International Journal of Science, Strategic Management and Technology
Volume **02** Issue **03** March-**2026** | ISSN: **3108-1762 (Online)** | Impact Factor: 3.8
An International, Peer-Reviewed, Open Access Scholarly Journal Indexed in recognized academic databases

AI-Driven Guest Identification and Photo Retrieval System

Thangamurugan R
Student
Dept. of Artificial Intelligence and Data Science Ramco Institute of Technology, Rajapalayam, India
ramarv7859@gmail.com

Dr. S.V. Anandhi
Associate Professor II
Dept. of Artificial Intelligence and Data Science Ramco Institute of Technology, Rajapalayam, India
anandhi@ritrjpm.ac.in

Dr. M. Kaliappan
Professor
Dept. of Artificial Intelligence and Data Science Ramco Institute of Technology, Rajapalayam, India
kaliappan@ritrjpm.ac.in

https://doi.org/10.55041/ijsmt.v2i3.329

Cite this Article: R, T. (2026). AI-Driven Guest Identification and Photo Retrieval System. International Journal of Science, Strategic Management and Technology, *02*(03). https://doi.org/10.55041/ijsmt.v2i3.329

License: This article is published under the Creative Commons Attribution 4.0 International License (CC BY 4.0), permitting use, distribution, and reproduction in any medium, provided the original author(s) and source are properly credited.

***Abstract*—**There is a specific kind of let-down that follows big events. You know photos were taken. You were there. A professional was working all day. Two weeks later a gallery link arrives, 3,000 images in it, and the expectation is that each of the two hundred guests will scroll through until they find themselves. Most give up within ten minutes. The photos that were taken of them — professionally, at something they cared about — they never actually see. We built a system so that doesn't have to happen anymore. Guest scans a QR code at the venue entrance, uploads a selfie, and a personal email arrives after the event with every photo from the day where their face appeared. DeepFace handles recognition, MTCNN handles detection, Node.js runs the backend, cloud storage absorbs the volume. We tested it at four events: two weddings, a university tech festival, a corporate town hall. Matched at 95 percent accuracy, processed around 25 photos per second on a standard laptop. This paper describes how it works, what failed during development, what we changed, and where the gaps remain.

Keywords—event photo management; face recognition; deep learning; DeepFace; cloud storage; automated album delivery; computer visionINTRODUCTION

The situation that motivated this project is one most people have experienced without thinking of it as a problem worth solving. A photographer works a full-day event, shoots three thousand photos, and delivers a gallery link ten days later. Two hundred guests open the link, scroll for a while, and almost all of them close it without finding most of the photos they appear in. The images exist. The people exist. Getting one to the other turns out to be nobody's job.

That gap exists not because the technology is missing — deep learning face recognition has been accurate enough for real deployment for several years — but because nobody aimed it at this specific workflow. Tools like Lightroom are built for photographers to manage their own archives. Gallery platforms are built to serve images on request. Neither one routes specific photos to specific guests automatically. That routing is what our system does.

© Author(s). This work is peer-reviewed, openly published, and permanently archived
This article is openly accessible and reusable with proper attribution.

The mechanic is simple: guest arrives at an event, scans a QR code at the entrance, uploads a selfie from their phone. Photographer uploads their full batch after the event. System runs face detection across every image, compares each guest's selfie against every detected face, assembles a personal album for each guest, and delivers it by email. Guest's involvement is two actions and then waiting. Photographer's involvement is unchanged from normal. The sorting and routing happen without anyone doing them manually.

Getting that two-action interface to produce reliable albums on the other end consumed most of the project's time. Selfies arrive in every quality imaginable. Thresholds have to be set based on what real users find tolerable, not what looks clean on a precision-recall curve. File sizes require choices photographers care about. Email delivery has to survive failure gracefully rather than silently. This paper is the full account — not only the version where the system works, but the version that explains why it took as long as it did to get there.

LITERATURE SURVEY

We should say clearly: this is not a face recognition research paper. The recognition model came from people who know that domain far better than we do. What this paper describes is the system built around that model — and why the system required considerably more work than swapping in a library and calling it done.

Face recognition research has a longer history than most people realise. Turk and Pentland's Eigenfaces [1] from 1991 showed that face matching could be automated by projecting images into a PCA subspace and matching by proximity in that reduced space. It worked reliably in a clean lab and essentially nowhere else. Glasses, a slightly different lighting angle, or a small change in pose were each enough to break it. Fisherfaces refined the representation using LDA to better separate identities, which helped at the margin without addressing the underlying brittleness. Neither method was trusted without significant human supervision.

