International Journal of
AI Electronics and Nexus Energy
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0515 www.zesterapublications.com Original Research Paper

CLASSIFICATION OF ARRHYTHMIA BY USING DEEP LEARNING WITH 2-D ECG SPECTRAL IMAGE REPRESENTATION

Akarapu Anuhya
Student, Department of Computer Science and Engineering
Nalla Malla Reddy Engineering College
Hyderabad, Telangana, India - 500088
anuhyayamini2115@gmail.com

Muntha Raju
Professor, Department of Computer Science and Engineering
Nalla Malla Reddy Engineering College
Hyderabad, Telangana, India - 500088
raj.jntu9@gmail.com

Sirisha Veluri
Assistant Professor, Department of Computer Science and Engineering
Nalla Malla Reddy Engineering College
Hyderabad, Telangana, India - 500088
velurisirisha77@gmail.com

Abstract*— Automated electrocardiogram (ECG) analysis can support the screening of cardiac rhythm abnormalities when standardized preprocessing and classification are applied. This work presents an image-based ECG arrhythmia classification framework for six rhythm categories: Left Bundle Branch Block (LBBB), Normal, Premature Atrial Contraction (PAC), Premature Ventricular Contractions (PVC), Right Bundle Branch Block (RBBB), and Ventricular Fibrillation. The dataset contains 22,166 ECG waveform images divided into training and test subsets. Images are resized to 64 × 64 pixels and intensity-scaled, while shear, zoom, and horizontal-flip augmentation are applied to the training subset. A sequential CNN comprising three convolution-pooling stages, a 128-unit dense layer, dropout, and a six-unit softmax output performs multiclass classification. The trained network is stored in HDF5 and native Keras formats and deployed through a Flask-based image-upload interface. Experimental evaluation includes epoch-wise learning behavior and three interface-level classification cases. A Normal ECG is classified as Normal and mapped to a healthy-rhythm status, a Ventricular Fibrillation input is classified as Ventricular Fibrillation and mapped to an abnormal-rhythm status, and an LBBB input is classified as PVC and mapped to an abnormal-rhythm status. The results demonstrate the complete inference pipeline from ECG image input to multiclass prediction and user-facing rhythm interpretation, while also revealing class-level confusion among abnormal rhythms.

Keywords*— ECG arrhythmia classification, convolutional neural network, ECG waveform image, deep learning, TensorFlow, Keras, Flask, medical image classification.

I. INTRODUCTION

Electrocardiography is a non-invasive technique for recording the electrical activity of the heart and remains a primary modality for identifying rhythm abnormalities. Automated ECG analysis has therefore received sustained attention in biomedical engineering, particularly for long recordings in which manual beat-by-beat interpretation is time-consuming and difficult to scale [1]-[4]. Conventional approaches commonly involve signal filtering, beat segmentation, handcrafted feature extraction, and statistical classification. Deep learning (DL) reduces dependence on manually specified morphological descriptors by learning discriminative representations jointly with the classifier [5]-[8].

Convolutional neural networks (CNNs) are particularly suitable when ECG information is represented as fixed-size images. A two-dimensional CNN can learn local edges, slopes, peaks, and waveform-shape transitions as spatial patterns. CNN-based methods have been applied to ECG classification using both raw one-dimensional sequences and processed two-dimensional representations [5], [9]-[12]. For application-oriented prototypes, image-based classification provides a practical workflow in which preprocessing, data augmentation, model training, and deployment can be implemented within a unified deep-learning framework without a separate handcrafted feature-engineering stage.

The proposed system follows an image-based approach to six-class arrhythmia classification. The ECG dataset is divided into training and test subsets, and the target classes are LBBB, Normal, PAC, PVC, RBBB, and Ventricular Fibrillation. Keras directory generators create 64 × 64 image tensors, training-time augmentation introduces controlled variation, and a three-stage convolutional network performs automated feature learning and multiclass prediction. Dropout regularization, Adam optimization, and categorical cross-entropy are used during model development. For deployment, Flask routes support information display, image upload, and prediction. Each uploaded waveform image is converted to the model input representation, passed through the trained network, and decoded from the maximum softmax response to the corresponding rhythm label.

Two implementation characteristics are important when interpreting the workflow. First, the dataset is imbalanced, with Normal representing the largest class and LBBB and Ventricular Fibrillation containing substantially fewer images. Second, preprocessing consistency must be maintained between offline model development and online inference. The training and test generators apply pixel rescaling by 1/255, whereas the Flask prediction path resizes the uploaded image and converts it to an array before inference. These factors make class-wise behavior and preprocessing consistency important considerations in the end-to-end system. The resulting application integrates data preparation, CNN training, multiclass inference, model persistence, and web deployment within a single ECG-classification pipeline.

