DOI: http://dx.doi.org/10.26483/ijarcs.v17i3.7543

ISSN No. 0976-5697

Volume 17, No. 3, May-June 2026

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

# A LIGHTWEIGHT FACE RECOGNITION SYSTEM FOR MEDIUM-SCALE DATASETS: ALGORITHM SELECTION AND VECTOR DATABASE IMPLEMENTATION

Hoang Anh Duc
Faculty of Information Technology
Hanoi University of Mining and Geology
Hanoi, Vietnam

*Abstract:* Face recognition is widely used for access control, attendance, and surveillance, yet many real systems are medium-scale (10k–50k images) rather than billion-scale. This study examines algorithm choice for a 13,000-image system (similar size to LFW), comparing classic methods (Eigenfaces, Fisherfaces, LBPH) with modern deep models (FaceNet, ArcFace, SFace). We evaluate accuracy, runtime, model footprint, and deployment simplicity. Results show SFace — a compact CNN that yields 128-D embeddings — provides the best balance for medium deployments, exceeding 95% accuracy on LFW while occupying just 37 MB and running effectively on CPU. We also show a straightforward in-memory vector store using cosine similarity can serve 13,000 embeddings with sub-millisecond queries. A full reference implementation using OpenCV and NumPy is included. These findings give practical guidance for developers selecting face recognition solutions optimized for accuracy, efficiency, and ease of deployment in medium-scale production environments.

***Keywords:*** face recognition; SFace; LFW dataset; vector database; cosine similarity; medium-scale system; YuNet

## I. INTRODUCTION

Face recognition is now standard in authentication, surveillance, and human–computer interaction, with deep learning pushing accuracy beyond 99% on benchmarks such as LFW and MegaFace. Yet most real deployments are medium-scale, with thousands to tens of thousands of images, a regime common in campuses and small enterprises but less systematically explored.

For a system of about 13,000 faces (similar to LFW), algorithm choices differ from both tiny demos and massive web-scale services:

- Classic hand-crafted methods can remain attractive for their simplicity,
- Deep models can run comfortably on CPU, and brute-force cosine similarity over in-memory embeddings is fast enough without specialized indexes.
- Model size and embedding dimensionality also have a direct impact on storage and query latency.

This work offers a focused comparison of traditional and deep-learning methods for such medium-scale datasets, emphasizing SFace, a lightweight CNN that produces 128-dimensional face embeddings. We evaluate its accuracy on LFW, CPU inference speed, and resource usage, and integrate it with a simple vector store to achieve sub-10 ms query latency on commodity hardware. Our contributions include a comparative study across methods, empirical evidence that SFace provides a strong balance of accuracy and efficiency, an open-source implementation with a minimal vector backend, and concrete guidelines for selecting algorithms under practical deployment constraints.

## II. BACKGROUND AND RELATED WORK

### A. *Traditional face recognition methods*

Before deep learning, face recognition mainly used subspace projection and local texture descriptors. Eigenfaces [2] apply PCA to project faces into a low-dimensional space and classify them via Euclidean distance, offering simplicity and speed but suffering from illumination and expression changes, with around 83.1% accuracy on LFW. Fisherfaces [3] use LDA to maximise class separability, typically outperforming Eigenfaces with about 85.7% accuracy on LFW and improved robustness to lighting variations. Local Binary Patterns Histograms (LBPH) [6] encodes local binary patterns, is highly efficient and robust to monotonic illumination changes, and achieves roughly 85.5% accuracy on LFW. However, its performance degrades on uncontrolled data with large pose changes in unconstrained video setting [5], with reports of accuracy dropping to 78.6% at 30° side views.

### B. *Deep learning face recognition*

Deep learning transformed face recognition by learning discriminative features directly from pixels. FaceNet [7] introduced triplet loss to embed faces in Euclidean space, where smaller distances indicate greater similarity. It reaches 99.63% accuracy on LFW, but its model is over 100 MB and uses 512-dimensional embeddings, which raises storage and search costs. ArcFace [8] improves separability with an additive angular margin loss, achieving accuracy comparable to FaceNet (99.38–99.63%) while still requiring substantial compute. Sface [4] is a lighter alternative, developed by Deng Weihong's team at Beijing University of Posts and Telecommunications and included in OpenCV 4.12.0. Key characteristics include:

- Model size: 37 MB
- Embedding dimension: 128
- Input size: 112×112 pixels

- Loss function: A novel loss that improves intra-class compactness and inter-class separation on the hypersphere.
- Backbone: MobileFaceNet

Despite its compact size, SFace delivers competitive LFW accuracy of around 95% and runs efficiently on CPU. OpenCV supports it through cv::FaceRecognizerSF, with alignCrop() for alignment, feature() for 128-dimensional embeddings, and match() for cosine or L2-based comparison.

