CS532: Final Project Presentation

Distributed Image Processing System

Isha Gohel(igohel@umass.edu)
Lavanika Srinivasaraghavan (lsrinivasara@umass.edu)
Sivaraaman Balakrishnan (sivaraamanba@umass.edu)
Sri Ram Bandi(sbandi@umass.edu)

Goals

Project Goals

- Build a distributed image processing system using Hadoop and Spark
- Train and test CNN models (custom and ResNet50) on classification of cat and dog images
- Dataset used: large-scale **Kaggle Dogs vs. Cats**.
- Process images in a distributed manner
- Benchmark:
    - **Time taken** – time taken to execute the inference for the subset of test images
    - **Throughput** – images classified per second.
    - **Scalability** – impact of increasing nodes

Goal Achievement:

- All major goals achieved: training, distributed execution, integration with HDFS.
- Minor tuning and evaluation improvements possible.

Overall Approach

- **Data Storage:** Load data(tar) into **HDFS**.
- **Pre-processing:** Resize (224×224) and normalize using **OpenCV**.
- **Distributed Processing:** Use **PySpark** to parallelize inference across nodes.
- **Model Inference:** Use **Custom CNN** with **PyTorch** with **4** layers trained on the training dataset and **inference on the testing set**.
- **Run Experiments:** Vary node count (1 to 3) for benchmarking.
- **Benchmarking and Aggregation:** Collect metrics(throughput and time taken) on performance.

Design Decisions

Modeling Choices:

- Fine-tuned a pre-trained ResNet50 model on the training dataset for baseline validation.
- Trained a custom CNN model from scratch to evaluate performance under different configurations.
- The custom CNN consistently matched the ResNet50's classification results across all test runs.

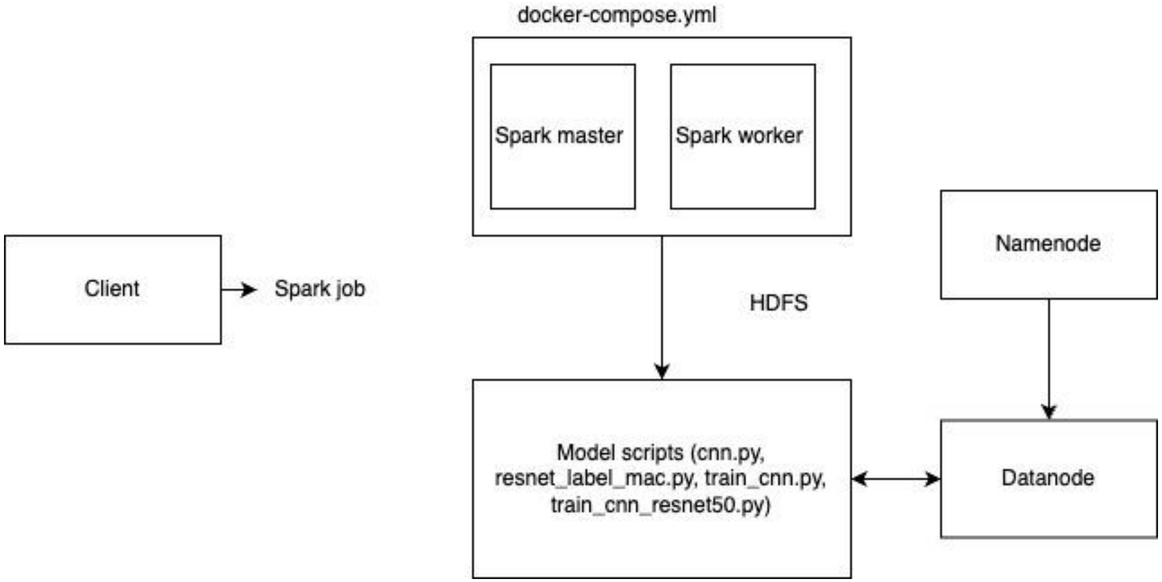
Frameworks Used:

- **OpenCV**: Preprocessing
- **HDFS**: Distributed storage
- **PyTorch**: Model training and inference
- **PySpark**: Parallel processing

High-Level Architecture

Components:

- app/: Spark job using CNN/ResNet models.
- docker-compose.yaml: Defines Spark master, worker, Hadoop Namenode/Datanode.
- hadoop-config/: HDFS configuration files.
- cnn.py, resnet_label_mac.py, train_cnn_resnet50.py: model scripts.


docker-compose.yml
Spark master
Spark worker
Client
Spark job
HDFS
Namenode
Model scripts (cnn.py, resnet_label_mac.py, train_cnn.py, train_cnn_resnet50.py)
Datanode

File and class overview

- CNN model: cnn.py : simple CNN for classification
- ResNet model: resnet_label_mac.py : loads fine-tuned ResNet50
- Main app: main.py : loads image, HDFS read, applies model, runs in Spark job
- **Trade-off:** ResNet is accurate but heavier; CNN is lightweight but less accurate.

