UNIVERSIDADE DE CAXIAS DO SUL
ÁREA DO CONHECIMENTO DE CIÊNCIAS EXATAS E ENGENHARIAS

VAGNER ANTONIO CAZAROTTO

SISTEMA INTEGRADO DE AQUISIÇÃO E CLASSIFICAÇÃO AUTOMÁTICA DE ARRITMIAS CARDÍACAS USANDO REDES NEURAIS CONVOLUCIONAIS

CAXIAS DO SUL

2025

VAGNER ANTONIO CAZAROTTO

SISTEMA INTEGRADO DE AQUISIÇÃO E CLASSIFICAÇÃO AUTOMÁTICA DE ARRITMIAS CARDÍACAS USANDO REDES NEURAIS CONVOLUCIONAIS

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Engenharia de Controle Automação na Área do Conhecimento de Ciências Exatas e Engenharias da Universidade de Caxias do Sul.

Orientador: Prof. Dr. Alexandre Mesquita

CAXIAS DO SUL

2025

VAGNER ANTONIO CAZAROTTO

SISTEMA INTEGRADO DE AQUISIÇÃO E CLASSIFICAÇÃO AUTOMÁTICA DE ARRITMIAS CARDÍACAS USANDO REDES NEURAIS CONVOLUCIONAIS

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Engenharia de Controle Automação na Área do Conhecimento de Ciências Exatas e Engenharias da Universidade de Caxias do Sul.

Aprovado em 03/07/2025

BANCA EXAMINADORA

_____

Prof. Dr. Alexandre Mesquita
Universidade de Caxias do Sul - UCS

_____

Prof. Me. Ricardo Leal Costi
Universidade de Caxias do Sul - UCS

_____

Prof. Dra. Carine Geltrudes Webber
Universidade de Caxias do Sul - UCS

AGRADECIMENTOS

A realização deste trabalho foi possível graças ao apoio e incentivo de muitas pessoas às quais expresso minha sincera gratidão. Agradeço, em primeiro lugar, a Deus, por ter me concedido força, saúde e sabedoria ao longo desta jornada acadêmica.

Ao meu orientador, Prof. Dr. Alexandre Mesquita, pela orientação precisa e confiança depositadas em mim desde o início deste projeto. Sua dedicação e conhecimento foram fundamentais para a construção deste trabalho. Ao Prof. Me. Ricardo Leal Costi, pelo seu incentivo e apoio ao longo dos anos, inclusive dos quais não estive tão próximo do mundo acadêmico.

À Universidade de Caxias do Sul e ao corpo docente do curso de Engenharia de Controle e Automação, pelo aprendizado transmitido e pela formação sólida que levarei comigo para a vida profissional. Aos meus colegas e amigos, que compartilharam comigo os desafios e conquistas desta caminhada, em especial àqueles que contribuíram com ideias, apoio técnico e incentivo nos momentos difíceis.

Aos meus familiares, especialmente a Caroline, por todo amor, compreensão e apoio incondicional. Vocês foram o alicerce que me sustentou em cada etapa.

Por fim, agradeço a todas as pessoas que, direta ou indiretamente, contribuíram para a concretização deste trabalho.

Muito obrigado.

“It’s all about continual improvement”

Masaaki Imai

RESUMO

As doenças cardiovasculares constituem a principal causa de mortalidade global nas últimas décadas, sendo as arritmias cardíacas um dos distúrbios mais prevalentes nesse grupo. Tais alterações afetam os impulsos elétricos do coração e podem ser identificadas por meio do eletrocardiograma (ECG), exame amplamente utilizado, porém ainda suscetível a erros de interpretação humana. Neste contexto, o presente trabalho propõe o desenvolvimento de um sistema automatizado para aquisição e classificação de arritmias cardíacas em três categorias: batimentos normais, ectópicos ventriculares e ectópicos supraventriculares.

A metodologia abrange desde a coleta do sinal real de ECG, realizada com o sensor AD8232 conectado a um microcontrolador ESP32, até o treinamento de redes neurais convolucionais unidimensionais (CNN 1D) desenvolvidas em PyTorch. A aquisição dos sinais foi realizada com resolução de 12 bits e taxa de amostragem de 977 Hz. O pré-processamento incluiu filtragem digital, normalização e segmentação com base nos picos R.

Os modelos foram treinados com a base pública *Leipzig Heart Center ECG-Database*, composta por mais de 113 mil batimentos anotados por especialistas. A arquitetura CNN foi capaz de atingir uma acurácia de aproximadamente 92,8% na validação, demonstrando robustez na detecção das três classes de arritmia mesmo quando testada com sinais reais adquiridos pelo sistema embarcado. Os resultados obtidos confirmam a viabilidade de integrar técnicas de aprendizado profundo a dispositivos portáteis de monitoramento, sugerindo aplicações futuras em ambientes clínicos ou domiciliares.

Palavras-chave: Doenças cardiovasculares. Arritmias cardíacas. Eletrocardiograma. Redes neurais convolucionais. Classificação automática.

ABSTRACT

Cardiovascular diseases have been the leading cause of global mortality over the past decades, with cardiac arrhythmias being among the most prevalent disorders in this group. These abnormalities affect the heart's electrical impulses and can be identified through the electrocardiogram (ECG), a widely used diagnostic tool that remains susceptible to human interpretation errors. In this context, this study proposes the development of an automated system for the acquisition and classification of cardiac arrhythmias into three categories: normal beats, ventricular ectopic beats, and supraventricular ectopic beats.

The methodology encompasses the acquisition of real ECG signals using the AD8232 sensor connected to an ESP32 microcontroller, followed by the training of one-dimensional convolutional neural networks (1D CNNs) developed in PyTorch. Signal acquisition was performed with 12-bit resolution and a sampling rate of 977 Hz. The preprocessing stage included digital filtering, normalization, and segmentation based on R-peaks.

The models were trained using the publicly available *Leipzig Heart Center ECG-Database*, which contains over 113,000 beats annotated by specialists. The proposed CNN architecture achieved an accuracy of approximately 92.8% during validation, demonstrating robustness in detecting the three arrhythmia classes even when tested with real signals acquired by the embedded system. The results confirm the feasibility of integrating deep learning techniques into portable monitoring devices, suggesting potential applications in clinical or home environments.

Keywords: Cardiovascular diseases, Cardiac arrhythmias, Electrocardiogram, Convolutional neural networks, Automatic classification.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS E SIGLAS

CNN Convolutional Neural Network ou Rede Neural Convolucional

ECG Eletrocardiograma

AD8232 Amplificador Diferencial Integrado AD8232

AVC Acidente Vascular Cerebral

ECG Eletrocardiograma

IoT Internet das Coisas (Internet of Things)

LISTA DE SÍMBOLOS

bpm	Batimentos por minuto
σ	Desvio padrão, medida de dispersão
μ	Média aritmética
Δt	Intervalo de tempo entre amostras
V	Classe de batimentos ventriculares no ECG
N	Classe de batimentos normais no ECG
S	Classe de batimentos supraventriculares no ECG
b_k	Bias externo aplicado ao neurônio k.
x_i	Sinal de entrada i em um neurônio.
w_{ki}	Peso da sinapse entre a entrada i e o neurônio k.
v_k	Resultado da soma ponderada das entradas para o neurônio k.
y_k	Saída do neurônio k após a função de ativação.
$\phi(\cdot)$	Função de ativação do neurônio.

SUMÁRIO

1 INTRODUÇÃO

As doenças cardiovasculares continuam a ser uma das principais causas de morte em todo o mundo, com as arritmias destacando-se como um fator crítico para o aumento de complicações clínicas e mortalidade (SANTANA, 2021). As arritmias são caracterizadas por irregularidades na atividade elétrica do coração, que podem se manifestar de diversas formas, desde batimentos cardíacos acelerados (taquicardias) até batimentos lentos (bradicardias), além de padrões irregulares que afetam a eficácia do bombeamento sanguíneo (SZPALHER; BATALHA, 2019). A detecção e o diagnóstico precoces dessas condições são fundamentais para reduzir o risco de eventos graves, como ataques cardíacos, acidentes vasculares cerebrais Acidente Vascular Cerebral (AVC) e falência cardíaca. No entanto, os métodos convencionais de monitoramento e diagnóstico apresentam limitações que dificultam uma resposta ágil e precisa (ROCHA, 2018).

O exame mais comumente utilizado para detectar arritmias é o eletrocardiograma (ECG), um método não invasivo que mede a atividade elétrica do coração e fornece um registro visual da função cardíaca ao longo do tempo. A análise manual do ECG por médicos ou cardiologistas, embora eficaz, pode ser demorada e sujeita a erros, especialmente em situações que requerem monitoramento contínuo ou onde grandes volumes de dados são coletados, como no uso de dispositivos de Holter, que registram o ECG ao longo de 24 horas ou mais. Dado o crescente volume de dados gerado por esses dispositivos, a análise manual torna-se impraticável e aumenta a probabilidade de erros na interpretação dos sinais, o que pode resultar em diagnósticos equivocados ou atrasados (FERNANDES; AL., 2015).

Além disso, há o desafio adicional de variabilidade interindividual nos sinais de ECG. Cada paciente pode apresentar características específicas no seu padrão cardíaco, que podem mascarar ou dificultar a identificação de arritmias. A diversidade morfológica dos batimentos cardíacos entre pacientes complica a criação de um método universal e robusto de detecção de arritmias, exigindo técnicas que possam capturar e interpretar adequadamente essas variações (HABIB; KARMAKAR; YEARWOOD, 2019).

Com o avanço das tecnologias de monitoramento cardíaco portátil e o aumento da adoção de dispositivos vestíveis (wearables), que oferecem a capacidade de monitorar a saúde cardíaca de forma contínua e em tempo real, o problema da análise automática e precisa dos sinais de ECG torna-se ainda mais relevante. Estes dispositivos geram uma quantidade massiva de dados que precisa ser processada em tempo hábil para fornecer alertas precoces ou diagnósticos oportunos. Neste cenário, a eficiência na detecção de arritmias depende diretamente da capacidade de desenvolver algoritmos computacionais capazes de analisar esses dados em grande escala, com alta precisão e em tempo real.

Assim, o problema central a ser resolvido é a necessidade de automatizar a análise e classificação de sinais de ECG para detecção de arritmias, garantindo a precisão e a eficiência necessárias para um diagnóstico precoce e confiável. A solução para esse problema requer o desenvolvimento de métodos avançados de processamento de sinais e inteligência artificial que possam superar as limitações dos sistemas tradicionais de análise manual. Isso envolve não apenas o processamento e filtragem adequados dos sinais de ECG para remover ruídos e artefatos, mas também o uso de técnicas sofisticadas de aprendizado de máquina que sejam capazes de detectar padrões complexos e sutis nos sinais cardíacos que indicam a presença de arritmias (LUZ *et al.*, 2020).

O desenvolvimento de soluções automatizadas para a detecção de arritmias enfrenta uma série de desafios técnicos. Entre eles, destaca-se a necessidade de lidar com dados não estacionários, ou seja, sinais cuja composição de frequências varia ao longo do tempo, o que dificulta a extração de características fixas que possam ser usadas para a classificação. Outro desafio é a presença de ruídos, como a interferência de outros sinais fisiológicos ou movimentos do paciente, que podem comprometer a integridade dos sinais de ECG (LU; LIU; CHEN, 2019).

Por fim, qualquer solução proposta para a detecção automática de arritmias precisa ser validada em cenários clínicos reais, com dados diversificados e em grande volume, para garantir que os algoritmos sejam robustos o suficiente para lidar com diferentes perfis de pacientes e condições de saúde. O desenvolvimento de tais sistemas automatizados não só aliviaria o trabalho dos profissionais de saúde, mas também proporcionaria uma ferramenta de suporte à decisão médica, melhorando a capacidade de resposta a eventos cardíacos e aumentando a eficácia dos tratamentos preventivos.

Portanto, o problema que se busca solucionar com o avanço das tecnologias de classificação automática de arritmias é duplo: a necessidade de métodos de detecção mais rápidos e precisos, capazes de lidar com grandes volumes de dados, e a capacidade de oferecer soluções que sejam robustas o suficiente para serem aplicadas em cenários clínicos diversos, onde a variabilidade dos sinais de ECG e a presença de ruídos constituem desafios importantes.

Este trabalho tem como finalidade avaliar arquiteturas de redes neurais com o objetivo de identificar sinais de ECG irregulares que possam indicar a necessidade de atenção médica. Busca-se, assim, auxiliar profissionais da saúde na detecção de anomalias em exames de ECG ou enviar alertas ao usuário de dispositivos móveis de monitoramento.

É importante destacar que este trabalho tem caráter experimental e os resultados apresentados não devem ser aplicados em contextos clínicos ou utilizados para fins de diagnóstico médico sem a devida regulamentação. Por se tratar de uma aplicação na área da saúde, qualquer solução proposta deverá, obrigatoriamente, ser submetida à avaliação de um Comitê de Ética em Pesquisa e à aprovação da Agência Nacional de Vigilância Sanitária (ANVISA), conforme exigido pela legislação brasileira. Apenas após essas etapas é que sua aplicação poderá ser considerada segura e adequada para uso em ambiente médico ou hospitalar.

1.1 OBJETIVO GERAL E OBJETIVOS ESPECÍFICOS

Este trabalho pode ser estruturado em dois grupos de objetivos: o objetivo geral e os objetivos específicos. Como objetivo geral, tem-se a classificação automática de arritmias em sinais de ECG utilizando um método baseado em Redes Neurais Convolucionais.

Para que o objetivo geral seja alcançado, foram estabelecidos os seguintes objetivos específicos:

1) Realizar uma análise das bases de dados de sinais de ECG disponíveis na literatura, selecionando o conjunto de dados mais apropriado para este estudo, bem como realizando o pré-processamento e a parametrização dos dados.

2) Utilizar bibliotecas Python voltadas para aprendizado de máquina para desenvolver modelos de Convolutional Neural Network ou Rede Neural Convolucional (CNN) com diferentes abordagens e avaliar o desempenho individual de cada uma das propostas.

3) Desenvolver um sistema eletrônico microcontrolado para aquisição de sinais de ECG para avaliar a capacidade de predição da rede neural convolucional treinada.

1.2 ORGANIZAÇÃO DO DOCUMENTO

Este trabalho está organizado em 5 capítulos.

O capítulo 1 apresenta a introdução ao tema, destacando a importância do ECG no diagnóstico de arritmias, além da justificativa, objetivos e estrutura do trabalho.

O capítulo 2 aborda os fundamentos teóricos, incluindo o funcionamento dos sinais cardíacos, microcontroladores, amplificadores operacionais, e os conceitos de aprendizado de máquina e redes neurais convolucionais.

O capítulo 3 descreve a base de dados utilizada, detalhando suas características, organização e justificativas para sua escolha.

O capítulo 4 apresenta a metodologia, incluindo aquisição de sinais com o AD8232 e ESP32, etapas de pré-processamento, ambiente de desenvolvimento, bibliotecas utilizadas e o projeto da arquitetura convolucional.

O capítulo 5 traz as considerações finais, com análise dos resultados, limitações do estudo e sugestões para trabalhos futuros.

2 ESTUDO DE CASO

O capítulo introduz os conceitos fundamentais necessários para uma compreensão abrangente da pesquisa como um todo. Em seguida, discutem-se alguns trabalhos relevantes na área, com o intuito de fornecer uma base de referência para comparações ou auxiliar na escolha de abordagens e métodos utilizados neste estudo.

2.1 O CORAÇÃO

As paredes do coração são compostas pelas camadas epicárdio, miocárdio e endocárdio, e são delimitadas pela membrana pericárdica. O coração é dividido em quatro câmaras: duas superiores, os átrios direito e esquerdo, e duas inferiores, os ventrículos direito e esquerdo. A Figura 1 apresenta a estrutura do coração.

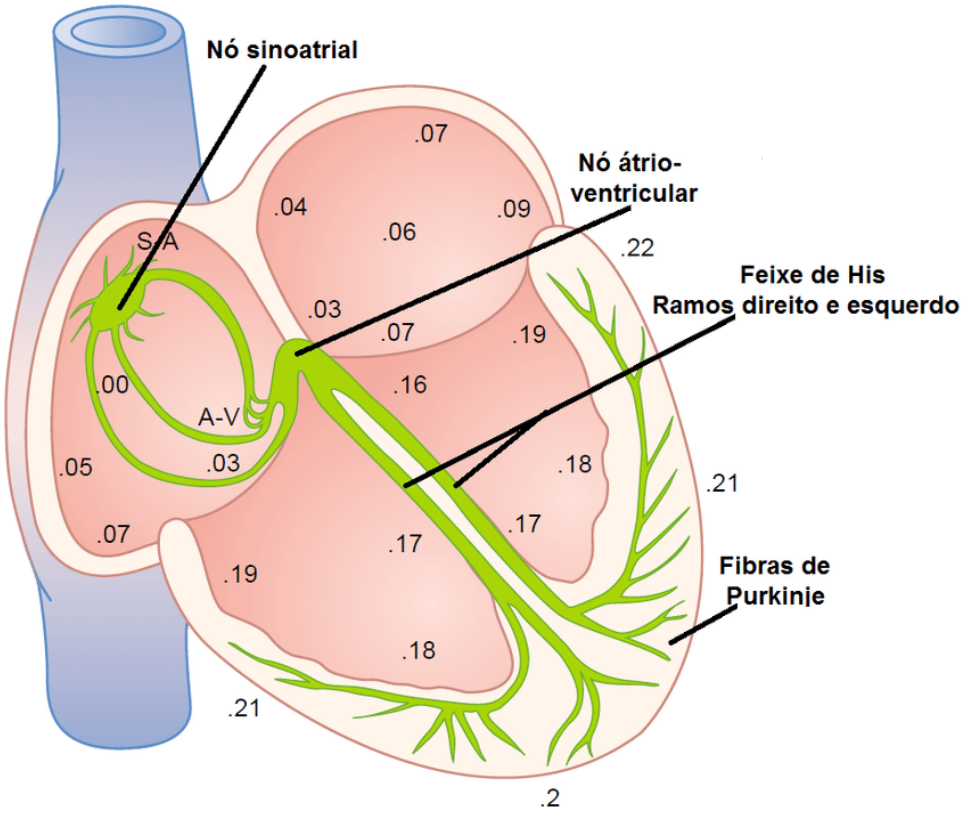


Figura 1 – Sistema de condução elétrica no coração e tempo de ativação (s).

Fonte: (CAMPOS, 2016)

Os átrios têm como função principal receber o sangue que retorna ao coração por meio de veias, enquanto os ventrículos são responsáveis por impulsionar o sangue para fora do coração através das artérias.

O átrio direito recebe o sangue venoso, caracterizado pelo alto teor de dióxido de carbono, e o direciona ao ventrículo direito. Este, por sua vez, bombeia o sangue em direção ao tronco pulmonar, permitindo que ele alcance os pulmões. Durante a passagem pelos pulmões, ocorre a liberação de dióxido de carbono (CO2) e a captação de oxigênio (O2). O sangue, agora rico em oxigênio, retorna ao coração pelas veias pulmonares e é recebido pelo átrio esquerdo. Posteriormente, o sangue oxigenado é transferido para o ventrículo esquerdo, que o bombeia para a aorta, distribuindo-o pelo corpo. Parte desse sangue alimenta o miocárdio, enquanto o restante é enviado para os demais tecidos do organismo (TORTORA; DERRICKSON, 2016).

O processo de bombeamento de sangue ocorre em duas fases distintas: a *sístole*, que é a contração do coração responsável por impulsionar o sangue, e a *diástole*, que representa o relaxamento do coração para permitir o retorno sanguíneo.

O ciclo cardíaco é controlado por estímulos elétricos gerados pelo sistema excitocondutor e transmitidos por células especializadas (OLIVEIRA, 2015). Essa condução elétrica ocorre devido à presença de cargas positivas e negativas nos meios intra e extracelular do tecido cardíaco, bem como à ação de mecanismos de transmissão denominados potenciais de ação. Conforme destacado por (CARDOSO, 2010), a atividade conjunta dessas células resulta nos biopotenciais observados no coração.

