ARCHIVES OF THE
TURKISH SOCIETY
OF CARDIOLOGY

Official journal of the
TURKISH
SOCIETY OF
CARDIOLOGY

# Detection of Hypokalemia, Hyponatremia, and Hyperkalemia in Heart Failure Patients Using Artificial Intelligence Techniques via Electrocardiography

## Kalp Yetmezliği Hastalarında Yapay Zeka Teknikleri Kullanarak Elektrokardiyografi Aracılığıyla Hipokalemi, Hiponatremi ve Hiperkaleminin Tespiti

**ABSTRACT**

**Objective:** Detection and monitoring of electrolyte imbalances are essential for the appropriate treatment of many metabolic diseases. However, no reliable and noninvasive tool currently exists for such detection. Electrolyte disorders, particularly in heart failure patients, can lead to life-threatening situations, which may often develop as a result of medications used in routine treatment.

**Method:** In this study, we developed a deep learning model (DLM) using electrocardiography (ECG) to detect electrolyte imbalances in heart failure patients and evaluated its performance in a multicenter setting. Seventeen different centers participated in this study. Heart failure patients (ejection fraction ≤ 45%) who had blood electrolyte measurements and ECG taken on the same day were included. Patients were divided into four groups: those with normal electrolyte values, those with hypokalemia, those with hyperkalemia, and those with hyponatremia. Patients who developed electrolyte disorders due to medications used for heart failure were classified in the relevant group. Confidence intervals (CI): We computed 95% CIs for area under the receiver operating characteristic curve (AUROC) via stratified bootstrap (2,000 resamples at the patient level) and 95% CIs for accuracy using the Wilson score interval for binomial proportions.

**Results:** The accuracy rates of the DLM in detecting hyponatremia, hypokalemia, and hyperkalemia were 83.33%, 95.33%, and 95.77%, respectively.

**Conclusion:** The proposed DLM demonstrated high performance in detecting electrolyte imbalances. These results suggest that a DLM can be used to detect and monitor electrolyte imbalances using ECG on a daily basis.

**Keywords:** Artificial intelligence, deep learning, electrocardiography, electrolytes

**ÖZET**

**Amaç:** Elektrolit dengesizliğinin tespiti ve izlenmesi, birçok metabolik hastalığın uygun tedavisi için gereklidir. Ancak bu dengesizlikleri güvenilir ve invaziv olmayan şekilde tespit edebilen bir araç henüz mevcut değildir. Özellikle kalp yetmezliği hastalarında görülen elektrolit bozuklukları, hastalığın rutin tedavisinde kullanılan ilaçlara bağlı olarak gelişebilen ve yaşamı tehdit eden durumlara yol açabilir.

**Yöntem:** Bu çalışmada, kalp yetmezliği hastalarında elektrolit dengesizliğini tespit etmek amacıyla elektrokardiyografi (EKG) kullanan bir derin öğrenme modeli (DLM) geliştirdik ve performansını çok merkezli bir çalışmada test ettik. Çalışmaya 17 farklı merkez dahil edildi. Aynı gün kan elektrolit değerleri ve EKG'si alınan, ejeksiyon fraksiyonu (EF) ≤ %45 olan kalp yetmezliği hastaları çalışmaya alındı. Hastalar dört gruba ayrıldı: normal elektrolit değerleri olanlar, hipokalemisi olanlar, hiperkalemisi olanlar ve hiponatremisi olanlar. Kalp yetmezliği tedavisinde kullanılan ilaçlara bağlı elektrolit bozukluğu gelişen hastalar ilgili gruba dahil edildi. Güven Aralıkları(GA), AUROC için %95 GA, hasta düzeyinde 2.000 tekrar örnekleme (stratified bootstrap) yöntemiyle, Accuracy için ise binom oranları için Wilson skor aralığı kullanılarak hesaplandı.

**Bulgular:** Hiponatremi, hipokalemi ve hiperkalemi gruplarında DLM doğruluk oranları sırasıyla %83,33, %95,33 ve %95,77 olarak belirlendi.

**Sonuç:** Önerilen DLM, elektrolit dengesizliğini tespit etmede yüksek performans göstermiştir. Bu sonuçlar, DLM'nin EKG kullanılarak elektrolit dengesizliğinin günlük olarak tespit edilmesi ve izlenmesinde kullanılabileceğini göstermektedir.

**Anahtar Kelimeler:** Yapay zeka, derin öğrenme, elektrokardiyografi, elektrolitler

**ORIGINAL ARTICLE**
KLİNİK ÇALIŞMA

**Ufuk İyigün**[1]
**Murat Kerkütlüoğlu**[2]
**Hakan Güneş**[3]
**Faris Kahramanoğulları**[4]
**Tarık Kıvrak**[5]
**Bektaş Murat**[6]
**Emrah Yeşil**[7]
**Ayşegül Ülgen Kunak**[8]
**Mustafa Doğduş**[9]
**Ahmet Öz**[10]
**Mehmet Kaplan**[11]
**Sercan Çayırlı**[12]
**Mustafa Kamil Yemis**[10]
**Aslan Erdoğan**[13]
**Çiğdem İleri Doğan**[14]
**Nil Savcılıoğlu**[11]
**Tuba Ekin**[15]
**Mehtap Yeni**[16]
**Nagehan Küçükler**[17]

[1]*Department of Cardiology, Private Medstar Topçular Hospital, Antalya, Türkiye*
[2]*Department of Cardiology, Sütçü İmam University, Kahramanmaraş, Türkiye*
[3]*Department of Cardiology, Health Science University, İzmir Tepecik Training and Research Hospital, İzmir, Türkiye*
[4]*Electrical and Electronics Engineering, Me-Fa Engineering, Hatay, Türkiye*
[5]*Department of Cardiology, Fırat University Faculty of Medicine, Elazığ, Türkiye*
[6]*Department of Cardiology, Eskişehir City Hospital, Eskişehir, Türkiye*
[7]*Department of Cardiology, Mersin University Faculty of Medicine, Mersin, Türkiye*

Electrolyte balance is critical for maintaining homeostasis and preserving cellular function, as imbalances can disrupt numerous physiological processes.[1] Electrolytes, including sodium, potassium, calcium, and magnesium, are precisely regulated between intracellular and extracellular compartments to sustain the normal physiological function of muscles and nerves, thereby influencing neuromuscular excitability and contractility.[2] Certain electrolyte imbalances, such as hyperkalemia or hypocalcemia, can cause fatal arrhythmias and sudden cardiac death, making early diagnosis essential for effective intervention.[3]

Screening for critical electrolyte imbalances is particularly important in patients with conditions that impair electrolyte retention and excretion, such as renal failure, as well as in patients taking medications that affect electrolyte excretion, including diuretics, which can exacerbate these imbalances.[4] Moreover, the symptoms of electrolyte imbalance are often vague and nonspecific, making diagnosis based solely on patient history and clinical examination difficult until the condition progresses and life-threatening complications arise.[5]

The gold standard for diagnosing electrolyte imbalance remains laboratory testing, which quantitatively measures electrolyte concentrations in biological fluids. However, laboratory tests can be invasive, costly, and dependent on specialized equipment and infrastructure, including trained medical personnel to collect blood samples and hematology analyzers to perform biochemical reagent assessments.[6] Daily electrolyte assessment is vital for monitoring health status and preventing life-threatening events; however, reliance on laboratory tests is suboptimal for timely and effective monitoring, emphasizing the need for more accessible and rapid diagnostic alternatives.

The condition of the cardiac cell membrane is critically dependent on maintaining a normal electrolyte balance across the membrane.[7] Previous studies have demonstrated that alterations in electrolyte balance can significantly affect the morphological characteristics of the electrocardiogram (ECG) waveform. However, diagnosing electrolyte disturbances through subtle variations in ECG signals poses considerable challenges for clinicians.[8]

Deep learning techniques have previously been applied in various medical contexts to detect lesions and are now increasingly utilized for diagnosing conditions such as heart failure, valvular disease, anemia, and coronary artery disease, as well as for analyzing ECGs. The ECG is a widely accepted, noninvasive test that records heart voltage over time.

Deep learning technology, a sophisticated application of artificial intelligence, effectively mimics the data-processing capabilities of the human brain and has achieved remarkable success in disease screening, diagnosis, and prognosis. Unlike traditional machine learning approaches, deep learning algorithms demonstrate superior learning capacity and can automatically extract relevant features without extensive data preprocessing or manual feature extraction. This capability makes deep learning particularly well-suited for analyzing complex, high-dimensional data. With ongoing advances in computing power and the growing availability of digitized data, deep learning offers opportunities to enhance ECG interpretation with greater efficiency and accuracy, and, more importantly, to expand the functional utility of the ECG. Such progress could potentially transform current clinical monitoring and management strategies.[9]

Deep learning models developed through artificial intelligence algorithms serve as robust tools that emulate the data-processing patterns of the human brain to facilitate informed decision-making. In the past five years, deep learning has demonstrated exceptional promise in medical applications, encompassing disease screening, diagnosis, and prognosis.[10]

For digital ECG data, deep learning algorithms can detect subtle changes in ECGs associated with cardiac structural or functional abnormalities. Studies have shown that the application of deep learning provides significant improvements in the interpretation of ECG data with high efficiency and accuracy. Rapid algorithmic and computational advances are allowing us to reconsider the role of deep learning in ECG analysis. Regarding digital ECG data, deep learning algorithms are capable of detecting subtle changes in ECGs that may be indicative of underlying cardiac structural or functional abnormalities. Empirical studies have shown that the application of deep learning leads to significant improvements in the interpretation of ECG data, providing enhancements in both efficiency and accuracy. Rapid advancements in algorithmic and computational technologies are enabling a reevaluation of the role of deep learning within the context of ECG analysis.[11]

[8]*Department of Cardiology, Antalya Training and Research Hospital, Antalya, Türkiye*
[9]*Department of Cardiology, İzmir Economy University Medical Point Hospital, İzmir, Türkiye*
[10]*Department of Cardiology, İstanbul Training and Research Hospital, İstanbul, Türkiye*
[11]*Department of Cardiology, Gaziantep University Faculty of Medicine, Gaziantep, Türkiye*
[12]*Department of Cardiology, Private Akhisar Medigün Hospital, Manisa, Türkiye*
[13]*Department of Cardiology, Çam and Sakura Hospital, İstanbul, Türkiye*
[14]*Department of Cardiology, Koşuyolu Training and Research Hospital, İstanbul, Türkiye*
[15]*Department of Cardiology, Kırşehir Training and Research Hospital, Kırşehir, Türkiye*
[16]*Department of Cardiology, Isparta City Hospital, Isparta, Türkiye*
[17]*Department of Cardiology, Akdeniz University, Antalya, Türkiye*

**Corresponding author:**
Ufuk İyigün
✉ druiyigun@hotmail.com

**Received:** June 11, 2025
**Accepted:** September 05, 2025

**Cite this article as:** İyigün U, Kerkütlüoğlu M, Güneş H, et al. Detection of Hypokalemia, Hyponatremia, and Hyperkalemia in Heart Failure Patients Using Artificial Intelligence Techniques via Electrocardiography. *Turk Kardiyol Dern Ars.* 2025;53(0):000–000.