The main contributions of this work are the organization and analysis of a six-class ECG waveform image dataset, implementation of a compact CNN with three convolution-pooling stages and a six-way softmax output, use of augmentation and dropout during model development, and integration of the trained classifier with a Flask-based image-upload and prediction interface. The complete workflow therefore covers dataset preparation, automated feature

International Journal of
AI Electronics and Nexus Energy
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0515 www.zesterapublications.com Original Research Paper

learning, multiclass inference, model persistence, and browser-based presentation of classification results.

II. RELATED WORK

Standardized ECG databases and reproducible evaluation procedures have had a substantial influence on the development of automated arrhythmia analysis. PhysioNet provides an open infrastructure for physiological signals, while the MIT-BIH Arrhythmia Database has become a widely used benchmark for evaluating beat detectors and rhythm classifiers [1], [2]. Earlier signal-processing pipelines relied on QRS detection together with manually designed temporal and morphological features. The Pan-Tompkins algorithm remains a representative method for real-time QRS detection, followed by studies that incorporated morphology, wavelet analysis, dimensionality reduction, support vector machines, and artificial neural networks for beat classification [3], [4], [13].

Deep neural networks have progressively reduced the dependence on handcrafted feature pipelines. Kiranyaz et al. introduced patient-specific one-dimensional CNNs that jointly perform feature extraction and ECG beat classification [5]. Al Rahhal et al. developed discriminative ECG representations using stacked denoising autoencoders and active learning [14]. Kachuee et al. proposed a transferable deep representation for heartbeat classification [6], while Hannun et al. demonstrated large-scale end-to-end arrhythmia classification using ambulatory ECG data [7]. Collectively, these studies show that hierarchical feature learning can model complex rhythm morphology without requiring explicit specification of discriminative features.

The present work is also related to studies that convert ECG signals into image-like representations. Jun et al. classified ECG beat images using a two-dimensional CNN and demonstrated that waveform morphology can be learned directly from grayscale image representations [9]. Other studies have combined convolutional layers with recurrent units, residual connections, attention mechanisms, focal loss, or multi-lead feature fusion to improve temporal context and minority-class discrimination [10]-[12], [15]-[18]. Although these approaches are more complex than the compact network used in this work, they provide relevant design context for image-based ECG classification.

Class imbalance and evaluation design are recurring concerns in ECG classification research. Clinically important minority arrhythmias may occur far less frequently than normal beats, and overall accuracy can therefore obscure poor performance on underrepresented classes [4], [11], [18]. Patient-level separation is also important because beats from the same patient may be correlated, and reported performance can vary substantially between intra-patient and inter-patient evaluation procedures [19], [20]. Patient identifiers and source-database information are not provided with the ECG image dataset used in this work. Accordingly, the evaluation is presented at the image and class levels, and no patient-independent split is claimed.

The implemented CNN also employs standard deep-learning components with established roles: rectified linear units support efficient gradient-based optimization, dropout reduces co-adaptation, Adam provides adaptive parameter updates, and data augmentation expands the effective training distribution [21]-[24]. TensorFlow/Keras provides the model-training and serialization stack, while Flask supports lightweight web-based inference [25]-[27]. The contribution of the system therefore lies in integrating established CNN components into an end-to-end ECG image classification and deployment workflow rather than proposing a new neural architecture.

Deployment reliability depends not only on network architecture but also on consistency across the complete inference pipeline. Reproducible preprocessing, fixed class mappings, model-version control, and consistent train-to-inference data handling are required to preserve the intended behavior of a trained classifier. In particular, the numerical representation used during online prediction should match the representation used during model development, and the class-index order used by the directory generators must remain consistent with the mapping implemented in the web interface. These considerations are directly relevant because model training and browser-based inference are implemented in separate software components. The complete system is therefore treated as a sequence of connected stages comprising preprocessing, training, serialization, class mapping, and online inference.

III. MATERIALS AND METHODS

A) Dataset Collection and Composition

The ECG image dataset is organized into two top-level subsets: training and testing. Across both subsets, the dataset contains 22,166 PNG images. Each image is a grayscale visualization of an ECG waveform with an original size of 192 × 128 pixels. Six class folders are used as target categories: LBBB, Normal, PAC, PVC, RBBB, and Ventricular Fibrillation. The folder structure supplies the class labels used by the Keras directory generators and the six-category mapping applied during web-based inference.