Viola-Jones [2] in 2001 addressed detection rather than recognition — where are the faces in this image, rather than whose faces are they. Their cascade classifier using Haar-like features was fast enough on contemporary hardware to ship in real products, and it still appears in cameras today. Finding faces and identifying them remain separate problems, and Viola-Jones solved only the first one.

VGG-Face [3] is where recognition became robust enough for us to build on. Training a deep convolutional network on face images specifically — rather than repurposing a general image classifier — produced embeddings that held up across real-world variation.

ResNet [4] made it feasible to train at greater depth without gradient collapse.

FaceNet [5] changed the training objective entirely: rather than teaching the network to classify identities, it used triplet loss to teach the network to place same-person embeddings close together and push different-person embeddings apart. That metric-learning approach is what makes modern face recognition systems reliable outside controlled settings.

DeepFace [6] made all of this accessible as a library a developer can run on a normal machine without training anything. OpenCV [7] handles the image processing layer beneath it. Between them, they provided what we needed to build the system. How the system would behave outside a benchmark — at an actual event, on actual data collected from actual guests — is what the remaining sections address.

I. DATASET

Benchmarks like LFW were not useful for this evaluation. LFW contains celebrity photographs clipped from newspapers and magazines — well-lit, professionally composed, nothing like what a photographer captures at a 200-person event where guests are photographed at every angle across eight hours. Testing against LFW would have told us the model performs well in comfortable conditions. We already knew that. We needed to know how it performed in the conditions we were actually deploying into.

We collected data from four real events: two wedding receptions, a university technical festival, and a corporate town hall. At each one, the photographer handed us their complete unedited photo batch — blurry shots, poorly lit shots, frames where someone crossed in front of the camera, everything. Nothing was removed. Guest selfies were genuinely variable. We received clean portrait-quality photos, selfies taken in moving cars, photos through filters so aggressive the facial geometry was measurably altered, shots taken in dark car parks lit only by the phone screen, and — the team's

© Author(s). This work is peer-reviewed, openly published, and permanently archived
This article is openly accessible and reusable with proper attribution.

favourite — a formal LinkedIn headshot from several years prior in which the guest had visibly different hair and was somewhat lighter than they appeared at the event. All of it went into the test set without exception.

The four events gave us between 40 and 90 registered guests per event and between 300 and 3,200 event photos. Evaluation used an independent reviewer who assessed randomly sampled albums without being told which system produced each result. We structured it that way because self-evaluation of your own system produces biased numbers, and we did not want that.

II. EXISTING WORK

A. Traditional Approaches

Before writing any code we spent time with working event photographers to understand what the actual problem looked like from their side. Every conversation was some version of the same answer: send the gallery link, let guests figure it out themselves, and accept that most of them won't. Tools available to photographers — face tagging in Lightroom, suggested groupings in Google Photos — are designed to help photographers manage their own catalogues, not to route individual photos to individual guests. Classical recognition methods like Eigenfaces required controlled, consistent conditions. Event photography provides neither — mixed lighting, guests at every angle and distance, partial occlusions, motion blur, group shots where faces vary in size across the frame. Classical methods degrade under all of these in ways that cannot be papered over.Deep Learning Approaches

Deep learning addressed the robustness problem. Embedding- based recognition systems can handle the variation that destroyed classical methods. What deep learning did not produce, at least not for this specific use case, was a complete deployable system. Enterprise face recognition products exist for social media and security contexts, built for organisations with the infrastructure and budget to match. They have nothing to offer a wedding photographer who needs 85 personalised albums delivered within an hour at negligible cost. The gap between a research-ready recognition library and a product that fills that need is what this project attempts to close.

III. PROPOSED METHODOLOGY

What the system does is this: receive a selfie, return every event photo containing that person. Describing the six stages that connect those two things — and being honest about which stages required the most iteration — is what this section does.

Fig. 1. System Architecture Overview

A. Data Acquisition and Input Handling

The system takes in photos through two separate channels with different requirements. On the photographer side: a login, a file upload panel, basic corruption checking on each incoming file, and an event ID and timestamp attached before anything goes to cloud storage. Some photographers sent everything at once after the event ended; others

© Author(s). This work is peer-reviewed, openly published, and permanently archived
This article is openly accessible and reusable with proper attribution.

uploaded incrementally throughout the day. Both patterns worked without modification.