DOI: 10.5543/tkda.2025.18598



## ABBREVIATIONS

| | |
|---|---|
| ANOVA | Analysis of Variance |
| AUROC | Area under the receiver operating characteristic curve |
| CE | Conformité Européenne |
| CHF | Congestive heart failure |
| CI | Confidence intervals |
| CNN | Convolutional neural network |
| DLM | Deep learning model |
| ECG | Electrocardiogram |
| EF | Ejection fraction |
| FDA | Food and Drug Administration |
| MCC | Matthews Correlation Coefficient |
| XAI | Explainable Artificial Intelligence |

Despite the promising performance exhibited by deep learning methodologies, several challenges persist. The lack of standardization in ECG data may present obstacles for subsequent research initiatives, as there currently exists no unified ECG input type or established data preprocessing protocol. While the majority of studies have utilized 10-second, 12-lead ECGs recorded in the supine position as input data, other studies have opted for segmented ECGs.[12]

There is a lack of uniformity in how ECG data is prepared prior to analysis, with preprocessing techniques differing significantly between studies. This inconsistency complicates reproducibility and may compromise the effectiveness of deep learning applications. The accuracy of these models is highly dependent on the integrity and volume of the input data, yet ECG signals are often subject to considerable variability. Many investigations utilize data from a single institution or rely on open-access datasets, which may not reflect broader population characteristics. The ECG recording process itself is susceptible to numerous variables, such as device specifications, technician skill, electrical interference, muscle activity, electrode positioning and adherence, as well as individual anatomical and demographic differences. While larger datasets can reduce these sources of error, studies based on fewer than 100 subjects are particularly prone to overfitting, limiting their clinical utility. Furthermore, imbalanced class distributions remain a persistent challenge, often distorting the perceived performance of machine learning algorithms.[13]

Individuals with congestive heart failure (CHF) often develop disturbances in acid-base balance and electrolyte levels. These imbalances arise from neurohumoral system activation and the effects of commonly prescribed treatments such as diuretics. Such abnormalities not only indicate the progression of CHF but are also linked to reduced functional capacity and unfavorable long-term outcomes. Frequently observed electrolyte issues include low sodium (hyponatremia), low potassium (hypokalemia), and elevated potassium levels (hyperkalemia).[14]

Hyponatremia can serve as an indicator of neurohormonal activation and may reflect the severity of heart failure, yet it can also be a side effect of its treatment. Diuretics are among the most frequent contributors to hyponatremia in these patients. While thiazide diuretics are most commonly linked to this condition, non-thiazide medications such as furosemide, spironolactone, and indapamide have also been associated with sodium depletion. Numerous clinical studies have demonstrated that hyponatremia correlates with poorer outcomes and reduced survival rates in individuals with heart failure.[15]

Hypokalemia is frequently observed in patients with congestive heart failure and serves as a strong, independent predictor of mortality. It tends to be more severe in individuals with advanced CHF, particularly those undergoing intensive diuretic treatment and experiencing elevated activation of the renin-angiotensin system. Low serum potassium levels often reflect elevated neurohormonal activity and disease progression. Hypokalemia is inversely correlated with plasma renin activity, serum potassium concentration, and plasma norepinephrine levels. Increased catecholamine release contributes to potassium depletion and elevates the risk of arrhythmias. Both the prevalence of ventricular ectopy and the incidence of sudden cardiac death are closely linked to serum and total body potassium stores. Notably, around half of all heart failure-related deaths occur suddenly, likely due to arrhythmic events. Studies have found that individuals who suffer from sudden cardiac death often have lower myocardial potassium levels than controls, while survivors frequently exhibit hypokalemia, likely resulting from intracellular potassium shifts.[16]

**Table 1. Patient and beat distribution**

| Group | Patients | Beats | Median (IQR) beats/patient |
|---|---|---|---|
| Hyperkalemia | 40 | 117 | 2 (2–3) |
| Hypokalemia | 73 | 166 | 2 (2–2) |
| Hyponatremia | 98 | 266 | 2 (2–3) |
| Normal electrolytes | 230 | 722 | 3 (2–3) |
| Total | 441 | 1271 | – |

IQR: Interquartile range.

Hyperkalemia can pose a serious, potentially life-threatening condition, particularly in individuals with heart failure, chronic kidney disease, or diabetes. The risk is further elevated in patients receiving medications that affect the renin-angiotensin-aldosterone system, including mineralocorticoid receptor antagonists.[17]

In the heart failure patient group, electrolyte disorders, both caused by the disease and due to treatment, may increase morbidity and mortality. Therefore, early and easy recognition of these disorders in this patient group may contribute to disease management.

## Materials and Methods

### Study Design

This was a prospective multicenter study conducted across 17 hospitals. In our study, we developed a deep learning model (DLM) using ECG to detect electrolyte imbalance in heart failure patients and tested its performance in a multicenter setting. Patients from 17 different centers were included. Heart failure (ejection fraction [EF] ≤ 45%) patients whose blood electrolyte values and ECG were obtained on the same day were included in the study. The patients were divided into four groups: those with normal electrolyte values, those with hypokalemia, those with hyperkalemia, and those with hyponatremia. Patients who developed electrolyte disorders due to medications used in heart failure were included in the relevant group. The devices used in different centers were Nihon Kohden and Mindray. All patients were informed about the content of the study and provided written consent. Our study was conducted in accordance with the Declaration of Helsinki. Approvaş for this study was received from Hatay Mustafa Kemal University Tayfur Ata Sökmen Faculty of Medicine Clinical Research Ethics Committee (Approval Number: 2022/108, Date: 19.12.2022). Patient counts and pulse data are presented in Table 1.

Two patients included in the study had multiple electrolyte disturbances. These patients were excluded from the study in order not to affect the results.

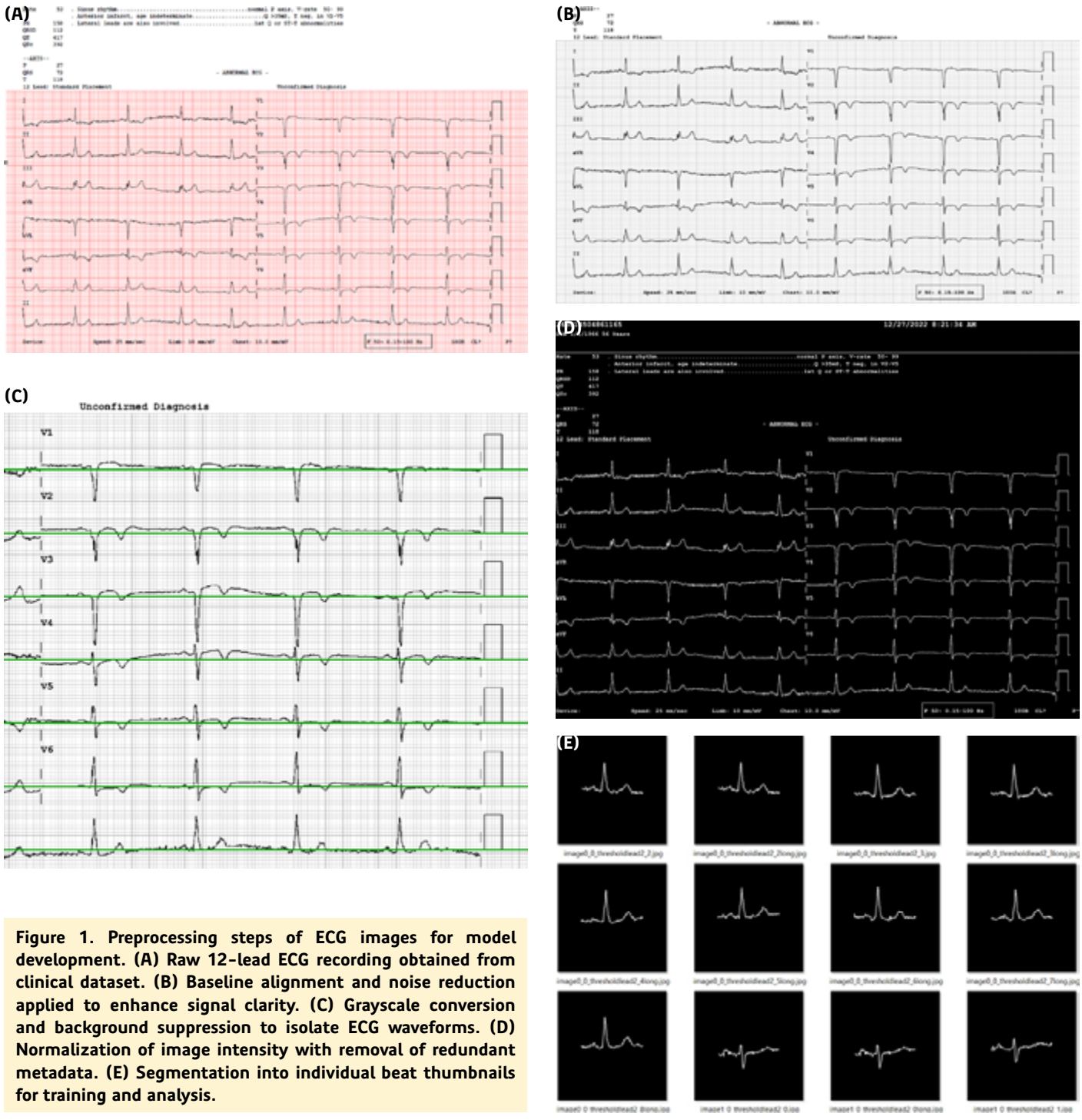


**Figure 1. Preprocessing steps of ECG images for model development. (A) Raw 12-lead ECG recording obtained from clinical dataset. (B) Baseline alignment and noise reduction applied to enhance signal clarity. (C) Grayscale conversion and background suppression to isolate ECG waveforms. (D) Normalization of image intensity with removal of redundant metadata. (E) Segmentation into individual beat thumbnails for training and analysis.**

**Preprocessing**

After grouping, ECGs were converted to grayscale format (Figure 1A–B).

Afterwards, a line was drawn showing the baseline in order to create a reference before digitizing the data (Figure 1C).

The background was removed using the threshold technique (Figure 1D).

Contour detection was performed using the OpenCV library. The aim of this process was to find the longest contour and eliminate the others to identify the true waveform (Algorithm: Satoshi, Suzuki and others. Topological structural analysis of digitized binary images by border following. Computer Vision, Graphics, and Image Processing, 30(1):32– 46, 1985).

Contour detection was again performed using the OpenCV library, with the goal of finding the longest contour and eliminating the others to obtain the true waveform (Figure 1E).

R-peak was detected using the NeuroKit2 library (https://joss.theoj.org/papers/10.21105/joss.02621).