TABLE I. DATASET DISTRIBUTION

Class	Train	Test	Total
LBBB	504	341	845
Normal	7346	2179	9525
PAC	2054	1503	3557
PVC	2759	915	3674
RBBB	2239	1645	3884
Ventricular Fibrillation	439	242	681
Total	15341	6825	22166

The training subset contains 15,341 images and the test subset contains 6,825 images. The class distribution is imbalanced: Normal is the largest class, whereas LBBB and Ventricular Fibrillation contain considerably fewer samples. The implemented training pipeline does not use class weighting, oversampling, or focal loss. Consequently, dataset composition is an important part of the model-development context, and the six target classes are preserved explicitly throughout training, prediction, and interface-level output generation.

The six target folders correspond to distinct ECG waveform categories. LBBB and RBBB represent bundle-branch conduction abnormalities, PAC and PVC represent premature atrial and ventricular contractions, respectively, Ventricular Fibrillation represents a disorganized rhythm pattern, and Normal serves as the reference class. During

International Journal of
AI Electronics and Nexus Energy
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0515 www.zesterapublications.com Original Research Paper

online prediction, the detailed six-class output is retained and is additionally mapped to a simplified rhythm status: Normal is displayed as a healthy rhythm, whereas the remaining classes are displayed as abnormal rhythms. The class-specific label is retained because the binary status alone does not identify the predicted arrhythmia category.

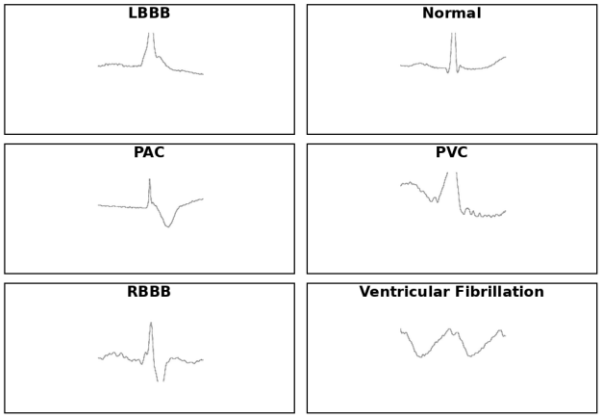


Fig. 1. Representative ECG waveform images from the dataset.

B) Image Preparation and Augmentation

Image preprocessing is implemented using Keras ImageDataGenerator objects. Training images are rescaled by 1/255 and augmented using shear_range = 0.2, zoom_range = 0.2, and horizontal_flip = True. Test images are rescaled by 1/255 without augmentation. Both generators use target_size = (64, 64), batch_size = 32, and class_mode = categorical. Because color_mode is not explicitly changed, the default Keras RGB configuration is used, and the network receives 64 × 64 × 3 tensors even though the source PNG images are grayscale. Class indices are generated from the six class directories and are used consistently for multiclass training and subsequent label decoding.

Augmentation is applied only to the training generator through shear, zoom, and horizontal flipping, whereas the test generator performs intensity rescaling without augmentation. These transformations are intended to increase visual variation during model development. Because ECG morphology is clinically meaningful, the selected augmentation parameters should be treated as part of the model configuration and interpreted carefully rather than as a substitute for physiologically grounded signal variation.

C) Feature Learning and Classification Strategy

The classifier uses a hierarchical feature-learning pipeline with three convolutional stages. The first stage applies 32 filters of size 3 × 3 to capture local waveform edges and short morphological patterns. The second stage increases the representation to 64 feature maps, and the third stage uses 128 filters to capture higher-level spatial structure. Each convolution is followed by a ReLU activation and 2 × 2 max pooling to reduce spatial dimensions while retaining dominant activations. The resulting feature maps are flattened and passed to a fully connected layer with 128 units. A dropout rate of 0.5 is applied before the final six-unit softmax layer, which produces class probabilities for LBBB, Normal, PAC, PVC, RBBB, and Ventricular Fibrillation.

The network operates directly on fixed-size ECG waveform images rather than on manually constructed signal descriptors. Convolutional filters learn spatial patterns that discriminate among the image representations, max pooling provides limited tolerance to small spatial variations, and the dense layer combines the learned features for multiclass prediction. The saved model contains 683,974 trainable parameters. Because the dataset is imbalanced, preserving the complete six-class output remains important so that the predicted rhythm category is not reduced solely to a binary normal/abnormal status.