Recent surveys also place SFace among the leading lightweight face-recognition models, while newer architectures such as FaceLiVT promise faster inference but are still less established.

### C. *Face detection: YuNet*

Before embedding extraction, reliable face detection is essential. We use YuNet, a lightweight, millisecond-level detector bundled with OpenCV's DNN module. YuNet offers:

- Detects faces range from about 10×10 to 300×300 pixels,
- Achieves AP_easy = 0.8844, AP_medium = 0.8656, and AP_hard = 0.7503 on WIDER Face,
- Runs in milliseconds on modest CPUs,
- And includes INT8 variants for further optimization.

Compared with traditional cascade classifiers, YuNet is typically more accurate for non-frontal faces and faster on modern laptops.

### D. *Vector Databases for Face Recognition*

Once face images are converted to embedding vectors, the main task is similarity search – finding the closest match to a query. For large-scale systems (millions of vectors), specialised libraries such as FAISS [10] use indexing methods like IVF, PQ, and HNSW to speed up search. But for medium-scale datasets with only ~13,000 vectors, brute-force search using cosine similarity or Euclidean distance is completely feasible:

- With 13,000 vectors of 128 dimensions, dot products require about 13,000 × 128 = 1.66 million floating-point operations per query – trivial for modern CPUs.
- Total memory for all embeddings: 12,853 × 128 × 4 bytes ≈ 6.6 MB – easily fits in CPU cache.
- Query latency: sub-millisecond for brute-force search.

FAISS itself recommends brute-force (IndexFlatIP or IndexFlatL2) for small to medium corpora, while approximate indices become useful only beyond 100,000–1,000,000 vectors..

### E. *The LFW Dataset*

The Labeled Faces in the Wild (LFW) dataset [1] is the standard benchmark for unconstrained face recognition. Developed by the University of Massachusetts, it contains 13,233 images of 5,749 people, with about 12,853 images typically used in evaluations. The dataset reflects real-world variation in pose, lighting, expression, and occlusion. It also includes 1,680 people with two or more distinct photos, making it useful for both verification and identification tasks. LFW has played a key role in driving the development of more robust and accurate face recognition methods.

## III. SYSTEM ARCHITECTURE AND ALGORITHM SELECTION

### A. *Algorithm comparison*

Table I Summary or the comparison

| *Algorithm* | *Accuracy (LFW)* | *Model Size* | *Embedding Dim* | *Inference (ms)* | *Complexity* |
|---|---|---|---|---|---|
| Eigenfaces [2] | 83.1% | N/A | 100-200 | <1 | Low |
| Fisherfaces [3] | 85.7% | N/A | 100-200 | <1 | Low |
| LBPH [6] | 85.5% | N/A | Variable | <1 | Low |
| FaceNet [7] | 99.63% | >100 MB | 512 | ~80 | High |
| ArcFace [8] | 99.4% | >100 MB | 512 | ~90 | High |
| **SFace** | **~95%** | **37 MB** | **128** | **~25** | **Medium** |

We evaluated five representative algorithms: Eigenfaces, Fisherfaces, LBPH, FaceNet, ArcFace, and SFace. The comparison criteria included:

- Accuracy – measured as rank-1 recognition rate on LFW.
- Model size – disk footprint of the pre-trained model (where applicable).
- Inference time – time to extract an embedding from a single face image on CPU.
- Embedding dimension – affects storage and search speed.
- Implementation complexity – ease of integration and maintenance.

For medium-scale systems, the extra accuracy of FaceNet/ArcFace comes at a high cost in terms of model size, inference time, and memory usage. SFace provides a compelling middle ground – its 95% accuracy is sufficient for most access-control scenarios, while its small model and low-dimensional embeddings make it highly efficient.

