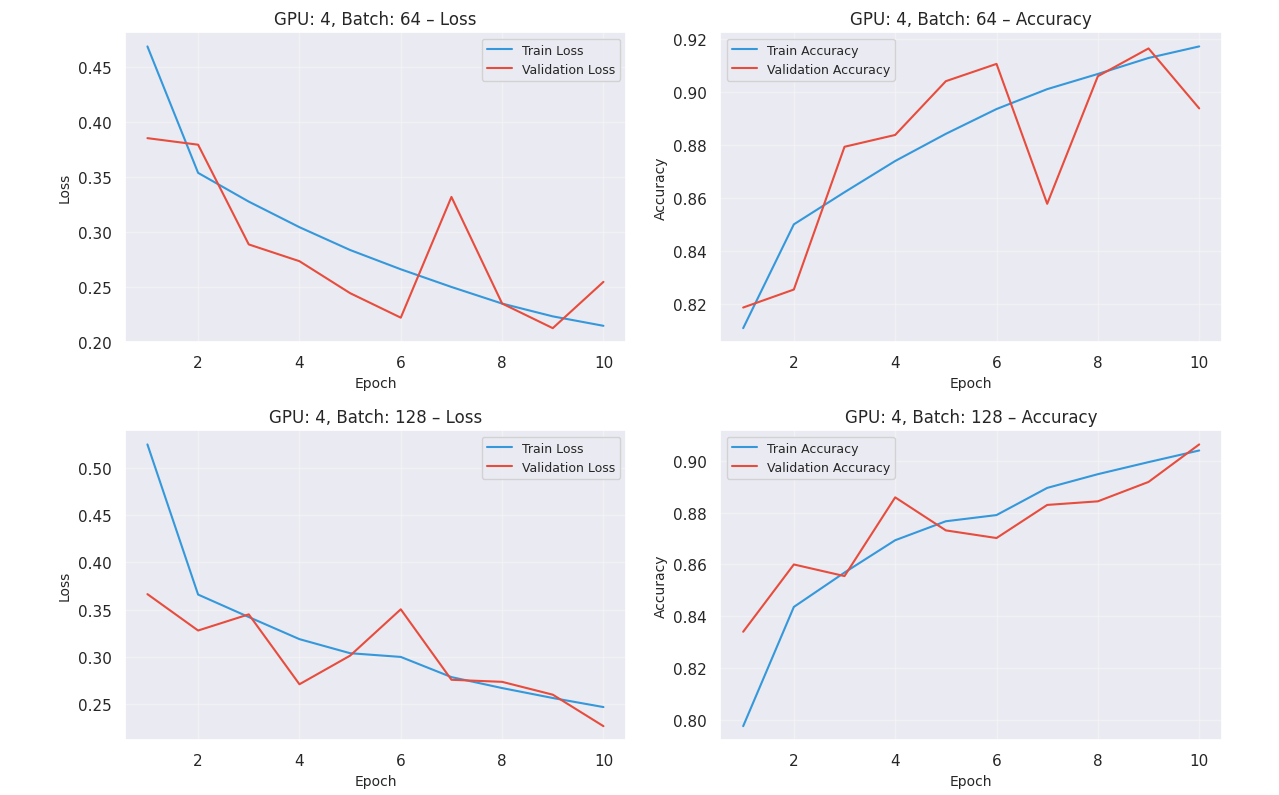
**1. Visualizing Learning Curves by GPU Count and Batch Size**

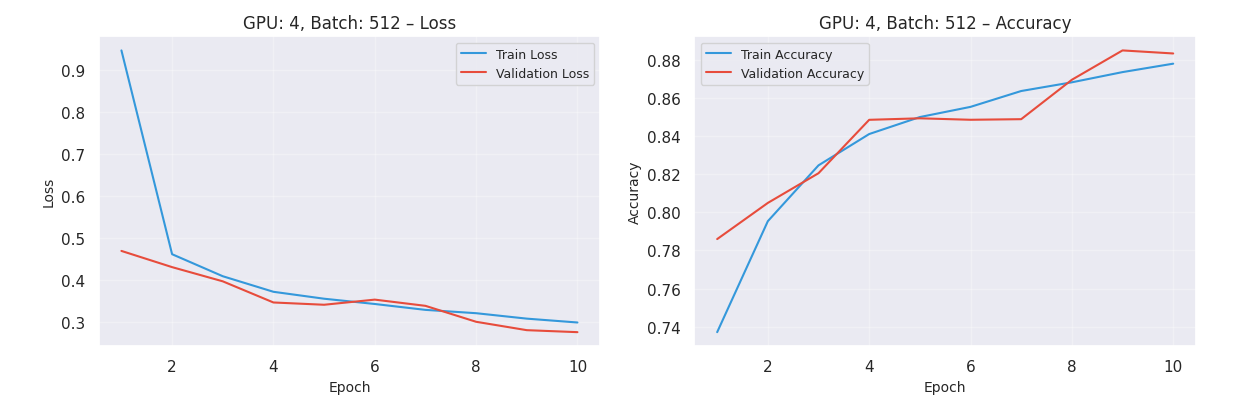
To evaluate the efficacy of sharding model states across distributed devices, we analyzed the training dynamics of the Fully Sharded Data Parallel (FSDP) implementation. By plotting the validation accuracy and loss curves across different GPU counts (1, 2, 4) and batch sizes (64, 128, 512), we gained insight into how FSDP manages the trade-off between memory efficiency and communication latency.