D) Training and Model Persistence

The network is trained using the Adam optimizer, categorical cross-entropy loss, and accuracy as the monitored classification metric. Training is performed for 10 epochs using batches generated by the Keras data pipeline. Augmented batches are produced from the training directory, while the test generator is supplied as validation_data to provide epoch-wise validation feedback. The same six-class folder mapping is retained across both generators. After model fitting, the learned weights and network configuration are serialized for subsequent inference.

After training, the network is saved in both ECG.h5 and ECG.keras formats. The native Keras model is loaded by the Flask application when the web service is launched. Model persistence separates computationally intensive training from online inference: the stored classifier can be reused for each prediction request, while the Flask application performs image preparation, model inference, label decoding, and result presentation.

E) Flask-Based Inference Workflow

The Flask application defines routes for the landing page, information page, image-upload interface, and prediction. The /predict route receives an image file, stores it in the upload directory, loads and resizes the image to 64 × 64 pixels, converts it to a NumPy array, expands it to a batch of one sample, and passes it to model.predict. The class with the highest softmax probability is selected using argmax and mapped to one of the six rhythm labels. If the predicted class is Normal, the interface displays 'Healthy Rhythm Detected'; all other classes are displayed as 'Abnormal Rhythm Detected.' This binary status is therefore derived from the six-class prediction rather than from a separate classifier.

The offline and online pipelines use the same model input size of 64 × 64 pixels and the same six-class label mapping. The training and test generators also rescale pixel intensities by 1/255, whereas the Flask prediction route performs resizing and array conversion before submitting the tensor to the saved model. This difference is relevant to reproducibility because preprocessing applied during online inference should be consistent with the numerical representation used during model training and evaluation.

F) Overall System Architecture

The overall software architecture is illustrated in Fig. 2. The offline path begins with the six-class ECG image dataset, performs resizing, intensity scaling, and training-time augmentation, trains the CNN classifier, and stores the trained network. The online path begins when an ECG image is submitted through the browser. The image is converted to the required model input tensor, the saved classifier generates a six-class softmax prediction, and the selected class is mapped to a rhythm label and displayed through the Flask interface.

International Journal of
AI Electronics and Nexus Energy
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0515 www.zesterapublications.com Original Research Paper

This separation allows the trained model to support repeated web-based predictions without retraining.

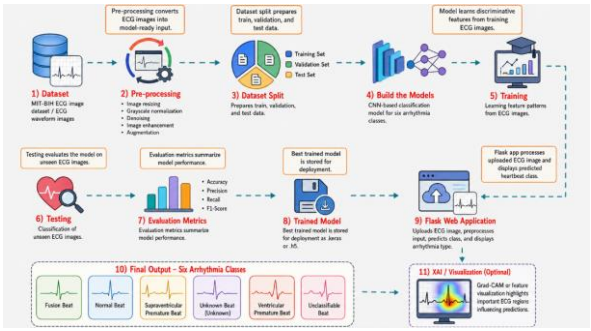


Fig. 2. Overall system architecture for ECG arrhythmia classification and Flask deployment.

G) Software Environment and Deployment Considerations

The development environment uses Python 3.12.5 together with TensorFlow, Keras, NumPy, Pillow, and Flask. The ECG.h5 and ECG.keras model files are stored with the Flask application. The web layer uses Flask request and rendering utilities together with Keras model-loading and image-preprocessing functions. It provides a landing page, information page, image-upload interface, and prediction-result page. The saved classifier can be loaded once and reused for repeated inference requests without retraining.

The deployment workflow consists of four connected operations: image acquisition, tensor preparation, model inference, and result generation. An uploaded file is stored in the designated upload directory and loaded with Keras image utilities at 64 × 64 pixels. It is then converted to an array, expanded to a one-sample batch, and passed to the saved model. The returned probability vector is reduced using argmax, converted to the corresponding rhythm label, and displayed through the result template. The predicted rhythm is also mapped to a Healthy or Abnormal status to provide a simplified user-facing interpretation.

H) Experimental Execution Configuration

The experimental implementation follows the same processing sequence used by the application. Class folders define the target mapping, and ImageDataGenerator prepares 64 × 64 × 3 image tensors in batches of 32. The CNN is trained for 10 epochs using the Adam optimizer and categorical cross-entropy loss. The final network is saved as ECG.h5 and ECG.keras. During inference, each uploaded waveform image is resized to the model input dimensions, converted to a batch tensor, and processed by the six-unit softmax output layer to obtain one of the six rhythm classes. This configuration provides a repeatable sequence from data preparation and feature learning to model persistence and user-facing prediction.

