International Journal of Engineering Research and Science & Technology

ISSN 2319-5991 www.ijerst.com
© 2026 IJERST. All Rights Reserved
Peer Reviewed, Referred & Indexed Journal

Research Paper

Sports Image Classification and Batch Management Using Convolutional Neural Network (CNN)

Tanna Divya Sri
PG MCA Scholar
Department of Computer Science
Sir C R Reddy College, Eluru,
Andhra Pradesh, India-534005
divyasrithanna@gmail.com

Anuradha Komati
Assistant Professor
Department of Computer Science
Sir C R Reddy College, Eluru
mail id: anu.komati@gmail.co

Musunuru Ratnakar
HOD
Department of Computer Science
Sir C R Reddy College, Eluru,
Andhra Pradesh, India-534005

Abstract— The rapid growth of digital sports media creates a practical requirement for automatically classifying and organizing large image collections. This paper presents the Sports Image Classification and Batch Management system documented in the project report. The desktop application combines a custom Keras convolutional neural network with a MobileNetV2 fallback, OpenCV-based preprocessing, a Tkinter graphical interface, SQLite authentication, and a multithreaded batch organizer. The custom model operates on 150×150 RGB inputs for Cricket, Football, and Basketball, whereas the fallback path uses 224×224 inputs and maps predictions to Cricket, Football, Basketball, Volleyball, and Tennis. Single-image prediction exposes RGB conversion, resizing, normalization, tensor expansion, class probability, and confidence output. Batch mode scans mixed folders, applies a configurable confidence threshold, creates category-specific directories, copies or moves files, routes uncertain samples to an Unclassified folder, and reports progress through a live log and classification-ratio chart. The documented application output classifies a Football image with 95.5% confidence and confirms successful end-to-end inference. The project demonstrates a compact and scalable workflow for sports academies, photographers, media archives, educational institutions, and event-management teams while retaining responsiveness through worker threads and thread-safe Tkinter updates.

Keywords— Sports image classification, convolutional neural network, MobileNetV2, batch image organization, Tkinter, SQLite, transfer learning.

I. INTRODUCTION

Sports photographs are produced continuously by academies, tournaments, broadcasters, photographers, news agencies, and social-media teams. A single event may generate thousands of files containing different games, camera angles, lighting conditions, uniforms, equipment, and backgrounds. Manual sorting requires repeated visual inspection, is difficult to scale, and can introduce inconsistent folder names or misplaced images. Automated classification converts this repetitive activity into a repeatable computer-vision workflow in which each image is assigned to a sport category and stored in a corresponding directory [1]–[4].

Convolutional neural networks are suited to this problem because they learn visual descriptors directly from pixels. Early layers respond to edges, colour transitions, and texture, while deeper layers represent objects and contextual patterns such as bats, wickets, goalposts, courts, jerseys, balls, and player posture. This hierarchical representation reduces dependence on handcrafted histograms or edge descriptors and supports classification under moderate changes in scale, background, and pose [1], [5]–[9].

The project report extends classification into a complete desktop management system rather than presenting only a training notebook. It provides user registration and login, single-image prediction, visualization of preprocessing matrices, directory selection, confidence-based routing, copy or move modes, progress monitoring, dynamic class-count visualization, and help instructions. The central contribution is therefore the integration of CNN inference with practical batch file management and responsive desktop interaction [30].

II. RELATED WORK

AlexNet demonstrated that deep convolutional learning could outperform conventional image-recognition pipelines at large scale [1]. VGGNet subsequently showed the effectiveness of increased depth using small 3×3 kernels [2], while GoogLeNet improved multiscale processing efficiency through inception modules [3]. Residual learning enabled substantially deeper networks by using identity shortcuts [4], and DenseNet promoted feature reuse through dense connectivity [9]. These architectures established the representation-learning principles used by contemporary image classifiers.

The transition from handcrafted features to learned representations is especially important in sports scenes. Colour histograms may distinguish an indoor court from a grass field, but they can fail when advertisements, spectators, lighting, or uniforms dominate the frame. Edge and texture descriptors may respond to goalposts, court lines, or bats, yet they do not directly model the relationship between equipment, players, and environment. CNNs progressively combine local responses into higher-level patterns, allowing a class decision to depend on several weak but complementary cues. Batch normalization stabilizes intermediate activations, dropout reduces co-adaptation, and transfer learning reuses generic visual filters when a sports-specific dataset is limited [11], [13]–[15].