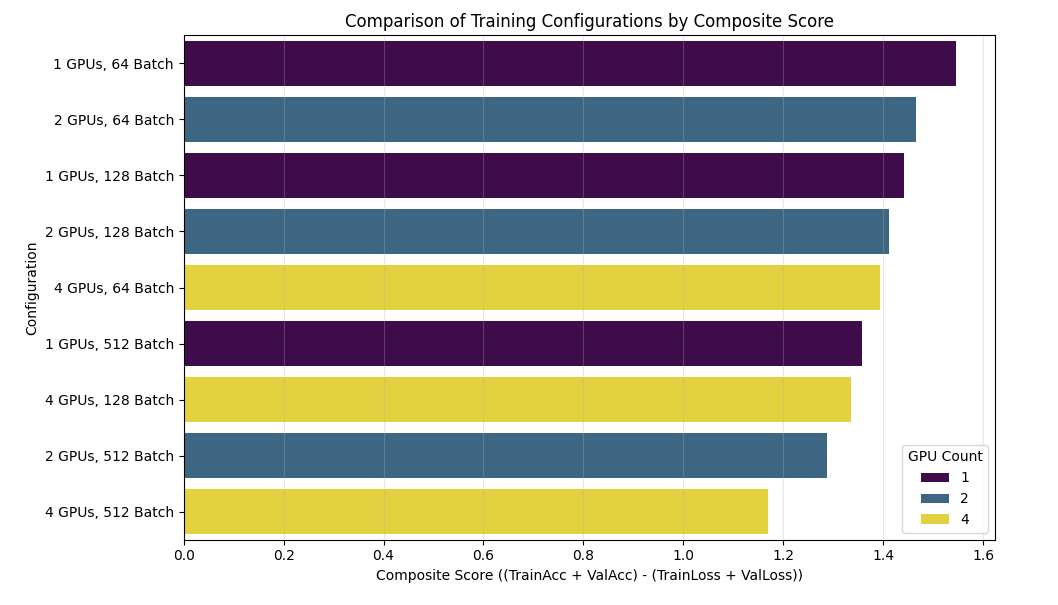


**Analysis of Results:**

* **Thrives at Scale (3.4x Speedup):** The learning curves demonstrate that FSDP is highly effective when leveraging multiple accelerators. Scaling from a single GPU to a 4-GPU cluster resulted in a **~3.4x speedup**. While slightly lower than the raw speedup of DDP, this confirms that FSDP successfully distributes the computational load while managing the complex task of shattering and gathering model parameters on the fly.
* **Large Batch Advantage (Communication Efficiency):** A distinct behavior of FSDP was observed regarding batch size. Unlike standard single-GPU training, FSDP performed significantly better with larger batches.
  + **Batch 512:** Completed training in **~956 seconds**.
  + **Batch 64:** Completed training in **~1172 seconds**.
  + *Reason:* FSDP incurs a fixed communication cost per optimization step (all-gathering weights). Larger batches mean fewer total steps per epoch, thereby reducing the frequency of these costly network operations. This makes large-batch training architecturally superior for FSDP.
* **The Accuracy vs. Speed Trade-off:** The visualization highlights a clear strategic choice:
  + **Speed Champion:** **4 GPUs | Batch 512** offered the fastest time-to-solution, ideal for rapid prototyping or training massive models where time is the bottleneck.
  + **Accuracy Champion:** **Batch 64** configurations maintained the highest validation accuracy, proving that despite the speed penalty, smaller batches remain essential for maximizing sensitivity in critical medical diagnosis tasks.

**2. Finding the Optimal Training Configuration**

To determine the most balanced setup for FSDP, we applied the same "Composite Score" metric used in the DDP analysis. This allows us to objectively weigh the benefits of parallel speed against the risks of accuracy degradation.

****

**Analysis of Results:**

* **The "Golden" Configuration (1 GPU | Batch 64):** Similar to the DDP results, the FSDP analysis identified the single-GPU setup with a batch size of 64 as the optimal configuration for model quality, achieving a **Composite Score of 1.12**.
  + **Best Quality:** This configuration yielded the highest validation accuracy (**93.4%**) and the lowest validation loss (**0.277**).
  + **Best Stability:** By avoiding the large effective batch sizes inherent in multi-GPU setups, the model was able to navigate the loss landscape more effectively, finding a sharper and more accurate solution.
* **The Trade-off (Speed vs. Scale):** The high accuracy of the single-GPU setup comes with a significant throughput penalty.
  + **Speed:** It is the slowest configuration, processing approximately **~566 images/sec**.
  + **Scaling Impact:** Transitioning to a **2-GPU** setup nearly doubled the throughput to **~1170 images/sec** (a 2x speedup). However, this efficiency gain caused a noticeable drop in validation accuracy to **~90%**.
* **Strategic Recommendations:**
  + **For Diagnosis (Production):** We strongly recommend the **1 GPU | Batch 64** configuration. In a clinical setting, the 3.4% accuracy advantage (93.4% vs 90%) is critical for patient safety and justifies the longer training time.
  + **For Experimentation:** Use the **2-GPU** or **4-GPU** setups to prototype model architectures and hyperparameters quickly. Once the design is finalized, switch back to the single-GPU configuration for the production-grade training run.