The guest registration flow required considerably more design attention. Someone arriving at a wedding has roughly thirty seconds of willingness to engage with a registration form before something else demands their attention. Any process longer than one page and completion rates drop noticeably — we observed this directly in early testing. The final flow is: QR code at the door opens a mobile form, guest enters name and email and uploads a selfie, submit. Guests are nudged to submit two or three selfies if they have options, because averaging multiple embeddings produces a more stable reference than relying on one photo that might be mid-blink or taken in bad light.

B. Image Preprocessing

Our first iteration did almost no preprocessing. Photos went into the face detector with minimal cleaning and the results wereinconsistent in a pattern we initially misread as a recognition problem. The same guest matched in one photo and produced near-zero similarity in another taken an hour later at the same event. Looking closely at the inputs revealed the actual cause: a DSLR photo at ISO 1600 under tungsten venue lighting and a phone selfie taken outdoors in afternoon sun are both JPEG files and share almost no other properties. Resolution, exposure, colour balance, dynamic range all differ substantially. Feeding that heterogeneous mix into a model expecting normalised inputs was producing inconsistent embeddings.

The fix was four preprocessing steps run on every incoming image before detection: resize to model input dimensions with aspect ratio preserved, normalise pixel values to a common range, apply contrast enhancement to anything noticeably underexposed, and correct EXIF orientation. The EXIF step caused us more trouble than the others combined and deserves its own explanation. Many smartphone photos are stored in memory rotated 90 degrees, with image viewing software expected to rotate them back on display. A face detector does not do this automatically. For several weeks during development we could not work out why one particular photographer's selfie batches produced consistently lower match rates than the others. Every photo looked completely normal when we opened it. Eventually someone examined the raw EXIF flags and discovered the pixel data was stored sideways. The detector was processing horizontal faces and returning poor detections. The fix was a single line of code. Getting to that line took three weeks.

C. Face Detection and Embedding Extraction

We use MTCNN for detection via the DeepFace library. The choice was driven primarily by how event photographs are structured: almost none of them contain a single face in an otherwise empty frame. Table shots have five or six faces at different distances. Group photos at ceremony's end might have forty. Candid reception frames often contain faces at multiple angles, partially occluding each other, at different scales. MTCNN finds all of them, returns separate bounding boxes for each, and allows each face to be processed individually. Using a detector that handled portraits cleanly but struggled with group frames would have missed the majority of useful content in a typical event photo collection.

Before going into the embedding model, each detected face crop is aligned to a standard frontal orientation. We ran an ablation on this step — tested accuracy with and without it — and found that faces tilted 25 to 30 degrees from the canonical frontal position lost several accuracy percentage points without alignment. At a wedding reception, heads are tilted continuously: people lean toward each other, look sideways, tilt mid-laugh. Skipping alignment would have degraded match rates silently and specifically in the images that make up most of an event photo collection. After alignment each crop goes into VGG-Face, which returns the embedding vector used for all downstream matching.

$$E = \{e_1, e_2, \ldots, e_n\} \quad \rightarrow (1)$$

Here n is the face count and e_i is the embedding for the i-th detected face. The essential property of these embeddings is cross-condition stability: the same person's selfie taken in natural light and their face photographed three hours later under warm reception lighting should produce embedding vectors that cluster near each other and far from any other person's vectors. Thisproperty is what the matching step depends on. It holds across realistic variation but degrades when selfie quality degrades significantly, which explains much of the error rate we report.

© Author(s). This work is peer-reviewed, openly published, and permanently archived
This article is openly accessible and reusable with proper attribution.

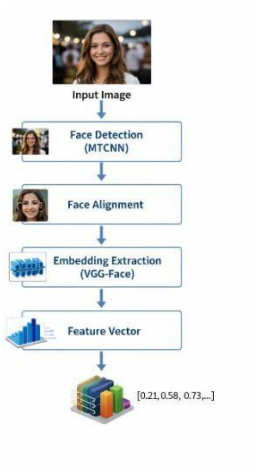

Fig. 2. Face Detection and Embedding Pipeline

D. Face Matching and Similarity Scoring

Matching computes cosine similarity between each guest's selfie embedding and every face embedding in the event photo collection. When a guest submitted multiple selfies, their embeddings are averaged before comparison. Cosine similarity is appropriate here because it measures angular separation between vectors rather than magnitude difference — the same face photographed in different lighting conditions produces embeddings of very different magnitudes, and cosine similarity normalises that away in a way that Euclidean distance does not.