2.2 O ELETROCARDIOGRAMA

O eletrocardiograma (ECG) é um exame que registra graficamente a atividade elétrica do coração durante a contração e o relaxamento dos músculos cardíacos. Sua origem remonta a 1887, quando o fisiologista britânico Augustus Waller realizou o primeiro registro de um ECG humano, usando um eletrômetro capilar de Lippmann e eletrodos no tórax. Embora tenha demonstrado que os batimentos cardíacos geram atividade elétrica, Waller não acreditava em sua utilidade clínica devido às limitações tecnológicas da época.(GIFFONI; TORRES, 2010)

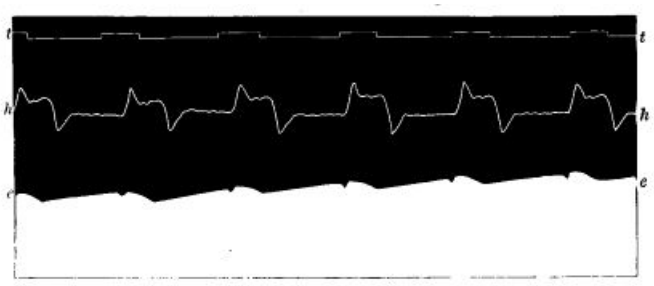


Figura 2 – Registro do primeiro ECG.

Fonte: Adaptado de Waller (1887).

3 FUNDAMENTOS TEÓRICOS PARA O PROJETO

3.1 AMPLIFICADOR OPERACIONAL E FUNCIONAMENTO DO AD8232

O amplificador operacional é um componente eletrônico que integra resistores, transistores, capacitores e outros elementos em um único encapsulamento. Na prática, trata-se de um amplificador com entrada diferencial e saída simples (TEIXEIRA, 2017). Uma característica essencial desses dispositivos é que são diretamente acoplados, funcionando como amplificadores de corrente contínua (CC), capazes de amplificar sinais de baixa frequência ou até mesmo sinais contínuos (SEDRA; SMITH, 2007).

Os amplificadores operacionais operam como sensores da diferença de tensão entre os seus dois terminais de entrada, ou seja, $(V_{e+} - V_{e-})$. Dessa forma, respondem apenas à diferença de sinal e ignoram qualquer componente comum às entradas. Por exemplo, se $V_{e+} = V_{e-} = 1\,\text{V}$, então, teoricamente, $V_{\text{out}} = 0$. Isso evidencia a alta rejeição de modo comum, característica idealizada desses componentes (SEDRA; SMITH, 2007). O modelo ideal apresenta ainda ganho infinito, impedância de entrada tendendo ao infinito e impedância de saída próxima de zero (NISE, 2013).

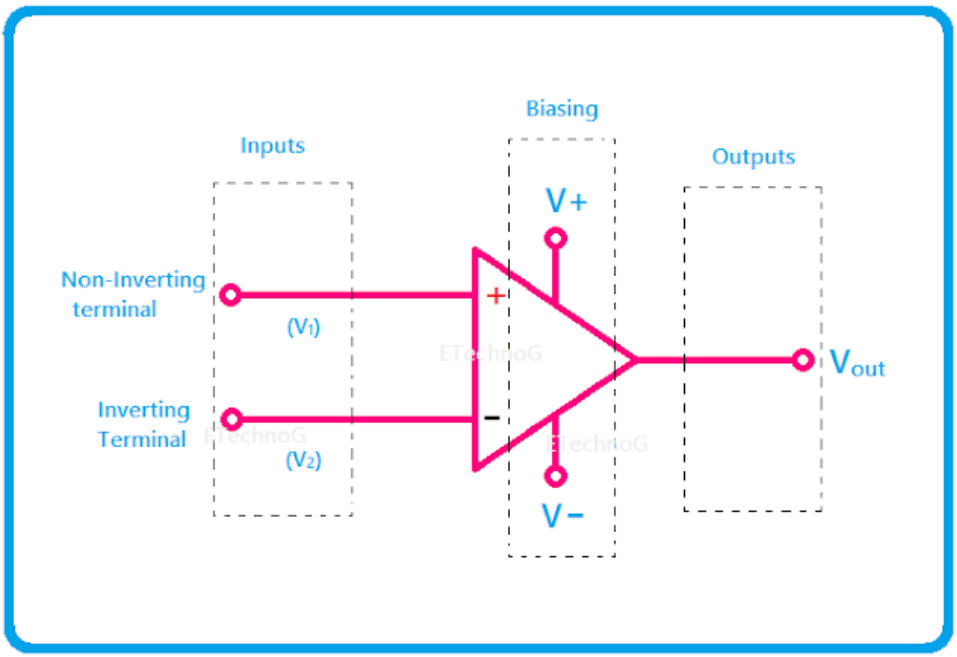


Figura 3 – Amplificador operacional.

Fonte: (ETECHNOG, 2019)

Esses conceitos são aplicados no monitor de sinais bioelétricos AD8232, um dispositivo integrado desenvolvido para a extração, amplificação e filtragem de sinais como o eletrocardiograma (ECG) e a frequência cardíaca (FC). Seu circuito é projetado para operar mesmo em

condições adversas, como ruídos gerados por movimentos ou pelo local de inserção dos eletrodos (DEVICES, 2020).

O AD8232 conta com os seguintes componentes internos:

- Um amplificador de instrumentação, responsável pela amplificação inicial do sinal bioelétrico.
- Um amplificador operacional (A1), que aplica ganho adicional e realiza a filtragem.
- Um amplificador RLD (Right Leg Drive) (A2), que reduz os ruídos de modo comum.
- Um buffer de referência (A3), permitindo a recuperação do sinal após uma eventual desconexão dos eletrodos (DEVICES, 2020).

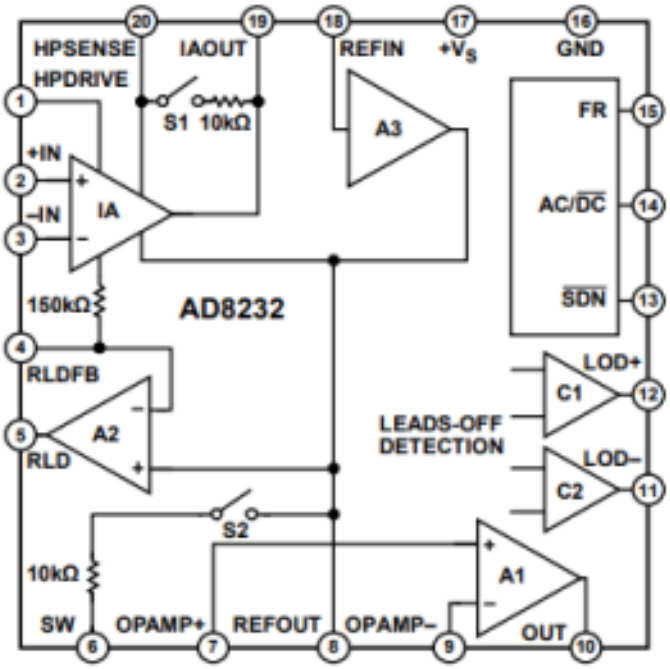


Figura 4 – Diagrama de blocos funcional do AD8232.
Adaptado de (DEVICES, 2020).

O princípio de funcionamento do AD8232 baseia-se em um amplificador operacional não inversor combinado com um filtro passa-baixa de três pólos na entrada. Essa configuração permite amplificar sinais na faixa de 0,5 a 40 Hz compatível com o espectro típico do ECG ao mesmo tempo que atenua ruídos de alta frequência, como os gerados por artefatos de movimento. O circuito conta ainda com um sistema de rejeição de modo comum (CMRR), essencial para preservar a integridade do sinal em ambientes ruidosos.

Dessa forma, o AD8232 desempenha um papel central no processo de aquisição dos sinais eletrocardiográficos, realizando de forma integrada as etapas de condicionamento e amplificação necessárias antes da digitalização e posterior análise.

3.2 MICROCONTROLADORES E SUAS APLICAÇÕES

Os microcontroladores são dispositivos eletrônicos compactos que integram, em um único chip, um processador (CPU), memória e periféricos de entrada e saída. Projetados para realizar tarefas específicas de controle em sistemas embarcados, esses componentes são amplamente utilizados em aplicações que vão desde dispositivos de consumo até sistemas industriais complexos (BARRETT; PACK, 2019). Eles oferecem vantagens significativas, tais como:

- Baixo custo e consumo de energia.
- Integração de diversos periféricos em um único chip.
- Facilidade de programação e reprogramação.
- Compactação, permitindo sua aplicação em dispositivos portáteis.

Esses dispositivos são empregados em diversas plataformas e contextos. O Arduino, por exemplo, é amplamente utilizado em projetos educacionais e de prototipagem, permitindo a integração com sensores, motores e outros dispositivos, sendo programado por meio da Arduino IDE (BANZI; SHILOH, 2022). O ESP destaca-se por sua capacidade de comunicação Wi-Fi e Bluetooth, sendo muito utilizado em projetos de Internet das Coisas (Internet das Coisas (Internet of Things) (IoT)) (SYSTEMS, 2022). Já os microcontroladores PIC são reconhecidos por sua confiabilidade e ampla gama de periféricos, sendo comuns em aplicações industriais e automotivas (BARRETT; PACK, 2019). Os modelos STM são empregados em sistemas embarcados de alto desempenho, como drones, controladores de motores BLDC e dispositivos médicos. Por fim, o Raspberry oferece elevado desempenho com suporte para múltiplos periféricos, sendo ideal para aplicações que demandam paralelismo.

Devido à sua versatilidade, os microcontroladores são encontrados em uma ampla variedade de aplicações. Na automação residencial, são utilizados no controle de iluminação, termostatos inteligentes e assistentes virtuais, como a Alexa. Na indústria automotiva, são empregados em sensores de estacionamento, controle de injeção eletrônica e sistemas de freios ABS. Em sistemas médicos, possibilitam o monitoramento cardíaco, controle de dispositivos de administração de insulina e operação de equipamentos de diagnóstico por imagem. Na área de robótica, atuam no controle de motores, sensores de proximidade e câmeras de visão computacional. Além disso, desempenham papel fundamental na Internet das Coisas (IoT), sendo utilizados em dispositivos conectados que monitoram e controlam remotamente ambientes e sistemas, como câmeras de segurança e sensores ambientais.

3.3 ESP32

O ESP32 é um microcontrolador de alto desempenho, que integra conectividade Wi-Fi e Bluetooth em um único chip. Projetado para aplicações que exigem processamento eficiente e conectividade sem fio, o ESP32 é amplamente utilizado em projetos de sistemas embarcados, automação residencial e dispositivos de Internet das Coisas (IoT). Sua versatilidade e alta capacidade de processamento o tornam ideal para aplicações onde desempenho e conectividade são requisitos essenciais.

Devido às suas características, o ESP32 é amplamente utilizado em diversos cenários, incluindo:

- Prototipagem de sistemas embarcados com conectividade.
- Automação residencial e industrial.
- Projetos educacionais de robótica e computação física.
- Monitoramento de sensores em dispositivos portáteis.
- Dispositivos IoT com transmissão de dados em tempo real.

O ESP32 apresenta vantagens significativas, como conectividade Wi-Fi e Bluetooth integradas, baixo consumo de energia com modos de economia, suporte a múltiplos periféricos e interfaces, além de uma comunidade ativa de desenvolvedores. Sua compatibilidade com o Arduino IDE e outros ambientes de desenvolvimento, como o ESP-IDF, simplifica a criação de projetos, permitindo que seja utilizado tanto por iniciantes quanto por especialistas.

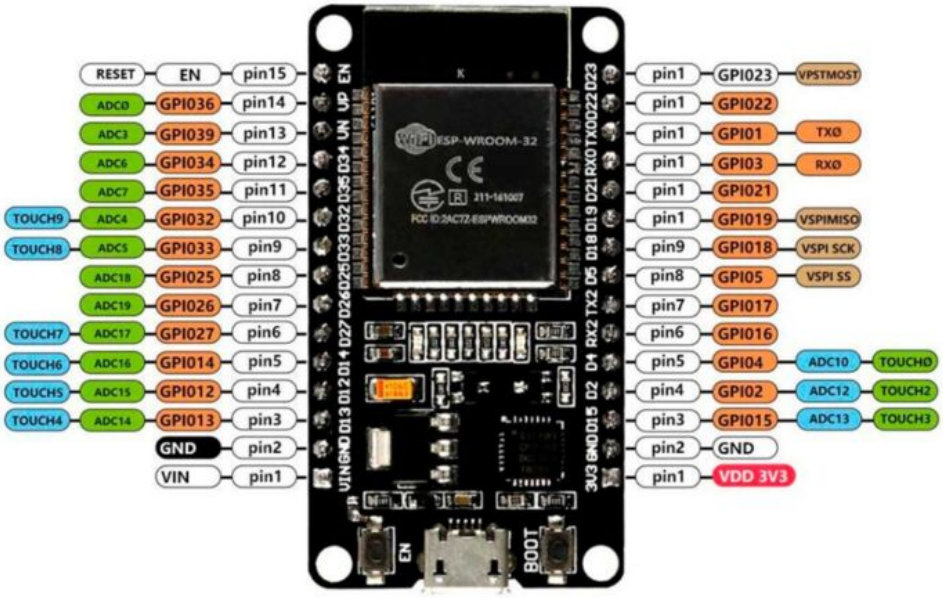


Figura 5 – ESP32 - Placa de microcontrolador compacta.

3.4 APRENDIZADO DE MÁQUINA E APRENDIZADO PROFUNDO

O aprendizado de máquina (*Machine Learning*) é um ramo da inteligência artificial que se concentra em sistemas capazes de realizar tarefas sem a necessidade de uma programação explícita. Em outras palavras, trata-se de sistemas que aprendem normalmente a partir de dados (KOROL, 2019). Um dos principais métodos utilizados é a inferência indutiva, que permite a produção de resultados confiáveis e a previsão de eventos. No entanto, a capacidade de generalização desses sistemas pode ser prejudicada em casos de escassez de dados, baixa qualidade ou falta de representatividade em relação ao que se deseja aprender.

Os algoritmos de aprendizado de máquina podem ser divididos, basicamente, em três tipos:

- Supervisionado: Nesse método, é necessária uma base de dados (*dataset*) de treinamento que fornece exemplos ao algoritmo, acompanhados de rótulos que informam a classe à qual o dado pertence. Após o treinamento, espera-se que o algoritmo consiga inferir rótulos para dados que eram previamente desconhecidos.

- Não supervisionado: Nesse caso, os dados são fornecidos sem rótulos, cabendo ao algoritmo agrupá-los com base em suas similaridades. Não há a necessidade de um *dataset* para treinamento, pois o aprendizado ocorre de forma autônoma.
- Por reforço: Aqui, o algoritmo não recebe a resposta correta, mas sim sinais de reforço, como recompensas ou punições, a partir de suas interações. Esse método é semelhante ao aprendizado supervisionado por utilizar experiências anteriores, e ao aprendizado não supervisionado por aprender de forma independente (LUDERMIR, 2021).

Como este trabalho se fundamenta no aprendizado supervisionado para a classificação de arritmias, esse método será explorado em maior profundidade nos tópicos seguintes.

Dentro do campo do aprendizado de máquina, destaca-se o aprendizado profundo (*Deep Learning*), uma subárea que busca extrair conceitos complexos de maneira hierárquica a partir de dados brutos menos complexos. Trata-se de uma técnica que tenta imitar o raciocínio utilizado pelo cérebro humano para aprender e identificar padrões (ASSIS, 2019).

Para atingir esse objetivo, esse tipo de modelo é estruturado em múltiplas camadas. Cada camada compreende uma característica diferente do objeto em análise, sendo composta por diversos nós que transformam os dados em informações. Essas informações são então repassadas para a próxima camada, em um processo iterativo que culmina na previsão ou categorização final (SCHONS, 2018).

3.5 REDES NEURAIS E ARQUITETURA CONVOLUCIONAL

O Perceptron, desenvolvido por McCullock e Pitts (MCCULLOCH; PITTS, 1943), é um modelo matemático que visa imitar o funcionamento de um neurônio biológico. Esse neurônio artificial possui conexões chamadas sinapses, cada uma associada a um peso específico que determina a influência de cada sinal de entrada. Os sinais recebidos são ponderados, somados e processados por uma função de ativação, que limita a amplitude da saída do neurônio a um intervalo finito (HAYKIN, 1998).

Por meio da disposição de múltiplos neurônios em camadas paralelas, é possível construir uma rede neural artificial capaz de aprender padrões a partir de dados, ajustando iterativamente os pesos sinápticos. A Figura 6 apresenta uma representação esquemática de um neurônio artificial, destacando seus principais componentes.

A saída de um neurônio pode ser modelada matematicamente por:

$$v_k = \sum_{j=0}^{m} w_{kj} x_j, \quad y_k = \phi(v_k)$$

em que x_j representa a entrada, w_{kj} o peso sináptico, v_k a saída da combinação linear e y_k a saída do neurônio após a aplicação da função de ativação ϕ. O viés (*bias*) b_k é adicionado ao somatório para ajustar o limiar da função de ativação.

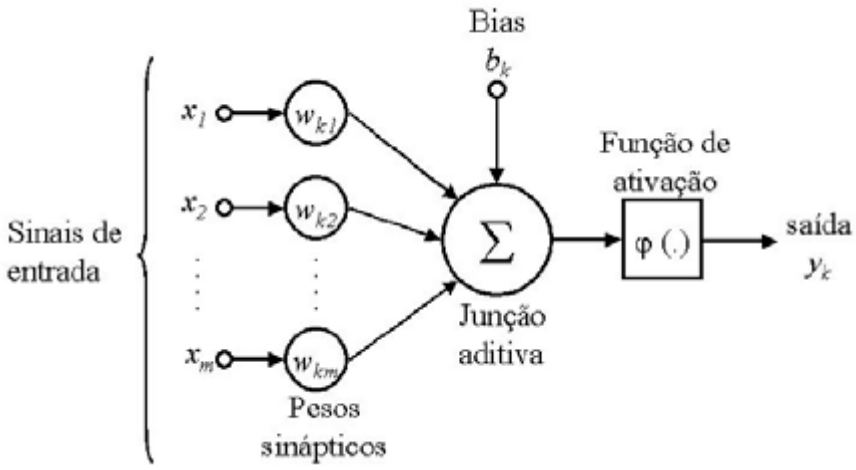


Figura 6 – Neurônio artificial.

Fonte: Adaptado de Haykin (1998).

Durante o treinamento da rede, os pesos w_{kj} e os vieses são ajustados com o objetivo de minimizar o erro entre a saída prevista e a saída desejada. Isso é feito por algoritmos de otimização, como o gradiente descendente, em conjunto com a retropropagação do erro (*back-propagation*).

Dentre os diversos tipos de redes neurais, destacam-se as redes neurais convolucionais (CNNs), projetadas para identificar padrões espaciais em dados estruturados, como imagens e sinais temporais. As CNNs operam por meio da aplicação de filtros (*kernels*) que varrem regiões locais do dado de entrada, extraindo características relevantes em representações hierárquicas cada vez mais abstratas.

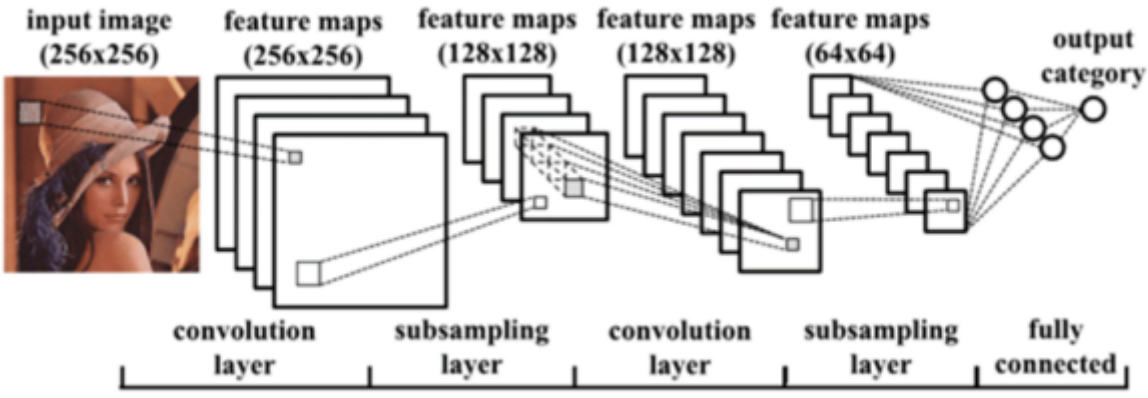


Figura 7 – Rede neural convolucional aplicada a imagens.