Resource-aware deployment motivated lightweight networks. MobileNet introduced depthwise separable convolutions for mobile and embedded systems [6], and MobileNetV2 added inverted residuals with linear bottlenecks to improve the accuracy–efficiency trade-off [7]. Xception also demonstrated the effectiveness of depthwise separable convolution [5], while EfficientNet formalized compound scaling of depth, width, and input resolution [10]. These works support the project report's selection of MobileNetV2 as the fallback model for local CPU inference.

Sports computer vision applies deep learning to player detection, ball identification, field localization, team discrimination, jersey recognition, scene understanding, and action analysis [20]–[26]. However, research prototypes often concentrate on model accuracy rather than the operational workflow required to organize user-owned files. The present

project addresses this practical gap by combining classification, authentication, preprocessing visualization, batch routing, disk operations, and progress reporting within one offline-capable application [30].

III. MATERIALS AND METHODS

The proposed system architecture is organized into four coordinated layers, as shown in Fig. 1. The presentation layer provides Tkinter screens for registration, login, the main dashboard, single-image prediction, batch organization, results, preprocessing views, logs, charts, and help. The application-control layer uses main.py for navigation and validation and a worker thread with a thread-safe queue for long-running batch operations. The AI inference layer performs OpenCV preprocessing and selects either the custom model.h5 CNN or the MobileNetV2 fallback before applying Softmax confidence and threshold-based routing. The data and storage layer maintains user accounts in SQLite, reads individual images or mixed source folders, and writes the organized category folders.

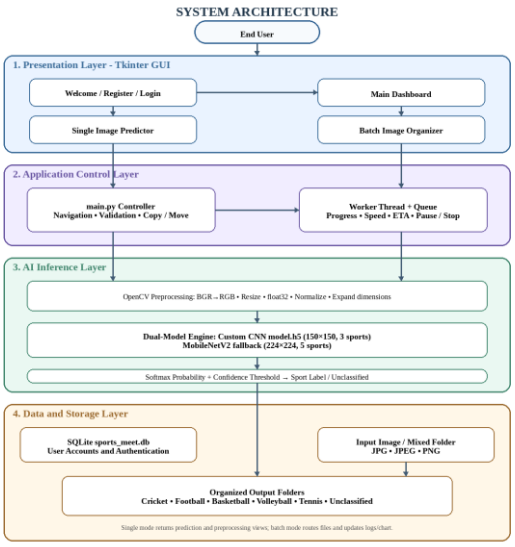


Fig. 1. System architecture for CNN-based sports image classification and batch management.

A) Dataset and Dual-Model Framework

i) Custom CNN Dataset: The report describes a manually curated dataset containing more than 100 labelled images for the custom three-class model. The primary categories are Cricket, Football, and Basketball. Representative visual cues include bats, stumps, helmets, pitches, goalposts, grass fields, footballs, indoor courts, hoops, and sport-specific uniforms. Data augmentation is used during training to introduce rotation, translation, shear, zoom, and horizontal reflection so that the network does not memorize only fixed backgrounds or camera positions [13], [30].

ii) MobileNetV2 Fallback: When the user-trained model.h5 file is unavailable, the application loads or caches MobileNetV2 and maps its top predictions to five configured categories: Cricket, Football, Basketball, Volleyball, and Tennis. This fallback preserves application usability and exploits ImageNet-pretrained visual features without requiring local retraining [7], [15], [30].

iii) Visual Class Characteristics: Cricket scenes may contain bats, stumps, helmets, gloves, pads, a prepared pitch, or a large oval field. Football scenes often contain rectangular grass regions, goal frames, nets, a spherical ball, and groups of players wearing shorts and jerseys. Basketball images typically contain an indoor wooden court, painted key areas, hoops, backboards, and sleeveless uniforms. Volleyball and tennis introduce nets, court markings, rackets, and characteristic ball sizes. The classifier does not receive these descriptions explicitly; instead, convolutional filters learn combinations of edges, textures, shapes, colours, and spatial arrangements that correlate with each category [30].

iv) Dataset Challenges: Sports images exhibit high intra-class variation because the same game can be photographed from close-up, sideline, aerial, broadcast, or spectator viewpoints. Inter-class overlap is also substantial: several sports use grass surfaces, multiple games contain nets, and player uniforms may look similar. Background clutter, scoreboards, advertisements, low-resolution crops, partial equipment, motion blur, and night lighting can cause the network to focus on nonessential details. Balanced class collection and augmentation should therefore be accompanied by careful validation on events, venues, and photographers that were not represented during training. This prevents a model from appearing accurate merely because it memorized a stadium background or a recurring team jersey.