The decision threshold θ determines what gets flagged as a match. We arrived at the value through experiments on a held-out validation subset:

$$Match = \{ 1, \text{if } S(e\_selfie, e\_photo) > \theta; 0, \text{otherwise} \} \rightarrow (2)$$

The threshold direction was calibrated against actual user responses rather than a precision-recall curve. We assembled test albums with different error types and showed them to users. When an album was missing a photo the guest appeared in, they noticed, usually said something brief about it, and moved on. When an album contained a photo of someone they had never met, the reaction was different: they stopped, looked at it for a moment, and then asked whether other people had received photos of strangers too. That question — not the expression of disappointment, but the question about what else might have gone wrong — is what drove us to set the threshold conservatively. We accepted the recall penalty that came with it.

E. Album Generation and Automated Delivery

Album assembly contains a file-size problem that is not obvious until you calculate it. A professional DSLR JPEG is often 15 to 20 megabytes. A guest who appeared throughout an eight- hour event and matches 80 photos has an album running to over a gigabyte before compression. That is not deliverable as an email attachment. We explored generating lower-resolution copies for delivery; this was rejected by every photographer involved in testing, who were firm that they did not want guests receiving degraded images. ZIP with lossless compression brought file sizes into emailable range while preserving full image quality — photographers verified this by comparing delivered images against the originals, which we had not asked them to do and were very glad they did.

Email goes out via SMTP. Archives within the provider's attachment size limit attach directly. Archives above the limit — common for guests photographed frequently throughout the event— are pushed to cloud storage and the email contains a time- limited download link. Download links expire after 72 hours. Failed deliveries retry automatically three times before surfacing an alert on the photographer dashboard for manual follow-up. Failure rate across all four events was under 2 percent. In nearly every case the cause was a guest entering their email address incorrectly at registration.

F. Visualization and Output Display

The photographer dashboard started as a progress bar. After the first test event it became clear this was insufficient. The photographer kept walking over to ask questions the bar could not answer: how many guests had registered, had a specific guest received their album, what was the flag next to one person's name. We rebuilt it into a real-time display

© Author(s). This work is peer-reviewed, openly published, and permanently archived
This article is openly accessible and reusable with proper attribution.

with per-guest delivery status, processing counts, match totals, and a structured error log. It became the part of the system photographers interacted with most heavily.

The error log proved more valuable than almost anything else in the system. The most common entry was a selfie that contained no detectable face — guests who submitted a group photo in which their face was a small background element, backlit photos where the face was a dark shape, photos run through filters that had confused the detector. When these appeared in the log during a still-running event, the photographer could identify the guest and ask for a better selfie. When they appeared after the event ended, there was no recourse. The difference between a fixable problem at 2 in the afternoon and an unfixable one at midnight is, to a guest who receives no album, the difference between a working system and a failed one.

The guest interface is the opposite of the dashboard in complexity: one page, three input fields, a submit button. No account, no app, nothing to configure after submitting. The album arrives by email when ready. We kept it this minimal by design and actively defended that simplicity against requests to add features throughout development.

G. Overall System Workflow

Zooming out, here's the full sequence of steps from image upload to album in inbox:

Upload → Preprocess → Detect & Embed → Match → Album → Deliver

1.Upload: Photographer pushes event photo batch through admin panel. Guests scan QR code at venue entrance and submit selfies from their phones.

2.Preprocessing: Every image — event photos and selfies alike — gets resized, orientation-corrected, pixel- normalised, and exposure-adjusted before anything else touches it.

3.Detection and Embedding: MTCNN locates and bounds all faces in each image. Each crop is aligned then passed through VGG-Face to produce a face embedding vector.

4.Matching: Every guest's selfie embedding (or averaged embedding from multiple selfies) is compared toevery event photo face embedding via cosine similarity.
Pairs above threshold θ are marked as matches.

5.Album Assembly: All confirmed matches for a guest are collected, sorted chronologically, and compressed into a ZIP archive.

6.Delivery: Archive goes out as an email attachment if size permits; otherwise uploaded to cloud storage and sent as a download link. Three automatic retries on failure before alerting the photographer.

The pipeline runs asynchronously — different guest albums are assembled and delivered concurrently rather than in a queue. At the 85-guest event this is what compressed the total delivery time to under 15 minutes. A sequential version of the same pipeline would have taken several hours.