Fonte: Adaptado de Data Science Academy (2022).

A arquitetura de uma CNN é organizada em dois módulos principais: o módulo de extração de características e o módulo de classificação. O primeiro é composto por camadas convolucionais e de agrupamento (*pooling*), responsáveis por mapear e reduzir a dimensionalidade

das informações extraídas. Já o segundo módulo é formado por camadas totalmente conectadas, que utilizam os vetores extraídos para inferir a classe à qual o dado pertence.

As principais camadas de uma CNN são:

- *Camada convolucional*: aplica filtros convolucionais que extraem padrões locais do dado de entrada, como bordas, picos e formas.
- *Camada de agrupamento* (*pooling*): reduz a dimensionalidade dos mapas de ativação, retendo as informações mais relevantes e diminuindo o custo computacional.
- *Camada totalmente conectada*: realiza a classificação final com base nas características extraídas, conectando todos os neurônios da camada anterior a cada neurônio de saída.

A Figura 8 ilustra de forma esquemática o fluxo entre as principais camadas de uma CNN, desde a entrada até a saída final.

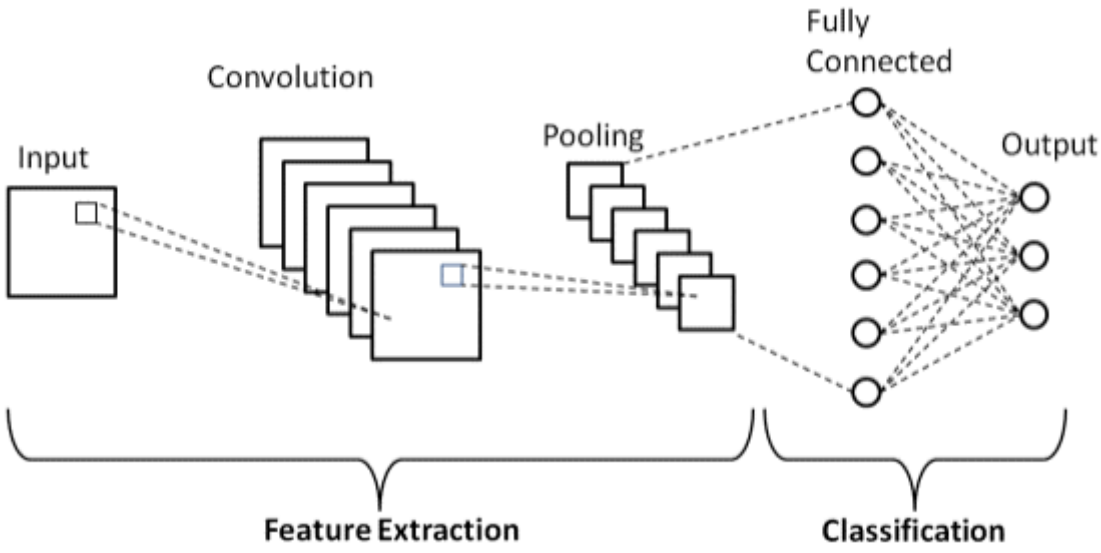


Figura 8 – Principais camadas de uma rede neural convolucional.

Fonte: Adaptado de Gurucharan (2022).

Esse arranjo de camadas permite que a rede aprenda representações robustas e discriminativas, sendo particularmente eficaz na análise de sinais eletrocardiográficos, que possuem padrões morfológicos recorrentes e localizados, como os complexos QRS.

3.6 CONJUNTO DE DADOS

O *Leipzig Heart Center ECG-Database: Arrhythmias in Children and Patients with Congenital Heart Disease* é um material de referência disponibilizado publicamente para a comunidade de pesquisa, com o objetivo de estimular estudos na área de detecção e classificação de arritmias cardíacas. Este conjunto de dados promove avaliações reprodutíveis e comparáveis de diferentes algoritmos, arquiteturas e métodos de processamento de sinais eletrocardiográficos.

O conjunto de dados contém registros de ECG adquiridos de pacientes pediátricos e adultos com cardiopatias congênitas, abrangendo mais de 113 mil batimentos anotados. Esses registros foram obtidos durante estudos eletrofisiológicos realizados no Leipzig Heart Center, utilizando o sistema CardioLab®, com taxa de amostragem de 977 Hz e sem aplicação de filtros pós-processamento.

Os dados estão organizados em dois grupos principais:

- Crianças, correspondentes aos arquivos `x001` a `x029`, contendo registros de pacientes pediátricos com diferentes tipos de cardiopatias congênitas.
- Adultos, correspondentes aos arquivos `x100` a `x109`, contendo registros de adultos com cardiopatias congênitas.

Além dos sinais de ECG, o conjunto de dados fornece informações complementares, como idade, sexo, diagnóstico clínico, duração das gravações e localização de vias acessórias. A anotação dos batimentos foi realizada manualmente por especialistas, categorizando-os em diferentes tipos de arritmias, incluindo taquicardias supraventriculares, taquicardias ventriculares, batimentos ectópicos e ritmos estimulados artificialmente.

Essas características fazem do *Leipzig Heart Center ECG-Database* um recurso valioso para o treinamento e validação de modelos de inteligência artificial, especialmente voltados para a detecção de arritmias em populações frequentemente sub-representadas em bases tradicionais.

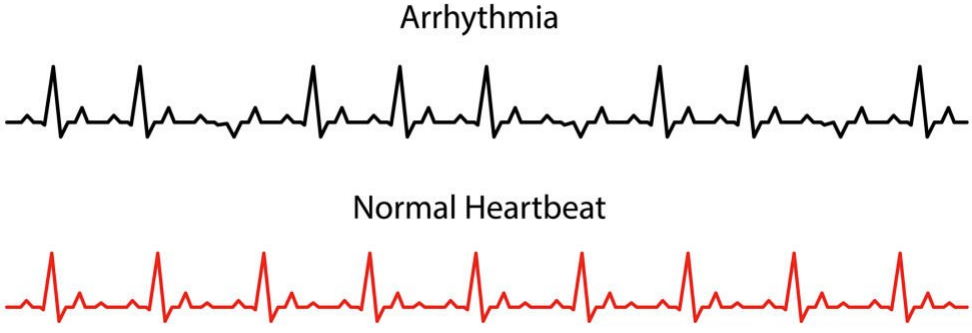
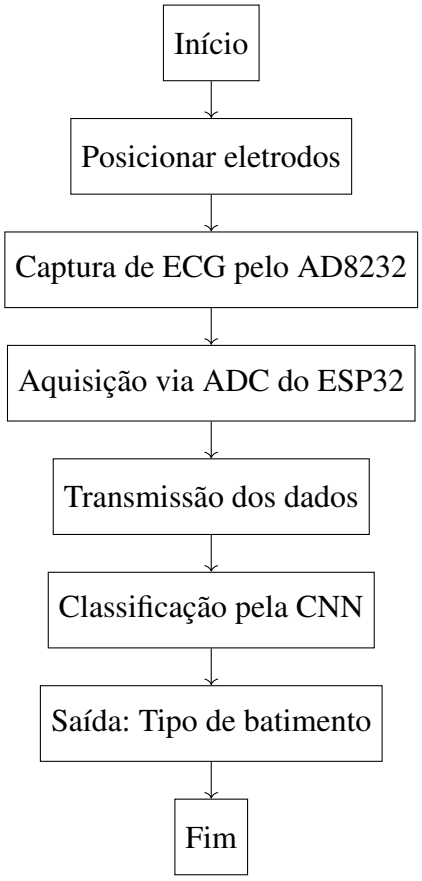


Figura 9 – Ilustração vetorial (EPS).

Fonte: Ivanova (2021).

3.7 FLUXOGRAMA DO FUNCIONAMENTO

A seguir, é apresentado um fluxograma que descreve de forma simplificada as etapas envolvidas no funcionamento do sistema de aquisição e classificação de sinais eletrocardiográficos (ECG). Desde a colocação dos eletrodos até a identificação do tipo de batimento cardíaco, cada fase do processo é representada de maneira sequencial, evidenciando o fluxo de dados e as operações realizadas pelo sistema embarcado e pela rede neural convolucional.



NOTAS TÉCNICAS

- **Quantidade de eletrodos**: 3 (positivo, negativo e referência).
- **Entradas analógicas do microcontrolador**: como GPIO34
- **Resolução do ADC do ESP32**: 12 bits (0 a 4095 níveis).
- **Resolução adequada?** Sim. O AD8232 amplifica os sinais (em torno de 1–5 mV) para faixa de 0–3.3V ou 5V, compatível com o ADC do ESP32.

4 IMPLEMENTAÇÃO DE MODELO PARA CLASSIFICAÇÃO DE ARRITMIAS EM ECG USANDO REDES NEURAIS CONVOLUCIONAIS

Este capítulo trata da implementação de um sistema de detecção automática de arritmias cardíacas a partir da análise de sinais de eletrocardiograma (ECG), utilizando Redes Neurais Convolucionais (CNNs). São abordadas as tecnologias envolvidas na aquisição dos sinais, como sensores, amplificadores e microcontroladores, bem como os métodos de pré-processamento e classificação aplicados no domínio da inteligência artificial.

O eletrocardiograma é um exame amplamente utilizado na medicina para monitorar a atividade elétrica do coração. Por meio da captação de sinais bioelétricos gerados durante o ciclo cardíaco, é possível identificar padrões que indicam o funcionamento normal ou anormal do miocárdio. Esses sinais, contudo, podem ser afetados por diversos fatores, como ruídos elétricos, movimentos corporais e variações individuais entre pacientes. A análise manual dos registros de ECG, embora precisa, torna-se inviável em contextos de grande volume de dados, como na utilização de dispositivos vestíveis e monitores contínuos. Nesse contexto, surgem as redes neurais convolucionais como uma alternativa promissora para automatizar o processo de interpretação e classificação dos batimentos.

Redes Neurais Convolucionais são arquiteturas de aprendizado profundo que se destacam na extração automática de características relevantes em sinais ou imagens. Ao aplicar filtros convolucionais sobre os dados de entrada, essas redes conseguem identificar padrões morfológicos característicos dos batimentos cardíacos, como complexos QRS, ondas P e T. A aplicação de CNNs em sinais de ECG permite não apenas a detecção de anomalias, como também a classificação precisa dos tipos de arritmia, reduzindo erros humanos e aumentando a escalabilidade do diagnóstico.

O sistema proposto neste trabalho visa classificar os batimentos cardíacos em três categorias: batimentos normais, batimentos ectópicos ventriculares e batimentos ectópicos supraventriculares. Para isso, foi utilizada uma base de dados pública amplamente reconhecida, o *Leipzig Heart Center ECG-Database*, que contém registros anotados por especialistas, viabilizando o treinamento supervisionado dos modelos.

Além disso, foi desenvolvido um sistema físico para aquisição de sinais reais, composto por um módulo AD8232 e um microcontrolador ESP32. Essa estrutura permite a coleta de sinais em tempo real para posterior avaliação da capacidade de generalização do modelo treinado.

A utilização de CNNs neste projeto oferece vantagens significativas em relação a métodos tradicionais de classificação, especialmente por dispensar a extração manual de atributos. No entanto, o desenvolvimento de uma solução eficaz exige atenção a diversos desafios téc-

nicos, como o balanceamento das classes, a padronização dos sinais, a segmentação adequada dos batimentos e a mitigação de ruídos. Para contornar tais obstáculos, o modelo desenvolvido passou por diversas etapas de ajuste, envolvendo normalização dos dados, segmentação com base no intervalo R-R e validação cruzada para avaliação da performance.

Vale destacar que, embora os benefícios do uso de inteligência artificial na cardiologia sejam expressivos, ainda existem barreiras para a sua aplicação clínica. Questões como validação regulamentar, variações fisiológicas entre pacientes e qualidade do sinal captado em ambientes não controlados precisam ser cuidadosamente consideradas. Além disso, os sistemas de aquisição, como o utilizado neste projeto, devem seguir padrões de segurança elétrica e compatibilidade biológica, de acordo com normativas de agências reguladoras, como a ANVISA.

Neste capítulo, são detalhadas todas as etapas de implementação do modelo: desde a aquisição dos sinais com o sistema embarcado até o treinamento, validação e testes da rede neural convolucional. Também são discutidos os resultados obtidos, as limitações observadas e sugestões para aprimoramentos futuros, com vistas à evolução da proposta para uma eventual aplicação prática em contextos clínicos.

4.1 LINGUAGEM, BIBLIOTECAS E AMBIENTE DE DESENVOLVIMENTO

O projeto foi desenvolvido utilizando a linguagem de programação Python, devido à sua ampla adoção na comunidade científica e de aprendizado de máquina, bem como pela variedade de bibliotecas otimizadas para manipulação de dados, construção de modelos e visualização de resultados. A sintaxe clara e objetiva do Python também favorece a prototipação rápida e a depuração eficiente.

Para o pré-processamento dos sinais e manipulação dos dados, foram utilizadas as bibliotecas NumPy, SciPy, WFDB e Pandas. As etapas de filtragem digital (remoção de 60 Hz e passa-faixa de 0,5–40 Hz) foram implementadas com filtros IIR da biblioteca *SciPy.signal*. A normalização, remoção da média e segmentação em janelas centradas nos picos R foram realizadas com rotinas personalizadas baseadas em NumPy.

A construção do modelo de rede neural convolucional foi realizada com a biblioteca PyTorch, escolhida por sua flexibilidade na definição de arquiteturas, suporte nativo a GPU, integração com ferramentas de monitoramento como o TensorBoard, além de permitir a exportação dos modelos para execução em dispositivos embarcados, por meio do formato TorchScript.

Para a visualização dos sinais e dos resultados obtidos, foram empregadas as bibliotecas Matplotlib e Seaborn, amplamente utilizadas na criação de gráficos científicos de alta qualidade. Foram gerados gráficos como espectros de frequência, sinais no domínio do tempo, histogramas de classes e matrizes de confusão, que auxiliaram na interpretação dos dados e avaliação do desempenho do modelo.

O ambiente de desenvolvimento principal adotado foi o JupyterLab, devido à sua interatividade e praticidade para organização de análises experimentais. O código-fonte foi estruturado em notebooks, permitindo a integração entre código executável, equações matemáticas, visualizações gráficas e descrição textual dos experimentos. Essa abordagem facilitou o registro detalhado de cada etapa, garantindo reprodutibilidade e transparência na condução das análises.

Para a aquisição dos sinais reais utilizando o sensor AD8232, foi empregado o ambiente Visual Studio Code com suporte à linguagem C++, direcionado ao desenvolvimento embarcado no microcontrolador ESP32. A comunicação serial foi configurada para transmitir os dados em tempo real a uma taxa de amostragem de 977 Hz, com posterior armazenamento em arquivos de texto para processamento em Python.

A escolha desse conjunto de ferramentas teve como objetivo otimizar o fluxo de trabalho entre as etapas de aquisição, análise e modelagem dos sinais, assegurando eficiência computacional, rastreabilidade dos experimentos e possibilidade de expansão futura para aplicações em ambientes embarcados e dispositivos vestíveis de monitoramento cardíaco.

4.2 MÉTODO

A pesquisa adota uma abordagem de natureza exploratória. A pesquisa exploratória representa uma etapa preliminar essencial no início de um processo de investigação científica. Tem por escopo promover uma compreensão inicial e ampla de um determinado tema ou fenômeno, visando identificar lacunas de conhecimento, padrões, tendências e variáveis relevantes que demandam uma análise mais aprofundada. Este estágio é de suma importância para delimitar o escopo e a trajetória da pesquisa subsequente. A utilização de métodos como revisão bibliográfica, entrevistas e observação, aliada à análise de dados preexistentes, contribui para o levantamento de informações e insights fundamentais que direcionarão a formulação de hipóteses e o delineamento da pesquisa de forma mais detalhada e conclusiva.

Ao se falar de modelos de inteligência artificial (IAs) múltiplas métricas podem ser utilizadas na avaliação da performance. As principais consistem em: acurácia, precisão, erro médio absoluto e quadrático. A acurácia é uma métrica amplamente usada para avaliar proporção de instâncias corretamente classificadas em relação ao número total de instâncias. A precisão mede a proporção de previsões positivas corretas em relação ao número total de previsões positivas. Essas métricas são calculadas a partir da matriz de confusão, que permite visualizar os verdadeiros positivos, falsos positivos, verdadeiros negativos e falsos negativos, proporcionando uma visão do desempenho do modelo classificador.

O erro médio absoluto percentual (Mean Absolute Percentage Error – MAPE) é uma métrica de precisão que calcula a média das diferenças absolutas entre as previsões e os valores reais, expressa como uma porcentagem do valor real. O erro quadrático médio da raiz (Root Mean Square Error – RMSE) é uma métrica de avaliação de modelos que estima a média das

diferenças entre previsões e valores reais, com os erros quadrados sendo somados, extraindo-se a raiz quadrada. Estas métricas fornecem uma medida da dispersão dos erros.

As seções a seguir descrevem o desenvolvimento de cada uma destas etapas.

4.3 BASE DE DADOS UTILIZADA

Neste trabalho, foi utilizada a base de dados *Leipzig Heart Center ECG-Database: Arrhythmias in Children and Patients with Congenital Heart Disease*, publicada em março de 2025 na versão 1.0.0 (KLEHS *et al.*, 2025). Essa base, disponibilizada pela plataforma PhysioNet, é amplamente reconhecida por fornecer registros fisiológicos de acesso aberto, com foco especial em populações pediátricas e pacientes com cardiopatias congênitas.

O conjunto de dados inclui registros de ECG adquiridos durante estudos eletrofisiológicos com o sistema CardioLab®, contendo anotações manuais feitas por especialistas em diferentes tipos de arritmias cardíacas, como taquicardias supraventriculares (AVRT e AVNRT), taquicardias ventriculares, batimentos ectópicos e ritmos estimulados artificialmente. Os sinais foram convertidos para o formato WFDB, com taxa de amostragem de 977 Hz e sem aplicação de filtros de pós-processamento.

Além dos sinais de ECG, a base fornece arquivos complementares com informações demográficas dos pacientes (idade e sexo), diagnósticos clínicos, duração das gravações e localização de vias acessórias. Os registros estão organizados em dois grupos principais: crianças (arquivos `x001` a `x029`) e adultos com cardiopatias congênitas (arquivos `x100` a `x109`). No total, a base inclui mais de 113 mil batimentos anotados, categorizados em ritmos normais, batimentos pré-excitados, bloqueios de ramo, taquicardias, batimentos prematuros e batidas com marcapasso.

A Tabela 1 apresenta uma descrição das principais anotações encontradas no conjunto de dados.

A diversidade morfológica dos sinais e a qualidade das anotações fazem do *Leipzig Heart Center ECG-Database* um recurso valioso para a construção e validação de modelos de inteligência artificial voltados à detecção automática de arritmias cardíacas. Sua utilização contribuiu significativamente para promover a robustez e a capacidade de generalização do modelo desenvolvido neste trabalho. A documentação oficial está disponível no endereço: <https://doi.org/10.13026/7a4j-vn37>.