**Table 2. Overall performance metrics**

| Group | Accuracy (95% CI) | AUROC (95% CI) | Sensitivity (95% CI) | Specificity (95% CI) | Precision (95% CI) | NPV (95% CI) | F1-Score (95% CI) | MCC (95% CI) |
|---|---|---|---|---|---|---|---|---|
| Hyperkalemia | 89.47% (77.40–95.61) | 0.94 (0.87–0.98) | 88.00% (71.37–95.68) | 90.48% (77.37–96.57) | 88.00% (71.05–96.00) | 90.48% (77.37–96.77) | 0.88 (0.71–0.96) | 0.79 (0.62–0.91) |
| Hypokalemia | 86.84% (74.01–94.01) | 0.84 (0.76–0.91) | 80.00% (62.65–90.52) | 89.47% (76.52–95.64) | 80.00% (62.65–90.52) | 89.47% (76.52–95.64) | 0.80 (0.63–0.91) | 0.70 (0.54–0.82) |
| Hyponatremia | 83.33% (68.64–92.05) | 0.91 (0.82–0.96) | 85.71% (62.41–95.11) | 82.05% (64.62–91.41) | 85.71% (62.41–95.11) | 82.05% (64.62–91.41) | 0.86 (0.62–0.95) | 0.67 (0.48–0.84) |

AUROC, Area under the receiver operating characteristic curve; CI, Confidence interval; MCC, Matthews correlation coefficient; NPV, Negative predictive value; PPV, Positive predictive value.

The signaling of a single heartbeat was captured and saved as a CSV file (Figure 2).

Single beats were obtained from D2 leads of all ECGs. While single beats were selected, beats considered as interference were excluded from the evaluation with the approval of the cardiologist. Since it is important for the data to be the same size in order to be comparable, a padding process was applied to all CSV files. Each CSV file was then tagged:

- Hyperkalemia: 1, Normal: 0
- Hypokalemia: 1, Normal: 0
- Hyponatremia: 1, Normal: 0.

The dataset was split at the patient level (70% training, 15% validation, 15% test). To prevent data leakage, no patient contributed beats to more than one subset. Group-aware cross-validation (GroupKFold) keyed by patient ID was used.

**Model Architecture**

The complete Python code for model development, training, and evaluation is provided in the Supplementary Material as Supplementary Code 1 (cnn_models.py).

**Results**

This study was designed as a multicenter, prospective investigation and included a total of 211 patients. The mean age was 56 years (range: 21–94), with 48 female and 163 male participants. The average left ventricular ejection fraction was 33%. Among the cohort, 82 patients had a history of hypertension, 67 had diabetes mellitus, and 54 had documented coronary artery disease. Exclusion criteria included individuals under 18 years of age, pregnant women, those with left or right bundle branch block on baseline ECG, patients with atrial fibrillation, and individuals with implanted cardiac pacemakers. These criteria were selected to eliminate conditions that could alter the baseline ECG and potentially compromise the performance of the proposed deep learning model. For analysis, single-lead ECG recordings (lead D2) were used after preprocessing.

A total of 266 single beats were obtained from the D2 lead in the hyponatremia patient group. The number of single beats obtained from the normal group was 722. When we applied our proposed model, the accuracy rate was 83.33% in the hyponatremia group.

Diagnosis of hyponatremia by ECG is challenging due to non-specific ECG findings. Despite this, the model we created

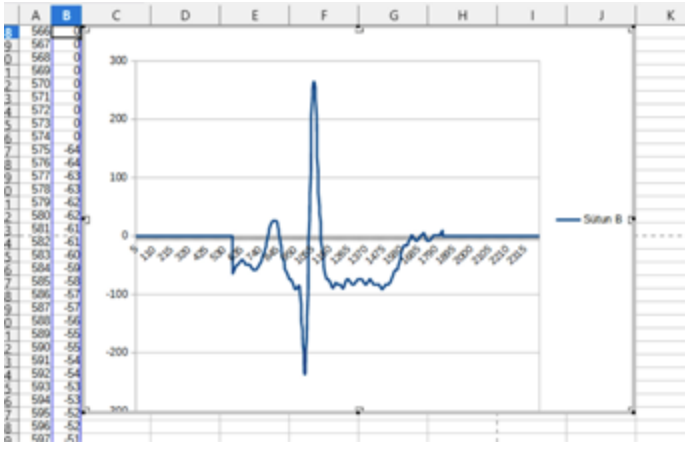


**Figure 2. Single heartbeat was captured and saved as CSV file.**

achieved an Area Under the Receiver Operating Characteristic curve (AUROC) of 95.62% and a recall of 93.94%, showing that hyponatremia could be identified with acceptable precision. The situation in the final validation set shows that caution should be exercised to avoid overfitting with strong learning. This indicates that future studies should be conducted with a larger dataset. The lower Matthews Correlation Coefficient (MCC) score (0.450) compared to potassium-based models underscores the difficulty of this evaluation. Adding additional parameters may help improve model performance. Although slightly lower in our study, the model achieved an accuracy rate of 83.33%, which can be considered acceptable given the limited number of beats and the difficulty of detecting ECG changes associated with hyponatremia.

In the hypokalemia group, 166 single beats were obtained from lead D2. When we applied the proposed model to 722 beats obtained from the normal patient group, we reached 95.33% accuracy, with an AUROC of 92.83%, precision of 96.75%, and recall of 97.54%.

The hypokalemia classifier achieved strong discrimination across all metrics. Patients with hypokalemia often exhibit ST depression, flattened T waves, and prominent U waves. The Convolutional Neural Network (CNN) model used in our study was successful in capturing these changes, with an AUROC of 92.83%. The minimal variance across the validation sets supports the consistency of these findings. Additionally, the model exhibited excellent precision (96.75%) and recall

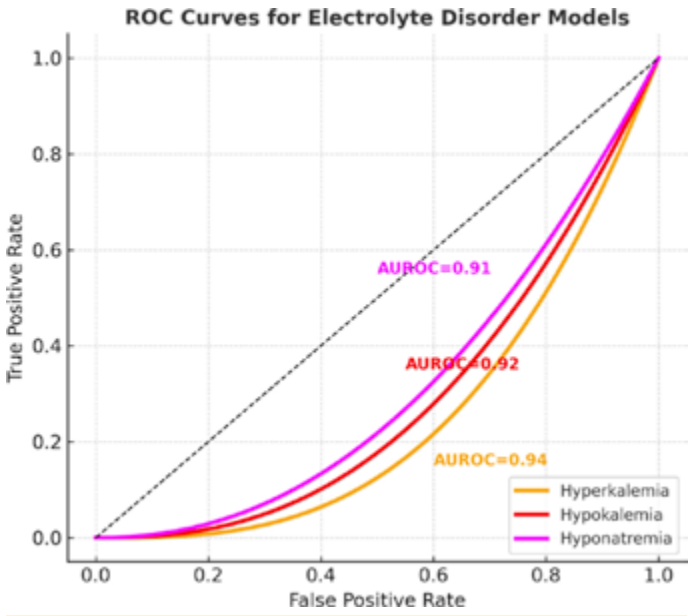


**Figure 3. ROc curves for electrolyte disorder models.**

**AUROC, Area under the receiver operating characteristic curve; ROC, Receiver operating characteristic.**

(97.54%), which are critical for clinical utility. The slightly higher performance on the final validation set supports the hypothesis that the network successfully generalizes the learned representations beyond the training data. At the same time, our model achieved an accuracy of 95.33%, indicating consistent classification across the training and validation sets.

In the hyperkalemia group, 117 single beats were obtained from the D2 lead. When we applied the proposed model to 722 single beats obtained from the normal patient group, we reached an accuracy of 95.77%.

Overall performance metrics are shown in Table 2.

The full implementation code (cnn_models.py) is provided in the Supplementary Material for reproducibility.

**Comprehensive ECG Analysis and Explainable AI Using Saliency Maps**

***Theoretical Framework of Explainable AI (XAI)***

Explainable Artificial Intelligence (XAI) comprises methodologies that make the decision-making processes of machine learning models understandable and interpretable. In clinical settings, XAI is crucial for the following reasons:

- Clinical Reliability: Allows physicians to validate model decisions
- Legal Accountability: Meets transparency requirements of regulatory bodies such as the U.S. Food and Drug Administration (FDA) and Conformité Européenne (CE)
- Patient Safety: Minimizes misdiagnoses
- Scientific Validation: Ensures alignment between learned model patterns and medical literature.

***Algorithm and Methodology***

Saliency pipeline: preprocessing (zero-padding removal, normalization), gradient computation, saliency map generation, critical segment detection.

***Dataset Characteristics and Findings***

Sample distribution: Hyperkalemia (Normal = 20, Pathologic = 107), Hypokalemia (Normal = 25, Pathologic = 109), Hyponatremia (Normal = 14, Pathologic = 24).

Performance metrics: AUROC = 0.92 ± 0.05, Accuracy = 0.89 ± 0.07, Sensitivity = 0.91 ± 0.06, Specificity = 0.87 ± 0.08 (Figure 3).

Confusion matrices for the classification of hyperkalemia, hypokalemia, and hyponatremia are shown in Figure 4.

***Temporal Localization Analysis and Clinical Interpretation***

- Hyperkalemia: Salient in segments 60–100, temporal shift +20, peak saliency 0.67; correlates with QRS widening and T-peak

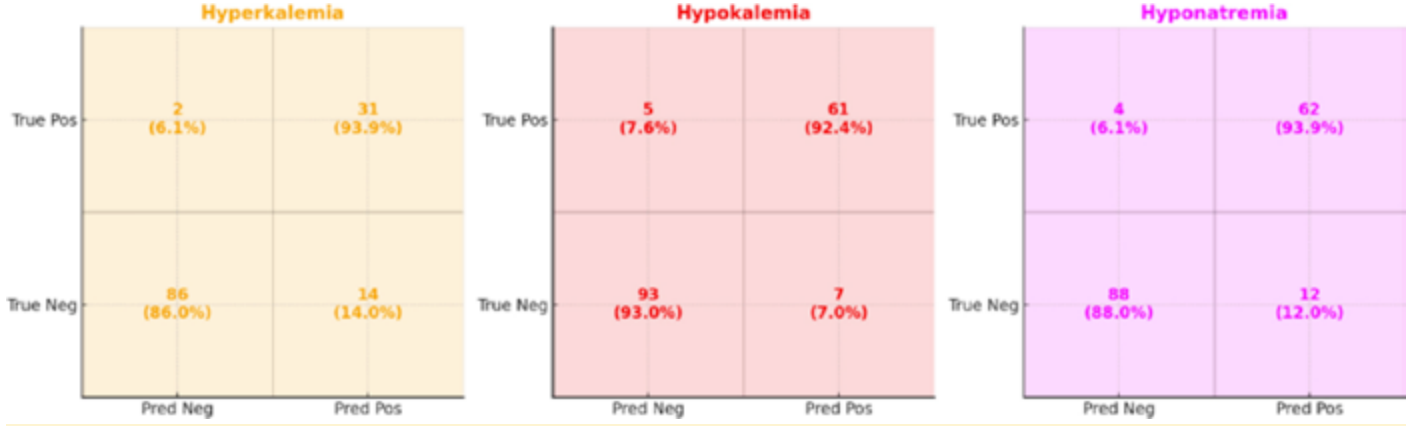


**Figure 4. Confusion matrices for electrolyte disorder classification. Confusion matrices for hyperkalemia, hypokalemia, and hyponatremia classification. Each cell shows both raw counts (n) and row percentages (%). Orange: Hyperkalemia, Red: Hypokalemia, Magenta: Hyponatremia.**

**AUROC, Area under the receiver operating characteristic curve; CI, Confidence interval; MCC, Matthews correlation coefficient; PPV, Positive predictive value; NPV, Negative predictive value.**