Table 1. Dataset, Model, and Output Configuration

Mode	Input / source	Classes or output	Purpose
Custom CNN	100+ labelled sports images; 150×150×3	Cricket, Football, Basketball	Project-trained classifier
MobileNetV2 fallback	ImageNet-pretrained; 224×224×3	Cricket, Football, Basketball, Volleyball, Tennis	Inference when model.h5 is absent
Batch organizer	Mixed JPG/PNG/JPEG folder	Category folders + Unclassified	Automatic copy/move organization

B) Image Pre-processing and Feature Learning

Images are read with OpenCV as BGR matrices, converted to RGB, resized to the active model's dimensions, converted to float32 arrays, normalized to the interval [0,1], and expanded to the batch form [1,H,W,3]. The preprocessing visualization module exposes these intermediate representations so that users can inspect the transformation process rather than receiving only a final class label [28], [30].

For input image I and kernel K, a two-dimensional convolution produces feature map S:

$$S(i,j) = \Sigma^{m}_{u=-m}\, \Sigma^{n}_{v=-n}\, I(i-u,j-v)K(u,v) \qquad (1)$$

The Rectified Linear Unit introduces non-linearity and suppresses negative activations:

$$f(x) = \max(0,x) \qquad (2)$$

After convolution and pooling, dense outputs z are transformed into class probabilities using Softmax:

$$p(c|x) = \exp(z_c) \,/\, \Sigma^{c}_{k=1} \exp(z_k) \qquad (3)$$

Training minimizes categorical cross-entropy, while Adam adapts the learning rate from first- and second-order gradient moments [11], [12]:

$$L = -\,\Sigma^{c}_{k=1}\, y_k \log(p_k) \qquad (4)$$

C) Training, Optimization, and Generalization

The report explains that the custom CNN is trained with labelled sports images and uses augmentation to increase visual diversity without changing the semantic class. Random rotation, horizontal and vertical shifts, shear, zoom, and horizontal flipping expose the network to altered camera placement and player orientation. Pooling reduces spatial resolution and computation, while dropout and regularization reduce dependence on individual activations. These controls are especially important for a small custom dataset because the training images may otherwise be memorized rather than

generalized to new venues, lighting conditions, and backgrounds [11]–[14], [30].

Adam updates network parameters from adaptive first- and second-moment estimates of the gradient. For gradient g_t, the moment estimates are:

$$m_t = \beta_1 m_(t-1) + (1-\beta_1) g_t ; \quad v_t = \beta_2 v_(t-1) + (1-\beta_2) g_t^2 \qquad (5)$$

After bias correction, the parameter vector θ is updated according to:

$$\theta_(t+1) = \theta_t - \eta \, \hat{m}_t / (\sqrt{\hat{v}_t} + \varepsilon) \qquad (6)$$

Training and inference are separated operationally. During training, the optimizer modifies the weights to reduce cross-entropy. During desktop prediction, the saved weights are frozen and only the forward pass is executed. The fallback MobileNetV2 path similarly uses pretrained weights and does not retrain during ordinary use, which keeps startup and prediction suitable for local CPU deployment [7], [12], [30].

D) Architecture, Security, and Threading

The implementation is modular. The presentation layer consists of Tkinter frames for the welcome page, authentication, menu, single predictor, preprocessing viewer, batch organizer, abstract, flowchart, and help screens. The control layer coordinates model loading, image inference, batch loops, file operations, progress calculation, and queued interface updates. The data layer stores accounts in the local sports_meet.db SQLite database and manages source and destination folders on disk [30].

Long-running classification and sorting are executed in worker threads so the graphical interface remains responsive. Because Tkinter widgets are not thread-safe, workers place update functions in a queue and the main event loop processes them through after(). This design separates computational work from rendering and prevents the application from appearing frozen during model loading or multi-image organization [30]. Parameterized SQLite queries are used for account operations, reducing the risk of malformed input affecting database commands.

IV. SYSTEM IMPLEMENTATION

A) Software Components

The application is developed in Python 3.10 using TensorFlow/Keras for neural inference, OpenCV and NumPy for image conversion, Tkinter for the desktop user interface, SQLite for local authentication, and threading and queue modules for asynchronous organization. The source is separated into configuration constants and mappings, model-engine functions, and the main interface and database controller, improving maintainability and allowing the model or category map to be replaced independently [29], [30].