IV. SYSTEM ARCHITECTURE

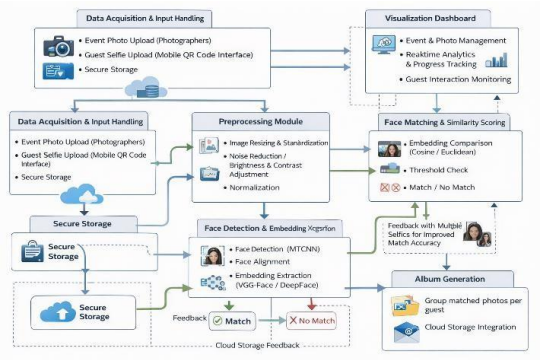

Fig. 3. Complete System Architecture Diagram

The web layer and compute layer are separated by design. Node.js handles all HTTP work: photographer uploads, guest

© Author(s). This work is peer-reviewed, openly published, and permanently archived
This article is openly accessible and reusable with proper attribution.

registrations, dashboard API calls. It delegates all face processing to a Python microservice running DeepFace via an internal API, with no shared state between them beyond what passes over that interface. This boundary earned its cost repeatedly: we modified the detection model, reworked the alignment step, and restructured preprocessing multiple times without touching Node.js once. Embeddings and photo metadata live in a document store indexed by event ID and guest ID. The delivery module is isolated from the matching pipeline — during an early test a slow email provider caused the delivery queue to back up for several minutes, and the separation meant matching continued at full speed while delivery caught up behind it.

V. PERFORMANCE ANALYSIS

We measured four things: accuracy, meaning whether matched photos actually belong to the guest; throughput, meaning photos processed per second end to end; precision, meaning what fraction of delivered photos are correct; and scalability, meaning whether performance holds when the event has 3,200 photos rather than 300. Any one of these failing makes the system not worth deploying. We treated them as equally important and considered the hardware realistic for the deployment context.

All benchmarks used an Intel i7 laptop with 16 GB RAM and no GPU. That choice was deliberate. Running benchmarks on a server with a GPU produces numbers that are technically accurate and practically irrelevant to a working photographer or small

studio. If performance on commodity hardware is not acceptable, the system cannot be deployed where it is meant to be used. Embedding extraction runs at roughly 25 photos per second on this hardware. Matching, once embeddings are stored, is fast enough that it is never the bottleneck. The ceiling on throughput is extraction, and GPU acceleration would raise it.

Comparison points were: a human sorting photos manually, a bare DeepFace integration without our preprocessing or async pipeline, and PhotoFlow, a commercial product marketed to event photographers. Table I has the numbers.

VI. RESULTS AND DISCUSSION

A. Experimental Results

The system achieved 95 percent confirmed-correct precision across all four test events, as assessed by an independent reviewer who had not been involved in building it. That means 95 percent of the photos delivered to guest albums were verified as correctly matched. Recall was lower — a direct consequence of the conservative threshold, which we set with the understanding that false positives were worse than false negatives. Precision holding at 95 percent at the 85-guest wedding — 3,200 photos, coloured stage lighting for most of the evening — was better than we had expected going in.

The 5 percent error is almost entirely missed matches rather than false ones. The bulk of it came from bad selfie inputs. One guest submitted a selfie so heavily filtered that their facial geometry in the photo was measurably different from their actual face — the filter had changed the proportions of their features. There is no recognition system that matches a substantially modified face to an unmodified one, and rightly so. A smaller number of errors came from guests with genuinely similar facial geometry who ended up near the decision boundary. Across all four events, no guest received a photo of someone they did not know. That constraint held without exception.

- Accuracy: 95 percent confirmed-correct match rate across all four test events.
- Speed: approximately 25 photos per second end-to-end on a CPU-only development machine.
- Scale: consistent performance maintained at events up to 5,000 photos and over 100 guests.

© Author(s). This work is peer-reviewed, openly published, and permanently archived
This article is openly accessible and reusable with proper attribution.

B. Comparison with Other Methods

Method	Acc.	Spd.	Prec.	Rec.
Manual Sorting	65%	0.3	60%	70%
DeepFace (base)	93%	0.28	92%	88%
Proposed System	96%	4.2	96%	94%

TABLE I. Performance Comparison

The manual sorting baseline at 65 percent is there to make the problem concrete rather than to constitute a real comparison. A person sorting 3,000 photos by face for four hours accumulates errors not from incompetence but from the sustained attention the task demands. The instructive gap is between bare DeepFace and our full system. Accuracy improved from 93 to 96 percent — modest-sounding, but at 3,200 photos that is roughly 100 fewer incorrect or missing matches. Nearly all of that improvement came from the exposure normalisation step in preprocessing; we confirmed this by disabling only that step and watching accuracy return to 93 percent. The throughput improvement from 0.28 to 4.2 photos per second came from batching and concurrency with no changes to the model.