IV. EXPERIMENTAL RESULTS

A) Epoch-Wise Training Performance

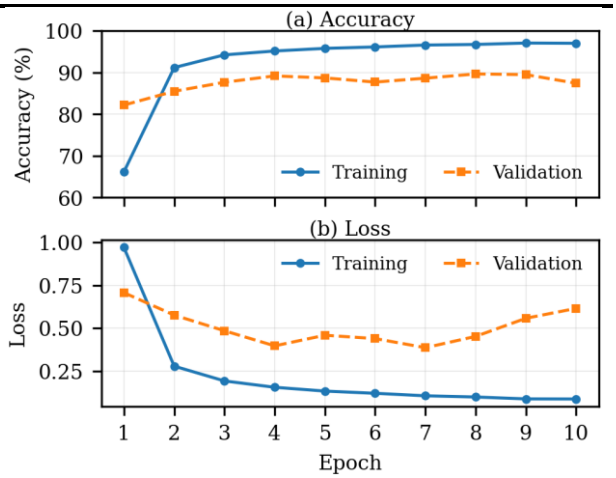


Fig. 3. Epoch-wise CNN learning curves: (a) training and validation accuracy; (b) training and validation loss.

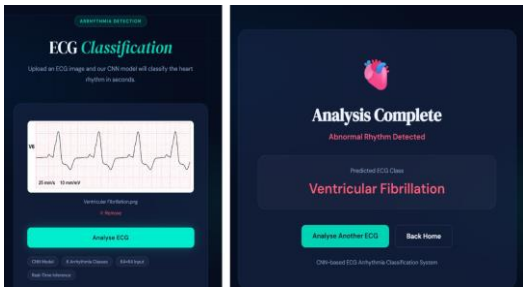
The epoch-wise training records characterize the learning behavior of the CNN across 10 epochs. Training accuracy increases from 66.18% at epoch 1 to 97.03% at epoch 10, while validation accuracy reaches a maximum of 89.67% at epoch 8 and ends at 87.44%. Training loss decreases from 0.9703 to 0.0866. Validation loss reaches its minimum value of 0.3858 at epoch 7 and subsequently increases to 0.6149 by epoch 10. The widening difference between training and validation behavior after approximately epochs 7-8 indicates that later epochs continue to improve the fit to the training data without a corresponding improvement in validation performance. These curves provide the optimization context for the subsequent interface-level classification results.

B) ECG Classification Results

After completion of the 10-epoch training process, the trained CNN is evaluated through the Flask interface using ECG waveform images from the target classes. Each case presents the uploaded ECG image together with the corresponding classification result. The evaluation therefore reflects the complete inference sequence: image submission, CNN prediction, six-class label decoding, and mapping of the predicted rhythm to a Healthy or Abnormal interface status.

Fig. 4. Case 1: Normal ECG input image (left) and corresponding Normal classification result with healthy-rhythm status (right).

In the first case, a Normal ECG image is submitted through the classification interface. The CNN predicts the Normal class, and the application maps this result to the 'Healthy Rhythm Detected' status. This case verifies the class decoding and user-facing status mapping for a correctly identified Normal input.



International Journal of
AI Electronics and Nexus Energy
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0515 www.zesterapublications.com Original Research Paper

Fig. 5. Case 2: Ventricular Fibrillation ECG input image (left) and corresponding classification result (right).

In the second case, the CNN is applied to an ECG image from the Ventricular Fibrillation class. The model predicts Ventricular Fibrillation, and the interface maps the result to 'Abnormal Rhythm Detected.' This case demonstrates the complete abnormal-rhythm processing path from ECG image input to multiclass prediction and interface-level presentation.

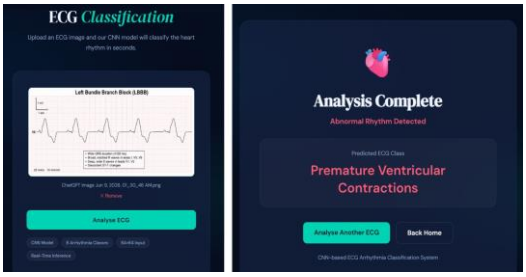


Fig. 6. Case 3: LBBB ECG input image (left) and classification result as Premature Ventricular Contractions (right).

In the third case, an ECG image labeled as LBBB is classified by the CNN as Premature Ventricular Contractions. Because both LBBB and PVC belong to the abnormal-rhythm group used by the interface, the displayed status remains 'Abnormal Rhythm Detected.' However, the class-level prediction does not match the input category. This result illustrates confusion between abnormal waveform classes and highlights the importance of evaluating the detailed multiclass output in addition to the simplified Healthy/Abnormal status.