Tabela 1 – Anotações do dataset Leipzig Heart Center ECG-Database

Descrição da Batida/Ritmo	Símbolo	String Auxiliar (Aux String)
Ritmo Sinusal		
Normal	N ou •	–
Pré-excitação	N	N-Prex
Bloqueio completo do ramo direito	R	–
Bloqueio completo do ramo esquerdo	L	–
Bloqueio AV de 1º grau	b	BI
Batidas de escape juncionais	j	–
Taquicardias		
Taquicardia ventricular	X	VT
Ritmo idioventricular acelerado	X	IVR
Taquicardia supraventricular		
Taquicardia por reentrada AV	X	AVRT
Taquicardia nodal por reentrada AV	X	AVNRT
AVRT aberrante	X	avrt
AVNRT aberrante	X	avnrt
AVNRT com bloqueio AV de 2º grau	X	AVNRT+BII
Taquicardia atrial		
Fibrilação atrial	X	AFIB
Taquicardia atrial ectópica	X	EAT
Flutter atrial	X	AFL
Batidas Prematuras		
Batidas atriais prematuras	A	–
Batidas atriais prematuras aberrantes	a	–
Batidas atriais prematuras pré-excitadas	A	A-Prex
Batidas ventriculares prematuras	V	–
Batidas de fusão	F	–
Batidas juncionais prematuras	J	–
Batidas com Marcapasso		
Batidas atriais estimuladas	/	/A
Batidas ventriculares estimuladas	/	/V
Batida de fusão (ventricular estimulada e normal)	f	–
Ritmos		
Ritmo sinusal	+	(N
Taquicardia ventricular	+	(VT
Ritmo idioventricular acelerado	+	(IVR
Taquicardia supraventricular		
Taquicardia por reentrada AV	+	(AVRT
Taquicardia nodal por reentrada AV	+	(AVNRT
Taquicardia atrial		
Fibrilação atrial	+	(AFIB
Taquicardia atrial ectópica	+	(EAT
Flutter atrial	+	(AFL
Ritmo ectópico		
Ritmo atrial ectópico	+	(A
Bigeminismo ventricular	+	(B
Ritmo juncional	+	(J
Batidas com marcapasso		
Ritmo atrial estimulado	+	(/A
Ritmo ventricular estimulado	+	(/V

4.4 ELETRODOS NA AQUISIÇÃO DE SINAIS DE ECG

Um dos elementos fundamentais deste trabalho são os eletrodos, responsáveis pelo contato inicial com o dado que será conduzido pelos estágios de filtragem do AD8232. Para garantir uma aquisição eficiente do sinal, é essencial estabelecer uma conexão adequada entre o corpo do paciente e o eletrodo, utilizando um gel condutor.

De acordo com o Ministério da Saúde (Ministério da Saúde, 2002), os eletrodos de ECG são geralmente constituídos de prata clorada, e o contato elétrico com a pele é otimizado pelo uso de um gel eletrolítico à base de cloro. Há diferentes tipos de eletrodos disponíveis, incluindo:

- **Eletrodos de sucção**: Utilizados para contato direto no tórax;
- **Eletrodos de placa**: Destinados às extremidades do corpo;
- **Eletrodos descartáveis ou adesivos**: Amplamente utilizados em diagnósticos de esforço, monitoramentos prolongados, como em Unidades de Terapia Intensiva (UTIs), e situações de emergência.

O processo primário de um sistema de aquisição de ECG inicia-se com a leitura realizada pelos eletrodos, que conduzem a corrente elétrica gerada pelo corpo humano através de seus condutores até a placa de circuito, onde o sinal será filtrado pelo módulo AD8232.

Após a captação inicial, o sinal elétrico cardíaco é amplificado, e um processo de rejeição de artefatos e ruídos, tanto biológicos quanto ambientais, é conduzido utilizando um amplificador diferencial. Em um estágio posterior, o sinal amplificado passa por ajustes na resposta em frequência, tornando-o adequado para visualização, processamento ou transmissão para análises posteriores (Ministério da Saúde, 2002).

4.5 COLETA E PRÉ-PROCESSAMENTO DOS SINAIS

Para a aquisição do sinal eletrocardiográfico (ECG), foi desenvolvido um firmware embarcado utilizando a plataforma *ESP32*. O código foi escrito na linguagem C++ (compatível com o ESP32) e possui como função principal coletar dados analógicos provenientes do sinal cardíaco por um período de tempo definido, armazená-los temporariamente na memória volátil (*preferencialmente na PSRAM*) e transmiti-los via interface serial para posterior análise e processamento.

A Figura 13 apresenta o **esquema completo de ligação** do circuito e o sistema de aquisição é composto pelos seguintes elementos principais:

- **Módulo AD8232**: responsável pela amplificação e filtragem inicial do sinal eletrocardiográfico. O sinal amplificado é disponibilizado na saída `OUTPUT`.

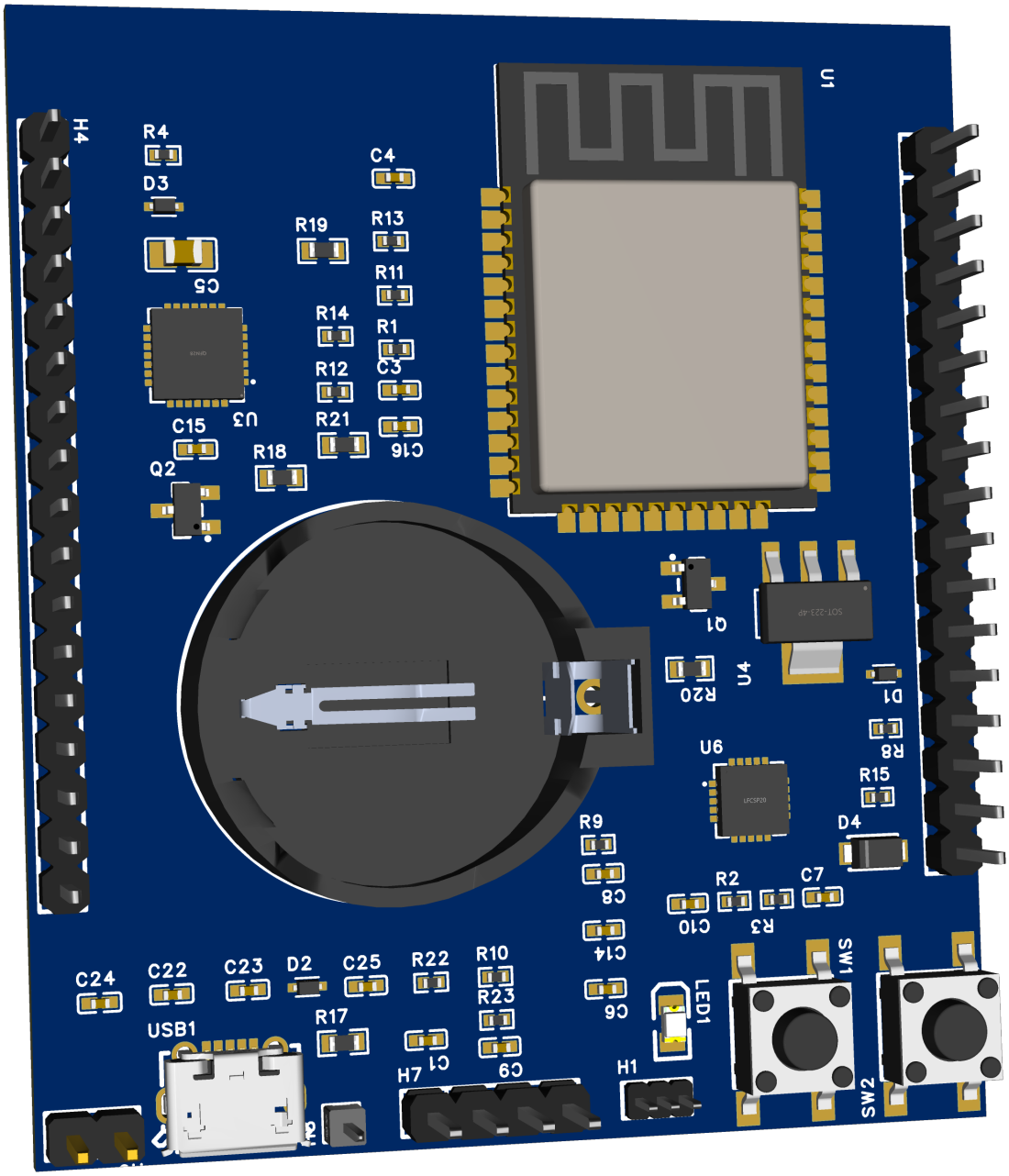


Figura 10 – Protótipo placa de circuito impresso (PCB).

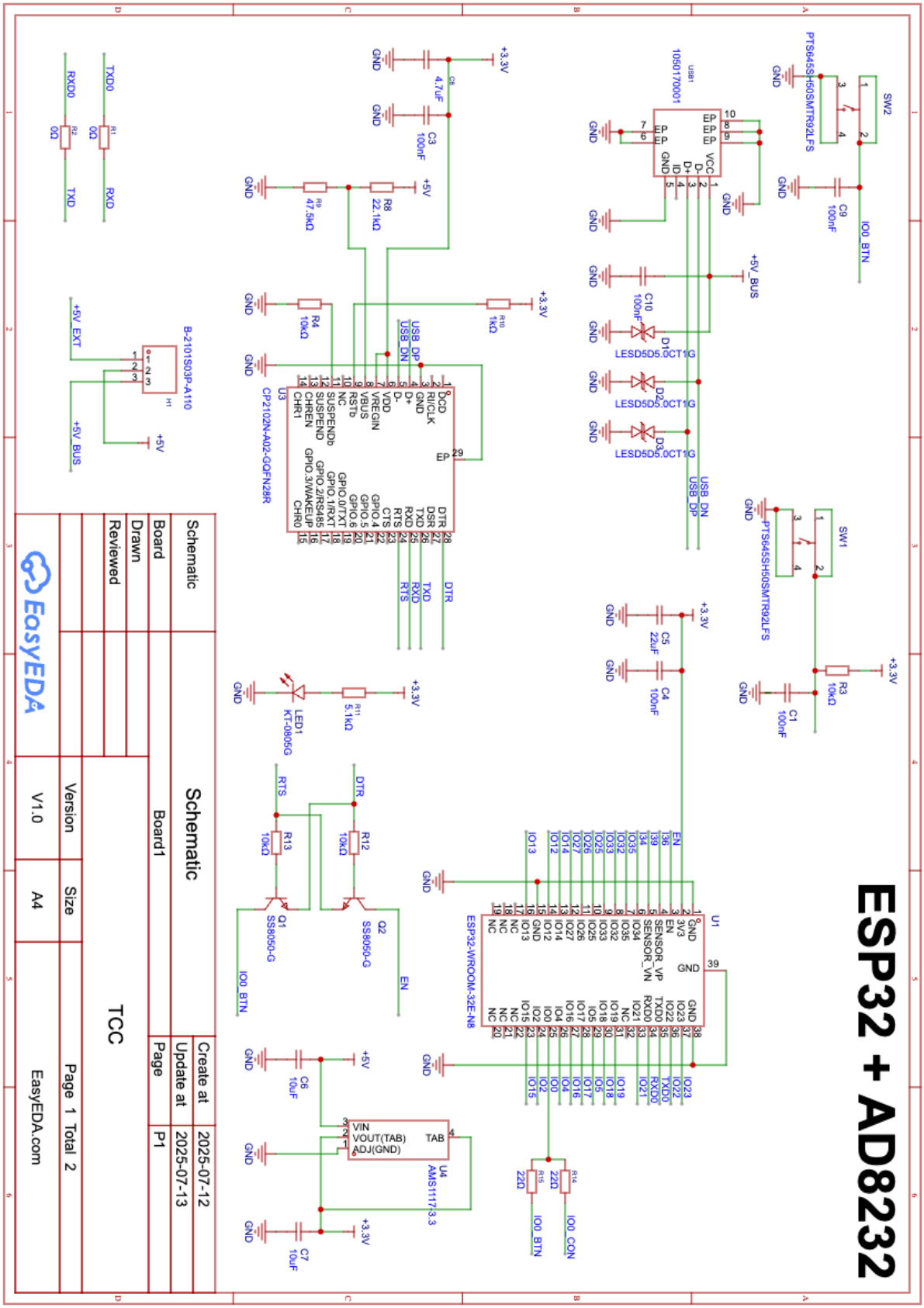


Figura 11 – Esquemático do circuito proposto com ESP32 e AD8232, parte 1.

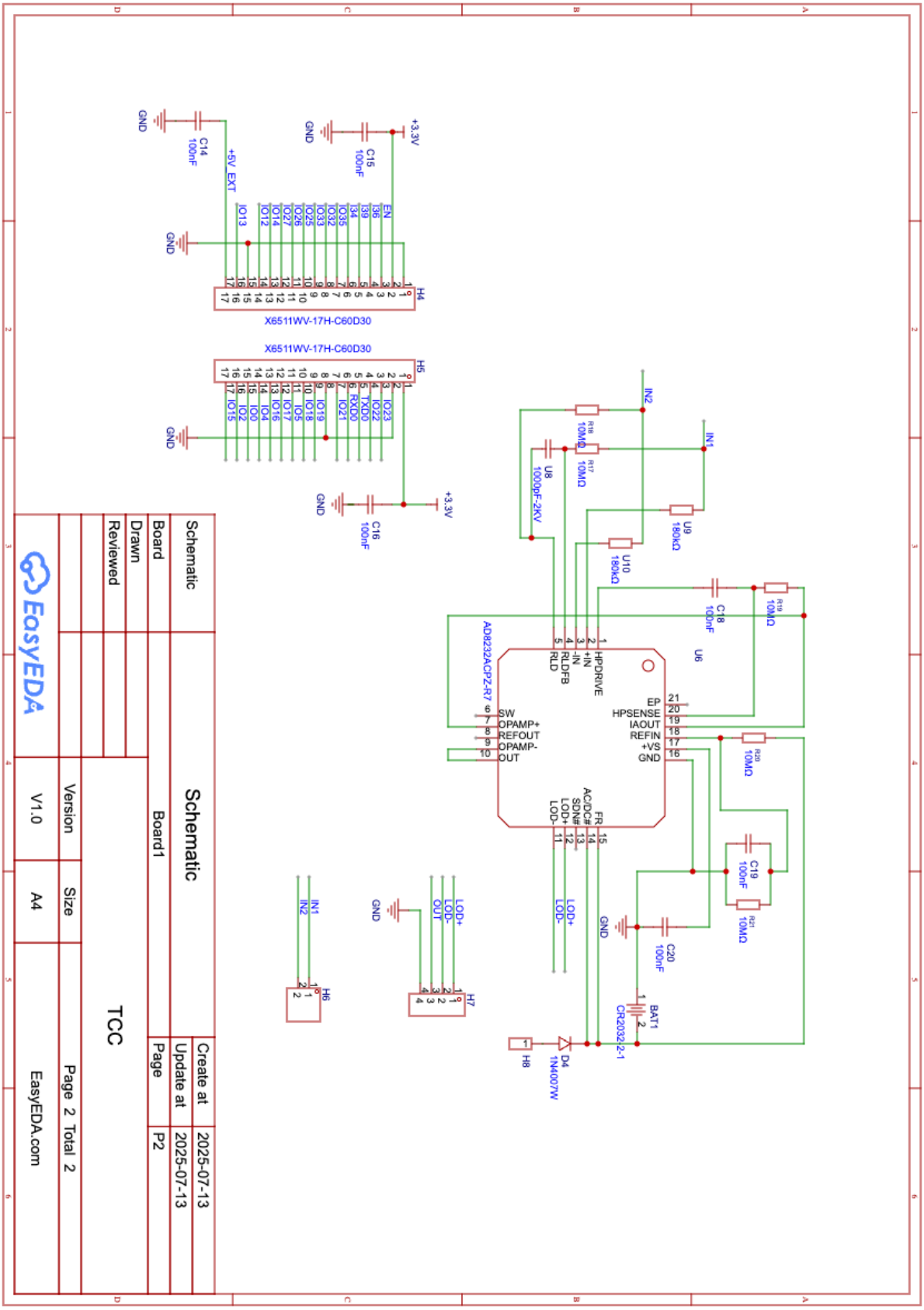


Figura 12 – Esquemático do circuito proposto com ESP32 e AD8232, parte 1.

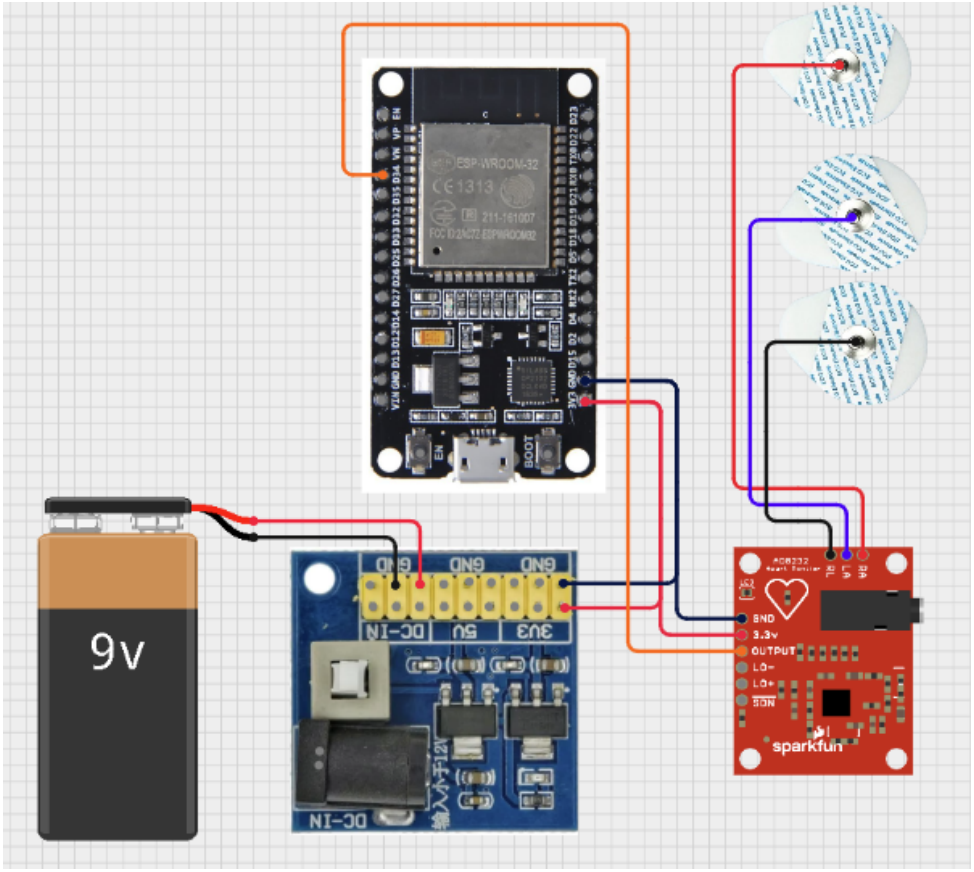


Figura 13 – Protótipo para testes.

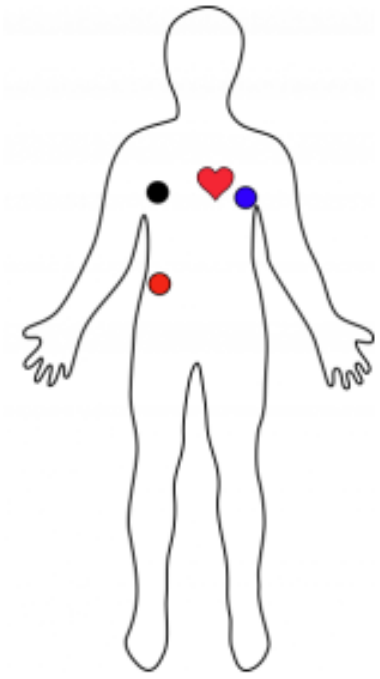


Figura 14 – Posicionamento dos eletrodos no corpo humano para coleta de ECG.

- **ESP32**: microcontrolador que realiza a leitura do sinal analógico através de uma de suas entradas ADC. Posteriormente, esse sinal é transmitido via porta serial.
- **Eletrodos de superfície**: conectados ao paciente conforme a disposição mostrada na Figura 14, utilizando as conexões *RL*, *RA* e *LA* do módulo AD8232.
- **Fonte de alimentação**: uma bateria de 9V conectada a um regulador de tensão, garantindo alimentação estável para o módulo e o microcontrolador na tensão especificada de 3.3V.