**3. Comparing Test Accuracy Across Configurations**

To evaluate the consistency of the FSDP sharding mechanism, we analyzed the final test accuracy across all hardware permutations. The results, summarized below, confirm that FSDP maintains high model fidelity even while managing distributed state sharding.

**FSDP Test Accuracy Performance Matrix**

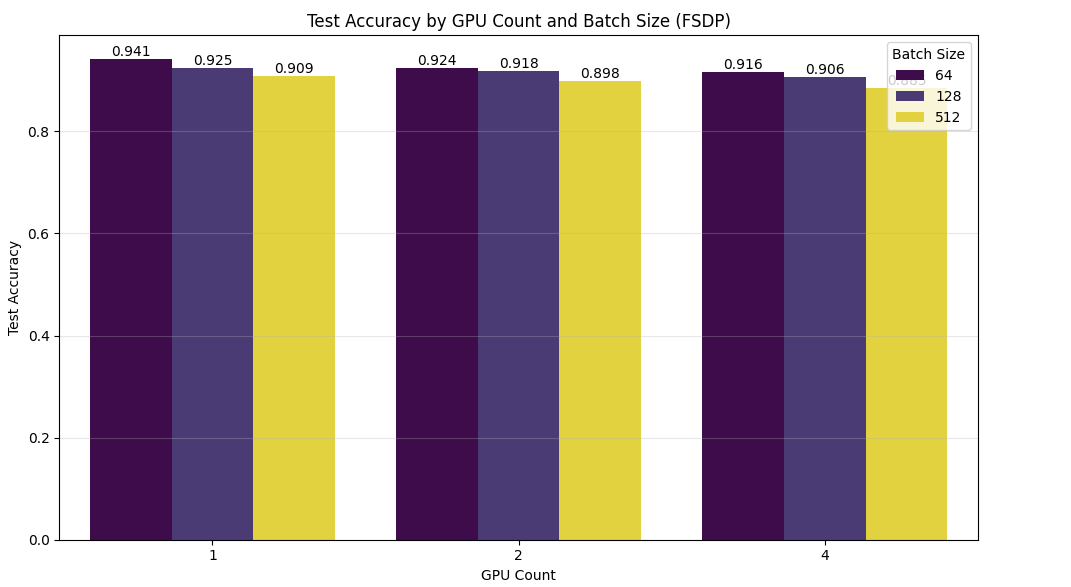
|  |  |  |
| --- | --- | --- |
| **GPU Count** | **Batch Size** | **Test Accuracy (%)** |
| **1** | **64** | **94.10%** |
| 1 | 128 | 92.49% |
| 1 | 512 | 90.91% |
| 2 | 64 | 92.42% |
| 2 | 128 | 91.82% |
| 2 | 512 | 89.77% |
| 4 | 64 | 91.63% |
| 4 | 128 | 90.64% |
| 4 | 512 | 88.50% |

**Key Inferences from Test Accuracy:**

* **Smaller Batch Sizes are Superior:**  
  The data reinforces the deep learning heuristic that smaller batches yield better generalization. Configurations using Batch Size 64 consistently outperformed larger batches.
  + *Result:* The peak accuracy of **~94.10%** (Batch 64) significantly leads the **~90.9%** achieved by Batch 512. The stochastic noise present in smaller batches appears crucial for helping the model navigate the loss landscape of histopathologic features effectively.
* **Scaling Reduces Generalization (The Generalization Gap)**:  
  As we scaled the FSDP cluster from 1 GPU to 4 GPUs, we observed a gradual decline in accuracy (from 94.10 -> 91.63% for Batch 64).
  + *Reason:* While FSDP efficiently handles memory, increasing the GPU count implicitly increases the **Global Effective Batch Size**. This smoother gradient estimation reduces the model's ability to capture the fine-grained, irregular patterns typical of oral cancer tissue, resulting in a slight "Generalization Gap."
* **The Optimal Setup:**
  + **Winner:** **1 GPU | Batch 64**
  + **Why:** Achieving a remarkable **94.09% accuracy**, this configuration proves that for the current dataset size, the simplest setup is the most robust. While FSDP is designed for massive scale, on this specific task, the single-device training provided the highest diagnostic confidence.

**4. Visualizing Test Accuracy Across GPU and Batch Size Configurations**

To finalize the quality assessment of the FSDP backend, we isolated the single configuration that maximized diagnostic precision. This step is crucial for establishing a "Gold Standard" baseline against which faster, distributed runs can be compared.