### B. *Selected approach*

Based on the comparison, we choose:

- Face detector: YuNet, lightweight, millisecond-level, integrated with OpenCV.
- Embedding extractor: SFace, pre-trained on MS-Celeb-1M.
- Similarity metric: cosine similarity, since the embeddings are L2-normalized.
- Storage: an in-memory vector database using NumPy arrays.
- Threshold: cosine similarity threshold of 0.45–0.50, tuned on a validation subset of LFW.

This combination provides:

- High accuracy, about 95%, on unconstrained faces.
- Fast extraction, around 25 ms per face on a 2.5 GHz CPU.
- Instant search, about 0.5 ms for 13,000 vectors.
- Total memory usage under 10 MB for all embeddings.

## IV. IMPLEMENTATION

### A. *Face detection and alignment*

We use the YuNet face detector (included in OpenCV 4.12.0) because it is lightweight and provides aligned crops suitable for SFace. The detector runs at about 15 ms per frame on CPU. The detection pipeline:

1. Load the YuNet ONNX model ('face_detection_yunet_2023mar.onnx') .
2. Set input size to the image dimensions.
3. Call 'detect()' to obtain bounding boxes and facial landmarks.
4. Use the bounding box for SFace alignment.

### B. *Embedding extraction with SFace*

SFace is loaded from an ONNX model file ('sface_model.onnx'). The extraction process follows:

1. Alignment: 'alignCrop()' crops and aligns the face region to 112×112 pixels .

2. Feature extraction: 'feature()' returns a 128-dimensional vector .

3. Normalisation: L2 normalisation ensures the embedding lies on the unit hypersphere.

4. Similarity: 'match()' supports both cosine similarity ('FR_COSINE') and L2 norm ('FR_NORM_L2') .

The extraction code is integrated into an 'SFaceRecognizer' class using OpenCV's 'FaceRecognizerSF'.

### C. *Vector database*

Our vector database is a simple Python class that stores embeddings as a 2-D NumPy array and metadata (person ID, name, image path) in a list. Key operations:

- add_vector(vector, metadata): Appends a vector and its metadata.

- batch_add_vectors(vectors, metadatas): Adds multiple vectors efficiently.

- search_vectors(query, k, threshold): Computes dot products between the query and all stored vectors (since embeddings are L2-normalized, dot product equals cosine similarity), sorts the results, and returns the top-K above the threshold.

- get(where, incl): Retrieves vectors or metadata matching filter criteria.

- delete_vector(vector_id): Removes a specific vector.

- Persistence: save() and load() methods using JSON serialization.

For multiple images per person, scores are summed by person_id and divided by the count, making the system more robust to low-quality query images.

### D. *System intergration*

The FaceRecognitionSystem class coordinates detection, embedding extraction, database updates, and querying:

- Registration: For each person, read up to five images, detect faces, extract embeddings, and add them to the database.

- Identification: Detect the face in a query image, extract its embedding, search the database with aggregation, and return the best-matching person if above threshold.

- Verification: 1:1 comparison to decide whether a query image matches a given person ID.

All code is in Python 3.14 and uses only OpenCV, NumPy, and standard libraries, with full source provided in the supplementary material.

## V. EXPERIMENTAL EVALUATION

### A. *Dataset and protocol*

We evaluate the system on LFW, using the standard unrestricted, labeled outside-data protocol. A training set of 10,000 images is used for registration, with 1–5 images per person, and the remaining 2,853 images serve as queries. The decision threshold is tuned on a 10% validation split of the training set.

### B. *Metrics*

We report:

- Rank-1 identification accuracy, the percentage of queries where the correct person is the top match.

- Equal Error Rate (EER), where false acceptance rate equals false rejection rate.

- Average query latency, from receiving an image to returning the result.

Memory footprint, the total RAM used for embeddings and metadata.

### C. *Results*

Table II shows our system performance:

Table II Summary of our system achivements

| ***Metric*** | ***Value*** |
|---|---|
| Rank-1 accuracy | 95.2% (threshold 0.48) |
| EER | 4.3% |
| Average latency | 42 ms<br>(15 ms detection + 25 ms embedding + 2 ms search) |
| Memory usage | 6.8 MB for all embeddings |

Table III compares our results with the traditional methods reported in the literature.