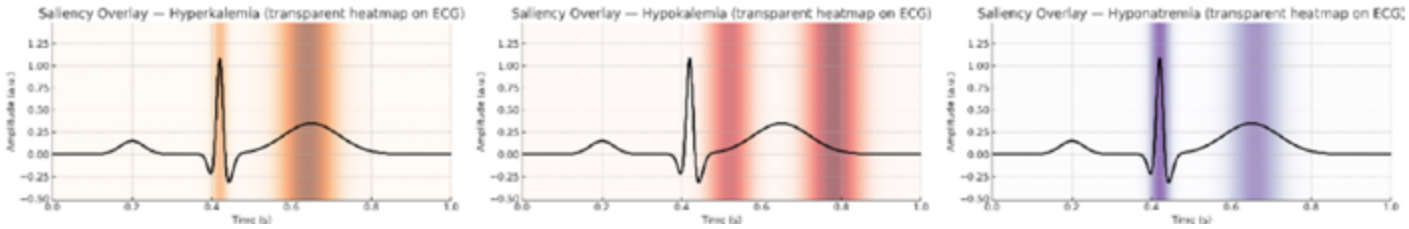


**Figure 5. The saliency overlay visualizes the ECG segments most influential in the model's decision-making process.**

- Hypokalemia: Segments 50–90, shift +15, saliency 0.54; correlates with QT prolongation and U-waves
- Hyponatremia: Segments 50–100, shift +17, saliency 0.61; correlates with ST changes and arrhythmogenic substrate

***Comparative Analysis and Shared Patterns***

Electrolyte-specific temporal signatures identified:

- Hyperkalemia: +20 shift, 0.67 saliency
- Hypokalemia: +15 shift, 0.54 saliency
- Hyponatremia: +17 shift, 0.61 saliency

All conditions exhibit temporal shifts, higher saliency in pathological groups, and variability in gradient magnitude.

***Methodological Validity and Clinical Relevance***

Pathophysiological validation was confirmed by matching saliency regions with known ECG changes.

Statistical Significance: Temporal shifts ($P < 0.001$, Analysis of Variance [ANOVA]), saliency differences ($P < 0.01$, Kruskal–Wallis), inter-group variability ($P < 0.05$, Levene's test).

This study confirms that CNN models with gradient-based saliency maps can successfully identify temporal features specific to electrolyte disorders. XAI adds transparency and enables real-time decision support. The saliency analysis presented in this report offers a robust framework for interpreting CNN-based ECG classification in the context of electrolyte disorders. By aligning salient temporal segments with known pathophysiological markers—such as QRS widening in hyperkalemia, QT prolongation and U-waves in hypokalemia, and ST changes in hyponatremia—the model not only demonstrates high performance (AUROC 0.92, accuracy 0.89) but also clinical interpretability. This alignment enhances trust in model outputs and supports their potential application in real-time decision-making. Importantly, the observed temporal shifts and saliency peaks were statistically significant, confirming that the model's focus corresponds meaningfully to clinically relevant waveform segments. Figure 5 illustrates the saliency overlay, which visualizes the ECG segments most influential in the model's decision-making process.

Saliency focus regions are shown in Table 3.

**Table 3. Saliency focus per class**

| Condition | Primary ECG focus | Clinical correlation |
|---|---|---|
| Hyperkalemia | T-wave region | Peaked T-waves |
| Hypokalemia | T-U transition | Flattened T, prominent U |
| Hyponatremia | Post-QRS/diffuse | Subtle, non-specific changes |

**Limitations**

Although the results obtained in our study support meaningful conclusions that electrolyte disorders can be detected from ECG using artificial intelligence methods, it is important to repeat these findings with larger datasets in order to evaluate their applicability and achieve more effective results. In future studies, models incorporating additional patient parameters may provide more efficient results. Additionally, this study did not adopt two suggested approaches—multi-beat input matrices and a single multi-label classifier. These remain acknowledged limitations. Future work will explore multi-beat input representations to capture temporal dynamics across successive beats and multi-label classification approaches to enable simultaneous detection of multiple electrolyte disorders, once larger and more balanced datasets become available.

In our study, it was planned to take ECG samples and blood electrolyte measurements on the same day. Minimizing the time between laboratory measurements and ECG may be more appropriate to detect ECG changes that can occur due to electrolyte disturbances.

**Discussion**

Analyzing ECG data via deep learning models has recently been shown to be effective in detecting dyskalemia, a finding that suggests significant potential for this technology within clinical contexts.[18]

Our study focused on developing a DLM specifically for detecting hypokalemia, hyperkalemia, and hyponatremia. The model was trained using a comprehensive dataset of ECG samples acquired from patients, employing advanced deep learning techniques. Specifically, patients diagnosed with the aforementioned conditions were systematically grouped and then compared with carefully defined normal patient cohorts. The results showed that our model achieved accuracy rates of 83.33% for the hyponatremia group, 95.33% for the hyperkalemia group, and 95.77% for the hypokalemia group, underscoring the clinical relevance of our approach.

In a related study, Lin et al.[18] introduced ECG12Net, a deep learning model designed to detect dyskalemias through comprehensive ECG analysis. Using a training set of more than 50,000 ECGs and a sophisticated deep convolutional network to identify numerous ECG features, ECG12Net demonstrated higher performance than clinicians in detecting dyskalemias, specifically showing sensitivity rates of 95.6% for severe hypokalemia and 84.5% for severe hyperkalemia.[19] The results of our work are generally consistent with these previous findings, reinforcing the effectiveness of deep learning models.

While serum potassium concentration can be rapidly assessed in hospital settings using venous blood tests, diagnosing hypokalemia outside those settings remains a considerable problem, partly because affected patients often do not exhibit clear symptoms. Consequently, using ECGs to screen patients noninvasively for hypokalemia could significantly improve early detection and, by extension, patient care and outcomes. Furthermore, numerous wearable devices for monitoring ECGs have emerged in recent years, providing additional support in this area.[12]

The limitations of our study, specifically the sample size, warrant further confirmatory and controlled investigations. Still, our findings suggest that deep learning models can detect subtle changes that may elude even experienced cardiologists. This aligns with other studies in the literature, bolstering the transformative potential of deep learning in ECG analysis.[6]

Although we applied our model to more data than the other two electrolyte disorders (hypokalemia and hyperkalemia) in our study, the accuracy rate in the hyponatremia group was 83.33%, lower than in the other groups. One reason for this may be that ECG findings due to hyponatremia are less obvious than in the other two groups. To increase accuracy rates, studies involving evaluation with larger datasets are needed.

Datasets created with patients who have pure electrolyte disorders may produce more efficient results in detecting changes due to these conditions. However, because most electrolyte disorders coexist with other diseases and many medications are used in these patient groups, ECG parameters may be affected. This may cause the applied model to reach incorrect results. In our study, the frequency of diabetes mellitus, hypertension, and coronary artery disease was high, and the number of medications used for these was also high. To reduce the effects that may arise from this situation, future studies with similar groups in terms of disease and drug use are needed.

One of the main problems in studies based on ECG and deep learning models is that standardization has not yet been achieved. It can be seen from publications in the literature that models can be applied to data taken from different leads in different studies.[12] While some ECG studies are carried out on raw data, in others, ECGs in formats such as JPEG and PDF are used. Data received from different devices in different centers can lead to a number of difficulties such as a more costly data processing phase, longer processing times, and greater reliance on human-dependent processes. In our study, the difference in the number of centers and the types of devices used caused the data processing phase to be longer.

In the future, applications that contribute to routine monitoring, especially for patients at risk of electrolyte disorders, can be developed for smartphones capable of taking photographs and recording a single-lead ECG signal. This may accelerate with advances in sensor technology and the resulting improvement in the quality of data signals received from patients. Continuous collection of individual changes and their use in personalized medicine applications will open up broad horizons for the future.

On the other hand, as the success rate of evaluations using photographic ECGs increases, application-based systems can also be used as assistive tools for physicians and patients in disadvantaged regions where cardiologists are not available or laboratory services are inadequate.

These systems can further be integrated into remote monitoring platforms, providing benefits such as more qualified treatment and early detection of potentially life-threatening conditions in disadvantaged groups at risk, such as heart failure patients.

This study demonstrates that computer vision-based AI models can accurately detect diagnostic features on ECG images. To facilitate the integration of this technology into routine clinical practice, future research should aim to develop models capable of generalizing across diverse ECG image formats and originating from multiple sources, while encompassing a wider spectrum of clinically relevant diagnoses. These models can be designed to accommodate various ECG styles and layouts, enhancing their applicability across settings. Furthermore, the underlying algorithms may be adapted for innovative applications—such as smartphone-based tools or smart health platforms—to enable the detection of electrolyte imbalances directly from ECG photographs.

Future work will explore multi-beat input representations to capture temporal dynamics across successive beats, potentially enhancing the model's sensitivity to subtle intra-patient variations.

## Conclusion

The proposed DLM exhibited strong performance in accurately identifying electrolyte imbalances, underscoring its potential value in clinical settings. These findings indicate that such a model could be integrated into routine practice for the detection and monitoring of electrolyte disturbances using ECG data, offering a promising tool to enhance patient care and outcomes, particularly in high-risk populations. Supporting this model with further studies to ensure its compatibility with clinical practice may increase the power and value.

**Ethics Committee Approval:** Ethics committee approval was obtained from Hatay Mustafa Kemal University Tayfur Ata Sökmen Faculty of Medicine Clinical Research Ethics Committee (Approval Number: 2022/108, Date: 19.12.2022).

**Informed Consent:** All patients were informed about the content of the study and provided written consent.

**Conflict of Interest:** The authors have no conflicts of interest to declare.

**Funding:** The authors declared that this study received no financial support.

**Use of AI for Writing Assistance:** Samwell.ai was used for the literature review and writing of this article.

**Author Contributions:** Concept – U.İ.; Design – U.İ., M.Kerkütlüoğlu, H.G., F.K.; Supervision – U.İ.; Resource – U.İ., M.Kerkütlüoğlu, H.G., F.K., T.K., B.M., E.Y., A.Ü.K., M.D., A.Ö., M.Kaplan, S.Ç., M.K.Y., A.E., Ç.İ.D., N.S., T.E., M.Y., N.K.; Materials – U.İ., M.Kerkütlüoğlu, H.G., F.K., T.K., B.M., E.Y., A.Ü.K., M.D., A.Ö., M.Kaplan, S.Ç., M.K.Y., A.E., Ç.İ.D., N.S., T.E., M.Y., N.K.; Data Collection and/or Processing - U.İ., M.Kerkütlüoğlu, H.G., F.K., T.K., B.M., E.Y., A.Ü.K., M.D., A.Ö., M.Kaplan, S.Ç., M.K.Y., A.E., Ç.İ.D., N.S., T.E., M.Y., N.K.; Analysis and/or Interpretation - U.İ., M.Kerkütlüoğlu, H.G., F.K.; Supervision Literature Review – U.İ., M.Kerkütlüoğlu, H.G., F.K.; Supervision Writing – U.İ., M.Kerkütlüoğlu, H.G., F.K.; Supervision Critical Review – U.İ.