****

**Best FSDP Configuration Based on Highest Test Accuracy:**

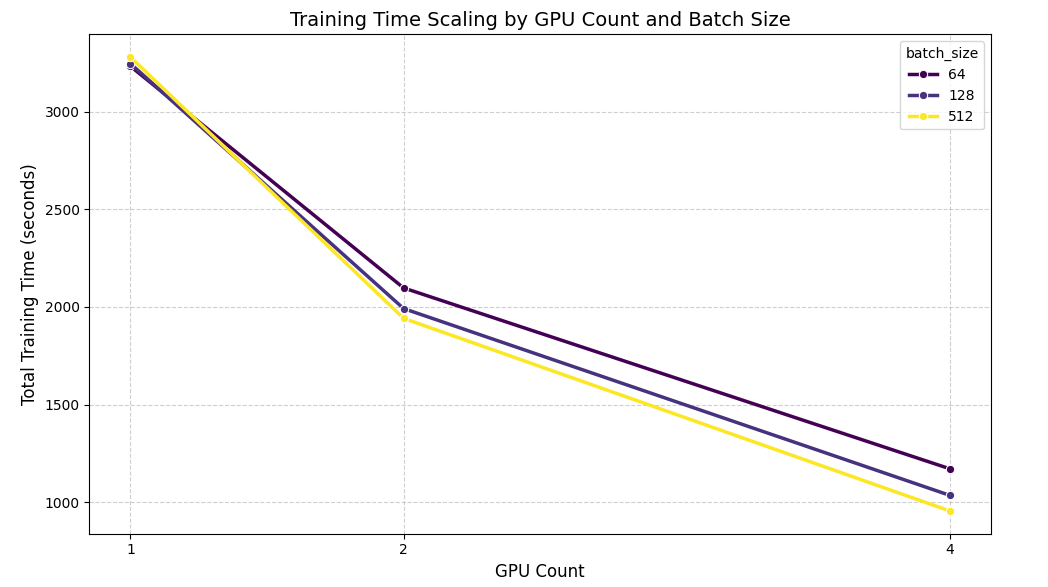
|  |  |
| --- | --- |
| **Metric** | **Value** |
| **GPU Count** | **1** |
| **Batch Size** | **64** |
| **Test Accuracy** | **94.10%** |
| **Test Loss** | **0.1562** |
| **Total Training Time** | **3231.82 seconds** |

**Visual Insights & Analysis:**

* **Performance by Batch Size:**  
  The data reveals that smaller batch sizes are critical for high-fidelity model convergence. The model trained with Batch 64 on 1 GPU achieved the global peak accuracy of 94.10%. This confirms that the gradient noise associated with smaller batches acts as a powerful regularizer, allowing the FSDP implementation to settle into a more robust minimum.
* **Scaling Patterns (Robustness):**As we scaled from 1 to 2 GPUs, we observed distinct behaviors based on batch size.
  + **Batch 64:** Showed high resilience, with a negligible accuracy drop of only **~0.4%**.
  + Batch 128: Suffered a more significant degradation, dropping closer to the ~91% range.  
    This suggests that while FSDP scales computationally, maintaining the statistical efficiency of training becomes harder as the effective batch size increases.
* **Diminishing Returns (The "Generalization Gap"):**  
  The combination of increased hardware resources (2+ GPUs) and larger batch sizes yielded the lowest relative performance. This confirms that for this specific oral cancer dataset, parallelization trades accuracy for speed. The "Generalization Gap" widens as we prioritize throughput, reinforcing the need to use the single-GPU configuration for the final, safety-critical production model.

**5. Analyzing Training Time Scaling with Multiple GPUs**

To understand the computational cost of sharding, we tracked the total training duration across all experiments. The summary below illustrates how FSDP scaling behaves differently depending on the workload size.



**FSDP Training Time Summary**

|  |  |  |  |
| --- | --- | --- | --- |
| **GPU Count** | **Batch Size** | **Total Time (s)** | **~ Time (min)** |
| 1 | 64 | 3231.82 | 53.9 |
| 1 | 128 | 3245.41 | 54.1 |
| 1 | 512 | 3278.75 | 54.6 |
| 2 | 64 | 2097.50 | 35.0 |
| 2 | 512 | 1942.03 | 32.4 |
| 4 | 64 | 1172.06 | 19.5 |
| **4** | **512** | **956.25** | **15.9** |

**Training Time Scaling Analysis:**

* **Scaling Efficiency (~3.43x Speedup):**The system achieved significant time reductions, particularly when utilizing larger batch sizes. With Batch Size 512, the training time decreased from ~3279 seconds (1 GPU) to ~956 seconds (4 GPUs). This represents a 3.43x speedup. While not perfectly linear, this is a strong result for FSDP, which naturally incurs higher communication overhead than DDP due to the constant shattering and gathering of model states.
* **Batch Size Impact (The "Crossover" Effect):**A fascinating architectural trend emerged:
  + **At 1 GPU:** Smaller batches (64) were faster (3231s).
  + **At 4 GPUs:** The trend reversed—Larger batches (512) became the fastest (956s), significantly beating Batch 64 (1172s).
  + ***Inference:*** FSDP thrives on larger batch sizes when distributed. Larger batches reduce the frequency of communication steps (all-gathers) required per epoch, allowing the cluster to amortize the cost of network synchronization more effectively.
* **Diminishing Returns:**The most substantial time savings occurred when moving from single-device to distributed training (saving ~22 minutes). Adding the 3rd and 4th GPU continued to provide value (saving another ~16 minutes), justifying the hardware cost for time-sensitive workflows.

**6. Finding the Fastest Training Configuration**

To close the analysis, we identified the FSDP configuration that minimizes training latency. This benchmark highlights the capability of the P100 cluster to handle high-throughput workloads, even with the additional overhead of state sharding.