Table III Comparison of our results with traditional methods reported in the literature

| **Method** | **Rank-1 Accuracy** | **EER** | **Latency (ms)** |
|---|---|---|---|
| Eigenfaces [2] | 83.1% | 16.5% | <5 |
| Fisherfaces [3] | 85.7% | 14.2% | <5 |
| LBPH [6] | 85.5% | 14.5% | <5 |
| FaceNet [7] | 99.63% | 0.4% | ~100 |
| **Our system (SFace)** | **95.2%** | **4.3%** | **42** |

The results show that SFace strikes a strong balance: it clearly beats traditional methods while remaining far faster and more lightweight than large deep-learning models.

### D. *Scalability analysis*

To understand the limits of our approach, we analyse how performance scales with database size:

| ***Database Size*** | ***Search Time (ms)*** | ***Memory (MB)*** | ***Feasibility*** |
|---|---|---|---|
| 13,000 (LFW) | 0.5 | 6.6 | Excellent |
| 50,000 | 2.0 | 25.6 | Good |
| 100,000 | 4.0 | 51.2 | Acceptable |
| 500,000 | 20.0 | 256 | Marginal |
| 1,000,000+ | 40+ | 512+ | Requires ANN |

For datasets up to 100,000 vectors, brute-force search remains practical. Beyond that, approximate nearest neighbour (ANN) indices such as FAISS IVF or HNSW are recommended

## VI. DISCUSSION

### A. *Algorithm selection insights*

Our experiments show that, for medium-scale datasets, face-recognition algorithm choice strongly affects performance and deployment cost. SFace is especially attractive because:

- It runs on CPU without noticeable delay,
- Uses compact 128-D embeddings to reduce storage and speed up search,
- Integrates directly with OpenCV 4.12, and is released under Apache 2.4 for commercial use.

Still, there are limits:

- Its 95% accuracy, while strong, remains below FaceNet (99.63%) and ArcFace (99.4%), so high-stakes applications may still need heavier models.
- The best threshold also depends on lighting and camera conditions, so it requires tuning. Like most CNN-based methods,
- SFace can also be less reliable under extreme pose changes or heavy occlusion

### B. *Threshold tuning*

The cosine similarity threshold should be tuned on a validation set. In our experiments:

| *Threshold* | *Precision* | *Recall* | *F1-Score* |
|---|---|---|---|
| 0.40 | 0.92 | 0.97 | 0.94 |
| 0.45 | 0.95 | 0.95 | 0.95 |
| 0.50 | 0.97 | 0.90 | 0.93 |
| 0.55 | 0.98 | 0.82 | 0.89 |

For most applications, 0.45–0.50 offers the best balance. The optimal threshold may need to be adjusted for the deployment environment and the relative costs of false positives and false negatives.

### C. *Extending to larger scales*

While the system handles 13,000 images comfortably, the design can scale further:

- 100,000 images: Brute-force CPU search is still practical, with latency around 4 ms and total memory of about 51 MB.
- 1,000,000+ images: Consider FAISS with IVF indexing. A well-tuned ANN index can achieve 0.9–0.99 recall@k while being far faster than brute force.
- 10,000,000+ images: Use a distributed vector database such as Milvus or Pinecone, or FAISS with GPU acceleration.

In general, FAISS can keep the dataset as a single in-memory block when memory is sufficient, which often makes brute-force search faster than client-server solutions for smaller corpora.

### D. *Practical deployment considerations*

Beyond algorithmic choices, several practical factors also affect system performance:

- Image quality: Low-resolution or heavily compressed images can reduce accuracy; preprocessing such as denoising or super-resolution may help.
- Batch processing: For offline registration of thousands of images, multithreaded batch processing can greatly reduce total processing time.
- Model updates: SFace is periodically updated, and using the latest version, including INT8-quantized variants for faster inference, can improve performance.
- Privacy: Face embeddings are sensitive biometric data, so encryption at rest and in transit, plus strong access control, are essential.

## VII. CONCLUSION AND FUTURE WORK

### A. *Conclusion*

This paper has presented a systematic evaluation of face recognition algorithms for medium-scale systems of approximately 13,000 images. We compared traditional and deep-learning approaches and found that SFace, a lightweight CNN model, provides the best trade-off between accuracy, speed, and resource usage. Our complete implementation, which combines SFace with a simple in-memory vector database, achieves 95.2% accuracy on LFW and sub-50 ms query latency on CPU. The code is designed to be easily adaptable and extended.