The single-image workflow begins when an authenticated user selects a JPG, JPEG, or PNG file. The interface validates that a file was selected and presents a preview region. When prediction starts, the application loads the active model, shows a progress screen, converts the image to RGB, applies the required spatial size, normalizes the array, and performs a forward pass. The predicted sport, confidence percentage, and success status are then rendered in a result frame. A separate control opens the preprocessing viewer so that the same input can be inspected at intermediate stages. Navigation controls allow the user to return to the predictor without restarting the application.

The batch workflow extends these operations to a directory. The source folder may contain a mixture of supported image types, while the destination folder stores the generated category directories. Before processing, the user selects copy mode, which preserves the originals, or move mode, which performs an in-place organizational transfer. Each file is validated, preprocessed, classified, and compared with the threshold. Successful categories increment their counters, uncertain items increment Unclassified, and failed or unsupported files are reported through the log. The worker periodically updates queue size, speed, elapsed progress, and estimated completion time. Pause and stop controls provide operational control during a large run, and the final chart summarizes how many images were routed to each category.

Table 2. Important Implementation Settings

Component	Documented setting	Operational role
Custom input	150×150 RGB, float32, normalized	Three-class model inference
Fallback input	224×224 RGB	MobileNetV2 inference
Authentication	SQLite users table	Registration and login
Batch threshold	Configurable; tests include 80%	Routes low-confidence images to Unclassified
File mode	Copy or move	Preserves or relocates source files
GUI updates	Worker thread + queue + after()	Responsive progress and logs

Table 3. Functional Modules and User-Visible Outputs

Module	Primary action	User-visible output
Registration / login	Validate account data	Authenticated access or clear error message
Single predictor	Select and classify one image	Sport label, confidence, status
Preprocessing viewer	Inspect transformation stages	RGB/resized/normalized/tensor views
Batch organizer	Classify and route a folder	Progress, speed, ETA, logs, category chart
Help / abstract	Display operating guidance	Instructions and project description

B) Confidence-Based Inference and Routing

The active model returns a probability vector. The highest probability determines the predicted class and is displayed as a confidence percentage. During batch mode, a configurable threshold τ separates confident predictions from uncertain inputs. Images below the threshold are not forced into a sport folder; they are directed to Unclassified, which reduces the operational impact of blurry, corrupted, or visually ambiguous samples. The editable routing rule is:

$$r(x) = \arg\max_c \, p(c|x), \ \text{if } \max_c \, p(c|x) \geq \tau; \quad \text{otherwise Unclassified} \qquad (7)$$

The organizer then creates destination directories, applies the selected copy or move operation, increments category counts, updates the progress bar, estimates completion time, writes real-time log messages, and redraws the classification-ratio chart. These steps convert model output into a usable asset-management workflow rather than leaving predictions as isolated labels [30].

C) Lightweight Model Selection

The report compares VGG16, ResNet50, InceptionV3, and MobileNetV2 for local deployment. VGG16 contains approximately 138 million parameters and is reported at about 1.2 s per CPU image, while MobileNetV2 contains approximately 3.4 million parameters and is reported at about 0.1 s per image. ResNet50 and InceptionV3 provide intermediate values. Fig. 2 combines parameter count and CPU inference time, showing why MobileNetV2 is appropriate for an offline desktop fallback [2]–[7], [30].

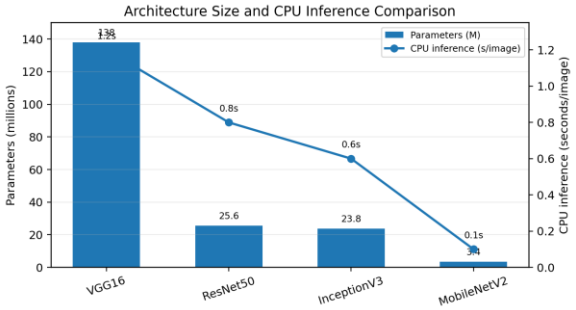


Fig. 2. Reported parameter count and CPU inference comparison.