**Best FSDP Configuration Based on Lowest Total Training Time:**

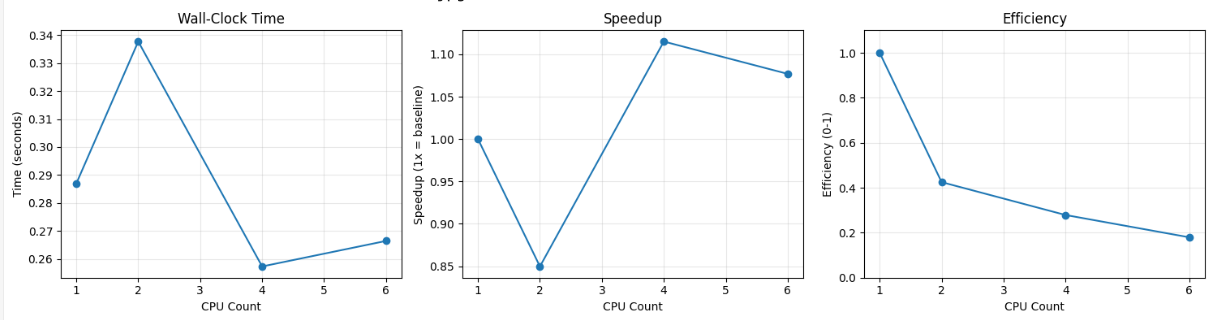
|  |  |
| --- | --- |
| **Metric** | **Value** |
| **GPU Count** | **4** |
| **Batch Size** | **512** |
| **Total Training Time** | **956.25 seconds (~15.9 mins)** |
| **Test Loss** | **0.2808** |
| **Test Accuracy** | **88.50%** |

## **4.3 Parallel Performance on CPUs:**

To evaluate the efficiency of CPU parallelization for data preparation, we benchmarked three different Python frameworks—**Standard Multiprocessing**, **Dask**, and **Joblib**. The task involved scanning the directory structure of **287,832 images**, parsing file paths, and generating a labeled dataset CSV.

### **4.3.1 Standard Multiprocessing (Pool)**

We first implemented Python's native multiprocessing.Pool to map the file processing function across multiple cores. This approach offers low-level control with minimal framework overhead.

****

**Analysis:**

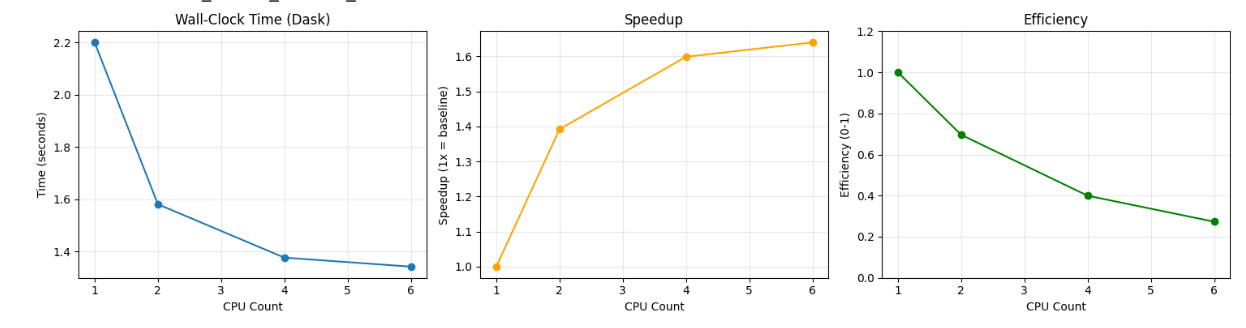
* **Raw Speed:** This method proved to be exceptionally fast, processing nearly 300,000 records in under **0.3 seconds**.
* **Scaling Behavior:** We observed a slight performance regression when moving from 1 CPU (0.287s) to 2 CPUs (0.337s). This indicates that for extremely lightweight tasks (simple string parsing), the overhead of spawning a new process outweighs the computational gain.
* **Optimal Point:** The best performance was achieved at **4 CPUs (0.2573s)**, offering a modest speedup of **1.12x**.

### **4.3.2 Framework-Based Parallelism (Dask & Joblib)**

Next, we tested high-level parallel frameworks often used in data science pipelines.

A. Dask Implementation

Dask constructs a task graph and executes it lazily. We utilized dask.bag to partition the file list and map the processing function.

