UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

SARAH MORGANA MEURER

DETECÇÃO E CLASSIFICAÇÃO DE PATOLOGIAS CARDÍACAS EM ELETROCARDIOGRAMAS UTILIZANDO REDES NEURAIS PROFUNDAS

TOLEDO

2022

SARAH MORGANA MEURER

DETECÇÃO E CLASSIFICAÇÃO DE PATOLOGIAS CARDÍACAS EM ELETROCARDIOGRAMAS UTILIZANDO REDES NEURAIS PROFUNDAS

Detection and classification of cardiac pathologies in electrocardiograms using deep neural networks

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia Eletrônica do Curso de Engenharia Eletrônica da Universidade Tecnológica Federal do Paraná.

Orientador: Eduardo Vinícius Kuhn

Coorientador: Daniel Gomes de Pinho Zanco

TOLEDO

2022

TERMO DE APROVAÇÃO

DETECÇÃO E CLASSIFICAÇÃO DE PATOLOGIAS CARDÍACAS EM ELETROCARDIOGRAMAS UTILIZANDO REDES NEURAIS PROFUNDAS

Sarah Morgana Meurer

Este Trabalho de Conclusão de Curso foi apresentado como requisito parcial para a obtenção do título Bacharel em Engenharia Eletrônica. Após deliberação da Banca Examinadora, composta pelos professores abaixo assinados, o trabalho foi considerado APROVADO.

Toledo, 14 de dezembro de 2022.

Prof. Felipe Walter Dafico Pfrimer
Coordenador do Curso

Banca Examinadora:

Alberto Vinícius de Oliveira, Dr.
Universidade Tecnológica Federal do Paraná (UTFPR)

Ranniery da Silva Maia, Dr.
Universidade Federal do Rio Grande do Norte (UFRN)

Eduardo Vinícius Kuhn, Dr. (orientador)
Universidade Tecnológica Federal do Paraná (UTFPR)

Daniel Gomes de Pinho Zanco, Me. (coorientador)
Profissional externo, Universidade Federal de Santa Catarina (UFSC)

AGRADECIMENTOS

Desejo expressar meus agradecimentos a todos que de alguma forma auxiliaram no desenvolvimento e conclusão do presente trabalho.

Em especial, agradeço a minha família, meus pais Davi Meurer e Marta Madalena Meurer, minha irmã Caroline Mayara Meurer Reolon e meu namorado Vinicius Alexandre Bertoldi Boff, por todo o suporte, apoio emocional e financeiro, pela paciência e por sempre acreditarem em mim e no sucesso deste trabalho. Sem vocês, nada disso seria possível.

Agradeço ao meu orientador Eduardo Kuhn e coorientador Daniel Zanco por terem aceito a tarefa de me orientar no decorrer desses anos de pesquisa, por todos os valiosos ensinamentos, experiências compartilhadas, confiança, apoio à pesquisa e disposição em me auxiliar sempre que necessário.

Agradeço à UTFPR – Campus Toledo e a todos os professores que foram fundamentais na minha formação acadêmica e profissional.

Agradeço aos membros da banca examinadora por terem aceito o convite de apreciar o trabalho, assim como pelas considerações, contribuições e sugestões.

Agradeço aos meus amigos Emilly Krepkij, Augusto Becker, Dyorgyo Valesan, Alef Fraga e Gabriel Pichek por toda a amizade, companhia e apoio no decorrer dos anos da graduação, sempre me fornecendo ajuda quando necessário.

Por fim, agradeço a todos que de alguma forma contribuíram para minha formação.