V. EXPERIMENTAL RESULTS AND DISCUSSION

A) Single-Image Classification Output

The finalized predictor accepts a user-selected sports image, displays a processing screen, executes the active CNN, and renders the class and confidence. The documented output in Fig. 3 reports Football with a confidence match of 95.5% and the status "Prediction Successful." The screen also provides direct navigation to the preprocessing matrices, connecting the prediction with its underlying RGB conversion, resizing, array generation, normalization, and tensor-shape stages [30].

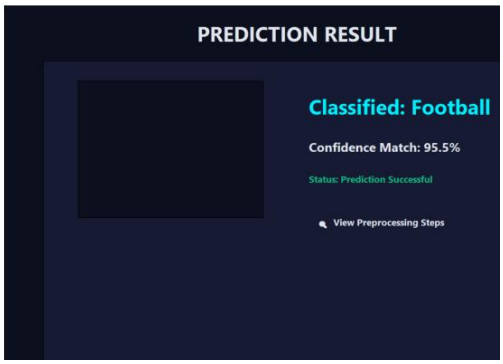


Fig. 3. Documented Football prediction with 95.5% confidence.

B) Pre-processing Visualization

The preprocessing viewer presents the internal image matrices in a dedicated interface rather than hiding all transformations inside the model engine. Users can move through the original RGB representation, resized image, normalized pixel array, and expanded tensor and observe the corresponding shape. This feature is particularly useful in an academic project because it links the visible image to the numerical representation processed by the CNN. It also helps identify incorrect colour conversion, unexpected dimensions, or failed image loading before a prediction is interpreted [30].

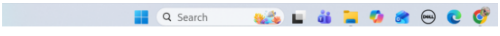


Figure 11: Single image classification prediction resu

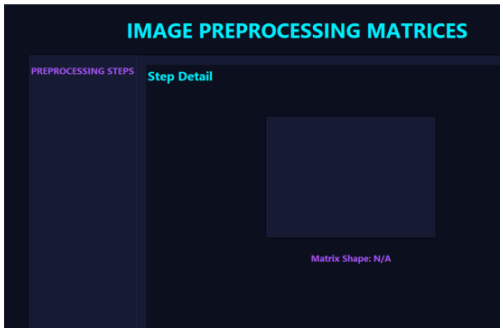


Fig. 4. Pre-processing matrix visualization interface.

C) Batch Organization Output

The Batch Event Image Organizer accepts a mixed source directory and an organized destination directory. Users select copy or move mode and choose a confidence threshold before starting the worker. The interface provides a queue count, processing speed, estimated completion time, progress bar, pause and stop controls, live operations log, and a class-ratio plot for Cricket, Football, Basketball, Volleyball, Tennis, and Unclassified. Fig. 4 shows the implemented organizer before processing begins [30].

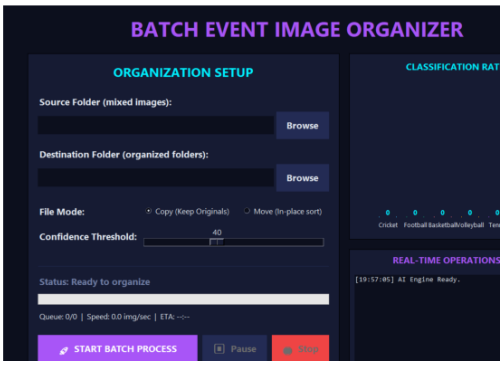


Fig. 5. Batch organizer with threshold, progress, logs, and class ratio.

D) Documented Results and Performance Indicators

The project report does not provide a dataset-level confusion matrix or measured Precision, Recall, and F1-Score. It states a non-functional target of at least 85% classification accuracy, documents a Football prediction with 95.5% confidence, and compares CNN architectures using parameter count and approximate CPU inference time. Accordingly, this paper reports only those source-supported observations and does not introduce unreported model metrics.

Table 4. Documented Output and Runtime Indicators

Indicator	Documented observation	Interpretation
Single prediction	Football; 95.5% confidence	Successful end-to-end inference
Preprocessing	RGB, resize, normalize, tensor expansion	Traceable input transformation
Threshold routing	Low-confidence sample → Unclassified	Avoids forced uncertain labels
Batch control	Progress, speed, ETA, pause/stop, live log	Operational transparency
Threading	GUI remains responsive during sorting	Background inference is decoupled from UI
File output	Copy/move into category directories	Automated asset organization

E) Report-Supported Model Performance

The report compares VGG16, ResNet50, InceptionV3, and MobileNetV2 for CPU-oriented deployment. The listed values are approximately 138 million parameters and 1.2 s per image for VGG16, 25.6 million and 0.8 s for ResNet50, 23.8 million and 0.6 s for InceptionV3, and 3.4 million and 0.1 s for MobileNetV2. These values describe model size and inference latency rather than classification Accuracy, Precision, Recall, or F1-Score.