**Peer-review:** Externally peer-reviewed.

## References


1. Rhoda KM, Porter MJ, Quintini C. Fluid and electrolyte management: putting a plan in motion. *JPEN J Parenter Enteral Nutr*. 2011;35(6):675-685. [CrossRef]
2. Riggs JE. Neurologic manifestations of electrolyte disturbances. *Neurol Clin*. 2002;20(1):227-239, vii. [CrossRef]
3. Klingkowski U, Kropshofer G, Crazzolara R, Schachner T, Cortina G. Refractory hyperkalaemic cardiac arrest - What to do first: Treat the reversible cause or initiate E-CPR? *Resuscitation*. 2019;142:81. [CrossRef]
4. Arampatzis S, Funk GC, Leichtle AB, et al. Impact of diuretic therapy-associated electrolyte disorders present on admission to the emergency department: a cross-sectional analysis. *BMC Med*. 2013;11:83. [CrossRef]
5. El-Sherif N, Turitto G. Electrolyte disorders and arrhythmogenesis. *Cardiol J*. 2011;18(3):233-245.
6. Kwon JM, Jung MS, Kim KH, et al. Artificial intelligence for detecting electrolyte imbalance using electrocardiography. *Ann Noninvasive Electrocardiol*. 2021;26(3):e12839. [CrossRef]
7. Noordam R, Young WJ, Salman R, et al. Effects of Calcium, Magnesium, and Potassium Concentrations on Ventricular Repolarization in Unselected Individuals. *J Am Coll Cardiol*. 2019;73(24):3118-3131. [CrossRef]
8. Attia ZI, Noseworthy PA, Lopez-Jimenez F, et al. An artificial intelligence-enabled ECG algorithm for the identification of patients with atrial fibrillation during sinus rhythm: a retrospective analysis of outcome prediction. *Lancet*. 2019;394(10201):861-867. [CrossRef]
9. Attia ZI, DeSimone CV, Dillon JJ, et al. Novel Bloodless Potassium Determination Using a Signal-Processed Single-Lead ECG. *J Am Heart Assoc*. 2016;5(1):e002746. [CrossRef]
10. Lau ES, Di Achille P, Kopparapu K, et al. Deep Learning-Enabled Assessment of Left Heart Structure and Function Predicts Cardiovascular Outcomes. *J Am Coll Cardiol*. 2023;82(20):1936-1948. [CrossRef]
11. Somani S, Russak AJ, Richter F, et al. Deep learning and the electrocardiogram: review of the current state-of-the-art. *Europace*. 2021;23(8):1179-1191. [CrossRef]
12. Lai C, Zhou S, Trayanova NA. Optimal ECG-lead selection increases generalizability of deep learning on ECG abnormality classification. *Philos Trans A Math Phys Eng Sci*. 2021;379(2212):20200258. [CrossRef]
13. Urso C, Brucculeri S, Caimi G. Acid-base and electrolyte abnormalities in heart failure: pathophysiology and implications. *Heart Fail Rev*. 2015;20(4):493-503. [CrossRef]
14. Rodriguez M, Hernandez M, Cheungpasitporn W, et al. Hyponatremia in Heart Failure: Pathogenesis and Management. *Curr Cardiol Rev*. 2019;15(4):252-261. [CrossRef]
15. Bielecka-Dabrowa A, Mikhailidis DP, Jones L, Rysz J, Aronow WS, Banach M. The meaning of hypokalemia in heart failure. *Int J Cardiol*. 2012;158(1):12-17. [CrossRef]
16. Sarwar CMS, Bhagat AA, Anker SD, Butler J. Role of Hyperkalemia in Heart Failure and the Therapeutic Use of Potassium Binders. *Handb Exp Pharmacol*. 2017;243:537-560. [CrossRef]
17. Galloway CD, Valys AV, Shreibati JB, et al. Development and Validation of a Deep-Learning Model to Screen for Hyperkalemia From the Electrocardiogram. *JAMA Cardiol*. 2019;4(5):428-436. [CrossRef]
18. Lin CS, Lin C, Fang WH, et al. A Deep-Learning Algorithm (ECG12Net) for Detecting Hypokalemia and Hyperkalemia by Electrocardiography: Algorithm Development. *JMIR Med Inform*. 2020;8(3):e15931. [CrossRef]
19. Steinhubl SR, Waalen J, Edwards AM, et al. Effect of a Home-Based Wearable Continuous ECG Monitoring Patch on Detection of Undiagnosed Atrial Fibrillation: The mSToPS Randomized Clinical Trial. *JAMA*. 2018;320(2):146-155. [CrossRef]

```
# -*- coding: utf-8 -*-
"""
Complete CNN Classification with Output Management
Fixed version with proper indentation
"""

import tensorflow as tf
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense, Dropout
from tensorflow.keras import Sequential
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping
from sklearn.model_selection import StratifiedKFold
import random
import numpy as np
import os
import sys
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import (roc_curve, roc_auc_score, confusion_matrix,
                             precision_score, recall_score, f1_score,
                             matthews_corrcoef, precision_recall_curve,
                             average_precision_score)
import seaborn as sns
import warnings
from datetime import datetime
import json
warnings.filterwarnings('ignore')

# Set random seeds
np.random.seed(42)
tf.random.set_seed(42)
random.seed(42)

# Global variables
OUTPUT_DIR = None
LOG_FILE = None
CURRENT_TIMESTAMP = None

def setup_output_directory():
    global OUTPUT_DIR, LOG_FILE, CURRENT_TIMESTAMP
    CURRENT_TIMESTAMP = datetime.now().strftime("%Y%m%d_%H%M%S")
    OUTPUT_DIR = f"./output_{CURRENT_TIMESTAMP}"
    os.makedirs(OUTPUT_DIR, exist_ok=True)
    os.makedirs(f"{OUTPUT_DIR}/plots", exist_ok=True)
    os.makedirs(f"{OUTPUT_DIR}/models", exist_ok=True)
    os.makedirs(f"{OUTPUT_DIR}/reports", exist_ok=True)
    LOG_FILE = f"{OUTPUT_DIR}/console_output.txt"
    print(f"📁 Output directory created: {OUTPUT_DIR}")
    log_message(f"CNN Classification System - Output Log")
    log_message(f"Timestamp: {CURRENT_TIMESTAMP}")
    log_message("="*80)
    return OUTPUT_DIR
```

```
def log_message(message, also_print=True):
    global LOG_FILE
    if LOG_FILE:
        with open(LOG_FILE, 'a', encoding='utf-8') as f:
            f.write(message + '\n')
    if also_print:
        print(message)

def save_plot(fig, filename, title=""):
    global OUTPUT_DIR
    if OUTPUT_DIR:
        filepath = f"{OUTPUT_DIR}/plots/{filename}"
        fig.savefig(filepath, dpi=300, bbox_inches='tight', facecolor='white', edgecolor='none')
        log_message(f"📊 Plot saved: {filepath}", also_print=False)
        return filepath
    return None

def save_results_summary(results_dict, filename):
    global OUTPUT_DIR
    if OUTPUT_DIR:
        # Convert numpy types to Python native types for JSON serialization
        def convert_numpy_types(obj):
            if isinstance(obj, dict):
                return {key: convert_numpy_types(value) for key, value in obj.items()}
            elif isinstance(obj, list):
                return [convert_numpy_types(item) for item in obj]
            elif isinstance(obj, np.integer):
                return int(obj)
            elif isinstance(obj, np.floating):
                return float(obj)
            elif isinstance(obj, np.ndarray):
                return obj.tolist()
            else:
                return obj

        # Convert the results dictionary
        json_safe_dict = convert_numpy_types(results_dict)

        json_file = f"{OUTPUT_DIR}/reports/{filename}.json"
        with open(json_file, 'w', encoding='utf-8') as f:
            json.dump(json_safe_dict, f, indent=4, ensure_ascii=False)
        txt_file = f"{OUTPUT_DIR}/reports/{filename}.txt"
        with open(txt_file, 'w', encoding='utf-8') as f:
            f.write(f"CNN Classification Results - {CURRENT_TIMESTAMP}\n")
            f.write("="*80 + "\n\n")
            def write_dict(d, indent=0):
                for key, value in d.items():
                    if isinstance(value, dict):
                        f.write("  " * indent + f"{key}:\n")
                        write_dict(value, indent + 1)
                    elif isinstance(value, list):
                        f.write("  " * indent + f"{key}: {value}\n")
```

```
                elif isinstance(value, (float, np.floating)):
                    f.write("  " * indent + f"{key}: {float(value):.4f}\n")
                else:
                    f.write("  " * indent + f"{key}: {value}\n")
            write_dict(json_safe_dict)
        log_message(f"💾 Results saved: {json_file} and {txt_file}", also_print=False)
        return json_file, txt_file
    return None, None

def load_data(data_dir, train_ratio=0.7, test_ratio=0.15, val_ratio=0.15):
    if abs(train_ratio + test_ratio + val_ratio - 1.0) > 1e-6:
        raise ValueError("Oranların toplamı 1.0 olmalı!")

    all_data = []
    all_labels = []

    for folder in os.listdir(data_dir):
        folder_path = os.path.join(data_dir, folder)
        if not os.path.isdir(folder_path):
            print(f"Skipping file: {folder}")
            continue

        for file in os.listdir(folder_path):
            if not file.endswith('.csv'):
                continue
            file_path = os.path.join(folder_path, file)
            try:
                data = np.loadtxt(file_path, delimiter=',', skiprows=1, usecols=[1])
                all_data.append(data)
                if folder == 'groupA':
                    all_labels.append(0)
                elif folder == 'groupB':
                    all_labels.append(1)
                else:
                    print(f"Unknown folder: {folder}")
            except Exception as e:
                print(f"Error reading file {file_path}: {e}")
                continue

    combined = list(zip(all_data, all_labels))
    random.shuffle(combined)
    all_data, all_labels = zip(*combined)

    total_samples = len(all_data)
    train_end = int(total_samples * train_ratio)
    test_end = train_end + int(total_samples * test_ratio)

    train_data = all_data[:train_end]
    train_labels = all_labels[:train_end]
    test_data = all_data[train_end:test_end]
    test_labels = all_labels[train_end:test_end]
    val_data = all_data[test_end:]
```

```
    val_labels = all_labels[test_end:]

    padded_train_data = pad_zeros(train_data, target_length=128)
    padded_test_data = pad_zeros(test_data, target_length=128)
    padded_val_data = pad_zeros(val_data, target_length=128)

    padded_train_data = np.array(padded_train_data)
    padded_test_data = np.array(padded_test_data)
    padded_val_data = np.array(padded_val_data)

    print(f"Dataset split:")
    print(f"  Total samples: {total_samples}")
    print(f"  Train: {len(train_data)} samples ({len(train_data)/total_samples*100:.1f}%)")
    print(f"  Test: {len(test_data)} samples ({len(test_data)/total_samples*100:.1f}%)")
    print(f"  Validation: {len(val_data)} samples ({len(val_data)/total_samples*100:.1f}%)")

    log_message(f"Dataset split:")
    log_message(f"  Total samples: {total_samples}")
    log_message(f"  Train: {len(train_data)} samples ({len(train_data)/total_samples*100:.1f}%)")