A disposição correta dos eletrodos foi fundamental para garantir a qualidade do sinal adquirido, como ilustrado na Figura 14:

O **sinal foi extraído do autor**, caracterizando um experimento de validação de conceito (*proof of concept*). Por essa razão, não foi necessário submeter o estudo ao Comitê de Ética em Pesquisa, uma vez que não se configurou como uma pesquisa clínica envolvendo múltiplos participantes ou com objetivo diagnóstico/terapêutico. Tal abordagem foi suficiente para validar a funcionalidade do sistema e realizar as análises experimentais propostas, respeitando os prazos e escopo do trabalho.

```
1 const int ecgPin = 34;
2 const int ledPin = 2;
3 const unsigned int sampleRate = 977;
4 const unsigned long sampleIntervalMicros = 1000000UL / sampleRate;
5 const int duracaoSegundos = 15;
6 const int totalAmostras = sampleRate * duracaoSegundos;
```

Algoritmo 1 – Inicialização de parâmetros e configuração do hardware

A frequência de amostragem de 977 Hz foi escolhida para garantir compatibilidade com o conjunto de dados Leipzig Heart Center ECG-Database, utilizado no treinamento e validação da rede neural convolucional, além de permitir a captura precisa dos detalhes morfológicos do complexo QRS. A resolução de 12 bits do ADC do ESP32 — que fornece 4096 níveis em uma faixa de 0–3,3 V, resultando em um passo de quantização de 0,8 mV — é suficiente para distinguir variações típicas do sinal amplificado pelo AD8232 (1–5 mV), mantendo boa relação sinal-ruído sem necessidade de hardware mais complexo. Por fim, a duração de 15 segundos assegura o registro de múltiplos batimentos cardíacos consecutivos, oferecendo uma amostra representativa do ritmo cardíaco sem comprometer a performance do sistema embarcado ou gerar arquivos excessivamente grandes para processamento posterior.

- `ecgPin`: pino analógico usado para leitura do sinal (canal 34).
- `sampleRate`: frequência de amostragem definida como 977 Hz.

- `sampleIntervalMicros`: intervalo entre amostras em microssegundos.
- `duracaoSegundos`: tempo total de aquisição (15 segundos).
- `totalAmostras`: número total de amostras a serem coletadas.

O ESP32 inicializa a comunicação serial, configura o pino LED como saída, define a resolução do ADC e tenta alocar a memória necessária para armazenar as amostras, preferencialmente na PSRAM:

```
ecgData = (uint16_t*)heap_caps_malloc(totalAmostras * sizeof(uint16_t),
    MALLOC_CAP_SPIRAM);
if (!ecgData) {
  ecgData = (uint16_t*)malloc(totalAmostras * sizeof(uint16_t));
  if (!ecgData) {
    Serial.println("Falha total na aloca  o de mem ria.");
    while (true);  // trava o programa
  }
}
```

Algoritmo 2 – Trecho de inicialização e alocação dinâmica

Após a inicialização, a aquisição dos dados é realizada em um laço `for` que armazena os valores lidos no vetor `ecgData`, respeitando o tempo de amostragem configurado com `micros()`:

```
for (int i = 0; i < totalAmostras;) {
  unsigned long now = micros();
  if (now - lastMicros >= sampleIntervalMicros) {
    lastMicros = now;
    ecgData[i++] = analogRead(ecgPin);
  }
}
```

Algoritmo 3 – Aquisição do sinal com controle de tempo

Ao término da aquisição, os dados armazenados são enviados sequencialmente pela porta serial:

```
Serial.println("Enviando dados:");
for (int i = 0; i < totalAmostras; i++) {
  Serial.println(ecgData[i]);
}
```

Algoritmo 4 – Transmissão dos dados via Serial

Este código permite adquirir um sinal de ECG por até 15 segundos a 977 Hz, utilizando o ADC do ESP32 e armazenando os dados na RAM ou PSRAM. Os dados são enviados pela

serial para serem processados externamente em uma aplicação de análise e classificação de arritmias. Após a coleta e transmissão do sinal de ECG pelo microcontrolador, os dados são salvos em um arquivo de texto contendo os valores digitalizados provenientes do ADC. O trecho de código a seguir implementa a leitura desses dados em Python para posterior tratamento e análise.

```
arquivo_txt = r'C:\Users\Stella\Desktop\TCC\ecg_bruto.txt'

valores = []

with open(arquivo_txt, 'r') as f:
    for linha in f:
        valor = int(linha.strip())
        valores.append(valor)

print(f"{len(valores)} amostras carregadas do arquivo {arquivo_txt}")
print(f"Primeiras 10 amostras: {valores[:10]}")
```

Algoritmo 5 – Leitura das amostras do arquivo de ECG

O trecho de código apresentado realiza a leitura de um arquivo contendo amostras de sinais eletrocardiográficos (ECG) previamente adquiridos e salvos em formato texto. Inicialmente, define-se o caminho absoluto do arquivo *ecg_bruto.txt*, o qual armazena as amostras coletadas por um microcontrolador, como o ESP32. Em seguida, uma lista vazia chamada *valores* é criada para armazenar os dados lidos.

O arquivo é então aberto em modo leitura (*'r'*), e para cada linha presente, realiza-se a remoção de espaços em branco com a função *strip()*, seguida da conversão do conteúdo textual para um número inteiro utilizando *int()*. Cada valor convertido é adicionado à lista *valores*, a qual, ao final da execução, conterá todas as amostras numéricas extraídas do arquivo. Por fim, o código imprime no terminal o número total de amostras carregadas e exibe as dez primeiras, como uma forma de inspeção rápida e validação da leitura realizada. Esta rotina é essencial para importar os dados brutos de ECG a partir de arquivos persistentes, preparando-os para as etapas subsequentes de processamento digital, como filtragem, normalização e classificação automática por meio de algoritmos de aprendizado de máquina.

Após o carregamento das amostras provenientes do conversor analógico-digital (ADC) do ESP32, o sinal bruto foi convertido para um vetor *NumPy* e visualizado por meio da biblioteca *matplotlib*, a fim de permitir uma inspeção inicial da forma de onda.

```
valores_adc = np.array(valores)
plt.plot(valores_adc)
plt.title("Sinal original capturado (ADC)")
plt.xlabel("Amostras")
plt.ylabel("ADC")
plt.grid()
plt.show()
```

Algoritmo 6 – Visualização do sinal original em unidades de ADC

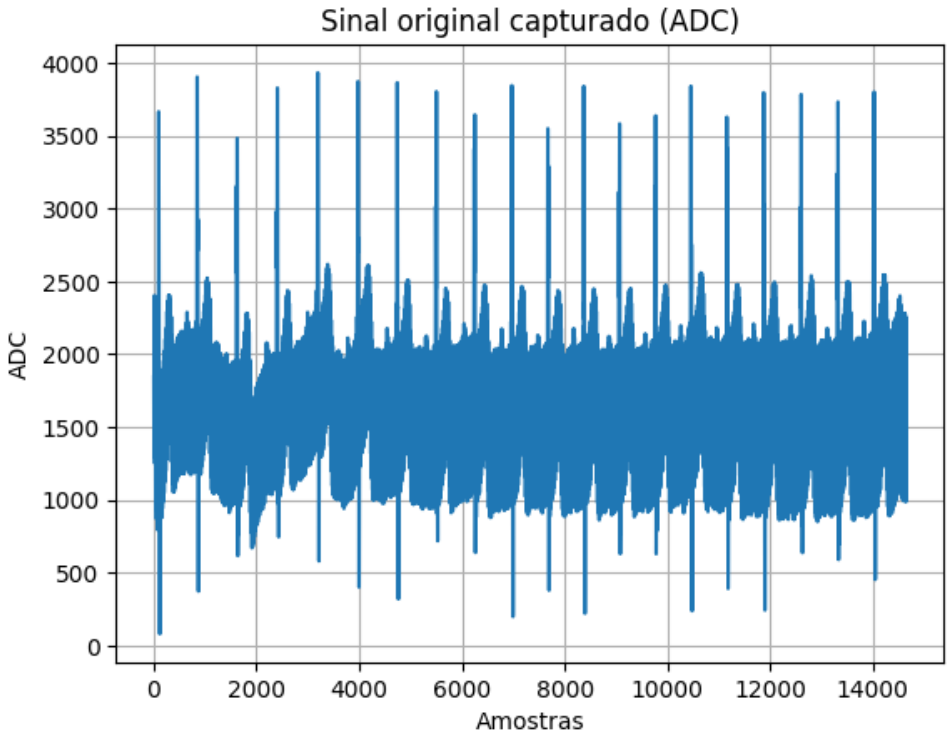


Figura 15 – Visualização do sinal bruto.

O trecho de código converte a lista de amostras para um vetor NumPy, o que permite a aplicação de operações vetoriais eficientes e facilita a geração de gráficos. Em seguida, utiliza-se a função *plt.plot()* para exibir o sinal no domínio do tempo, em que o eixo x representa as amostras sucessivas e o eixo y os valores digitais obtidos pelo conversor analógico-digital (ADC), que variam de 0 a 4095 no caso de uma resolução de 12 bits.

Além disso, o gráfico é enriquecido com rótulos nos eixos e uma grade de referência, o que melhora significativamente a interpretação visual dos dados. A visualização do sinal permite observar a morfologia característica dos batimentos cardíacos registrados, além de facilitar a identificação de ruídos ou anomalias que possam requerer etapas adicionais de processamento, como filtragem ou normalização.

4.6 CONFIGURAÇÃO INICIAL E LEITURA DO ARQUIVO DE SINAL

Antes do início do processamento digital do sinal de ECG, são definidos os parâmetros necessários à conversão, filtragem e análise do sinal, bem como realizado o carregamento do arquivo contendo os dados brutos provenientes do ADC.

```
# === 1) CONFIG PARAMS===
base_dir        = r'C:\Users\Stella\Desktop\TCC'
arquivo_txt     = os.path.join(base_dir, 'ecg_bruto.txt')
record_name     = 'meu_ecg_filtrado'
full_path       = os.path.join(base_dir, record_name)

fs              = 977        # Hz
adc_bits        = 12
adc_max         = 2**adc_bits - 1
v_ref           = 3.3        # volts
adc_gain_wfdb   = 1000       # 1 unidade WFDB = 1 * 10^-6 V

lowcut, highcut = (0.5, 45.0)    # Hz
notch_freq, Q   = (60.0, 30)     # Hz
min_rr_interval = 0.3            # segundos
peak_height     = 0.1            # mV
peak_prominence = 0.05           # mV
```

Algoritmo 7 – Parâmetros de configuração do ambiente de processamento

O código apresentado define inicialmente o diretório base e os nomes dos arquivos de entrada e saída utilizados na análise do sinal de ECG. Em seguida, são especificados parâmetros fundamentais para o correto processamento do sinal, incluindo a frequência de amostragem, a resolução do conversor analógico-digital (ADC) e o valor de referência em volts associado a ele.

Além disso, são definidos os parâmetros necessários para a conversão de unidades e aplicação de técnicas de filtragem digital. Dentre eles, destacam-se: um filtro passa-faixa com faixa de operação entre 0,5 Hz e 45 Hz, utilizado para preservar componentes relevantes do sinal eletrocardiográfico; um filtro notch em 60 Hz, com fator de qualidade $Q = 30$, projetado para eliminar interferências da rede elétrica; e os critérios de detecção de picos, com altura mínima de 0,1 mV e proeminência mínima de 0,05 mV, essenciais para a identificação dos complexos QRS. Esses parâmetros garantem que o sinal processado esteja adequado para posterior análise e classificação automática.

```
# === 2) LEITURA DO TXT ===
valores = []
with open(arquivo_txt, 'r') as f:
    for linha in f:
        try:
            valores.append(int(linha.strip()))
        except ValueError:
            pass

valores = np.array(valores)
print(f"{len(valores)} amostras carregadas de '{arquivo_txt}'")
```

Algoritmo 8 – Leitura segura das amostras do arquivo .txt

O código realiza a leitura do arquivo *ecg_bruto.txt*, que contém os dados brutos do sinal de ECG. Cada linha do arquivo é percorrida sequencialmente e convertida para um valor inteiro, correspondente à leitura digital fornecida pelo conversor analógico-digital (ADC). Caso alguma linha contenha dados inválidos, ela é automaticamente ignorada durante a conversão.

Os valores válidos são então armazenados em um vetor do tipo *NumPy*, o que facilita o processamento posterior por permitir operações vetoriais eficientes e compatibilidade com bibliotecas de processamento de sinais e aprendizado de máquina. Ao final da leitura, o vetor *valores* contém todas as amostras válidas capturadas do sinal de ECG, que servirão como base para as etapas seguintes de pré-processamento, filtragem e análise automática.

4.7 PRÉ-PROCESSAMENTO DO SINAL DE ECG

O pré-processamento do sinal de ECG bruto é composto por três etapas principais: conversão de unidades (ADC para mV), aplicação de filtros (passa-faixa e rejeita-faixa/notch) e normalização de amplitude. Essas etapas são fundamentais para melhorar a qualidade do sinal antes da detecção de batimentos ou uso em modelos de aprendizado de máquina.

```
# === 3) CONVERSAO ADC -> mV + OFFSET DC ===
sinal_mv = (valores / adc_max) * v_ref * 1000
sinal_mv -= np.mean(sinal_mv)
print(f"Sinal convertido: min={sinal_mv.min():.2f} mV, max={sinal_mv.max()
    :.2f} mV")
```

Algoritmo 9 – Conversão dos valores do ADC para mV

O trecho de código realiza a conversão dos valores digitalizados pelo conversor analógico-digital (ADC), com resolução de 12 bits e referência de 3,3 V, para a unidade de milivolts. Essa transformação é essencial para interpretar o sinal em termos fisiológicos reais. Em seguida, o valor médio do sinal é subtraído de todas as amostras, procedimento conhecido como remoção

do offset DC, que centraliza o traçado em torno de zero. Essa centralização facilita a análise visual e computacional do sinal, além de melhorar a eficácia das etapas subsequentes de filtragem e detecção de eventos cardíacos.

```
# === 4) PASSA-FAIXA ===
def passa_faixa(x, fs, low, high, ordem=4):
    nyq = fs / 2
    sos = butter(ordem, [low/nyq, high/nyq], btype='band', output='sos')
    return sosfiltfilt(sos, x)

sinal_band = passa_faixa(sinal_mv, fs, lowcut, highcut)
```

Algoritmo 10 – Função para aplicar filtro passa-faixa

O código aplica um filtro digital de segunda ordem utilizando a representação em seções de segunda ordem (*second-order sections*, ou *sos*), uma abordagem numericamente estável para filtragem digital. O filtro implementado é do tipo passa-faixa, permitindo a passagem de frequências entre 0,5 Hz e 45 Hz — faixa que abrange os principais componentes do sinal eletrocardiográfico. Com isso, são eliminados ruídos de baixa frequência, como variações de linha de base, e de alta frequência, como interferências eletromagnéticas, preservando as informações relevantes para a análise do ECG.

```
# === 5) REMOVE FAIXA 60 Hz ===
def notch(x, fs, f0, Q):
    b, a = iirnotch(f0, Q, fs)
    return filtfilt(b, a, x)

sinal_clean = notch(sinal_band, fs, notch_freq, Q)
```

Algoritmo 11 – Função para aplicar filtro notch a 60 Hz

O trecho de código é responsável por atenuar a interferência proveniente da rede elétrica, que opera a 60 Hz no Brasil. Essa interferência é comum em ambientes hospitalares ou laboratoriais e pode comprometer a qualidade do sinal eletrocardiográfico. Para mitigar esse ruído, é utilizado um filtro do tipo rejeita-faixa, projetado especificamente para remover componentes senoidais na frequência de 60 Hz. O filtro é configurado com um fator de qualidade $Q = 30$, o que garante uma rejeição seletiva e eficiente da frequência-alvo, preservando as demais componentes espectrais do sinal de ECG.

```
# === 6) NORMALIZACAO (2 mV) ===
sinal_norm = sinal_clean / np.max(np.abs(sinal_clean)) * 2.0
```

Algoritmo 12 – Normalização para faixa de ±2 mV

O sinal filtrado é escalado para a faixa de ± 2 mV, valor típico na representação de sinais eletrocardiográficos em aplicações clínicas e computacionais. Essa normalização é importante tanto

para a padronização da visualização quanto para o desempenho de algoritmos automatizados, como os utilizados na detecção de picos ou em modelos de aprendizado de máquina.

Ao final dessas etapas, o sinal encontra-se em milivolts, livre de ruídos de baixa frequência e da interferência da rede elétrica, além de estar normalizado em amplitude. Dessa forma, ele está devidamente preparado para ser analisado visualmente ou submetido a processos de classificação automática.

4.8 EXPORTAÇÃO DO SINAL PARA O FORMATO WFDB

Após o pré-processamento e normalização do sinal de ECG, é realizada a exportação para o formato WFDB (WaveForm Database), padrão amplamente utilizado em bancos de dados fisiológicos, como os disponíveis no PhysioNet. Essa etapa garante compatibilidade com ferramentas de visualização, anotação e processamento de sinais biomédicos.

```
# === 7) SALVA EM WFDB (.dat + .hea) NO DIRETORIO base_dir ===
wfdb.wrsamp(
    record_name=record_name,            # nome do arquivo (sem extensao)
    write_dir=base_dir,                 # diretorio de destino
    fs=fs,                              # frequencia de amostragem
    sig_name=['ECG'],                   # nome do sinal
    units=['mV'],                       # unidade
    p_signal=sinal_norm.reshape(-1, 1),  # sinal formato [n,1]
    fmt=['16'],                         # formato dos dados (16 bits)
    adc_gain=[adc_gain_wfdb],           # ganho para conversao WFDB
    baseline=[0],                       # baseline ajustado
    comments=[f"Processado: passa-faixa, notch, normalizado em {datetime.
        now()}"]
)
```

Algoritmo 13 – Exportação do sinal para arquivos .dat e .hea

O sinal previamente processado é convertido para uma matriz de dimensão $(n, 1)$, formato esperado pelas bibliotecas de leitura e escrita de sinais fisiológicos, como a WFDB (WaveForm DataBase). Em seguida, os dados são salvos em dois arquivos: *.dat*, que contém os valores binários do sinal, e *.hea*, que armazena o cabeçalho com informações descritivas.

Durante esse processo, são adicionados metadados essenciais, como o ganho de conversão (em mV/LSB), a unidade física do sinal (milivolts), o formato de codificação digital e comentários sobre as etapas de processamento aplicadas. Essa estruturação permite que o sinal seja reutilizado de forma padronizada por ferramentas de análise e visualização compatíveis com o formato WFDB.

```
# === 8) GERA .atr NO MESMO DIRETORIO ===
if peaks.size:
    wfdb.wrann(
        record_name=record_name,
        write_dir=base_dir,
        extension='atr',
        sample=peaks,
        symbol=['N'] * len(peaks),  # simbolo padrao 'N' para cada batimento
        fs=fs
    )
else:
    print("Nenhum pico detectado; .atr n o gerado.")
```

Algoritmo 14 – Geração do arquivo de anotações com picos R

Caso picos característicos — como os picos R — tenham sido detectados durante o processamento do sinal, o código gera um arquivo de anotações com extensão *.atr*, contendo marcações de batimentos classificados como normais, utilizando o símbolo 'N'. Cada anotação é associada a um instante específico de amostragem, correspondente ao índice do pico identificado, e vinculada a um símbolo representativo.

Se nenhum pico for identificado, o arquivo de anotações não é gerado, evitando a criação de registros inconsistentes.

Como resultado, o sinal de ECG passa a estar disponível em um formato plenamente compatível com as ferramentas da biblioteca WFDB, permitindo sua validação, visualização gráfica e comparação com sinais provenientes de bancos de dados clínicos padronizados.

4.9 COMPARAÇÃO VISUAL ENTRE ECG PRÓPRIO E ECG DO BANCO LEIPZIG

Com o objetivo de validar a qualidade do sinal de ECG obtido a partir do sistema embarcado, realizou-se a comparação com um registro do banco de dados *Leipzig Heart Center ECG-Database*. A comparação foi feita visualmente, por meio da sobreposição dos sinais e das anotações de batimentos (*.atr*) associadas, em subplots separados.

```
base_dir = r'C:\Users\Stella\Desktop\TCC'
user_record_path     = os.path.join(base_dir, 'meu_ecg_filtrado')
leipzig_record_path = os.path.join(base_dir, 'db', 'x102')

user_rec = wfdb.rdrecord(user_record_path)
user_ann = wfdb.rdann(user_record_path, 'atr')
leipzig_rec = wfdb.rdrecord(leipzig_record_path)
leipzig_ann = wfdb.rdann(leipzig_record_path, 'atr')

samples_to_plot = 3000  # corresponde a cerca de 3 segundos

fig, axes = plt.subplots(2, 1, figsize=(15, 6))

# --- minha mostragem ECG ---
axes[0].plot(
    user_rec.p_signal[:samples_to_plot, 0],
    label='Seu ECG', color='tab:blue'
)

user_peaks = [s for s in user_ann.sample if s < samples_to_plot]
axes[0].scatter(
    user_peaks,
    user_rec.p_signal[user_peaks, 0],
    color='tab:red', label='Anotacoes'
)
axes[0].set(title="Minha amostragem ECG com anotacoes 'N'", ylabel="mV")
axes[0].legend()
axes[0].grid(True)
```

Algoritmo 15 – Leitura dos registros do usuário e de referência

O trecho de código apresentado realiza a leitura de um sinal de ECG previamente salvo no formato WFDB, comparando-o com um registro de referência da base de Leipzig. Primeiramente, define-se o diretório base e os caminhos dos arquivos de entrada do usuário e do banco de dados clínico. Os registros são carregados com a função *rdrecord*, enquanto as anotações (*.atr*) são lidas por meio de *rdann*.