No Accuracy, Precision, Recall, or F1-Score formulas are included because the attached project report does not provide those formulas or measured values for the final model. The convolution, ReLU, Softmax, categorical cross-entropy, and Adam equations in Section III-B are retained because they are

explicitly described in the report and are directly related to CNN inference and training.

F) Discussion

The output demonstrates that the project is more than a three-class demonstration model. Authentication controls access, the preprocessing viewer provides educational transparency, the fallback model protects against a missing custom file, confidence routing creates a safe destination for uncertain samples, and the batch worker integrates inference with directory management. The separation between custom and fallback modes also supports gradual deployment: a user can operate the application immediately with MobileNetV2 and later replace it with a domain-trained model.

The main limitation is the absence of a reported independent test-set confusion matrix and per-class metrics. A single confidence score shows the certainty of one output but does not establish generalization across all sports, camera conditions, or datasets. The custom training set is small, and visually similar contexts, background clutter, class imbalance, and intra-class variation can affect robustness. Therefore, future evaluation should report class counts, train/validation/test separation, Accuracy, macro Precision, macro Recall, macro F1-Score, confusion matrix, calibration, and average batch throughput on specified hardware.

G) Usability, Reliability, and Security Observations

The report presents a consistent dark desktop interface with clearly separated pages for welcome, login, registration, system menu, prediction, preprocessing, organization, abstract, and help. This page-based structure reduces the amount of information displayed at one time and gives nontechnical users an obvious sequence: authenticate, select an operation, choose input, run inference, and inspect the result. Status labels, progress bars, folder browse controls, mode selectors, logs, and help directions provide feedback at each stage. The design also avoids exposing TensorFlow or file-system commands to the user, which is important when the application is used by students, photographers, or office staff rather than developers.

Reliability is improved by validating file paths and formats, retaining the last responsive interface state while a worker executes, and routing low-confidence images to a neutral folder. Copy mode provides an additional safeguard because the original collection remains unchanged. SQLite allows the program to operate without a separate database server or internet connection, although production deployment should store password hashes rather than plain credentials, enforce stronger account rules, and record login and file-operation events. The model file and category mapping should also be protected from accidental modification because an incorrect mapping could systematically place valid photographs in the wrong directories.

VI. APPLICATIONS AND FUTURE DIRECTIONS

The system can support sports academies that receive event photographs from multiple coaches, photographers who must deliver class-specific folders, educational laboratories demonstrating CNN preprocessing, media archives that require searchable categories, and event-management teams preparing digital assets. Automatic tagging can also serve as the first stage of content-based retrieval, highlight preparation, analytics, recommendation, and multi-sport archive maintenance [20]–[27].

The report identifies video categorization, cloud synchronization, object detection, and a local retraining interface as principal extensions. Video support could sample frames or segments and route highlights by sport. Cloud integration could synchronize organized folders to managed storage. YOLO or SSD could identify players, jerseys, balls, and sport-specific objects before or alongside scene classification [18], [19]. A retraining page could accept new labelled folders, apply augmentation, record training curves, export an updated model.h5 file, and display per-class evaluation before deployment.

Additional work should include password hashing, role-based accounts, duplicate-file detection, perceptual hashing, metadata indexing, resumable batch jobs, audit logs, configurable image formats, and model-version tracking. For larger deployments, a service-based inference layer could separate the desktop client from GPU-backed classification while preserving the current interface. Quantization or pruning could further reduce memory and latency for mobile or edge devices.

A recommended deployment process is to maintain separate development and production model files. A newly trained model should first be evaluated on a locked test set and then run in shadow mode on a copy of an event folder. Reviewers can inspect the Unclassified directory and a random sample from every predicted class before the model replaces the current version. The software should record the model name, threshold, processing date, and file counts in a summary report. These controls make batch organization auditable and allow an incorrect model or threshold to be rolled back without losing the original photographs.