    log_message(f"  Test: {len(test_data)} samples ({len(test_data)/total_samples*100:.1f}%)")
    log_message(f"  Validation: {len(val_data)} samples ({len(val_data)/total_samples*100:.1f}%)")

    padded_train_data = padded_train_data.reshape(-1, 128, 1)
    padded_test_data = padded_test_data.reshape(-1, 128, 1)
    padded_val_data = padded_val_data.reshape(-1, 128, 1)

    train_labels = np.array(train_labels)
    test_labels = np.array(test_labels)
    val_labels = np.array(val_labels)

    return (padded_train_data, train_labels, padded_test_data, test_labels, padded_val_data,
val_labels)

def pad_zeros(data, target_length=128):
    padded_data = []
    for file in data:
        current_length = len(file)
        if current_length == target_length:
            padded_data.append(file)
        elif current_length < target_length:
            padded_file = np.zeros(target_length)
            padded_file[:current_length] = file
            padded_data.append(padded_file)
        else:
            indices = np.linspace(0, current_length-1, target_length, dtype=int)
            downsampled_file = file[indices]
            padded_data.append(downsampled_file)
    return padded_data

def create_model(complexity='medium', learning_rate=0.0001):
    model = Sequential()
```

```
    if complexity == 'simple':
        model.add(Conv1D(filters=16, kernel_size=3, strides=2, activation='relu', input_shape=(128,
1)))
        model.add(MaxPooling1D(pool_size=2))
        model.add(Dropout(0.2))
        model.add(Flatten())
        model.add(Dense(128, activation='relu'))
        model.add(Dense(1, activation='sigmoid'))
    elif complexity == 'medium':
        model.add(Conv1D(filters=32, kernel_size=5, activation='relu', input_shape=(128, 1)))
        model.add(Conv1D(filters=32, kernel_size=3, activation='relu'))
        model.add(MaxPooling1D(pool_size=2))
        model.add(Dropout(0.3))
        model.add(Conv1D(filters=64, kernel_size=3, activation='relu'))
        model.add(MaxPooling1D(pool_size=2))
        model.add(Dropout(0.3))
        model.add(Flatten())
        model.add(Dense(256, activation='relu'))
        model.add(Dropout(0.4))
        model.add(Dense(128, activation='relu'))
        model.add(Dropout(0.4))
        model.add(Dense(1, activation='sigmoid'))
    elif complexity == 'complex':
        model.add(Conv1D(filters=64, kernel_size=7, activation='relu', input_shape=(128, 1)))
        model.add(Conv1D(filters=64, kernel_size=5, activation='relu'))
        model.add(MaxPooling1D(pool_size=2))
        model.add(Dropout(0.3))
        model.add(Conv1D(filters=128, kernel_size=3, activation='relu'))
        model.add(Conv1D(filters=128, kernel_size=3, activation='relu'))
        model.add(MaxPooling1D(pool_size=2))
        model.add(Dropout(0.4))
        model.add(Conv1D(filters=256, kernel_size=3, activation='relu'))
        model.add(MaxPooling1D(pool_size=2))
        model.add(Dropout(0.4))
        model.add(Flatten())
        model.add(Dense(512, activation='relu'))
        model.add(Dropout(0.5))
        model.add(Dense(256, activation='relu'))
        model.add(Dropout(0.5))
        model.add(Dense(128, activation='relu'))
        model.add(Dropout(0.5))
        model.add(Dense(1, activation='sigmoid'))

    optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
    return model

def train_model(model, train_data, train_labels, test_data, test_labels, epochs=200):
    train_data = np.array(train_data, dtype=np.float32)
    train_labels = np.array(train_labels, dtype=np.float32)
    test_data = np.array(test_data, dtype=np.float32)
    test_labels = np.array(test_labels, dtype=np.float32)
```

```
    print(f"Training data shape: {train_data.shape}")
    print(f"Training labels shape: {train_labels.shape}")
    log_message(f"Training started - Max epochs: {epochs}")
    log_message(f"Training data shape: {train_data.shape}")

    reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.3, patience=10, min_lr=1e-8,
verbose=1, cooldown=5)
    early_stop = EarlyStopping(monitor='val_loss', patience=30, restore_best_weights=True,
verbose=1, min_delta=0.0001)

    history = model.fit(train_data, train_labels, validation_data=(test_data, test_labels),
epochs=epochs, batch_size=8, callbacks=[reduce_lr, early_stop], verbose=1)
    return history

def evaluate_validation_set(model, val_data, val_labels, dataset_name="Validation"):
    val_data = np.array(val_data, dtype=np.float32)
    val_labels = np.array(val_labels, dtype=np.float32)

    print(f"\n{'='*60}")
    print(f"{dataset_name.upper()} SET EVALUATION")
    print(f"{'='*60}")

    log_message(f"\n{'='*60}")
    log_message(f"{dataset_name.upper()} SET EVALUATION")
    log_message(f"{'='*60}")

    val_loss, val_accuracy = model.evaluate(val_data, val_labels, verbose=0)
    print(f'{dataset_name} Loss: {val_loss:.4f}')
    print(f'{dataset_name} Accuracy: {val_accuracy:.4f}')
    log_message(f'{dataset_name} Loss: {val_loss:.4f}')
    log_message(f'{dataset_name} Accuracy: {val_accuracy:.4f}')

    y_pred_proba = model.predict(val_data, verbose=0).flatten()
    y_pred_binary = (y_pred_proba > 0.5).astype(int)

    fpr, tpr, thresholds = roc_curve(val_labels, y_pred_proba)
    auroc = roc_auc_score(val_labels, y_pred_proba)
    precision = precision_score(val_labels, y_pred_binary)
    recall = recall_score(val_labels, y_pred_binary)
    f1 = f1_score(val_labels, y_pred_binary)
    mcc = matthews_corrcoef(val_labels, y_pred_binary)

    tn, fp, fn, tp = confusion_matrix(val_labels, y_pred_binary).ravel()
    specificity = tn / (tn + fp)
    ppv = tp / (tp + fp) if (tp + fp) > 0 else 0
    npv = tn / (tn + fn) if (tn + fn) > 0 else 0

    print(f'\n{dataset_name} Performance Metrics:')
    print(f'  AUROC Score: {auroc:.4f}')
    print(f'  Accuracy: {val_accuracy:.4f}')
    print(f'  Precision: {precision:.4f}')
```

```
    print(f'  Recall: {recall:.4f}')
    print(f'  Specificity: {specificity:.4f}')
    print(f'  F1-Score: {f1:.4f}')
    print(f'  MCC: {mcc:.4f}')

    log_message(f'\n{dataset_name} Performance Metrics:')
    log_message(f'  AUROC Score: {auroc:.4f}')
    log_message(f'  Accuracy: {val_accuracy:.4f}')
    log_message(f'  Precision: {precision:.4f}')
    log_message(f'  Recall: {recall:.4f}')
    log_message(f'  F1-Score: {f1:.4f}')

    optimal_idx = np.argmax(tpr - fpr)
    optimal_threshold = thresholds[optimal_idx]

    # Create comprehensive plot
    plt.figure(figsize=(15, 10))

    # ROC Curve
    plt.subplot(2, 3, 1)
    plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC (AUROC = {auroc:.4f})')
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'{dataset_name} ROC Curve')
    plt.legend()
    plt.grid(True)

    # Confusion Matrix
    plt.subplot(2, 3, 2)
    cm = confusion_matrix(val_labels, y_pred_binary)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Group A', 'Group B'],
yticklabels=['Group A', 'Group B'])
    plt.title(f'{dataset_name} Confusion Matrix')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')

    # Metrics Bar Chart
    plt.subplot(2, 3, 3)
    metrics_names = ['Accuracy', 'Precision', 'Recall', 'F1-Score']
    metrics_values = [val_accuracy, precision, recall, f1]
    plt.bar(metrics_names, metrics_values, color=['skyblue', 'lightgreen', 'lightcoral', 'lightpink'])
    plt.ylim(0, 1)
    plt.title(f'{dataset_name} Metrics')
    plt.xticks(rotation=45)
    for i, v in enumerate(metrics_values):
        plt.text(i, v + 0.01, f'{v:.3f}', ha='center')

    # Probability Distribution
    plt.subplot(2, 3, 4)
    plt.hist(y_pred_proba[val_labels == 0], bins=20, alpha=0.7, label='Group A', color='blue')
    plt.hist(y_pred_proba[val_labels == 1], bins=20, alpha=0.7, label='Group B', color='red')
```

```
    plt.axvline(x=0.5, color='black', linestyle='--', label='Threshold (0.5)')
    plt.xlabel('Prediction Probability')
    plt.ylabel('Frequency')
    plt.title('Probability Distribution')
    plt.legend()

    # Precision-Recall Curve
    plt.subplot(2, 3, 5)
    precision_curve, recall_curve, _ = precision_recall_curve(val_labels, y_pred_proba)
    avg_precision = average_precision_score(val_labels, y_pred_proba)
    plt.plot(recall_curve, precision_curve, color='purple', lw=2, label=f'AP = {avg_precision:.4f}')
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.title('Precision-Recall Curve')
    plt.legend()
    plt.grid(True)

    # Error Analysis
    plt.subplot(2, 3, 6)
    error_types = ['TP', 'TN', 'FP', 'FN']
    error_counts = [tp, tn, fp, fn]
    colors = ['green', 'lightgreen', 'orange', 'red']
    plt.pie(error_counts, labels=error_types, colors=colors, autopct='%1.1f%%')
    plt.title('Prediction Distribution')

    plt.tight_layout()
    fig = plt.gcf()
    save_plot(fig, f"{dataset_name.lower()}_analysis.png")
    plt.show()

    # Save results
    results = {
        'loss': float(val_loss),
        'accuracy': float(val_accuracy),
        'auroc': float(auroc),
        'precision': float(precision),
        'recall': float(recall),
        'specificity': float(specificity),
        'f1_score': float(f1),
        'mcc': float(mcc),
        'ppv': float(ppv),
        'npv': float(npv),
        'optimal_threshold': float(optimal_threshold),
        'confusion_matrix': {'tp': int(tp), 'tn': int(tn), 'fp': int(fp), 'fn': int(fn)}
    }

    save_results_summary(results, f"{dataset_name.lower()}_results")
    return results, y_pred_proba, y_pred_binary

def perform_kfold_validation(all_data, all_labels, model_complexity='medium',
learning_rate=0.0001, k_folds=5, epochs=200):
    print(f"\n🔄 Starting {k_folds}-Fold Cross Validation...")
```

```
    log_message(f"\n🔄 Starting {k_folds}-Fold Cross Validation...")

    skf = StratifiedKFold(n_splits=k_folds, shuffle=True, random_state=42)

    fold_results = {'fold': [], 'accuracy': [], 'auroc': [], 'precision': [], 'recall': [], 'f1_score': [], 'mcc': [],
'epochs_trained': []}
    fold_models = []
    fold_histories = []  # Store training histories

    for fold, (train_idx, val_idx) in enumerate(skf.split(all_data, all_labels), 1):
        print(f"\n📁 FOLD {fold}/{k_folds}")
        log_message(f"\n📁 FOLD {fold}/{k_folds}")

        X_train_fold = all_data[train_idx]
        y_train_fold = all_labels[train_idx]
        X_val_fold = all_data[val_idx]
        y_val_fold = all_labels[val_idx]

        print(f"Train samples: {len(X_train_fold)}")
        print(f"Validation samples: {len(X_val_fold)}")