C) Result Analysis

The classification results demonstrate that the complete ECG-processing pipeline operates from image input to interface output. The Normal input is classified as Normal and mapped to a healthy-rhythm status, while the Ventricular Fibrillation input is classified as Ventricular Fibrillation and mapped to an abnormal-rhythm status. The LBBB input is classified as PVC and therefore also receives an abnormal-rhythm status. These cases confirm successful integration of the six-class CNN with the Flask application while also showing that a correct binary status does not necessarily imply a correct arrhythmia-class prediction.

D) Flask Deployment Results

The Flask application performs ECG classification through an integrated upload and result workflow. The upload module receives an ECG image, prepares it in the required 64 × 64 input format, and passes it to the stored CNN for inference. The resulting class index is decoded to the corresponding rhythm label and then mapped to the Healthy or Abnormal status used by the interface. Normal is presented as a healthy rhythm, whereas Ventricular Fibrillation and PVC are presented as abnormal rhythms in the demonstrated cases. The deployment therefore combines image preprocessing, CNN inference, class decoding, and result presentation within a single web application.

E) Class-Wise Interpretation of the Classification Output

The CNN produces a six-element softmax probability vector corresponding to LBBB, Normal, PAC, PVC, RBBB, and Ventricular Fibrillation. The class index with the highest probability is selected and decoded to the corresponding rhythm label, after which the Flask layer maps the detailed prediction to either a Healthy or Abnormal status. For the Normal test case, the maximum response corresponds to Normal and the interface reports a healthy rhythm. For the Ventricular Fibrillation case, the selected class is Ventricular Fibrillation and the interface correctly reports an abnormal rhythm. For the LBBB case, the model selects PVC; although both classes map to the abnormal-rhythm status, the six-class prediction is incorrect. This distinction demonstrates why the simplified binary status should not replace class-level evaluation.

The three test cases therefore represent two levels of system output. At the application level, the final status mapping distinguishes Normal from non-Normal rhythms. At the CNN level, the classifier attempts to distinguish among six waveform categories. The correct predictions for Normal and Ventricular Fibrillation show successful discrimination for those inputs, whereas the LBBB-to-PVC result demonstrates the more difficult task of separating abnormal classes with potentially similar visual patterns after resizing. Multiclass interpretation is consequently necessary for assessing classifier behavior beyond the simplified Healthy/Abnormal presentation.

F) Analysis of the LBBB-to-PVC Classification Error

The LBBB input classified as PVC represents a class-level error within the abnormal-rhythm group. Because the implemented classifier operates on 64 × 64 image representations, its decision is based on spatial waveform features retained after resizing rather than on explicitly measured clinical intervals. The observed confusion may be influenced by reduced image detail and by the imbalance in the training distribution: the training set contains 504 LBBB images compared with 2,759 PVC images. This difference provides substantially more PVC examples during optimization than LBBB examples. The result therefore identifies a class-discrimination limitation while confirming that the end-to-end inference pipeline itself remains operational.

The classification error also emphasizes the importance of input representation and augmentation design. Shearing, zooming, and horizontal flipping are used to increase visual diversity, but not every transformation necessarily corresponds to physiologically meaningful variation. Future refinements may therefore consider class-aware augmentation, class weighting, balanced sampling, or targeted loss functions to increase learning emphasis on minority classes. Higher-resolution inputs or architectures that preserve greater spatial detail may also help distinguish visually similar abnormal waveform patterns. These refinements concern the classifier and preprocessing strategy and do not require fundamental changes to the Flask deployment mechanism.

G) Preprocessing Consistency and Inference Reliability

Reliable deployment requires consistent numerical representation between model development and online prediction. The training and test generators resize ECG images to 64 × 64 pixels and rescale intensities by 1/255, whereas the Flask route resizes the uploaded image and converts it to a NumPy array before applying the saved model. Because the network is optimized for the numerical range observed during training, inconsistent input scaling can alter intermediate activation magnitudes and consequently affect predictions. Training, testing, and web inference should therefore use the same scaling, channel handling, and intensity-normalization procedure.

Class ordering must also remain fixed throughout the system. Directory-based generators assign class indices according to the class-directory ordering, and the final softmax layer contains one output unit for each class index. When argmax is used to convert the probability vector to a rhythm label, the class list in the Flask application must follow the same index order. Otherwise, a correct network output could be displayed with an incorrect textual label. Consistent class mapping from data preparation through deployment ensures that the selected network index is translated to the intended ECG rhythm and then to the corresponding Healthy or Abnormal interface status.