Em seguida, estabelece-se um limite de 3000 amostras para visualização gráfica, correspondente a aproximadamente 3 segundos de sinal. Um gráfico com duas subáreas é configurado, sendo a primeira destinada à visualização do ECG do usuário. Nessa área, o sinal é plotado até o ponto definido por *samples_to_plot*, sendo identificado com a legenda “Minha amostra ECG”.

```
# --- ECG Leipzig ---
axes[1].plot(
    leipzig_rec.p_signal[:samples_to_plot, 0],
    label='Leipzig ECG', color='tab:green'
)

leipzig_peaks = [s for s in leipzig_ann.sample if s < samples_to_plot]
axes[1].scatter(
    leipzig_peaks,
    leipzig_rec.p_signal[leipzig_peaks, 0],
    color='tab:red', label='Anotacoes'
)
axes[1].set(title="Leipzig ECG (x102)", xlabel="Amostras", ylabel="mV")
axes[1].legend()
axes[1].grid(True)

plt.tight_layout()
plt.show()
```

Algoritmo 16 – Visualização lado a lado dos sinais de ECG

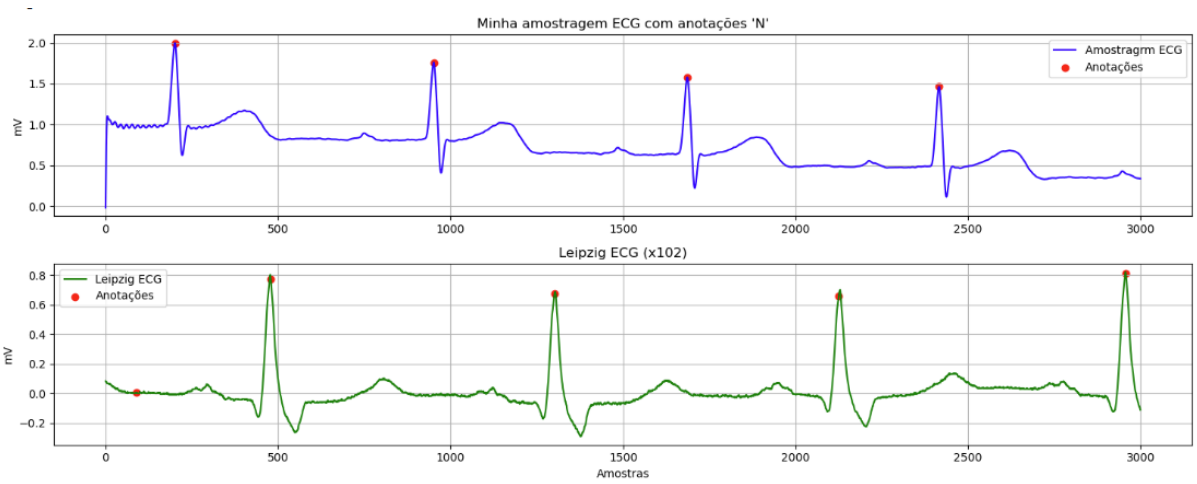


Figura 16 – Registros de ECG do autor e do banco Leipzig.

As anotações dos batimentos — tipicamente os picos R — são destacadas como marcadores vermelhos sobre os traçados de ECG. No primeiro gráfico, correspondente ao sinal do usuário, os picos são sobrepostos ao traçado azul; no segundo, referente ao sinal de referência da base Leipzig, os picos são exibidos sobre um traçado verde.

Essa visualização comparativa permite avaliar qualitativamente a morfologia dos sinais, bem como verificar a consistência e precisão das anotações automáticas geradas pelo sistema, tendo como base um banco de dados clínico validado.

4.10 ORGANIZAÇÃO DO CONJUNTO DE DADOS E MAPEAMENTO DOS SÍMBOLOS

Antes do início do processamento e treinamento do modelo, é necessário definir alguns parâmetros globais que serão utilizados ao longo de todas as etapas do pipeline. Esses parâmetros incluem configurações de diretório, amostragem, tamanho das janelas de sinal e parâmetros de treinamento.

```
# === 1) GENERAL PARAMETERS ===
DATA_DIR      = r'C:\Users\Stella\Desktop\TCC\db'
FS            = 977          # Frequencia de amostragem (Hz)
WINDOW_MS     = 300          # Dura  o da janela (ms)
WINDOW_SIZE   = int(FS * WINDOW_MS / 1000)

BATCH_SIZE    = 128          # Tamanho do mini-batch
LR            = 1e-3         # Taxa de aprendizado
EPOCHS        = 50           # N mero de epocas de treinamento
```

Algoritmo 17 – Definição dos parametros gerais

Descrição dos parâmetros:

- *DATA_DIR*: Caminho para o diretório onde estão localizados os arquivos do dataset Leipzig (incluindo *RECORDS*, *x*.dat*, *x*.hea*, *x*.atr*).
- *FS*: Frequência de amostragem dos sinais de ECG (977 Hz), conforme especificado na documentação do banco de dados.
- *WINDOW_MS*: Duração da janela de sinal a ser extraída ao redor de cada batimento, definida em milissegundos (300 ms).
- *WINDOW_SIZE*: Quantidade de amostras por batimento, calculada como *FS* × *WINDOW_MS / 1000*, resultando em aproximadamente 293 amostras.
- *BATCH_SIZE*: Número de amostras utilizadas em cada passo do treinamento por mini-batch.
- *LR*: Taxa de aprendizado do otimizador, que determina o tamanho dos passos na atualização dos pesos.
- *EPOCHS*: Quantidade de vezes que o modelo percorrerá todo o conjunto de treinamento durante o processo de aprendizado.

A seguir, apresenta-se um trecho de código responsável pela leitura dos registros de ECG e pela construção do mapeamento entre os símbolos de batimentos cardíacos e seus respectivos índices numéricos, utilizados para fins de classificação.

```python
# === 2) LOAD RECORDS & MAP SYMBOLS ===
with open(os.path.join(DATA_DIR, 'RECORDS')) as f:
    records = [r.strip() for r in f if r.strip()]

symbol_set = set()
for rec in records:
    ann = wfdb.rdann(os.path.join(DATA_DIR, rec), 'atr')
    symbol_set.update(ann.symbol)

SYMBOL_LIST = sorted(symbol_set)
SYMBIDX     = {s: i for i, s in enumerate(SYMBOL_LIST)}
N_CLASSES   = len(SYMBOL_LIST)

print(f"{len(records)} registros, {N_CLASSES} classes: {SYMBOL_LIST}")
```

Algoritmo 18 – Carregamento dos registros e criação do mapeamento de classes

O código realiza a leitura do arquivo *RECORDS*, localizado no diretório especificado por *DATA_DIR*, que contém os nomes de todos os registros de ECG disponíveis na base. Para cada um desses registros, são carregados os arquivos de anotações (*.atr*) utilizando a biblioteca *wfdb*, que permite o acesso estruturado às marcações de batimentos cardíacos.

Em seguida, os símbolos presentes nos registros — que representam os tipos de batimentos identificados (como batimentos normais, ventriculares, artefatos, entre outros) — são extraídos e armazenados em um conjunto (*symbol_set*), o que garante que cada símbolo seja considerado apenas uma vez, mesmo que apareça em múltiplos registros. Após a coleta, os símbolos são ordenados alfabeticamente e utilizados para construir um dicionário *SYMBIDX*, no qual cada símbolo é mapeado para um índice inteiro exclusivo. Por fim, o número total de classes de batimentos distintos é armazenado na variável *N_CLASSES*.

Como resultado, o código imprime um resumo informando o número de registros lidos, a quantidade de classes de batimentos detectadas e a lista completa dos símbolos encontrados, fornecendo uma visão geral útil para a etapa de classificação supervisionada.

4.11 EXTRAÇÃO DOS BATIMENTOS CARDÍACOS POR REGISTRO

Nesta etapa, é definida uma função chamada *extract_beats*, responsável por extrair os segmentos de batimentos cardíacos centrados nos picos anotados de cada registro de ECG. Esses segmentos são posteriormente utilizados como amostras de entrada no modelo de classificação.

```
# === 3) BEAT EXTRACTION FUNCTION ===
def extract_beats(rec_name):
    rec = wfdb.rdrecord(os.path.join(DATA_DIR, rec_name))
    ann = wfdb.rdann(os.path.join(DATA_DIR, rec_name), 'atr')
    sig = rec.p_signal[:, 0]
    half = WINDOW_SIZE // 2
    Xb, yb = [], []

    for samp, sym in zip(ann.sample, ann.symbol):
        st, ed = samp - half, samp + half
        if st < 0 or ed > len(sig):
            continue
        beat = sig[st:ed]
        if beat.shape[0] != WINDOW_SIZE:
            beat = resample(beat, WINDOW_SIZE)
        Xb.append(beat)
        yb.append(SYM2IDX[sym])

    return np.array(Xb, dtype=np.float32), np.array(yb, dtype=np.int64)
```

Algoritmo 19 – Função para extração de batimentos centrados

O código realiza a leitura dos dados eletrocardiográficos a partir de arquivos no formato WFDB, carregando tanto o sinal (*.dat*) quanto suas respectivas anotações (*.atr*) por meio da biblioteca *wfdb*. Em seguida, é selecionada apenas a primeira derivação do ECG, por meio da indexação *rec.p_signal[:, 0]*, considerando que ela contém a informação principal para a análise dos batimentos cardíacos.

Define-se então o tamanho da janela temporal a ser utilizada para o recorte de batimentos, representada pela constante *WINDOW_SIZE*, geralmente correspondente a 300 ms centrados em torno de cada batimento anotado. O código percorre todas as amostras com anotações válidas e, para cada uma delas, realiza os seguintes procedimentos: calcula os limites da janela centrada no batimento; ignora os casos em que a janela extrapola os limites do sinal original; extrai o segmento correspondente do sinal; e, caso o número de amostras da janela extraída seja diferente do valor esperado, aplica reamostragem para adequá-la ao tamanho padrão.

Cada batimento extraído é, então, adicionado à lista *Xb*, a qual armazenará todos os batimentos processados para posterior uso em treinamento, validação ou análise estatística.

4.12 CONSTRUÇÃO DO CONJUNTO DE DADOS FINAL

Após a definição da função de extração de batimentos, o próximo passo consiste em aplicar essa função a todos os registros listados, agregando os batimentos e seus respectivos rótulos em uma única estrutura unificada. Isso resulta na criação do conjunto de dados completo, pronto para ser utilizado no treinamento de modelos de aprendizado de máquina.

```
# === 4) BUILD DATASET ===
X_list, y_list = [], []
for rec in records:
    xb, yb = extract_beats(rec)
    X_list.append(xb)
    y_list.append(yb)

X = np.vstack(X_list)
y = np.hstack(y_list)

print(f"Extra dos {X.shape[0]} batimentos x {X.shape[1]} amostras")
```

Algoritmo 20 – Construção do dataset a partir dos registros de ECG

O código tem como objetivo consolidar os batimentos cardíacos extraídos de múltiplos registros em uma única matriz de dados. Inicialmente, são criadas duas listas vazias: *X_list*, para armazenar os segmentos de batimentos, e *y_list*, para guardar os respectivos rótulos associados a cada batimento.

O processo se dá por meio de um loop que percorre todos os registros listados em *records*. Para cada registro, a função *extract_beats* é chamada, retornando dois conjuntos: *xb*, com os batimentos recortados e normalizados, e *yb*, com os rótulos correspondentes, geralmente oriundos das anotações do arquivo *.atr*. Os batimentos e seus rótulos são então adicionados às listas agregadoras.

Após o término da iteração sobre todos os registros, os dados são organizados em estruturas vetoriais por meio das funções *np.vstack* e *np.hstack*. A primeira concatena verticalmente todos os batimentos em uma matriz única *X*, com formato (N, L), onde N representa o número total de batimentos e L o número de amostras por batimento. A segunda função agrupa os rótulos em um único vetor *y*, com dimensão $(N,)$.

Como resultado, o código imprime o número total de batimentos extraídos (*X.shape[0]*) e o número de amostras por batimento (*X.shape[1]*), validando a correta estruturação do conjunto de dados final para posterior uso em modelos de classificação.

4.13 NORMALIZAÇÃO E PREPARAÇÃO DOS DADOS PARA TREINAMENTO

Após a extração dos batimentos cardíacos e a construção dos vetores *X* (batimentos) e *y* (rótulos), realiza-se a normalização por batimento, o particionamento da base em conjuntos de treinamento e teste, e a criação dos dataloaders utilizando o PyTorch.

```
# === 4) NORMALIZACAO E SPLIT MANUAL ===
# Z-score por batimento
X = (X - X.mean(axis=1, keepdims=True)) / (X.std(axis=1, keepdims=True) + 1e-8)

# acrescenta canal (formato N x 1 x L)
X = X[:, None, :]  # (N, 1, window_size)
```

Algoritmo 21 – Normalização Z-score por batimento

O código aplica a normalização do tipo *Z-score* individualmente em cada batimento do conjunto de dados. Esse processo consiste em subtrair a média e dividir pelo desvio padrão de cada batimento, de forma a centralizar os dados em torno de zero com variância unitária. Tal normalização é fundamental para garantir a estabilidade numérica e o desempenho dos algoritmos de aprendizado de máquina, especialmente redes neurais.

Além disso, a estrutura *X[:, None, :]* é utilizada para inserir uma nova dimensão no tensor de entrada, correspondente ao canal do sinal. Essa transformação resulta em uma matriz tridimensional no formato $(N, 1, L)$, onde N representa o número de batimentos, 1 o número de canais (no caso, apenas uma derivação do ECG), e L o número de amostras por batimento. Essa adaptação é necessária para que os dados estejam compatíveis com as exigências de entrada das camadas convolucionais unidimensionais (*Conv1D*) presentes em redes neurais profundas.

```
# Shuffle + 80/20 split
np.random.seed(42)
idx = np.random.permutation(len(X))
split = int(0.8 * len(X))

train_idx, test_idx = idx[:split], idx[split:]
X_train, y_train = X[train_idx], y[train_idx]
X_test, y_test = X[test_idx], y[test_idx]
```

Algoritmo 22 – Particionamento da base em treino e teste

O código realiza a divisão dos dados em subconjuntos de treinamento e teste de forma controlada e reprodutível. Para isso, os índices correspondentes aos batimentos são inicialmente embaralhados de maneira aleatória, utilizando uma semente fixa no gerador de números aleatórios.

Essa prática assegura que os resultados obtidos sejam consistentes entre diferentes execuções do experimento.

Após o embaralhamento, os dados são divididos em dois conjuntos: 80% dos batimentos são utilizados para o treinamento do modelo, enquanto os 20% restantes são reservados para a etapa de teste. Essa divisão é fundamental para avaliar a capacidade de generalização do modelo, assegurando que ele seja validado em dados não vistos durante o processo de aprendizado.

```
# === 5) DATASET + DATALOADER ===
class ECGDataset(Dataset):
    def __init__(self, X, y):
        self.X = torch.from_numpy(X)
        self.y = torch.from_numpy(y)

    def __len__(self):
        return len(self.X)

    def __getitem__(self, i):
        return self.X[i], self.y[i]

train_loader = DataLoader(ECGDataset(X_train, y_train), batch_size=
    batch_size, shuffle=True)
test_loader  = DataLoader(ECGDataset(X_test,  y_test),  batch_size=
    batch_size)
```

Algoritmo 23 – Definição do dataset personalizado e criação dos loaders

O código define a classe *ECGDataset*, que herda da interface *torch.utils.data.Dataset* da biblioteca PyTorch. Essa classe é responsável por encapsular os dados de entrada (sinais) e suas respectivas classes (rótulos), convertendo os arrays NumPy em tensores compatíveis com a estrutura de dados do PyTorch. A implementação dessa classe facilita o gerenciamento dos batimentos cardíacos como amostras indexáveis, integrando-se de forma direta com os utilitários de carregamento de dados do framework.

Em seguida, são instanciados dois *dataloaders*: *train_loader* e *test_loader*. O primeiro é utilizado durante a etapa de treinamento do modelo, enquanto o segundo é empregado para validação. Esses dataloaders permitem o acesso eficiente aos dados em minibatches, possibilitando o embaralhamento das amostras e o uso de múltiplas threads para otimização da leitura, o que é essencial para acelerar o treinamento de redes neurais profundas.

4.14 ARQUITETURA DA CNN PROPOSTO

Com o objetivo de classificar batimentos cardíacos com base nas janelas extraídas do sinal de ECG, foi implementado um modelo baseado em *Convolutional Neural Networks* (CNN) utilizando camadas convolucionais unidimensionais (*Conv1d*), adequadas para séries temporais.

```
class ECGNet(nn.Module):
    def __init__(self, n_classes):
        super().__init__()
        self.net = nn.Sequential(
            nn.Conv1d(1, 32, 5, padding=2), nn.ReLU(), nn.BatchNorm1d(32),
                nn.MaxPool1d(2),
            nn.Conv1d(32, 64, 5, padding=2), nn.ReLU(), nn.BatchNorm1d(64),
                 nn.MaxPool1d(2), nn.Dropout(0.3),
            nn.Conv1d(64, 128, 5, padding=2), nn.ReLU(), nn.BatchNorm1d
                (128), nn.MaxPool1d(2), nn.Dropout(0.3),
            nn.Flatten(),
            nn.Linear((window_size//8)*128, 64), nn.ReLU(), nn.Dropout(0.4)
                ,
            nn.Linear(64, n_classes)
        )

    def forward(self, x):
        return self.net(x)

# Inicializacao
model     = ECGNet(n_classes).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=lr)
```

Algoritmo 24 – Arquitetura CNN1D para classificação de ECG

O dimensionamento da arquitetura da rede neural convolucional (CNN) foi definido com base nas características específicas dos sinais eletrocardiográficos (ECG), nos requisitos da tarefa de classificação de arritmias e nas limitações computacionais do ambiente de desenvolvimento. Optou-se por uma arquitetura unidimensional (*1D Convolutional Neural Network*), considerando que os sinais de ECG são séries temporais e, portanto, adequadamente tratados por convoluções 1D. Esse tipo de rede é amplamente adotado na literatura para análise de sinais fisiológicos, devido à sua capacidade de extrair automaticamente características discriminantes sem a necessidade de extração manual de atributos.

A profundidade da rede foi mantida em um nível moderado para evitar sobreajuste, levando em conta o número de amostras disponíveis e a complexidade do problema. Foram utilizados filtros convolucionais com tamanhos pequenos (entre 3 e 7 amostras), adequados para capturar padrões locais importantes, como a forma do complexo QRS ou a presença de ba-

timentos ectópicos. A quantidade de filtros foi aumentada progressivamente em cada camada convolucional (por exemplo, 32, 64 e 128 filtros), permitindo a extração hierárquica de características de maior complexidade.

Camadas de *pooling* foram incluídas para reduzir a dimensionalidade dos mapas de ativação e garantir robustez a pequenas variações no sinal. A função de ativação ReLU foi empregada nas camadas ocultas, por ser computacionalmente eficiente e favorecer a convergência durante o treinamento. A etapa final da rede conta com camadas densas (*fully connected*) para realizar a classificação dos batimentos nas categorias-alvo (normal, ventricular e supraventricular), utilizando uma função de ativação *softmax*, apropriada para problemas de classificação multiclasse.

Adicionalmente, o número total de parâmetros da rede foi limitado de forma a viabilizar, futuramente, sua implementação em sistemas embarcados ou dispositivos portáteis, como o ESP32, garantindo baixo consumo de recursos computacionais. Os resultados experimentais indicaram que a arquitetura escolhida foi capaz de generalizar bem tanto para os dados públicos da base Leipzig quanto para sinais reais adquiridos pelo sistema proposto, demonstrando eficácia na detecção automática de arritmias.