Team Contributions

1. Isha Gohel - Model architecture, Model Training and fine-tuning, Batched inference
2. Lavanika Srinivasaraghavan - Model architecture, Model Training and fine-tuning, Batched inference
3. Sivaraaman Balakrishnan - Spark/HDFS integration, Docker setup, Image preprocessing
4. Sri Ram Bandi - Pipeline coordination, single vs. multi-node testing, Model Training and fine-tuning

Code Demo

Tests and Validation

- Baseline Model:
    - Fine-tuned ResNet model on the train dataset. The classification labels of this model on the test set is our baseline ground truth.

- Tests:
    - We trained our custom CNN model from scratch using the training dataset. During our experiments, the model consistently produced classification results that matched the ground truth labels(Resnet50) across all test runs.
    - To evaluate scalability, we ran our program using different numbers of Spark worker nodes (1 to 3) and compared the throughput and execution time.
    - One known limitation is that increasing the number of images in the TAR file, especially when combined with a increased number of workers, can cause memory-related errors due to RAM constraints in the containerized environment.

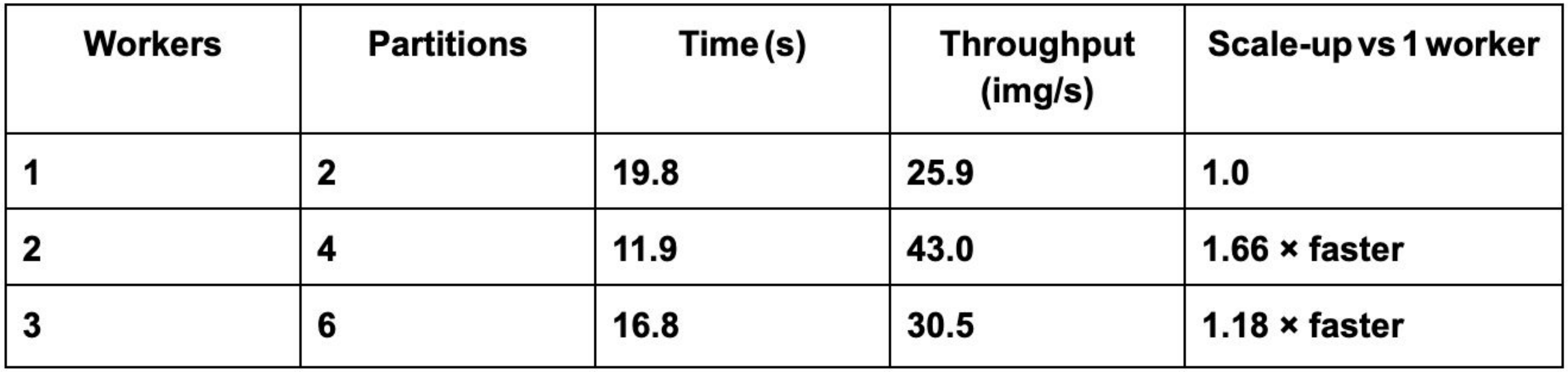
Experimental Results and Analysis

Accuracy of the models :

- CNN accuracy: 78.1%
- ResNet50 fine-tuned accuracy: 96.3%

Single node v/s multi node analysis :

- Analysis using 512 images and batch size 8



Workers	Partitions	Time (s)	Throughput (img/s)	Scale-up vs 1 worker
1	2	19.8	25.9	1.0
2	4	11.9	43.0	1.66 × faster
3	6	16.8	30.5	1.18 × faster

Possible Improvements

- We can extend to multi label instead of a 2 label classification
- Try implementing using bigger image dataset
- Instead of fixed number of workers we could make it dynamic using kubernetes

To improve model accuracy:

- Instead of using the pretrained ResNet50 as it is, we can fine tune all or some deeper layers on the dataset
- Automatically highlight images where CNN and ResNet50 disagree and visualize with confidence scores.

Thank you!