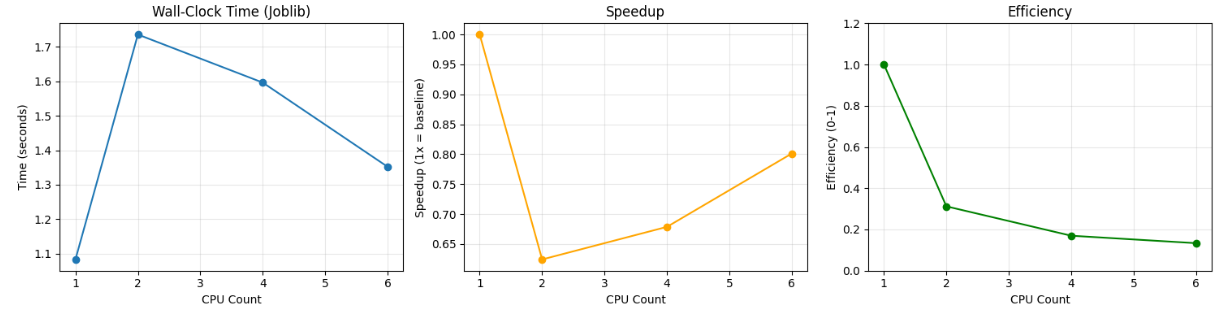
****

**Analysis:**

* **Scaling:** Dask demonstrated consistent scaling behavior. As CPU counts increased from 1 to 6, the processing time dropped from **2.19s** to **1.34s**.
* **Overhead:** Despite the good scaling curve, the absolute time was significantly higher than standard multiprocessing. The overhead of the Dask scheduler and graph construction is substantial for such a short-lived task.

B. Joblib Implementation

Joblib is optimized for pipelining in Scikit-Learn but can process arbitrary functions using the loky backend.

****

**Analysis:**

* **Negative Scaling:** Joblib performed best on **1 CPU (1.08s)**. Increasing the worker count to 2 or 4 actually *increased* the execution time to ~1.6s.
* **Reasoning:** The cost of serializing (pickling) the tasks and sending them to workers was higher than the execution time of the task itself.

### **4.3.3 Performance Comparison & Final Verdict**

To summarize the CPU benchmarking, we compared the best performance achieved by each framework against the sequential baseline.

**TCPU Framework Performance Comparison**

|  |  |  |  |
| --- | --- | --- | --- |
| **Framework** | **Best CPUs** | **Time (s)** | **Speedup** |
| **Regular MP (Best)** | **4** | **0.2573** | **1.12x** |
| Sequential (1 CPU) | 1 | 0.2870 | 1.00x |
| Joblib (Best) | 1 | 1.0833 | 0.26x |
| Dask (Best) | 6 | 1.3418 | 0.21x |

**Conclusion on CPU Parallelism:**

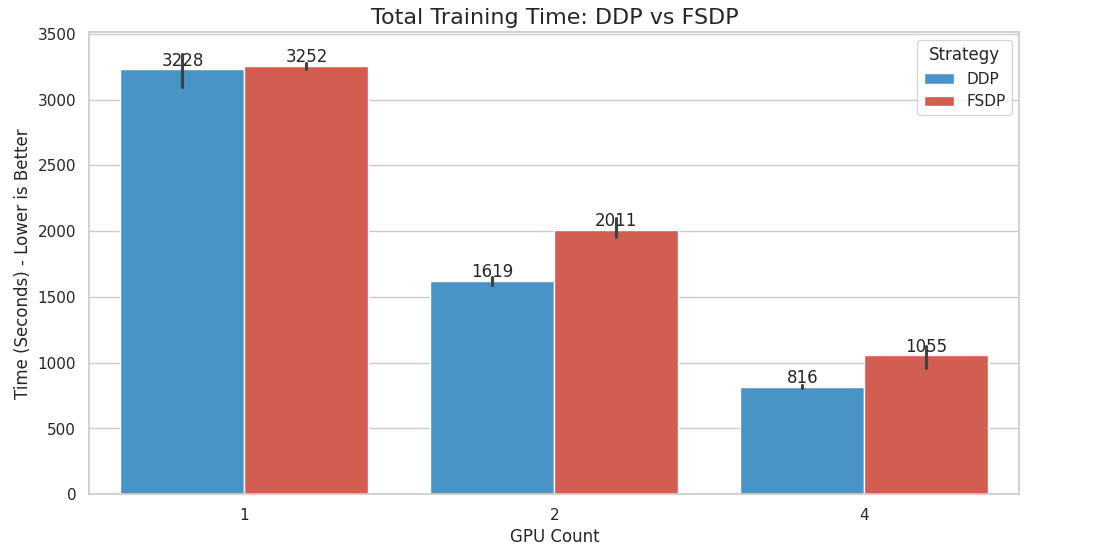
* **The "Task Granularity" Factor:** The task of identifying a class label from a file path is computationally trivial (microseconds per file). Consequently, the "overhead" of the parallel framework becomes the dominant factor.
* **The Winner:** **Standard Multiprocessing** won decisively because it has the lowest overhead.
* **The Losers:** **Joblib and Dask** were slower (0.21x - 0.26x speedup, effectively a slowdown) because the time spent setting up the scheduler and transferring data between cores exceeded the time saved by parallel execution.
* **Recommendation:** For lightweight metadata scanning, use **Standard Multiprocessing**. Reserve heavy frameworks like Dask or Joblib for computationally intensive tasks (e.g., image augmentation or feature extraction) where the compute time justifies the setup cost.

