**Lab 1 Performance Analysis Report**

**Course:** CS5130  
**Student:** Jingjing Lin  
**Assignment:** Lab 1 — Interactive Image Mosaic Generator Using Gradio  
**Date:** September 21, 2024

**Abstract**

This report presents a performance analysis of an Interactive Image Mosaic Generator that reconstructs input images using tiles from the CIFAR‑10 dataset. The system employs LAB color‑space matching with Euclidean (L2) distance, fully vectorized NumPy operations, and a real‑time Gradio interface. Empirical results on an Apple Silicon MacBook Pro show sub‑second end‑to‑end processing and a **71.77×** speedup versus a loop‑based baseline, demonstrating suitability for interactive use.

**Keywords:** image mosaic, CIFAR‑10, LAB color space, vectorization, NumPy, Gradio, performance analysis

**1. Introduction**

The Interactive Image Mosaic Generator transforms an input image into an artistic mosaic by replacing each grid cell with a semantically similar tile from a curated subset of CIFAR‑10. This report evaluates computational efficiency, output quality, and scalability, and provides configuration guidance for interactive deployments.

**1.1 Objectives**

* Quantify runtime performance across grid resolutions.
* Compare vectorized versus loop‑based implementations.
* Assess reconstruction quality using MSE, SSIM, and PSNR.
* Analyze scalability with respect to grid resolution and tile corpus size.

**1.2 System Overview**

* **Dataset:** CIFAR‑10 (5,000 selected tiles across 10 classes)
* **Matching Space:** LAB with L2 (Euclidean) distance
* **Computation:** Fully vectorized NumPy operations (no nested Python loops)
* **Interface:** Real‑time Gradio web application

**2. Methodology**

**2.1 Processing Pipeline**

1. **Image Preprocessing**  
   Resize to ensure divisibility by the grid; optional per‑channel color quantization (16 levels); normalize to RGB.
2. **Grid Partitioning**  
   Vectorized reshape, e.g., image.reshape(grid\_size, cell\_h, grid\_size, cell\_w, 3), to eliminate nested loops.
3. **Color Analysis**  
   Compute per‑cell mean RGB; convert all cell means to LAB in batch for perceptual uniformity.
4. **Tile Matching**  
   Compute pairwise distances via broadcasting:  
   np.linalg.norm(cell\_colors[:, None, :] - tile\_colors[None, :, :], axis=2); select argmin per cell.
5. **Mosaic Reconstruction**  
   Resize chosen tiles to cell dimensions and assemble into the final mosaic using vectorized operations.

**2.2 Metrics**

* **Quality:** Mean Squared Error (MSE), Structural Similarity Index (SSIM), Peak Signal‑to‑Noise Ratio (PSNR).
* **Computation:** End‑to‑end processing time (per request), initialization time (one‑time), and memory footprint.

**3. Experimental Setup**

* **Hardware:** MacBook Pro (Apple Silicon)
* **Tile Corpus:** 5,000 CIFAR‑10 images (≈500 per class)
* **Initialization Time:** 0.55 s (dataset load + preprocessing)
* **Test Input:** 256×256 synthetic gradient image

**4. Results**

**4.1 Grid Resolution Study**

| **Grid Size** | **Processing Time (s)** | **MSE** | **SSIM** | **PSNR (dB)** | **Total Cells** |
| --- | --- | --- | --- | --- | --- |
| 8×8 | 0.0135 | 2413.31 | 0.187 | 14.30 | 64 |
| 16×16 | 0.0493 | 2231.26 | 0.104 | 14.65 | 256 |
| 32×32 | 0.3854 | 1934.46 | 0.074 | 15.27 | 1024 |
| 64×64 | 0.7797 | 1505.54 | 0.103 | 16.35 | 4096 |

**Findings:** Processing time scales roughly with the number of cells (O(n²) in grid size). Higher resolution lowers MSE and modestly improves PSNR. A 32×32 grid offers a strong quality‑latency trade‑off for interactive use.

**4.2 Vectorization vs. Loops**

| **Implementation** | **Processing Time (s)** | **MSE** | **Speedup** |
| --- | --- | --- | --- |
| Vectorized | 0.0397 | 2231.26 | **71.77×** |
| Loop‑based | 2.8499 | 2231.26 | 1.00× |

**Observation:** Vectorization yields identical reconstruction quality with a 71.77× runtime reduction, enabling real‑time responsiveness.

**4.3 Quality Summary**

* **MSE:** ~1505–2413 (decreases with resolution)
* **SSIM:** 0.074–0.187 (moderate structural similarity)
* **PSNR:** +2.05 dB improvement from 8×8 to 64×64

**5. Optimization & Scalability**

**5.1 Computational Efficiency**

* **Precomputed LAB Means:** Compute tile LAB colors once at startup.
* **View‑based Ops:** Prefer reshapes/views to minimize allocations.
* **Batching/Broadcasting:** Replace Python loops with NumPy broadcasting in distance calculations.
* **Advanced Indexing:** Use argmin indices to gather tiles efficiently.

**5.2 Complexity Analysis**

* **Grid Resolution:** O(n²) cells.
* **Tile Corpus:** O(m) with respect to number of tiles.
* **Matching:** O(n²·m) pairwise distance computation.

**5.3 Practical Performance Targets**

* **Interactive budget:** ≲100 ms for highly responsive UIs (tight target).
* **Measured:** ~13.5 ms (8×8) → ~779.7 ms (64×64).
* **Recommendation:** 32×32 grid (≈385 ms) balances quality and latency for typical interactive apps.

**6. System Architecture Performance**

**6.1 Initialization (One‑time)**

* CIFAR‑10 loading and preprocessing (5,000 tiles): ~0.55 s.
* Vectorized LAB conversion for tile means.
* **Memory footprint:** ~200 MB (tiles + precomputed features).

**6.2 Per‑Request Runtime**

* **Preprocessing:** <1 ms (resize/format).
* **Grid Partitioning:** ~1 ms (reshape).
* **Tile Matching:** 13–780 ms (grid‑dependent).
* **Reconstruction:** ~3 ms (tile placement/assembly).

**7. Conclusions and Recommendations**

**7.1 Summary**

The system achieves sub‑second mosaicking for moderate grid sizes, with a **71.77×** acceleration from vectorization while preserving reconstruction quality (identical MSE to baseline). LAB‑space matching provides perceptually aligned selection, and the architecture scales well to large tile corpora under interactive workloads.