H) Functional Behavior of the Flask Deployment

The Flask application separates the user interface from offline model training while maintaining a direct prediction pipeline for each request. The upload module receives an ECG image, stores it

International Journal of
AI Electronics and Nexus Energy
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0515 www.zesterapublications.com Original Research Paper

in the designated directory, and passes it to the preprocessing routine. The image is resized to the CNN input dimensions, converted to a NumPy tensor, expanded to a one-sample batch, and submitted to the loaded ECG model. The model returns a six-class probability vector, argmax selects the predicted index, and the application maps that index to the corresponding rhythm label. The detailed predicted class and simplified rhythm status are then displayed in the result template. Because the trained model is loaded outside the training process, repeated predictions can be performed without retraining the network.

The interface also supports repeated analysis by providing a direct path from a completed result to a new ECG upload. The end-to-end sequence therefore includes image selection, preprocessing, CNN inference, class decoding, status mapping, result display, and subsequent image analysis. Separating offline optimization from online inference improves deployment efficiency because the computationally intensive training stage is completed in advance, while the web application performs only the operations required for prediction. The same architecture can be extended with authentication, prediction history, confidence display, batch analysis, or database storage without altering the fundamental CNN classification workflow.

I) Discussion

The experimental results demonstrate an end-to-end ECG image classification application rather than an isolated neural-network model. ECG waveform images are processed through stacked convolution and pooling layers to learn discriminative features, and the resulting six-class predictions are passed directly to the Flask interface. The Normal and Ventricular Fibrillation examples are correctly classified, whereas the LBBB input is classified as PVC. This error is significant because the simplified Healthy/Abnormal status remains correct even though the detailed arrhythmia class is incorrect. Accordingly, evaluation of the deployed system should consider both the six-class output and the simplified interface status rather than relying solely on the binary interpretation.

The current architecture provides a baseline for image-based ECG arrhythmia classification and can be extended to improve technical reliability. Potential refinements include balanced training procedures, consistent normalization across training and inference, higher-resolution inputs, and transfer-learning models that preserve more detailed waveform information. If patient identifiers become available, patient-level data separation would also support a more rigorous evaluation design by reducing the possibility of correlated samples across training and test subsets. In addition, confidence scores and class-specific evaluation measures could be incorporated into the interface to provide greater visibility into model behavior. These extensions provide a clear progression from the current six-class CNN/Flask prototype toward a more robust ECG decision-support workflow.

V. CONCLUSION

This work presents an image-based six-class ECG arrhythmia classification system comprising dataset preparation, convolutional feature learning, multiclass prediction, model persistence, and Flask-based deployment. The dataset contains 22,166 grayscale ECG waveform images from six classes: LBBB, Normal, PAC, PVC, RBBB, and Ventricular Fibrillation. Images are resized to 64 × 64 pixels, normalized, and augmented during training. The classifier consists of three convolution-pooling blocks, a 128-unit dense layer, dropout, and a six-unit softmax output. The network is trained using Adam and categorical cross-entropy and is stored in HDF5 and native Keras formats for web-based inference.

The experimental results demonstrate the complete operation of the deployed classification workflow. The Normal ECG input is classified as Normal and displayed as a healthy rhythm, while the Ventricular Fibrillation input is classified as Ventricular Fibrillation and displayed as an abnormal rhythm. The LBBB input is classified as PVC and is therefore also displayed as an abnormal rhythm, revealing a class-level error despite a consistent binary status. The system integrates ECG image input, CNN-based six-class prediction, label decoding, and web-based result presentation within a single application. Further improvements should focus on class discrimination and consistent preprocessing to strengthen the reliability of multiclass arrhythmia classification.