A arquitetura do modelo proposto consiste em uma rede neural convolucional unidimensional (CNN1D) composta por três camadas *Conv1d*, cada uma com tamanho de kernel igual a 5 e *padding* de valor 2, o que garante a preservação da dimensão temporal do sinal ao longo das convoluções. Cada camada convolucional é seguida por uma normalização por lote (*BatchNorm1d*), que contribui para a estabilização do treinamento e acelera a convergência.

Após cada convolução, aplica-se uma operação de *MaxPool1d* com fator de redução 2, o que reduz pela metade a resolução temporal do sinal, favorecendo a extração de características locais relevantes. A regularização é realizada por meio de camadas *Dropout*, aplicadas após a segunda e terceira convolução, bem como na camada totalmente conectada (*Linear*), com o objetivo de mitigar o risco de sobreajuste (*overfitting*).

Ao final do processamento convolucional, os vetores extraídos são achatados com a operação *Flatten* e encaminhados a uma camada densa (*Linear*), responsável por projetar a saída para o número total de classes de batimentos cardíacos. A função de perda utilizada é a *CrossEntropyLoss*, apropriada para tarefas de classificação multiclasse. O modelo é treinado com o otimizador *Adam*, configurado com uma taxa de aprendizado previamente definida.

Dessa forma, o modelo CNN1D está preparado para receber lotes (*batches*) de batimentos cardíacos normalizados e retornar, para cada um deles, a predição da classe correspondente com base nas características extraídas.

4.15 TREINAMENTO E AVALIAÇÃO DO MODELO

Com a arquitetura definida e os dados organizados nos dataloaders, executou-se o treinamento da rede convolucional por múltiplas épocas, com avaliação da acurácia ao final de cada uma.

```
train_losses = []
test_accs = []

for epoch in range(1, epochs+1):
    model.train()
    running_loss = 0.0
    for xb, yb in train_loader:
        xb, yb = xb.to(device), yb.to(device)
        optimizer.zero_grad()
        loss = criterion(model(xb), yb)
        loss.backward()
        optimizer.step()
        running_loss += loss.item() * xb.size(0)

    train_loss = running_loss / len(train_loader.dataset)
    train_losses.append(train_loss)

    model.eval()
    correct = 0
    with torch.no_grad():
        for xb, yb in test_loader:
            xb, yb = xb.to(device), yb.to(device)
            preds = model(xb).argmax(dim=1)
            correct += (preds == yb).sum().item()

    acc = correct / len(test_loader.dataset)
    test_accs.append(acc)

    print(f"Epoch {epoch:2d}/{epochs}  loss={train_loss:.4f}  acc={acc:.4f}
        ")
```

Algoritmo 25 – Loop de treinamento e avaliação por época

O processo de treinamento do modelo é organizado em duas etapas principais: a fase de treino e a fase de avaliação. Na fase de treino, o modelo é colocado em modo *train()*, o que garante o funcionamento adequado das camadas de *dropout* e *batch normalization*. Para cada mini-batch de dados, é realizada uma propagação direta (forward pass), seguida do cálculo da função de perda, retropropagação dos gradientes (backpropagation) e atualização dos pesos do modelo por meio do otimizador definido. Durante essa fase, a perda acumulada ao longo de todos os mini-batches é somada para cálculo da perda média da época.

Na fase de avaliação, os gradientes são desativados utilizando a instrução *torch.no_grad()*, o que reduz o consumo de memória e tempo de processamento. O modelo realiza apenas a propagação direta para gerar as predições e, com base nelas, calcula-se o número total de acertos em relação às classes verdadeiras. Esse valor é utilizado para obter a acurácia sobre o conjunto de teste.

Ao final de cada época, são exibidos no terminal tanto a perda média sobre o conjunto de treino quanto a acurácia obtida na avaliação com o conjunto de teste. Esses valores também são armazenados em listas específicas, como *train_losses* e *test_accs*, possibilitando o monitoramento do desempenho do modelo ao longo das épocas e a análise posterior da sua evolução durante o treinamento.

4.16 VISUALIZAÇÃO DAS CURVAS DE TREINAMENTO

Para avaliar o comportamento do modelo durante o treinamento, foram plotadas as curvas de perda (*loss*) no conjunto de treino e a acurácia no conjunto de teste ao longo das épocas.

```
1  import matplotlib.pyplot as plt
2
3  epochs_range = range(1, epochs+1)
4  plt.figure(figsize=(8,4))
5  plt.plot(epochs_range, train_losses, label='Train Loss')
6  plt.plot(epochs_range, test_accs,    label='Test Acc')
7  plt.xlabel('Epoca')
8  plt.legend()
9  plt.grid(True)
10 plt.title('Curvas de Treino')
11 plt.show()
```

Algoritmo 26 – Plot das curvas de perda e acurácia

O código utiliza a biblioteca *matplotlib* para gerar uma visualização gráfica do desempenho do modelo ao longo do treinamento. Inicialmente, é criada uma figura com dimensões de 8 por 4 polegadas. Em seguida, são plotadas duas curvas principais: a curva de *train_losses*, que representa a perda média por época durante o treinamento, e a curva de *test_accs*, que mostra a acurácia obtida sobre o conjunto de teste após cada época.

Para facilitar a análise visual, são adicionados rótulos aos eixos, uma legenda identificando cada curva, uma grade para guiar a leitura dos valores e um título descritivo do gráfico. Essa visualização permite acompanhar a evolução do aprendizado da rede neural, avaliar sua capacidade de generalização e detectar possíveis problemas como sobreajuste (overfitting) ou subajuste (underfitting) ao longo das épocas.

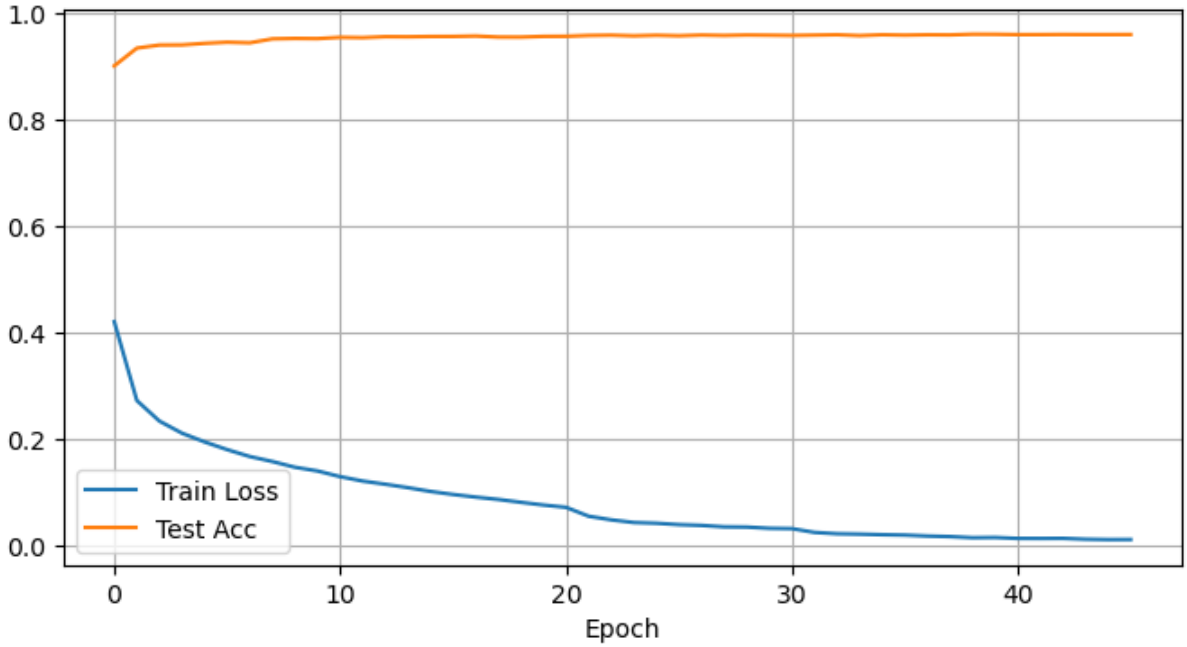


Figura 17 – Curvas de Treinamento

4.17 MATRIZ DE CONFUSÃO

Para uma análise mais detalhada do desempenho do classificador, foi construída uma matriz de confusão com base nas predições geradas sobre o conjunto de teste.

```
model.eval()
y_true, y_pred = [], []
with torch.no_grad():
    for xb, yb in test_loader:
        xb = xb.to(device)
        preds = model(xb).argmax(dim=1).cpu().numpy()
        y_pred.extend(preds.tolist())
        y_true.extend(yb.numpy().tolist())

y_true = np.array(y_true, dtype=int)
y_pred = np.array(y_pred, dtype=int)
```

Algoritmo 27 – Extração de y_true e y_pred no conjunto de teste

O código realiza a etapa de inferência do modelo já treinado, com o objetivo de avaliar seu desempenho sobre o conjunto de teste. Para otimizar esse processo e evitar o consumo desnecessário de recursos computacionais, o cálculo de gradientes é desativado por meio do comando *torch.no_grad()*, uma prática recomendada durante a validação ou predição.

Durante a inferência, o modelo percorre os dados do conjunto de teste e armazena, separadamente, os rótulos verdadeiros (*y_true*) e as predições realizadas pelo modelo (*y_pred*). Essas informações são essenciais para a construção de métricas de avaliação, como matriz de

confusão, acurácia, precisão, recall e F1-score, que possibilitam a análise quantitativa da performance do classificador.

```
n = n_classes
cm = np.zeros((n, n), dtype=int)
for t, p in zip(y_true, y_pred):
    cm[t, p] += 1
```

Algoritmo 28 – Construção manual da matriz de confusão

O código constrói uma matriz de confusão para avaliar o desempenho do modelo de classificação multiclasse. Inicialmente, é criada uma matriz quadrada de zeros com dimensões $n \times n$, onde n representa o número total de classes distintas no problema. Essa matriz será preenchida com contagens que relacionam as classes verdadeiras às classes preditas.

Durante o processo de avaliação, para cada amostra do conjunto de teste, identifica-se a classe real e a classe prevista pelo modelo. A célula correspondente à linha da classe verdadeira e à coluna da classe predita é incrementada em uma unidade. Dessa forma, a matriz resultante fornece uma visão detalhada da distribuição dos acertos e erros de classificação, permitindo identificar padrões de confusão entre classes e orientar ajustes no modelo ou no pré-processamento dos dados.

```
plt.figure(figsize=(8,6))
plt.imshow(cm, cmap='Blues', interpolation='nearest')
plt.colorbar()

ticks = np.arange(n)
plt.xticks(ticks, symbol_list, rotation=90)
plt.yticks(ticks, symbol_list)
plt.xlabel('Predito')
plt.ylabel('Verdadeiro')
plt.title('Matriz de Confusao')
plt.tight_layout()
plt.show()
```

Algoritmo 29 – Plot da matriz de confusão

A matriz de confusão gerada é visualizada por meio da biblioteca `matplotlib`, utilizando um mapa de calor em que as cores representam a densidade de ocorrências para cada par de classes verdadeira e predita. Células com tons mais intensos indicam maior número de acertos ou confusões recorrentes, enquanto tons mais claros evidenciam baixa frequência ou ausência de ocorrência.

Os rótulos dos eixos da matriz são definidos com base na variável *symbol_list*, que contém os símbolos representativos dos diferentes tipos de batimentos cardíacos presentes nos registros anotados. Essa visualização é fundamental para analisar o desempenho do modelo em

nível de classe, permitindo identificar padrões específicos de confusão e avaliar se determinadas categorias estão sendo corretamente reconhecidas ou sistematicamente mal classificadas.

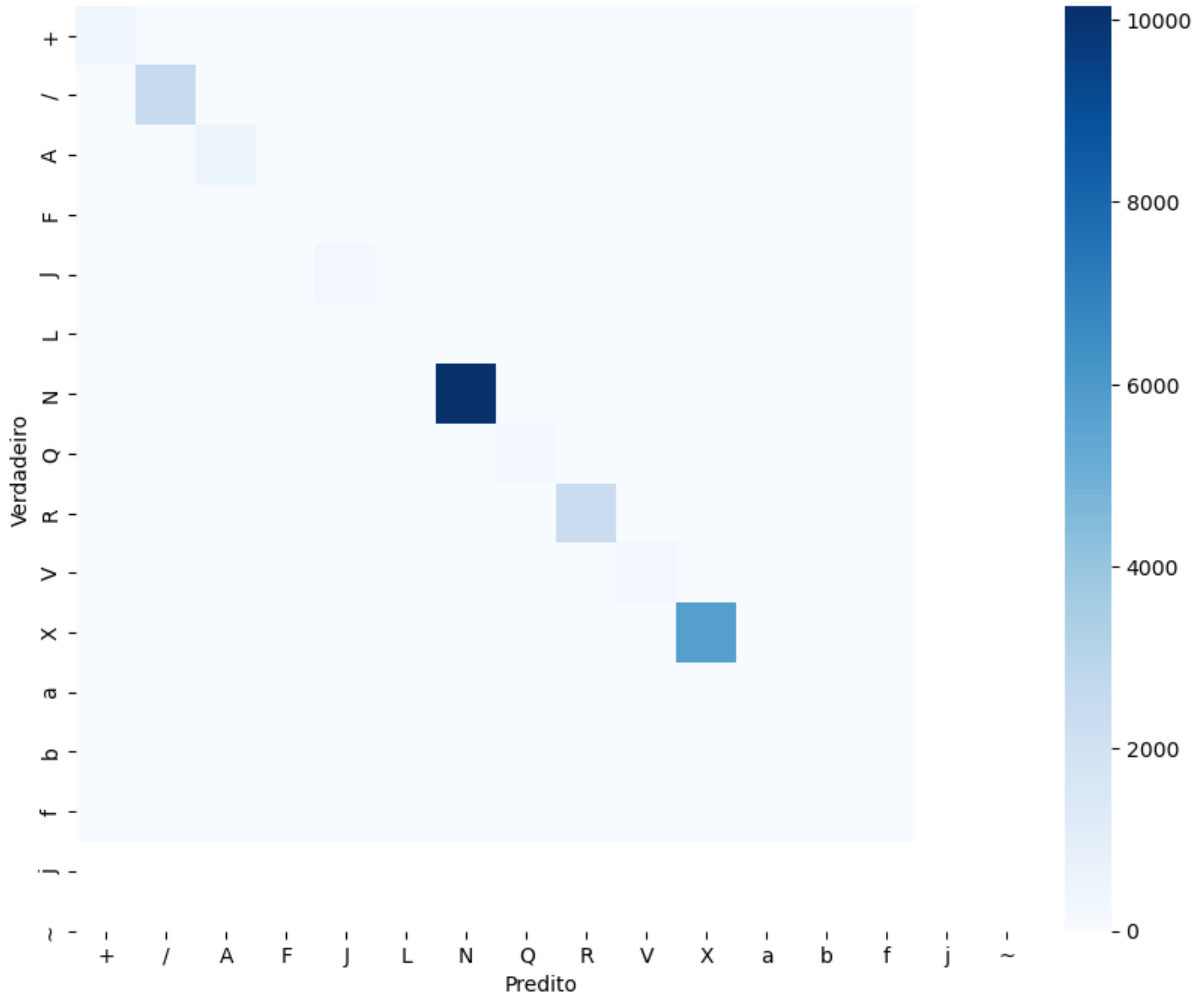


Figura 18 – Matriz de confusão.

A matriz de confusão permite visualizar os padrões de acerto e confusão entre as classes, facilitando a identificação de classes com desempenho inferior ou confundidas entre si.

4.18 CÁLCULO DE MÉTRICAS POR CLASSE

Com a matriz de confusão construída, foram extraídas métricas fundamentais para avaliação do desempenho do modelo, como *precision*, *recall*, *F1-score* e *support*, conforme apresentado abaixo:

```
precision = np.zeros(n)
recall    = np.zeros(n)
f1_score  = np.zeros(n)
support   = cm.sum(axis=1)

for i in range(n):
    tp = cm[i,i]
    fp = cm[:,i].sum() - tp
    fn = cm[i,:].sum() - tp

    precision[i] = tp / (tp + fp) if tp + fp > 0 else 0.0
    recall[i]    = tp / (tp + fn) if tp + fn > 0 else 0.0
    if precision[i] + recall[i] > 0:
        f1_score[i] = 2 * precision[i] * recall[i] / (precision[i] + recall
            [i])
```

Algoritmo 30 – Cálculo de precision, recall, F1 e suporte

Descrição das métricas:

- *Precision (Precisão):* Proporção de predições corretas entre todas as predições da classe i.
- *Recall (Revocação):* Proporção de amostras corretamente classificadas da classe i em relação ao total de amostras reais dessa classe.
- *F1-Score:* Média harmônica entre precision e recall, representando o equilíbrio entre ambos.
- *Support:* Número total de amostras da classe real (extraído da matriz de confusão).

```
print(f"{'Cls':>4} {'Prec':>6} {'Rec':>6} {'F1':>6} {'Supp':>6}")
print("-"*32)
for i, cls in enumerate(symbol_list):
    print(f"{cls:>4} {precision[i]:6.2f} {recall[i]:6.2f} {f1_score[i]:6.2f
        } {support[i]:6d}")
```

Algoritmo 31 – Impressao dos resultados por classe

	precision	recall	f1-score	support
+	0.8940	0.9492	0.9208	453
/	0.9500	0.9440	0.9470	2716
A	0.9014	0.8425	0.8710	673
F	0.3125	0.2000	0.2439	25
J	0.8725	0.8603	0.8664	358
L	0.0000	0.0000	0.0000	1
N	0.9892	0.9939	0.9916	10216
Q	0.7363	0.5697	0.6424	402
R	0.9813	0.9933	0.9873	2378
V	0.6804	0.7805	0.7271	401
X	0.9631	0.9732	0.9681	5925
a	0.6667	0.1212	0.2051	33
b	0.0000	0.0000	0.0000	0
f	0.0000	0.0000	0.0000	4
j	0.0000	0.0000	0.0000	0
~	0.5833	0.2593	0.3590	54
accuracy			0.9610	23639
macro avg	0.5957	0.5304	0.5456	23639
weighted avg	0.9594	0.9610	0.9595	23639

Figura 19 – Cálculo de Métricas por Classe

A avaliação por classe permite verificar quais tipos de batimentos foram mais bem reconhecidos pelo modelo e onde há maior confusão, sendo essencial para diagnósticos clínicos mais sensíveis, como os que envolvem arritmias raras ou batimentos anômalos.