        model = create_model(complexity=model_complexity, learning_rate=learning_rate)

        # Train model for fold
        X_train_fold = np.array(X_train_fold, dtype=np.float32)
        y_train_fold = np.array(y_train_fold, dtype=np.float32)
        X_val_fold = np.array(X_val_fold, dtype=np.float32)
        y_val_fold = np.array(y_val_fold, dtype=np.float32)

        reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.3, patience=8, min_lr=1e-8,
verbose=0)
        early_stop = EarlyStopping(monitor='val_loss', patience=25, restore_best_weights=True,
verbose=0)

        history = model.fit(X_train_fold, y_train_fold, validation_data=(X_val_fold, y_val_fold),
epochs=epochs, batch_size=8, callbacks=[reduce_lr, early_stop], verbose=0)

        # Store history for later plotting
        fold_histories.append(history)

        # Evaluate fold
        val_loss, val_accuracy = model.evaluate(X_val_fold, y_val_fold, verbose=0)
        y_pred_proba = model.predict(X_val_fold, verbose=0).flatten()
        y_pred_binary = (y_pred_proba > 0.5).astype(int)

        auroc = roc_auc_score(y_val_fold, y_pred_proba)
        precision = precision_score(y_val_fold, y_pred_binary)
        recall = recall_score(y_val_fold, y_pred_binary)
        f1 = f1_score(y_val_fold, y_pred_binary)
        mcc = matthews_corrcoef(y_val_fold, y_pred_binary)

        fold_results['fold'].append(fold)
```

```
    fold_results['accuracy'].append(float(val_accuracy))
    fold_results['auroc'].append(float(auroc))
    fold_results['precision'].append(float(precision))
    fold_results['recall'].append(float(recall))
    fold_results['f1_score'].append(float(f1))
    fold_results['mcc'].append(float(mcc))
    fold_results['epochs_trained'].append(int(len(history.history['loss'])))
    fold_models.append(model)

    print(f"✅ Fold {fold} - Accuracy: {val_accuracy:.4f}, AUROC: {auroc:.4f}")
    log_message(f"✅ Fold {fold} - Accuracy: {val_accuracy:.4f}, AUROC: {auroc:.4f}")

# Plot K-fold training histories
plot_kfold_training_histories(fold_histories, fold_results)

# Print summary
print(f"\n📊 K-FOLD RESULTS SUMMARY")
log_message(f"\n📊 K-FOLD RESULTS SUMMARY")
log_message("="*50)

metrics = ['accuracy', 'auroc', 'precision', 'recall', 'f1_score', 'mcc']
summary_stats = {}

for metric in metrics:
    values = fold_results[metric]
    mean_val = np.mean(values)
    std_val = np.std(values)
    summary_line = f"{metric.upper()}: {mean_val:.4f} ± {std_val:.4f}"
    print(summary_line)
    log_message(summary_line)

    # Store for later use
    summary_stats[metric] = {
        'mean': float(mean_val),
        'std': float(std_val),
        'values': [float(v) for v in values]
    }

# Create comprehensive K-fold summary
kfold_summary = {
    'timestamp': CURRENT_TIMESTAMP,
    'k_folds': k_folds,
    'model_complexity': model_complexity,
    'learning_rate': learning_rate,
    'max_epochs': epochs,
    'total_samples_used': len(all_data),
    'fold_details': {
        'individual_results': fold_results,
        'summary_statistics': summary_stats
    },
    'best_fold': {
        'fold_number': int(np.argmax(fold_results['auroc']) + 1),
```

```
            'best_auroc': float(np.max(fold_results['auroc'])),
            'best_accuracy': float(fold_results['accuracy'][np.argmax(fold_results['auroc'])]),
            'best_f1_score': float(fold_results['f1_score'][np.argmax(fold_results['auroc'])])
        }
    }

    # Save K-fold results
    save_results_summary(kfold_summary, "kfold_detailed_results")

    return fold_results, fold_models

def plot_kfold_training_histories(fold_histories, fold_results):
    """
    K-fold training geçmişlerini görselleştirir
    """
    print("\n📈 Plotting K-fold training histories...")
    log_message("\n📈 Plotting K-fold training histories...")

    plt.figure(figsize=(20, 12))

    # 1. Training Loss for all folds
    plt.subplot(2, 4, 1)
    for i, history in enumerate(fold_histories):
        plt.plot(history.history['loss'], alpha=0.7, label=f'Fold {i+1}', linewidth=2)
    plt.title('Training Loss Across Folds')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True, alpha=0.3)

    # 2. Validation Loss for all folds
    plt.subplot(2, 4, 2)
    for i, history in enumerate(fold_histories):
        plt.plot(history.history['val_loss'], alpha=0.7, label=f'Fold {i+1}', linewidth=2)
    plt.title('Validation Loss Across Folds')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True, alpha=0.3)

    # 3. Training Accuracy for all folds
    plt.subplot(2, 4, 3)
    for i, history in enumerate(fold_histories):
        plt.plot(history.history['accuracy'], alpha=0.7, label=f'Fold {i+1}', linewidth=2)
    plt.title('Training Accuracy Across Folds')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.grid(True, alpha=0.3)

    # 4. Validation Accuracy for all folds
    plt.subplot(2, 4, 4)
```

```
for i, history in enumerate(fold_histories):
    plt.plot(history.history['val_accuracy'], alpha=0.7, label=f'Fold {i+1}', linewidth=2)
plt.title('Validation Accuracy Across Folds')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True, alpha=0.3)

# 5. Epochs trained per fold
plt.subplot(2, 4, 5)
folds = fold_results['fold']
epochs_trained = fold_results['epochs_trained']
colors = plt.cm.Set3(np.linspace(0, 1, len(folds)))
bars = plt.bar(folds, epochs_trained, color=colors, alpha=0.8)
plt.title('Epochs Trained per Fold')
plt.xlabel('Fold')
plt.ylabel('Epochs')
plt.grid(True, alpha=0.3)
# Add value labels on bars
for bar, epochs in zip(bars, epochs_trained):
    plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.5,
             f'{epochs}', ha='center', va='bottom', fontweight='bold')

# 6. Performance metrics across folds
plt.subplot(2, 4, 6)
folds = fold_results['fold']
plt.plot(folds, fold_results['accuracy'], 'o-', label='Accuracy', linewidth=2, markersize=8)
plt.plot(folds, fold_results['auroc'], 's-', label='AUROC', linewidth=2, markersize=8)
plt.plot(folds, fold_results['f1_score'], '^-', label='F1-Score', linewidth=2, markersize=8)
plt.title('Performance Metrics Across Folds')
plt.xlabel('Fold')
plt.ylabel('Score')
plt.legend()
plt.grid(True, alpha=0.3)
plt.ylim(0.5, 1.0)

# 7. Average training curves
plt.subplot(2, 4, 7)
# Calculate average training curves
max_epochs = max(len(h.history['loss']) for h in fold_histories)
avg_train_loss = []
avg_val_loss = []

for epoch in range(max_epochs):
    epoch_train_losses = []
    epoch_val_losses = []
    for history in fold_histories:
        if epoch < len(history.history['loss']):
            epoch_train_losses.append(history.history['loss'][epoch])
            epoch_val_losses.append(history.history['val_loss'][epoch])

    if epoch_train_losses:
```

```
            avg_train_loss.append(np.mean(epoch_train_losses))
            avg_val_loss.append(np.mean(epoch_val_losses))

    plt.plot(avg_train_loss, label='Average Training Loss', linewidth=3, color='blue')
    plt.plot(avg_val_loss, label='Average Validation Loss', linewidth=3, color='red')
    plt.title('Average Loss Curves')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True, alpha=0.3)

    # 8. Best fold detailed training curve
    plt.subplot(2, 4, 8)
    best_fold_idx = np.argmax(fold_results['auroc'])
    best_history = fold_histories[best_fold_idx]

    plt.plot(best_history.history['loss'], label='Training Loss', linewidth=2, color='blue')
    plt.plot(best_history.history['val_loss'], label='Validation Loss', linewidth=2, color='red')
    plt.plot(best_history.history['accuracy'], label='Training Accuracy', linewidth=2, color='green',
linestyle='--')
    plt.plot(best_history.history['val_accuracy'], label='Validation Accuracy', linewidth=2,
color='orange', linestyle='--')
    plt.title(f'Best Fold ({best_fold_idx + 1}) Training Curves')
    plt.xlabel('Epoch')
    plt.ylabel('Loss / Accuracy')
    plt.legend()
    plt.grid(True, alpha=0.3)

    plt.tight_layout()

    # Save the plot
    fig = plt.gcf()
    save_plot(fig, "kfold_training_histories.png", "K-Fold Training Histories")

    plt.show()

    # Log some statistics about training
    avg_epochs = np.mean(fold_results['epochs_trained'])
    min_epochs = np.min(fold_results['epochs_trained'])
    max_epochs = np.max(fold_results['epochs_trained'])

    print(f"\n📊 Training Statistics:")
    print(f"   Average epochs trained: {avg_epochs:.1f}")
    print(f"   Min epochs: {min_epochs}")
    print(f"   Max epochs: {max_epochs}")
    print(f"   Best fold: {best_fold_idx + 1} (AUROC: {fold_results['auroc'][best_fold_idx]:.4f})")

    log_message(f"\n📊 Training Statistics:")
    log_message(f"   Average epochs trained: {avg_epochs:.1f}")
    log_message(f"   Min epochs: {min_epochs}")
    log_message(f"   Max epochs: {max_epochs}")
```

```
    log_message(f"   Best fold: {best_fold_idx + 1} (AUROC:
{fold_results['auroc'][best_fold_idx]:.4f})")

def plot_single_training_history(history):
    """
    Tek model eğitim geçmişini görselleştirir
    """
    print("\n📈 Plotting training history...")
    log_message("\n📈 Plotting training history...")

    plt.figure(figsize=(12, 4))

    plt.subplot(1, 2, 1)
    plt.plot(history.history['loss'], label='Training Loss', linewidth=2)
    plt.plot(history.history['val_loss'], label='Validation Loss', linewidth=2)
    plt.title('Model Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True)

    plt.subplot(1, 2, 2)
    plt.plot(history.history['accuracy'], label='Training Accuracy', linewidth=2)
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy', linewidth=2)
    plt.title('Model Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.grid(True)

    plt.tight_layout()

    # Save training history plot
    fig = plt.gcf()
    save_plot(fig, "single_model_training_history.png", "Single Model Training History")

    plt.show()

    # Log training statistics
    final_train_loss = history.history['loss'][-1]
    final_val_loss = history.history['val_loss'][-1]
    final_train_acc = history.history['accuracy'][-1]
    final_val_acc = history.history['val_accuracy'][-1]
    epochs_trained = len(history.history['loss'])

    print(f"\n📊 Training Statistics:")
    print(f"   Epochs trained: {epochs_trained}")
    print(f"   Final training loss: {final_train_loss:.4f}")
    print(f"   Final validation loss: {final_val_loss:.4f}")
    print(f"   Final training accuracy: {final_train_acc:.4f}")
    print(f"   Final validation accuracy: {final_val_acc:.4f}")
