# Parallel Final Project Proposal(Parallelizing Scene Recognition Using Bag of Words Model in Computer Vision)

AndrewID: Kevin Kyi(kwkyi), Derek Kim(derekk)

**Project URL:** https://github.com/kevin-kyi/Parallel-Final-Project

## Summary:

We plan to implement a parallelized version of a scene recognition system using the bag-of-words model on a heterogeneous hardware system. This will involve parallelizing key tasks in the image processing pipeline, involving feature extraction, visual word dictionary construction, image classification, to achieve speedups in both algorithmic processing and model inference.

## Background:

Our project is based on a bag-of-words approach to classify scene images from a large open source dataset, where images are represented as a histogram of features("visual words"). The image processing pipeline includes extracting features using filter banks, constructing visual word dictionaries using K-means clustering on feature vectors, and mapping pixels to a dictionary for image histogram representation. For classification, we will utilize k-Nearest Neighbors(kNN) techniques allowing us to categorize images and tune our performance based on k-neighbors.

The image processing pipeline pertaining to our application is highly parallelizable because of the inherent independence in tasks such as feature extraction(per pixel computations) and histogram formulations(parallelizable visual word aggregation). Additionally, more computationally expensive portions of the algorithm such as applying filters, and generating dictionaries using K-means clustering, and kNN classifications can utilize different parts of our heterogeneous system to achieve speedup. By leveraging parallel computing on CPUs and GPUs we aim to reduce the runtime of these algorithmic processes, making the system scalable for large datasets.

## The Challenge:

This problem is challenging because the different steps for classifying a scene each have their own complexities, especially when parallelizing.

For feature extraction, per-pixel operation is very computationally intensive, especially when images are high resolution and the dataset is large. However, there is a large benefit to parallelization for this step because the per-pixel computations are independent and it has good

spatial locality (neighboring pixels are accessed sequentially) so a large portion of memory accesses will be local.

For the visual word dictionary construction, we use K-means clustering which is a very memory and compute intensive process. Additionally, since K-means clustering relies on data from the previous iteration's cluster assignments, it requires synchronization at each iteration. K-means will likely involve divergent paths because of varying features so synchronization will be a big challenge.

For classification with kNN, although it is a data-parallel algorithm, it requires access to the entire dataset because each test image must be compared with all images in the dataset; this can lead to constraints on bandwidth and memory. There is a lot of complexity involved with distributing data while minimizing inter-process communication costs. Similar to K-means, kNN may also involve divergent paths due to varying distances when comparing images which will likely lead to synchronization complexities.

Overall, distributing the workload in an efficient manner while pipelining to hide latency will be a big challenge because there are many options for optimization. Additionally, since we are using large datasets, another challenge will be minimizing latency for memory accesses.

## Resources:

- Starter Code: Python-based image processing algorithm for feature extraction and kNN classifications.
- Dataset: SUN(Scene UNderstanding) database that contains 899 categories and 130,519 images which we will further partition for our input scene recognition dataset. https://vision.princeton.edu/projects/2010/SUN/
- Hardware: Access to NVIDIA GPUs for GPU-based parallelism to accelerate image processing algorithms and a multi-core CPU for non image processing parallelism

## Goals and Deliverables:

GOALS:

The initial portion of our work deals with dataset partitioning from an open source library. This includes implementing a basic data partitioning scheme to divide the image dataset into manageable subsets which will be crucial for later parallel execution. This goal will ensure we have a structured way to handle data distribution in classification across cores or GPU threads. After dividing our data into subsets we aim to focus on developing a fully functional sequential version of the entire pipeline. This will act as our performance baseline for our overall system

and parts of our image processing pipeline. Issues with this may pertain to CPP library limitations on computer vision functions that are readily available in languages such as python and java.
We then want to set our focus on parallelization across each part of the system. First we want to optimize the filter response extraction function to operate in parallel on each color channel(CIE Lab Color space) and pixel.
Second we want to optimize word map conversions which include distance calculations for pixel mapping using k-means clustering in the visual dictionary.
Third we want to parallelize histogram feature extraction by running visual word calculations in parallel which will utilize both multi-core CPU and GPU setups. This will leverage per-thread histogram updates and requires careful synchronizations/communications to combine partial histograms.
Lastly, we want to parallelize our nearest neighbors classification(kNN) by parallelizing the distance computations between the test image histograms and training set histograms. We will decide which distance calculation algorithm(euclidean, chi^2) to use in this portion based on computational exploration on multi-core CPUs and GPUs.

DEMO:
In terms of data output I want to graph performance breakdowns based on speedups obtained in different parts of the image processing pipeline. For each stage of the pipeline(filter responses, dictionary building, classifications, etc) I want to record and graph runtime across different configurations and speculate on why we chose a certain parallel strategy.
A low bar goal for our presentation would be showing a pre-selected image and showing its classification based on our system with high accuracy and speed. A more desirable goal would be a real-time demo where we take a new picture during the presentation(that can be classified) and show the classification results on the spot. This will not only show accuracy but also the responsiveness and real world applications of our optimized system.