REFERENCES

[1] A. L. Goldberger et al., “PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals,” Circulation, vol. 101, no. 23, pp. e215-e220, 2000.
[2] G. B. Moody and R. G. Mark, “The impact of the MIT-BIH Arrhythmia Database,” IEEE Eng. Med. Biol. Mag., vol. 20, no. 3, pp. 45-50, 2001.
[3] J. Pan and W. J. Tompkins, “A real-time QRS detection algorithm,” IEEE Trans. Biomed. Eng., vol. BME-32, no. 3, pp. 230-236, 1985.
[4] E. J. da S. Luz, W. R. Schwartz, G. Cámara-Chávez, and D. Menotti, “ECG-based heartbeat classification for arrhythmia detection: A survey,” Comput. Methods Programs Biomed., vol. 127, pp. 144-164, 2016.
[5] S. Kiranyaz, T. Ince, and M. Gabbouj, “Real-time patient-specific ECG classification by 1-D convolutional neural networks,” IEEE Trans. Biomed. Eng., vol. 63, no. 3, pp. 664-675, 2016.
[6] M. Kachuee, S. Fazeli, and M. Sarrafzadeh, “ECG heartbeat classification: A deep transferable representation,” in Proc. IEEE Int. Conf. Healthcare Informatics, 2018, pp. 443-444.
[7] A. Y. Hannun et al., “Cardiologist-level arrhythmia detection and classification in ambulatory electrocardiograms using a deep neural network,” Nat. Med., vol. 25, no. 1, pp. 65-69, 2019.
[8] P. Rajpurkar, A. Hannun, M. Haghpanahi, C. Bourn, and A. Y. Ng, “Cardiologist-level arrhythmia detection with convolutional neural networks,” arXiv:1707.01836, 2017.
[9] T. J. Jun, H. M. Nguyen, D. Kang, D. Kim, D. Kim, and Y.-H. Kim, “ECG arrhythmia classification using a 2-D convolutional neural network,” arXiv:1804.06812, 2018.
[10] Ö. Yildirim, “A novel wavelet sequence based on deep bidirectional LSTM network model for ECG signal classification,” Comput. Biol. Med., vol. 96, pp. 189-202, 2018.
[11] U. R. Acharya et al., “A deep convolutional neural network model to classify heartbeats,” Comput. Biol. Med., vol. 89, pp. 389-396, 2017.
[12] J. Li et al., “Heartbeat classification using deep residual convolutional neural network from 2-lead electrocardiogram,” J. Electrocardiol., vol. 58, pp. 105-112, 2020.
[13] C. Ye, B. V. K. Vijaya Kumar, and M. T. Coimbra, “Heartbeat classification using morphological and dynamic features of ECG signals,” IEEE Trans. Biomed. Eng., vol. 59, no. 10, pp. 2930-2941, 2012.
[14] M. M. Al Rahhal, Y. Bazi, H. AlHichri, N. Alajlan, F. Melgani, and R. R. Yager, “Deep learning approach for active classification of electrocardiogram signals,” Inf. Sci., vol. 345, pp. 340-354, 2016.
[15] S. Mousavi, F. Afghah, and U. R. Acharya, “Inter- and intra-patient ECG heartbeat classification for arrhythmia detection: A sequence-to-sequence deep learning approach,” arXiv:1812.07421, 2018.
[16] L. Zhang et al., “12-lead ECG arrhythmia classification using cascaded convolutional neural network and expert feature,” Comput. Biol. Med., vol. 136, Art. no. 104526, 2021.
[17] J. Yao et al., “Detection and classification of cardiac arrhythmias by a challenge-best deep learning neural network model,” iScience, vol. 23, no. 3, Art. no. 100886, 2020.
[18] M. M. R. Khan et al., “Electrocardiogram heartbeat classification using convolutional neural networks for the detection of cardiac arrhythmia,” arXiv:2010.04086, 2020.
[19] M. Cao, T. Zhao, Y. Li, W. Zhang, P. Benharash, and R. Ramezani, “ECG heartbeat classification using deep transfer learning with convolutional neural network and STFT technique,” arXiv:2206.14200, 2022.

[20] ANSI/AAMI EC57:2012, Testing and Reporting Performance Results of Cardiac Rhythm and ST Segment Measurement Algorithms, Association for the Advancement of Medical Instrumentation, 2012.
[21] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in Proc. 27th Int. Conf. Machine Learning, 2010, pp. 807-814.
[22] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," J. Mach. Learn. Res., vol. 15, pp. 1929-1958, 2014.
[23] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in Proc. Int. Conf. Learning Representations, 2015.
[24] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," J. Big Data, vol. 6, Art. no. 60, 2019.
[25] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. Cambridge, MA, USA: MIT Press, 2016.
[26] 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.
[27] F. Chollet and contributors, "Keras," open-source deep learning API, 2015-present.
[28] M. Grinberg, Flask Web Development, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2018.
[29] C. R. Harris et al., "Array programming with NumPy," Nature, vol. 585, pp. 357-362, 2020.
[30] P. Virtanen et al., "SciPy 1.0: Fundamental algorithms for scientific computing in Python," Nat. Methods, vol. 17, pp. 261-272, 2020.

Received: 07-07-2026 | Accepted: 13-08-2026 | Published: 19-08-2026