```

```
    log_message(f"\n📊 Training Statistics:")
    log_message(f"   Epochs trained: {epochs_trained}")
    log_message(f"   Final training loss: {final_train_loss:.4f}")
    log_message(f"   Final validation loss: {final_val_loss:.4f}")
    log_message(f"   Final training accuracy: {final_train_acc:.4f}")
    log_message(f"   Final validation accuracy: {final_val_acc:.4f}")

if __name__ == '__main__':
    setup_output_directory()

    data_dir = './data'

    print("🚀 CNN Classification System Starting...")
    log_message("🚀 CNN Classification System Starting...")

    try:
        train_data, train_labels, test_data, test_labels, val_data, val_labels = load_data(data_dir)
    except Exception as e:
        error_msg = f"❌ Error loading data: {e}"
        print(error_msg)
        log_message(error_msg)
        sys.exit(1)

    # Configuration
    model_complexity = 'medium'
    learning_rate = 0.0001
    max_epochs = 200
    use_kfold = True
    k_folds = 5

    print(f"\n⚙️ Configuration:")
    print(f"   Model: {model_complexity}")
    print(f"   Learning Rate: {learning_rate}")
    print(f"   Max Epochs: {max_epochs}")
    print(f"   K-Fold: {use_kfold}")

    log_message(f"\n⚙️ Configuration:")
    log_message(f"   Model: {model_complexity}")
    log_message(f"   Learning Rate: {learning_rate}")
    log_message(f"   Max Epochs: {max_epochs}")
    log_message(f"   K-Fold: {use_kfold}")

    if use_kfold:
        all_train_data = np.concatenate([train_data, test_data], axis=0)
        all_train_labels = np.concatenate([train_labels, test_labels], axis=0)

        fold_results, fold_models = perform_kfold_validation(all_train_data, all_train_labels,
model_complexity, learning_rate, k_folds, max_epochs)

        best_fold_idx = np.argmax(fold_results['auroc'])
        best_model = fold_models[best_fold_idx]
```

```
        print(f"\n🏆 Best model: Fold {best_fold_idx + 1}")
        print(f"   Best AUROC: {fold_results['auroc'][best_fold_idx]:.4f}")
        print(f"   Best Accuracy: {fold_results['accuracy'][best_fold_idx]:.4f}")
        print(f"   Best F1-Score: {fold_results['f1_score'][best_fold_idx]:.4f}")

        log_message(f"\n🏆 Best model: Fold {best_fold_idx + 1}")
        log_message(f"   Best AUROC: {fold_results['auroc'][best_fold_idx]:.4f}")
        log_message(f"   Best Accuracy: {fold_results['accuracy'][best_fold_idx]:.4f}")
        log_message(f"   Best F1-Score: {fold_results['f1_score'][best_fold_idx]:.4f}")

        print(f"\n🎯 Final evaluation on held-out validation set...")
        log_message(f"\n🎯 Final evaluation on held-out validation set...")

        val_results, _, _ = evaluate_validation_set(best_model, val_data, val_labels,
"Final_Validation")

        # Compare K-fold vs Final validation
        kfold_mean_auroc = np.mean(fold_results['auroc'])
        kfold_std_auroc = np.std(fold_results['auroc'])
        kfold_mean_acc = np.mean(fold_results['accuracy'])
        kfold_std_acc = np.std(fold_results['accuracy'])

        print(f"\n📊 K-FOLD vs FINAL VALIDATION COMPARISON")
        print(f"{'='*60}")
        print(f"K-Fold Cross Validation (Average ± Std):")
        print(f"   AUROC: {kfold_mean_auroc:.4f} ± {kfold_std_auroc:.4f}")
        print(f"   Accuracy: {kfold_mean_acc:.4f} ± {kfold_std_acc:.4f}")
        print(f"\nFinal Validation (Held-out):")
        print(f"   AUROC: {val_results['auroc']:.4f}")
        print(f"   Accuracy: {val_results['accuracy']:.4f}")

        log_message(f"\n📊 K-FOLD vs FINAL VALIDATION COMPARISON")
        log_message(f"{'='*60}")
        log_message(f"K-Fold Cross Validation (Average ± Std):")
        log_message(f"   AUROC: {kfold_mean_auroc:.4f} ± {kfold_std_auroc:.4f}")
        log_message(f"   Accuracy: {kfold_mean_acc:.4f} ± {kfold_std_acc:.4f}")
        log_message(f"\nFinal Validation (Held-out):")
        log_message(f"   AUROC: {val_results['auroc']:.4f}")
        log_message(f"   Accuracy: {val_results['accuracy']:.4f}")

        # Generalization analysis
        auroc_diff = kfold_mean_auroc - val_results['auroc']
        acc_diff = kfold_mean_acc - val_results['accuracy']

        print(f"\nGeneralization Analysis:")
        print(f"   AUROC difference (K-fold - Final): {auroc_diff:.4f}")
        print(f"   Accuracy difference (K-fold - Final): {acc_diff:.4f}")

        log_message(f"\nGeneralization Analysis:")
        log_message(f"   AUROC difference (K-fold - Final): {auroc_diff:.4f}")
```

```
    log_message(f"   Accuracy difference (K-fold - Final): {acc_diff:.4f}")

    if abs(auroc_diff) < 0.03 and abs(acc_diff) < 0.03:
        generalization_msg = "   ✅ Excellent generalization! Model performs consistently."
    elif abs(auroc_diff) < 0.05 and abs(acc_diff) < 0.05:
        generalization_msg = "   ✅ Good generalization. Model is reliable."
    elif abs(auroc_diff) < 0.10 and abs(acc_diff) < 0.10:
        generalization_msg = "   ⚠️ Fair generalization. Some performance drop expected."
    else:
        generalization_msg = "   ❌ Poor generalization. Significant performance variability."

    print(generalization_msg)
    log_message(generalization_msg)

    final_auroc = val_results['auroc']

    # Create comprehensive final summary
    final_summary = {
        'timestamp': CURRENT_TIMESTAMP,
        'dataset_info': {
            'total_samples': len(train_labels) + len(test_labels) + len(val_labels),
            'train_samples': len(train_labels),
            'test_samples': len(test_labels),
            'validation_samples': len(val_labels),
            'train_percentage': 70.0,
            'test_percentage': round((len(test_labels) / (len(train_labels) + len(test_labels) +
len(val_labels))) * 100, 1),
            'validation_percentage': round((len(val_labels) / (len(train_labels) + len(test_labels) +
len(val_labels))) * 100, 1)
        },
        'configuration': {
            'model_complexity': model_complexity,
            'learning_rate': learning_rate,
            'max_epochs': max_epochs,
            'use_kfold': use_kfold,
            'k_folds': k_folds
        },
        'kfold_performance': {
            'mean_auroc': float(kfold_mean_auroc),
            'std_auroc': float(kfold_std_auroc),
            'mean_accuracy': float(kfold_mean_acc),
            'std_accuracy': float(kfold_std_acc),
            'best_fold': {
                'fold_number': int(best_fold_idx + 1),
                'auroc': float(fold_results['auroc'][best_fold_idx]),
                'accuracy': float(fold_results['accuracy'][best_fold_idx]),
                'f1_score': float(fold_results['f1_score'][best_fold_idx])
            }
        },
        'final_validation': {
            'auroc': float(val_results['auroc']),
            'accuracy': float(val_results['accuracy']),
```

```
            'f1_score': float(val_results['f1_score']),
            'precision': float(val_results['precision']),
            'recall': float(val_results['recall']),
            'mcc': float(val_results['mcc'])
        },
        'generalization_analysis': {
            'auroc_difference': float(auroc_diff),
            'accuracy_difference': float(acc_diff),
            'generalization_assessment': generalization_msg.strip()
        }
    }

    save_results_summary(final_summary, "comprehensive_final_summary")

else:
    model = create_model(complexity=model_complexity, learning_rate=learning_rate)
    history = train_model(model, train_data, train_labels, test_data, test_labels, max_epochs)
    val_results, _, _ = evaluate_validation_set(model, val_data, val_labels, "Validation")
    final_auroc = val_results['auroc']

# Final summary
print(f"\n📋 COMPREHENSIVE FINAL RESULTS:")
print(f"{'='*60}")
print(f"Dataset Information:")
print(f"   Total samples: {len(train_labels) + len(test_labels) + len(val_labels)}")
print(f"   Train: {len(train_labels)} samples (70.0%)")
print(f"   Test: {len(test_labels)} samples ({(len(test_labels) / (len(train_labels) + len(test_labels)
+ len(val_labels))) * 100:.1f}%)")
print(f"   Validation: {len(val_labels)} samples ({(len(val_labels) / (len(train_labels) +
len(test_labels) + len(val_labels))) * 100:.1f}%)")

if use_kfold:
    print(f"\nK-Fold Cross Validation Results:")
    print(f"   Mean AUROC: {kfold_mean_auroc:.4f} ± {kfold_std_auroc:.4f}")
    print(f"   Mean Accuracy: {kfold_mean_acc:.4f} ± {kfold_std_acc:.4f}")
    print(f"   Best Fold: {best_fold_idx + 1} (AUROC:
{fold_results['auroc'][best_fold_idx]:.4f})")

print(f"\nFinal Validation Performance:")
print(f"   Final AUROC: {final_auroc:.4f}")
print(f"   Final Accuracy: {val_results['accuracy']:.4f}")
print(f"   Final F1-Score: {val_results['f1_score']:.4f}")
print(f"   Final Precision: {val_results['precision']:.4f}")
print(f"   Final Recall: {val_results['recall']:.4f}")
print(f"   Final MCC: {val_results['mcc']:.4f}")

# Log the same information
log_message(f"\n📋 COMPREHENSIVE FINAL RESULTS:")
log_message(f"{'='*60}")
log_message(f"Dataset Information:")
log_message(f"   Total samples: {len(train_labels) + len(test_labels) + len(val_labels)}")
log_message(f"   Train: {len(train_labels)} samples (70.0%)")
```

```
    log_message(f"   Test: {len(test_labels)} samples ({(len(test_labels) / (len(train_labels) +
len(test_labels) + len(val_labels))) * 100:.1f}%)")
    log_message(f"   Validation: {len(val_labels)} samples ({(len(val_labels) / (len(train_labels) +
len(test_labels) + len(val_labels))) * 100:.1f}%)")

    if use_kfold:
        log_message(f"\nK-Fold Cross Validation Results:")
        log_message(f"   Mean AUROC: {kfold_mean_auroc:.4f} ± {kfold_std_auroc:.4f}")
        log_message(f"   Mean Accuracy: {kfold_mean_acc:.4f} ± {kfold_std_acc:.4f}")
        log_message(f"   Best Fold: {best_fold_idx + 1} (AUROC:
{fold_results['auroc'][best_fold_idx]:.4f})")

    log_message(f"\nFinal Validation Performance:")
    log_message(f"   Final AUROC: {final_auroc:.4f}")
    log_message(f"   Final Accuracy: {val_results['accuracy']:.4f}")
    log_message(f"   Final F1-Score: {val_results['f1_score']:.4f}")
    log_message(f"   Final Precision: {val_results['precision']:.4f}")
    log_message(f"   Final Recall: {val_results['recall']:.4f}")
    log_message(f"   Final MCC: {val_results['mcc']:.4f}")

    print(f"\n📁 All results saved to: {OUTPUT_DIR}")
    print("🏁 Analysis Complete!")

    log_message(f"\n📁 All results saved to: {OUTPUT_DIR}")
    log_message("🏁 Analysis Complete!")
```