Learning:
From working on this project I want to gain a deeper understanding of which parts of a computer vision pipeline are suitable for parallelization and why. For instance, we aim to learn why certain stages benefit more from GPU processing(filter responses, etc.) compared to CPU-based multi-threading(highly synchronizable portions). Additionally, by considering multi-core CPU and GPU implementations we learn how to leverage each hardware type effectively. This will involve understanding when to switch between CPU and GPU processing to maximize pipeline efficiency in heterogeneous systems. Additionally, our project will have many communication challenges and by exploring different load balancing techniques we can ensure tasks are optimally divided across processing units to avoid common parallel problems such as bottlenecks, communication costs, and deadlock.

System Capabilities:
Our proposed system will perform scene recognition on a dataset of images, classifying each image into one of several predefined categories. The system will be capable of processing high-resolution images by leveraging a parallelizable pipeline. By implementing parts of the pipeline on both multi-core CPUs and GPUs, our system will adaptively allocate tasks to the optimal processing unit, allowing for efficient scaling in large-scale image datasets. For our overall system we hope to achieve at least a 5x speedup over the sequential implementation by optimizing our algorithm based on our hardware components. Within our reach goals, the system should classify images almost instantaneously due to our utilization of high-end GPU hardware, enabling real-time classification on lower resolution images. Lastly, we want our system to exhibit strong scalability, with consistent performance gains across different datasets.

## Platform Choice:

A heterogeneous system that utilizes both CPUs and GPUs, paired with C++ and CUDA, is ideal for implementing a parallelized scene recognition pipeline due to the workload's computational intensity and need for efficient parallel processing. The CPU manages coordination tasks, such as workload distribution, data transfers, and complex control flows that we will likely face when implementing the K-means clustering and kNN algorithm, making it a good choice for synchronization and managing memory access efficiently. The GPU is in charge of accelerating computation-intensive steps for per-pixel feature extraction, K-means clustering, and kNN classification. Its high throughput and ability to handle thousands of parallel threads allow it to process many image regions and feature vectors concurrently, which aligns well with the independence in per-pixel and per-feature tasks in the pipeline.

Using a combination of CPUs and GPUs maximizes speedup potential by offloading computationally intensive tasks to the GPU, while the CPU takes care of inter-stage communication. With this system, our goal is to balance workload and optimize resource utilization. Additionally, using C++ and CUDA gives us better control over memory and parallel processing. Overall, this system not only offers significant speedup for complex image processing but also provides an opportunity to explore real-world challenges in load balancing, memory management, and parallel computation on a scalable platform.

## Schedule:

### Week 2: Nov 11 - Nov 17

- **Wednesday, Nov 13**: Submit the project proposal.
- **Tasks**:

    - Begin implementing the **sequential baseline** for each stage (filter response extraction, dictionary creation, word map generation, and classification) to establish a performance baseline.
    - Measure and document runtime for each stage of the sequential pipeline for use in later comparisons.
    - Finalize the dataset partitioning scheme and test initial code functionality.

## Week 3: Nov 18 - Nov 24

- **Tasks**:
    - Begin **parallelizing the filter response extraction** stage using both multi-core CPU and GPU. Implement basic threading for CPU and initial CUDA kernels for GPU.
    - Document performance metrics and compare parallel speedup vs. sequential execution.
    - Start setting up initial code structure for dictionary creation and word mapping, laying groundwork for parallel implementation.

## Week 4: Nov 25 - Dec 1

- **Wednesday, Nov 27**: Submit Milestone Report.
- **Tasks**:
    - Continue parallelizing **dictionary creation with K-means clustering**, implementing parallel K-means on both CPU and GPU and gathering initial performance metrics.
    - Write and submit the milestone report, detailing completed sequential and filter response parallelization stages, initial dictionary creation parallelization results, and any encountered challenges or revised approaches.
    - Begin refining the parallelized filter response code based on feedback from the milestone report.

## Week 5: Dec 2 - Dec 8

- **Tasks**:
    - Implement **word map conversion** using parallelized distance computations. Apply batch processing and distance calculations on both CPU and GPU for efficiency.
    - Conduct detailed performance analysis on word map conversion, including comparisons across different hardware configurations (CPU vs. GPU).
    - Begin parallelizing histogram feature extraction and gather data on its speedup potential.

## **Week 6: Dec 9 - Dec 15**

- **Tasks**:
    - Finalize parallelization of **nearest neighbor classification** by optimizing distance calculations between test and training images.
    - **Run complete end-to-end tests** of the system with parallel implementations of each stage to verify functionality and performance improvements.
    - Prepare data visualizations (graphs and charts) for speedup across system stages and different parallelization strategies.

For our schedule we plan on tracking our progress around a week early to allow buffer for potential downfalls in our project goals. Additionally, we plan on exploring different languages and parallelization strategies that may introduce different goals that may affect our planned schedule. Lastly, we will spend the rest of week 2(Nov 11-17) researching different libraries to leverage certain computer vision computations, and doing more research on which parts of our image processing pipeline we want to parallelize.