For research extension, the batch organizer can become a dataset-building tool. Users could correct misclassified files, and the corrections could be stored as verified labels for later retraining. Active-learning strategies may prioritize low-confidence or disagreement samples, reducing the effort needed to expand the dataset. A future comparison should evaluate the custom CNN, MobileNetV2, ResNet50, and EfficientNet under the same split and augmentation policy, measuring macro F1-Score, calibration, CPU latency, memory use, and files processed per minute. This would connect classification quality directly with the operational requirements of local desktop deployment.

VII. CONCLUSION

This paper transformed the attached project report into a compact conference-paper description of a CNN-based sports image classifier and batch management application. The system combines custom three-class inference, a five-category MobileNetV2 fallback, OpenCV preprocessing, confidence-based routing, SQLite authentication, Tkinter visualization, worker-thread execution, and physical folder organization. The documented Football prediction reaches 95.5% confidence, and the batch interface provides threshold control, progress, speed, ETA, logs, and class-count visualization. By integrating visual classification with file management, the project offers a practical foundation for intelligent sports-media organization. Future work should expand the dataset, introduce standardized classification metrics, and extend the application to video, object detection, cloud storage, and user-driven retraining.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in Proc. Adv. Neural Inf. Process. Syst., vol. 25, 2012, pp. 1097–1105.

[2] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in Proc. Int. Conf. Learn. Representations, 2015.

[3] C. Szegedy et al., "Going deeper with convolutions," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2015, pp. 1–9.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2016, pp. 770–778.

[5] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2017, pp. 1251–1258.

[6] A. G. Howard et al., "MobileNets: Efficient convolutional neural networks for mobile vision applications," arXiv:1704.04861, 2017.

[7] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2018, pp. 4510–4520.

[8] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception architecture for computer vision," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2016, pp. 2818–2826.

[9] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2017, pp. 4700–4708.

[10] M. Tan and Q. V. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in Proc. Int. Conf. Mach. Learn., 2019, pp. 6105–6114.

[11] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in Proc. Int. Conf. Mach. Learn., 2015, pp. 448–456.

[12] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in Proc. Int. Conf. Learn. Representations, 2015.

[13] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," J. Big Data, vol. 6, Art. no. 60, 2019.

[14] S. J. Pan and Q. Yang, "A survey on transfer learning," IEEE Trans. Knowl. Data Eng., vol. 22, no. 10, pp. 1345–1359, Oct. 2010.

[15] J. Deng et al., "ImageNet: A large-scale hierarchical image database," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2009, pp. 248–255.

[16] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," Proc. IEEE, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[17] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in Proc. Int. Conf. Mach. Learn., 2010, pp. 807–814.

[18] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2016, pp. 779–788.

[19] W. Liu et al., "SSD: Single shot MultiBox detector," in Proc. Eur. Conf. Comput. Vis., 2016, pp. 21–37.

[20] K. Lu, J. Chen, J. J. Little, and H. He, "Light cascaded convolutional neural networks for accurate player detection," arXiv:1709.10230, 2017.

[21] V. Reno, N. Mosca, R. Marani, M. Nitti, T. D'Orazio, and E. Stella, "Convolutional neural networks based ball detection in tennis games," in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops, 2018.

[22] A. Senocak, T.-H. Oh, J. Kim, and I. S. Kweon, "Part-based player identification using deep convolutional representation and multi-scale pooling," in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops, 2018.

[23] M. Koshkina, H. Pidaparthy, and J. H. Elder, "Contrastive learning for sports video: Unsupervised player classification," in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops, 2021, pp. 4528–4536.

[24] N. Patsiouras, P. Nousi, and I. Pitas, "Few-shot image recognition for UAV sports cinematography," in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops, 2020.

[25] T. Nonaka et al., "Rugby scene classification enhanced by vision-language model," in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops, 2024.

[26] A. Fani, S. S. Mirzaei, and A. Etemad, "Hockey action recognition via integrated deep networks," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops, 2017.

[27] N. Homayounfar, S. Fidler, and R. Urtasun, "Sports field localization via deep structured models," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2017, pp. 4012–4020.

[28] G. Bradski, "The OpenCV library," Dr. Dobb's J. Software Tools, 2000.

[29] M. Abadi et al., "TensorFlow: A system for large-scale machine learning," in Proc. 12th USENIX Symp. Operating Systems Design and Implementation, 2016, pp. 265–283.

[30] T. Divya Sri, "Sports image classification and batch management using convolutional neural networks (CNN)," MCA Project Report, Sir C R Reddy Autonomous College for PG Courses, Eluru, India, 2026.