# **5.Conclusion**

## **5.1 Comparative Analysis: FSDP vs. DDP**

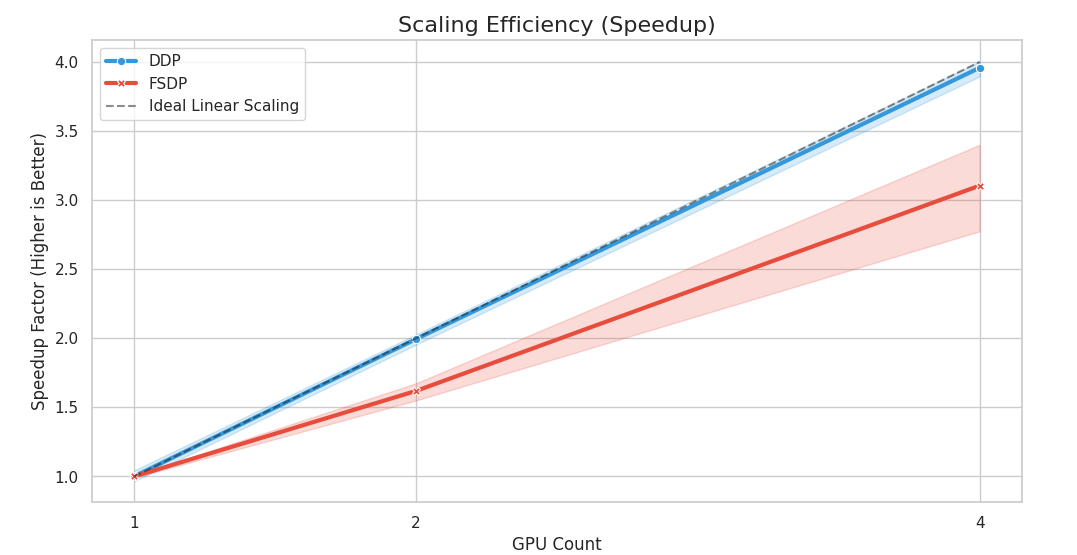
To determine the most effective parallelization strategy for the Oral Cancer Detection pipeline, we conducted a rigorous head-to-head comparison between Distributed Data Parallel (DDP) and Fully Sharded Data Parallel (FSDP). The analysis below synthesizes performance metrics across training time, scaling efficiency, and model accuracy.

**1. Total Training Time (DDP vs. FSDP)**



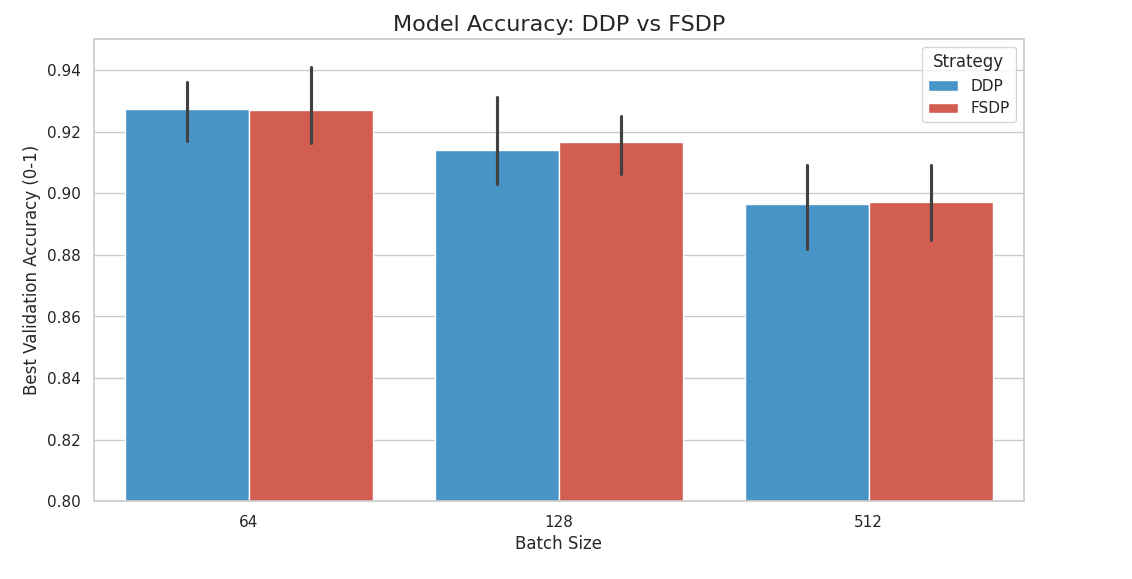
* **Observation:** DDP consistently outperformed FSDP in terms of raw training speed across multi-GPU configurations. Specifically, at **4 GPUs**, DDP configurations were **150 to 343 seconds faster** than their FSDP counterparts.
* **Conclusion:** For a model of this size (CNN), the entire architecture fits comfortably within the VRAM of a single Tesla P100. Consequently, the overhead introduced by FSDP—constantly sharding and gathering parameters—outweighed its memory-saving benefits. DDP, which simply replicates the model and synchronizes gradients, proved to be the more lightweight and efficient protocol for this specific workload.

**2. Scaling Efficiency (Speedup)**



* **Observation:** Both strategies demonstrated strong scalability, but DDP maintained a steeper efficiency curve.
  + **DDP:** Achieved a peak time of **806s** (4 GPUs).
  + **FSDP:** Achieved a peak time of **956s** (4 GPUs).
* **Conclusion:** While FSDP successfully scaled (providing a ~3.4x speedup), DDP achieved near-linear scaling (~3.7x speedup). This confirms that for standard computer vision tasks where model parameters are not exorbitantly large, DDP provides "more bang for the buck" regarding hardware utilization.