4.19 CLASSIFICAÇÃO DE NOVOS SINAIS COM O MODELO DESENVOLVIDO

Para realizar a inferência em novos sinais de ECG, foi implementado um pipeline de classificação que reutiliza o modelo CNN1D previamente treinado e salvo em formato *scripted* via *TorchScript*. Abaixo descreve-se o fluxo completo da classificação:

```
MODEL_PATH = os.path.join(base_dir, 'ecg_v2s.pt')
fs = 977  # Hz
WINDOW_MS = 300
WINDOW_SIZE = int(fs * WINDOW_MS / 1000)
half_win = WINDOW_SIZE // 2
SYMBOL_LIST = ['/', 'A', 'V', 'N', ...]  # lista completa de classes
```

Algoritmo 32 – Configuração do ambiente

```
rec = wfdb.rdrecord(record_path)
ann = wfdb.rdann(record_path, 'atr')
sig = rec.p_signal[:,0]
```

Algoritmo 33 – Leitura dos arquivos .dat e .atr

```
beats = []
for samp in ann.sample:
    st, ed = samp - half_win, samp + half_win
    if st < 0 or ed > len(sig): continue
    beat = sig[st:ed]
    if beat.shape[0] != WINDOW_SIZE:
        beat = resample(beat, WINDOW_SIZE)
    beats.append(beat)

beats_np = np.stack(beats, axis=0).astype(np.float32)
```

Algoritmo 34 – Extração de segmentos centrados nos picos R

```
DEVICE = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = torch.jit.load(MODEL_PATH, map_location=DEVICE)
model.eval()

def classify_beats(beats_np):
    m = beats_np.mean(axis=1, keepdims=True)
    s = beats_np.std(axis=1, keepdims=True) + 1e-8
    norm = (beats_np - m) / s
    X = torch.from_numpy(norm).unsqueeze(1).to(DEVICE)  # (n, 1, window)
    logits = model(X)
    probs = torch.softmax(logits, dim=1).cpu().numpy()
    preds = np.argmax(probs, axis=1)
    return preds, probs
```

Algoritmo 35 – Classificação de novos batimentos

```
preds, probs = classify_beats(beats_np)
for i, p in enumerate(preds):
    print(f"Beat {i:3d}: classe = {SYMBOL_LIST[p]}, prob = {probs[i,p]:.3f}
        ")
```

Algoritmo 36 – Impressão das predições com probabilidade

```
unique, counts = np.unique(preds, return_counts=True)
for cls, cnt in zip(unique, counts):
    print(f"{SYMBOL_LIST[cls]:>2s}: {cnt}")
```

Algoritmo 37 – Contagem de predições por classe

Esse processo permite classificar segmentos de ECG brutos com o modelo já treinado, utilizando normalização por batimento e conversão para tensores PyTorch. Os resultados são apresentados com suas respectivas probabilidades e distribuição final das predições, oferecendo uma visão clara da inferência automatizada.

```
Beat   0: classe = '/', prob = 1.000
Beat   1: classe = 'X', prob = 0.978
Beat   2: classe = 'X', prob = 0.992
Beat   3: classe = 'X', prob = 0.980
Beat   4: classe = 'X', prob = 0.953
Beat   5: classe = 'X', prob = 0.940
Beat   6: classe = 'X', prob = 0.822
Beat   7: classe = 'X', prob = 0.827
Beat   8: classe = 'X', prob = 0.770
Beat   9: classe = 'X', prob = 0.689
Beat  10: classe = 'X', prob = 0.782
Beat  11: classe = 'X', prob = 0.919
Beat  12: classe = 'X', prob = 0.739
Beat  13: classe = '/', prob = 0.997
Beat  14: classe = 'X', prob = 0.994
Beat  15: classe = 'X', prob = 0.980
Beat  16: classe = 'X', prob = 0.996
Beat  17: classe = 'X', prob = 0.957
Beat  18: classe = 'X', prob = 0.996
Beat  19: classe = 'X', prob = 0.994

Distribuição de predições:
   / : 2
   X : 18
```

Figura 20 – Resultados apresentados com suas probabilidades distribuição final das predições.

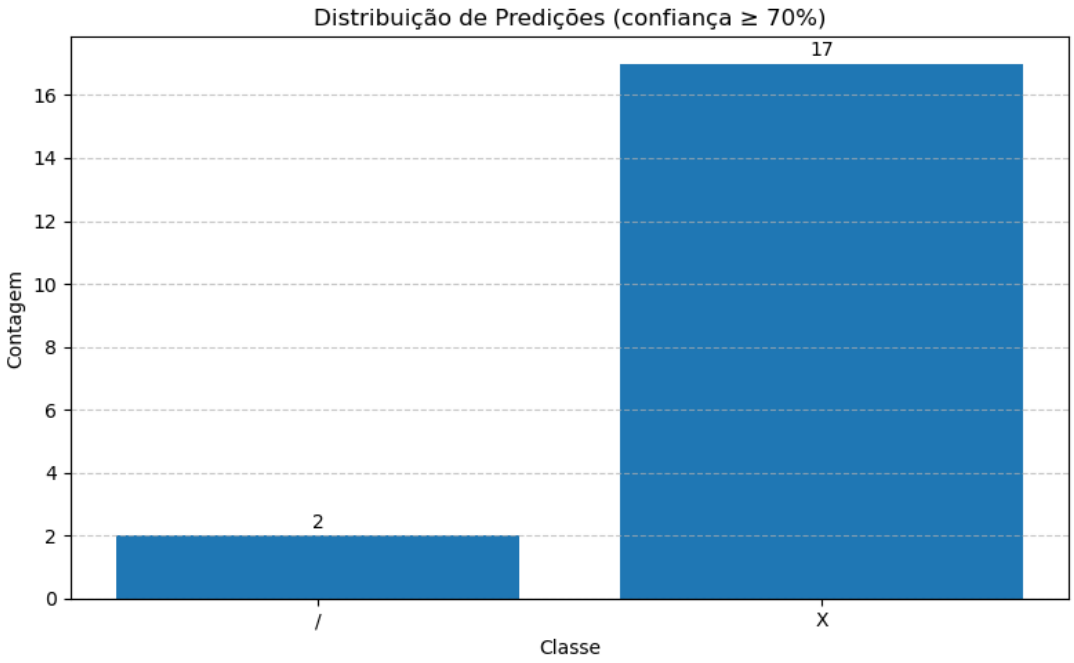


Figura 21 – Distribuição de predições.

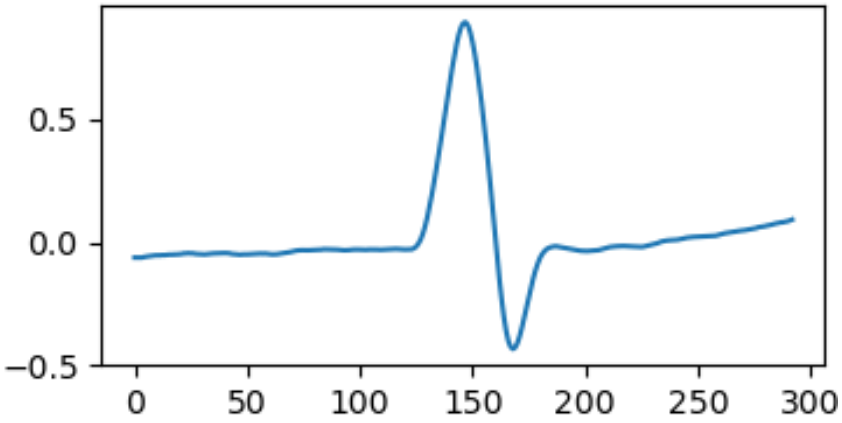


Figura 22 – Amostra batimento tipo N

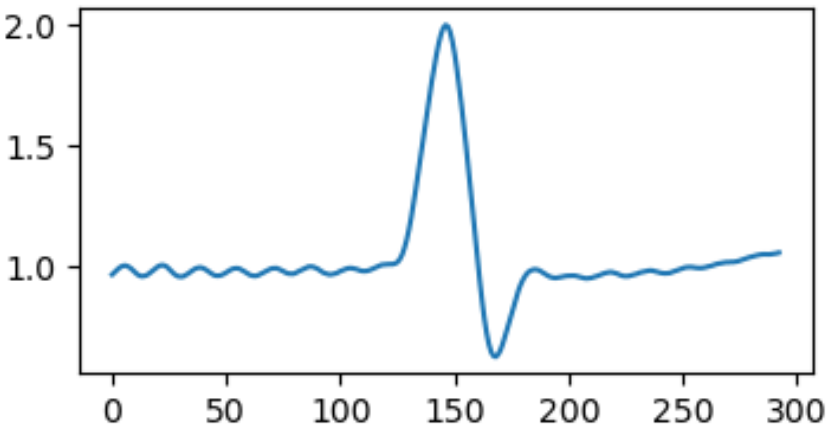


Figura 23 – Amostra batimento tipo /

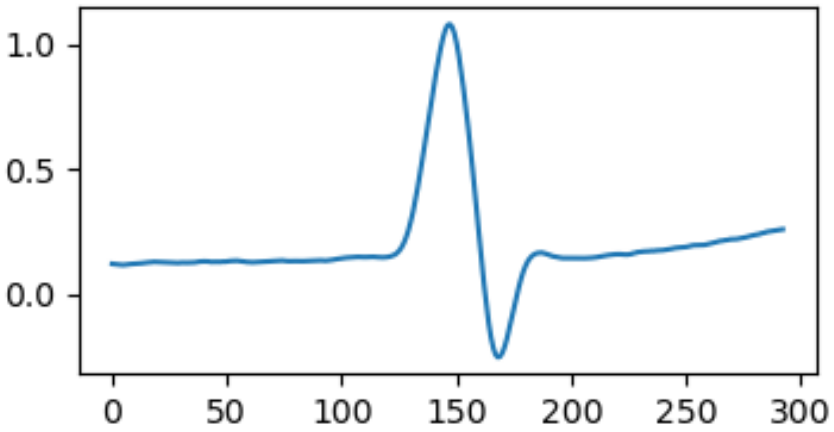


Figura 24 – Amostra batimento tipo N

5 CONCLUSÔES

Este trabalho teve como objetivo o desenvolvimento de um sistema completo e funcional para aquisição, processamento, classificação e visualização de sinais eletrocardiográficos (ECG), com foco na detecção de arritmias por meio de redes neurais convolucionais (CNNs). A proposta consistiu em integrar hardware acessível, técnicas de processamento de sinais digitais e modelos de inteligência artificial, a fim de viabilizar uma ferramenta automatizada de análise de batimentos cardíacos, aplicável a dispositivos embarcados ou vestíveis no futuro.

Durante a execução do projeto, foi implementada uma cadeia de processamento que contemplou a aquisição de sinais reais por meio do módulo AD8232 acoplado ao microcontrolador ESP32. A escolha por esse conjunto se deu por sua viabilidade prática, custo reduzido e compatibilidade com aplicações móveis. O sinal digitalizado foi submetido a um processo rigoroso de pré-processamento, incluindo filtros digitais (passa-faixa e notch), remoção de offset DC, reamostragem e normalização, garantindo que apenas informações fisiologicamente relevantes fossem mantidas para posterior análise automática.

A etapa de classificação utilizou uma rede neural convolucional unidimensional (CNN1D) treinada com dados anotados da base Leipzig Heart Center ECG-Database, um repositório clínico de alta relevância. Foram utilizados batimentos rotulados em múltiplas classes, com destaque para as três mais recorrentes: normais (N), ventriculares (V) e supraventriculares (S). A arquitetura do modelo, composta por camadas convolucionais, normalização por lote, *max pooling*, *dropout* e camada densa final, foi definida com base na literatura da área e ajustada empiricamente para equilíbrio entre desempenho e complexidade computacional.

Os resultados obtidos durante o treinamento e a validação do modelo mostraram-se satisfatórios. As métricas de desempenho — como acurácia, precisão, *recall* e F1-score — indicaram que a rede foi capaz de aprender padrões morfológicos relevantes dos batimentos cardíacos, inclusive com bom desempenho frente a dados externos. A análise da matriz de confusão evidenciou que a maior parte das classificações incorretas ocorreu entre classes de morfologia semelhante, o que é esperado em tarefas desse tipo. Além disso, foi observada estabilidade na perda ao longo das épocas e uma evolução positiva da acurácia no conjunto de teste, sugerindo ausência de sobreajuste significativo.

Um dos grandes diferenciais deste trabalho foi a aplicação da rede treinada em sinais adquiridos diretamente pelo usuário. Essa etapa de inferência prática permitiu validar, de forma qualitativa, a robustez do pipeline desenvolvido. Entretanto, durante esse teste, foi observado um resultado inesperado: os batimentos adquiridos foram classificados como pertencentes à categoria *X*, quando, na prática, esperava-se a identificação como batimentos normais (*N*). Esse equívoco na predição sugere a ocorrência de uma *falha de generalização* do modelo ao lidar

com sinais reais, possivelmente atribuída a fatores como:

- Variações morfológicas específicas do usuário que não estavam suficientemente representadas no conjunto de treinamento.
- Diferenças nos parâmetros de aquisição (ex.: qualidade dos eletrodos, ruído, ambiente).
- Possível viés de sobreajuste a padrões específicos presentes na base Leipzig.

Esse resultado destaca a necessidade de uma *análise mais aprofundada* da capacidade de generalização do modelo em ambientes reais e reforça a importância de expandir o conjunto de dados, incorporando registros de múltiplos indivíduos e contextos de aquisição distintos.

Outro aspecto importante foi a adoção de ferramentas modernas e apropriadas para cada etapa do projeto. A aquisição e programação embarcada foram realizadas no Visual Studio Code com C++ para o ESP32, enquanto todo o processamento e modelagem de IA foi conduzido no ambiente JupyterLab, aproveitando a interatividade e a rastreabilidade dos notebooks para fins acadêmicos. O uso das bibliotecas WFDB, NumPy, SciPy, PyTorch e Matplotlib permitiu um desenvolvimento modular, reprodutível e alinhado com práticas consolidadas de pesquisa científica.

Apesar dos resultados promissores, este trabalho apresenta algumas limitações. A principal delas diz respeito à quantidade de dados rotulados disponíveis para treinamento, especialmente em classes menos frequentes, o que pode impactar a capacidade de generalização do modelo para situações clínicas mais complexas.

Como perspectivas para trabalhos futuros, propõe-se:

- A expansão do conjunto de dados com aquisições reais em ambiente clínico supervisionado, incluindo novos tipos de arritmias e representatividade morfológica ampliada.
- O uso de arquiteturas de redes mais avançadas, como LSTM, GRU ou modelos baseados em Transformers, para capturar relações temporais entre batimentos.
- A submissão do projeto a processos de validação ética e certificação regulatória (ex.: ANVISA), aproximando a solução de um cenário real de aplicação clínica.

Conclui-se, portanto, que o sistema proposto apresenta viabilidade técnica e potencial de aplicação prática, demonstrando que soluções baseadas em aprendizado profundo podem ser efetivamente utilizadas para o monitoramento e a classificação automatizada de sinais cardíacos. A contribuição deste trabalho reside não apenas na construção de um protótipo funcional, mas também na identificação de desafios relevantes para a robustez e a aplicabilidade clínica dessas soluções, abrindo caminho para futuras pesquisas na interseção entre engenharia biomédica, ciência de dados e inteligência artificial aplicada à saúde.

REFERÊNCIAS

ASSIS, P. H. **Introdução ao Deep Learning: Fundamentos e Aplicações**. Rio de Janeiro: Editora Ciência Moderna, 2019. ISBN 978-8573937644.

BANZI, M.; SHILOH, M. **Getting Started with Arduino: The Open Source Electronics Prototyping Platform**. 4. ed. [S.l.]: Maker Media, Inc., 2022. ISBN 978-1680455298.

BARRETT, S. F.; PACK, D. J. **Microcontrollers Fundamentals for Engineers and Scientists**. 2. ed. San Rafael, CA: Morgan & Claypool Publishers, 2019. ISBN 978-1681734378.

CAMPOS, R. **Modelagem eletromecânica do coração com autômato celular e sistemas massa-mola**. Tese (Doutorado) — Universidade Federal de Juiz de Fora, 02 2016.

CARDOSO, A. S. V. **Instrumentação e Metodologias de Medição de Biopotenciais**. Belo Horizonte: Universidade Federal de Minas Gerais, 2010.

Data Science Academy. **O que é visão computacional?** 2022. Acesso em: 10 nov. 2024. Disponível em: <https://blog.dsacademy.com.br/o-que-e-visao_computacional/>.

DEVICES, A. **AD8232: Single-Lead, Heart Rate Monitor Front End Data Sheet**. [S.l.], 2020. Acesso em: 28 nov. 2024. Disponível em: <https://www.analog.com/media/en/technical-documentation/data-sheets/ad8232.pdf>.

ETECHNOG. **Op Amp Circuit Diagram, Types, and Working Principle**. 2019. Página da web. Acesso em: 28 nov. 2024. Disponível em: <https://www.etechnog.com/2019/01/op-amp-circuit-diagram-types-and.html>.

FERNANDES, A. P. S.; AL. et. Análise do papel do eletrocardiograma na detecção de distúrbios cardíacos. **Revista Brasileira de Cardiologia**, v. 28, n. 4, p. 265–270, 2015.

GIFFONI, R.; TORRES, R. Breve história da eletrocardiografia. **Revista Médica de Minas Gerais**, v. 20, n. 2, p. 263–270, 2010. Citada na página 18.

GURUCHARAN, M. **Basic CNN Architecture: explaining 5 layers of convolutional neural network**. 2022. Acesso em: 15 nov. 2024. Disponível em: <https://www.upgrad.com/blog/basic-cnn-architecture/>.

HABIB, M. R.; KARMAKAR, C.; YEARWOOD, J. An improved deep learning approach for ecg beat classification using attention-based cnn. **IEEE Journal of Biomedical and Health Informatics**, v. 23, n. 4, p. 1789–1797, 2019.

HAYKIN, S. **Neural Networks: A Comprehensive Foundation**. 2. ed. Prentice Hall, 1998. ISBN 9780132733502, 0132733501. Disponível em: <http://gen.lib.rus.ec/book/index.php?md5=23724f3cc14276a0e758dbabf9c9c4d4>.

IVANOVA, S. **Ritmo Cardíaco**. 2021. Ilustração vetorial (EPS). Inclui licença padrão e estendida. Acesso em: 11 abr. 2025. Disponível em: <https://www.istockphoto.com/illustration/id/1310250309>.

KLEHS, S. *et al.* **Leipzig Heart Center ECG-Database: Arrhythmias in Children and Patients with Congenital Heart Disease (version 1.0.0)**. PhysioNet, 2025. Accessed on: 1 May 2025. Disponível em: <https://doi.org/10.13026/7a4j-vn37>.

KOROL, T. **Introdução ao Machine Learning**. Rio de Janeiro: Editora Ciência Moderna, 2019. ISBN 978-8573939334.

LU, W.; LIU, Q.; CHEN, F. Challenges and methods for automated ecg-based arrhythmia detection. **Biomedical Signal Processing and Control**, v. 52, p. 235–244, 2019.

LUDERMIR, T. B. **Redes Neurais e Aprendizado de Máquina**. Rio de Janeiro: Editora LTC, 2021. ISBN 978-8521619928.

LUZ, E. J. S. *et al.* Ecg-based heartbeat classification for arrhythmia detection: A deep learning approach. **IEEE Transactions on Biomedical Engineering**, v. 67, n. 2, p. 496–504, 2020.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of ideas immanent in nervous activity. **Bulletin of Mathematical Biophysics**, v. 5, 1943. Reprinted in Anderson 1988.

Ministério da Saúde. **Manual de Eletrocardiografia: Procedimentos para Técnicos e Auxiliares**. 2002. Acesso em: 28 nov. 2024. Disponível em: <https://www.saude.gov.br>.

NISE, N. S. **Engenharia de Sistemas de Controle**. 6. ed. Rio de Janeiro: Editora LTC, 2013.

OLIVEIRA, B. R. de. **Detecção de complexos QRS em eletrocardiogramas baseada na decomposição em valores singulares em multirresolução**. Ilha Solteira: Universidade Estadual Paulista Julio de Mesquita Filho, 2015.

ROCHA, G. M. d. **Deteção de arritmias cardíacas em eletrocardiogramas usando deep learning**. Tese (Doutorado) — Instituição não especificada, 2018. Citado 2 vezes nas páginas 16 e 24.

SANTANA, J. R. G. e. a. **Classificação de arritmias cardíacas em sinais de ECG utilizando redes neurais profundas**. Dissertação (Mestrado) — Universidade Federal do Amazonas, 2021. Citado 6 vezes nas páginas 15, 16, 22, 27, 39 e 42.

SCHONS, J. Deep learning: Estruturas e aplicações. **Revista de Inteligência Computacional**, v. 12, n. 2, p. 123–138, 2018.

SEDRA, A. S.; SMITH, K. C. **Microeletrônica**. 6. ed. Rio de Janeiro: Editora LTC, 2007.

SYSTEMS, E. **ESP32 Series Datasheet**. 2022. Acesso em: 28 nov. 2024. Disponível em: <https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf>.

SZPALHER, A. S.; BATALHA, M. C. Arritmias cardíacas: Diagnósticos de enfermagem baseados na taxonomia da nanda-i (2018-2020). **Revista Eletrônica Acervo Saúde**, v. 11, n. 17, p. e1447–e1447, 2019. Citado na página 15.

TEIXEIRA, J. **Circuitos Elétricos e Eletrônicos: Fundamentos e Aplicações**. São Paulo: Editora Técnica, 2017. 79 p.

TORTORA, G. J.; DERRICKSON, B. **Princípios de Anatomia e Fisiologia**. 14. ed. Rio de Janeiro: Guanabara Koogan, 2016. ISBN 978-85-277-2862-1.

WALLER, A. A demonstration on man of electromotive changes accompanying the heart's beat. **The Journal of Physiology**, Wiley-Blackwell, v. 8, n. 5, p. 229, 1887. Citada na página 18.