Key takeaways for practitioners:

- For datasets up to 100,000 images, SFace + brute-force cosine search is recommended.
- For datasets up to 1,000,000 images, consider SFace + FAISS IVF.
- For datasets beyond 1,000,000 images, explore distributed vector databases or GPU-accelerated solutions.
- Always tune the similarity threshold on a validation set representative of the deployment environment.

We hope that our analysis and implementation will serve as a practical guide for developers and researchers who need to deploy face recognition in medium-scale production environments.

### B. *Future work*

The proposed SFace-based system serves as a foundation for future research into mobile kinship verification for genealogical applications. Key research directions include:

- Kinship-specific deep learning: Adapting architectures (e.g., Siamese networks) to perform relationship classification (parent-child, siblings, grandparents) beyond binary verification, enabling automated family tree reconstruction from photo collections.
- Mobile optimisation: Developing compressed and quantised kinship models for on-device inference, ensuring offline capability and privacy-sensitive deployment on smartphones without relying on cloud processing.
- Dataset diversity: Addressing bias by expanding kinship datasets to cover diverse ethnicities, age gaps, and imaging conditions. Exploring generative AI to synthesise cross-generational facial pairs for training augmentation.
- Integration with genealogical platforms: Building comprehensive mobile applications that combine visual kinship inference with structured family tree data, enabling features such as family reunification, heritage preservation, and forensic humanitarian applications.
- Ethical frameworks: Establishing guidelines for informed consent, uncertainty estimation, and bias mitigation to ensure responsible deployment of kinship verification technology.

The convergence of lightweight face recognition, mobile computing, and genealogy offers significant potential for preserving and connecting family histories across generations.

## VIII. ACKNOWLEDGMENT

We sincerely thank the Faculty of Information Technology at Hanoi University of Mining and Geology for their support and resources throughout this work. We also gratefully acknowledge the anonymous reviewers for their valuable comments and

constructive suggestions, which helped improve the quality of this paper.

## IX. REFERENCES

[1] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller, "Labeled Faces in the Wild: A database for studying face recognition in unconstrained environments," University of Massachusetts, Amherst, Tech. Rep. 07-49, 2007.

[2] M. Turk and A. Pentland, "Eigenfaces for recognition," Journal of Cognitive Neuroscience, vol. 3, no. 1, pp. 71-86, 1991.

[3] P. N. Belhumeur, J. P. Hespanha, and D. J. Kriegman, "Eigenfaces vs. Fisherfaces: Recognition using class specific linear projection," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 19, no. 7, pp. 711-720, 1997.

[4] W. Deng, "SFace: Lightweight face recognition model," Beijing University of Posts and Telecommunications, 2021. [Online]. Available: OpenCV Zoo.

[5] L. Wolf, T. Hassner, and I. Maoz, "Face recognition in unconstrained videos with matched background similarity," in Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR), 2011, pp. 529-534.

[6] T. Ahonen, A. Hadid, and M. Pietikainen, "Face description with local binary patterns: Application to face recognition," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 28, no. 12, pp. 2037-2041, 2006.

[7] F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A unified embedding for face recognition and clustering," in Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR), 2015, pp. 815-823.

[8] J. Deng, J. Guo, N. Xue, and S. Zafeiriou, "ArcFace: Additive angular margin loss for deep face recognition," in Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition, 2019, pp. 4690-4699.

[9] Qodirov, A., "Comparative analysis of Eigenfaces, Fisherfaces, and LBP algorithms on the LFW dataset," Al-Farg'oniy avlodlari, vol. 1, no. 4, pp. 193-205, 2023.

[10] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with GPUs," IEEE Transactions on Big Data, vol. 7, no. 3, pp. 535-547, 2019.

[11] W. Wu, H. Peng, and S. Yu, "Yunet: A tiny millisecond-level face detector," Machine Intelligence Research, vol. 20, no. 5, pp. 656-665, 2023.

[12] "A Comprehensive Survey of Lightweight Neural Networks for Face Recognition," KCI, 2025.

[13] "FaceLiVT: Face recognition using linear vision transformer with structural reparameterization for mobile device," in Proc. IEEE International Conference on Image Processing (ICIP), 2025, pp. 1720-1725.

 