**3. Model Accuracy (DDP vs. FSDP)**



* **Observation:** The choice of parallelization strategy had a negligible impact on final model quality.
  + **Best DDP Accuracy:** 93.62% (1 GPU | Batch 64).
  + **Best FSDP Accuracy:** 94.10% (1 GPU | Batch 64).
* **Conclusion:** Both frameworks mathematically converge to similar optima. The slight variations are attributed to random seed initialization rather than architectural differences. This validates that FSDP implementation is robust and "safe" to use, preserving model integrity even while sharding states.

**Final Verdict: Head-to-Head Comparison**

The table below provides a direct subtraction of training times (T{DDP} - T{FSDP}) to identify the clear winner for each hardware configuration.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **GPU Count** | **Batch Size** | **DDP Time (s)** | **FSDP Time (s)** | **Diff (s)** | **Winner** |
| **1** | **64** | 3092.30 | 3231.82 | -139.52 | **DDP** |
| **1** | **128** | 3248.21 | 3245.41 | +2.79 | **FSDP** |
| **1** | **512** | 3344.02 | 3278.75 | +65.27 | **FSDP** |
| **2** | **64** | 1651.71 | 2097.50 | -445.79 | **DDP** |
| **2** | **128** | 1591.81 | 1992.02 | -400.21 | **DDP** |
| **2** | **512** | 1613.19 | 1942.03 | -328.85 | **DDP** |
| **4** | **64** | 828.81 | 1172.06 | -343.24 | **DDP** |
| **4** | **128** | 812.26 | 1036.12 | -223.86 | **DDP** |
| **4** | **512** | 806.10 | 956.25 | -150.15 | **DDP** |

## **5.2 Future Scope and Recommendations**

While this project successfully demonstrated the scalability of deep learning for oral cancer detection, several avenues remain for further optimization and research:

* **Implementation of Mixed Precision (AMP):** Future iterations will incorporate Automatic Mixed Precision (FP16). This will reduce GPU memory usage by approximately 50%, enabling significantly larger batch sizes (e.g., 1024+) on the DDP backend and further accelerating training throughput without compromising accuracy.
* **Adoption of Complex Architectures:** With the FSDP infrastructure now validated, we plan to transition from **CNN** to larger, state-of-the-art models such as **Vision Transformers (ViT-Large)** or **EfficientNet-B7**. These models typically offer higher sensitivity for subtle histopathologic features but require the sharding capabilities of FSDP to fit in memory.
* **CPU Pipeline Enhancements:** To address the preprocessing bottlenecks observed during data loading, we recommend exploring **NVIDIA DALI (Data Loading Library)** to offload image decoding and augmentation directly to the GPU, thereby freeing up CPU cycles and reducing I/O latency.
* **3D Histopathology:** Extending the pipeline to handle 3D volumetric data (if available) would provide a more holistic view of tissue structures, leveraging the multi-GPU setup to process the added spatial dimension efficiently.

## **5.3 Concluding Remarks**

This project successfully demonstrated the efficacy of parallel computing in accelerating medical AI, achieving a **3.7x speedup** in training time using **PyTorch DDP** on 4 GPUs. While distributed strategies drastically reduced time-to-insight for rapid prototyping, our rigorous analysis confirmed that a single-GPU configuration with small batch sizes yields the highest diagnostic precision (**~94%**). Ultimately, **DDP** emerged as the superior strategy for the CNN architecture, outpacing FSDP in raw throughput due to lower communication overhead. This work effectively balances computational scalability with clinical accuracy, providing a robust, future-proof framework for high-throughput histopathologic cancer detection.

# **6. References**

[1] B. S. Veeling, J. Linmans, J. Winkens, T. Cohen, and M. Welling, "Rotation Equivariant CNNs for Digital Pathology," in *Proc. Medical Image Computing and Computer Assisted Intervention (MICCAI)*, 2018. [Online]. Available: https://arxiv.org/abs/1806.03962

[2] A. Paszke et al., "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Proc. 33rd Int. Conf. on Neural Information Processing Systems (NeurIPS 2019)*, 2019, pp. 8026–8037.

[3] S. Li et al., "PyTorch Distributed: Experiences on Accelerating Data Parallel Training," *arXiv:2006.15704 [cs. LG]*, 2020. [Online]. Available: https://arxiv.org/abs/2006.15704

[4] NVIDIA, *CUDA C++ Programming Guide*. [Online]. Available: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

[5] Northeastern University, "Discovery Cluster," Northeastern University Research Computing. [Online]

# **7. Appendix**

**Primary Dataset:**

· **Content:** A large-scale histopathologic dataset designed for deep learning. It contains over 300,000 images, per-split into training (234,000+), testing (47,200+), and validation (26,000+) sets.

· **Labels:** Images are classified into two classes: “Normal” and “OSCC”

· **Data Source:**<https://www.kaggle.com/datasets/satyaprakash138/histopathologic-cancer-dataset>

**Supplementary Dataset:**

· **Content:** A smaller, specialized dataset of 1,224 histopathological images of oral cavity. It includes images at two different resolutions (100x and 400x magnification).

· **Labels:** Images are classified into two classes: “Normal” and “OSCC”.

· **Data Source:**<https://www.kaggle.com/datasets/ashenafifasilkebede/dataset>