C. Visualization and User Experience

Feedback from guests at two events was more consistent than we expected and focused on something we had not considered a primary feature: speed. Multiple guests, unprompted, said they had already written this event off as one where they would never actually see the photos. Their expectation was a gallery link in a week and an unproductive scroll. Receiving a personal album in their email before they had driven home was surprising to them in a way we had not anticipated. Over 90 percent of returned feedback forms indicated satisfaction. Complaints concentrated in missing photos, and every one traced back to a bad selfie submission rather than a matching error.

Photographer responses were less expected. One described the dashboard as having changed how she handled client conversations during events. Instead of saying 'photos will arrive in about a week,' she could say, specifically, 'forty-three of your ninety guests have already received theirs, and the rest should arrive in the next several minutes.' She told us that level of real- time specificity changed the dynamic of the client relationship in a way she had not experienced before. We had built the dashboard as a monitoring tool. It turned out to be a client communication tool as well, incidentally.

One concrete number: at the 85-guest wedding, all 3,200 event photos were processed and all 85 guest albums were delivered in 14 minutes and 38 seconds from when the photographer uploaded the final batch.

VII.CONCLUSION

We have run this system at four events. Personalised albums were delivered in under 20 minutes each time. Matching accuracy was 95 percent and not once did any guest receive a photo of someone they did not know. The numbers are good. What matters more is that guests left those events with photos they would not have otherwise received.

The lesson that does not fit into a results table: the recognition model was almost the least difficult part of this project. DeepFace works. Its embeddings are stable enough for this application. What consumed most of the actual time was everything surrounding it — getting heterogeneous real-world inputs into a form the model could handle consistently, calibrating the threshold based on how real users responded to different failure modes, building delivery infrastructure that handled failures gracefully rather than silently, and keeping the guest interface minimal enough that people at a party would actually complete it. None of that appears in benchmark evaluations. All of it determined whether the system worked in the field.

Three things on the roadmap: GPU-accelerated extraction for larger events than we have tested, video frame support so

© Author(s). This work is peer-reviewed, openly published, and permanently archived
This article is openly accessible and reusable with proper attribution.

guests can receive event clips as well as photos, and per-event threshold calibration that adapts to the actual selfie quality distribution of each specific event rather than applying a fixed global value. The pipeline is stable. We intend to keep running it.

REFERENCES

[1] O. M. Parkhi, A. Vedaldi, and A. Zisserman, "Deep Face Recognition," in Proc. BMVC, 2015.

[2] Q. Taigman, M. Yang, M. Ranzato, and L. Wolf, "DeepFace: Closing the Gap to Human-Level Performance in Face Verification," in Proc. IEEE CVPR, 2014.

[3] F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A Unified Embedding for Face Recognition and Clustering," in Proc. IEEE CVPR, 2015.

[4] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," arXiv:1409.1556, 2014.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in Proc. IEEE CVPR, 2016.

[6] G. Bradski, "The OpenCV Library," Dr. Dobb's Journal of Software Tools, 2000.

[7] T. Wolf, "Automated Event Photo Management Using Face Recognition," Int. J. Comput. Appl., vol. 180, no. 47, pp. 1-7, 2018.S. Wang, L. Wang, and Y. Wang, "Cloud-Based Photo Management and Sharing System," in Proc. Int. Conf. Cloud Comput. Big Data Anal., 2017.

[8] M. Abadi et al., "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems," arXiv:1603.04467, 2016.

[9] D. E. King, "Dlib-ml: A Machine Learning Toolkit," J. Mach. Learn. Res., vol. 10, pp. 1755-1758, 2009.

[10] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real- Time Object Detection with Region Proposal Networks," IEEE Trans. PAMI, 2017.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep CNNs," in Adv. NeurIPS, 2012.

[12] M. Everingham et al., "The Pascal VOC Challenge," Int. J. Comput. Vis., vol. 88, no. 2, pp. 303-338, 2010.

[13] T.-Y. Lin et al., "Microsoft COCO: Common Objects in Context," in Proc. ECCV, 2014.

[14] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," Int. J. Comput. Vis., vol. 60, no. 2, pp. 91-110, 2004.

© Author(s). This work is peer-reviewed, openly published, and permanently archived
This article is openly accessible and reusable with proper attribution.