RESUMO

Doenças cardiovasculares vêm figurando, no decorrer das últimas décadas, dentre as principais causas de morte registradas no mundo. Tais doenças podem ser detectadas/diagnosticadas por um profissional de saúde através de um exame não invasivo denominado eletrocardiograma (ECG), o qual fornece uma representação da atividade elétrica do coração. Entretanto, a interpretação do ECG não é uma tarefa trivial, demandando considerável experiência do profissional de saúde na identificação apropriada de alterações morfológicas. Nesse contexto, técnicas de aprendizado de máquina (em especial, aprendizado profundo) vêm sendo amplamente utilizadas, na área de eletrofisiologia clínica, para criar modelos matemáticos capazes de detectar automaticamente padrões que caracterizam a presença ou ausência de patologias cardíacas. Contudo, apesar do bom desempenho alcançado, poucos trabalhos apresentados até então utilizam conjuntos de dados públicos no treinamento das arquiteturas, dificultando assim a validação dos resultados, comparações de desempenho e proposição de melhorias. Diante disso, o presente trabalho foca sobre o desenvolvimento de uma ferramenta computacional para detecção e classificação automática multirrótulos de 5 superclasses de patologias cardíacas. Especificamente, as arquiteturas introduzidas em (RAJPURKAR *et al.*, 2017) e (RIBEIRO *et al.*, 2020) são implementadas, utilizando linguagem Python juntamente as bibliotecas Tensorflow e Keras, e treinadas considerando um conjunto de dados público já rotulado por cardiologistas, denominado PTB-XL (WAGNER *et al.*, 2020). O desempenho dos modelos obtidos (após o treinamento) é avaliado através da matriz de confusão multirrótulos (MLCM), a partir da qual as métricas precisão, sensibilidade e F1-*score* assim como seus valores médios *micro*, *macro* e *weighted* podem ser computados, conforme metodologia introduzida em (HEYDARIAN; DOYLE; SAMAVI, 2022). Ainda, comparações envolvendo o custo computacional para a implementação e treinamento das arquiteturas são apresentadas, abordando a quantidade de parâmetros, o número de épocas e o tempo transcorrido até o encerramento do treinamento. Os resultados obtidos mostraram que o modelo de (RIBEIRO *et al.*, 2020) apresenta um desempenho similar, ou ainda ligeiramente melhor, para as diferentes métricas consideradas do que aquele obtido a partir da arquitetura de (RAJPURKAR *et al.*, 2017); sobretudo, ao se levar em conta o custo computacional envolvido.

Palavras-chave: aprendizado profundo; eletrocardiograma; patologias cardíacas; redes neurais convolucionais.

ABSTRACT

Cardiovascular diseases have been, in the last decades, among the main causes of mortality worldwide. Such diseases can be detected/diagnosed by a health professional through a noninvasive test called an electrocardiogram (ECG), which provides a representation of the electrical activity of the heart. Nonetheless, the interpretation of the ECG is not a trivial task, demanding considerable experience from the health professional to properly identify morphological changes. In this context, machine learning (especially, deep learning) techniques have been widely used, in the field of clinical electrophysiology, to construct mathematical models which are able to automatically detect patters associated with the presence or absence of cardiac pathologies. However, despite the good performance achieved, only a few research works published so far have used public datasets for training the introduced architectures, thus making it difficult to validate the results, to obtain fair performance comparisons, as well as to propose improvements. So, the present work focuses on the development of a computational tool for automatic detection and classification of 5 super-classes of cardiac pathologies. Specifically, the architectures introduced in (RAJPURKAR *et al.*, 2017) and (RIBEIRO *et al.*, 2020) are implemented here, using Python language along with Tensorflow and Keras modules, and trained with a public dataset already labeled by cardiologists, named PTB-XL (WAGNER *et al.*, 2020). The performance of the obtained models (after training) is assessed through the multilabel confusion matrix (MCLM), from which some metrics like precision, recall, and F1-score as well as their average values termed micro, macro, and weighted can be computed, according to the methodology described in (HEYDARIAN; DOYLE; SAMAVI, 2022). Still, comparisons involving the computational cost required for implementing and training both architectures are presented, highlighting the number of parameters, number of epochs, and time elapsed until the end of the training. Results showed that the model obtained from the architecture described by (RIBEIRO *et al.*, 2020) presents a similar performance, or even slightly better, for different metrics than the one obtained from the architecture introduced by (RAJPURKAR *et al.*, 2017); especially, when the involved computational cost is taken into account.

Keywords: deep learning; electrocardiogram; cardiac pathologies; convolutional neural networks.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

1D	Uma dimensão
2D	Duas dimensões
3D	Três dimensões
BN	*Batch normalization*
CD	Distúrbios de condução
CNN	Rede neural convolucional
Conv	Convolucional
DNN	Rede neural profunda
ECG	Eletrocardiograma
FN	Falso negativo
FP	Falso positivo
HYP	Hipertrofia
MI	Infarto do miocárdio
MLCM	Matriz de confusão multirrótulos
MLP	Perceptron multicamadas
NORM	Normal
NPL	*No predicted label*
NTL	*No true label*
ReLU	*Rectified linear unit*
ResNet	Rede neural residual
TN	Verdadeiro negativo
TP	Verdadeiro positivo

LISTA DE SÍMBOLOS

w_l	Coeficiente da camada de índice l
ϵ	Constante
$\mathcal{N}$	Distribuição gaussiana
x	Entrada da camada
$\hat{m}_t$	Estimativa não polarizada do primeiro momento
$\hat{v}_t$	Estimativa não polarizada do segundo momento
$\mathcal{L}$	Função custo de entropia cruzada binária
$f(x)$	Função de ativação
$\tilde{x}_i$	i-ésimo dado ajustado pela normalização
x_i	i-ésimo dado de entrada da camada
$\mathbf{M}$	Matriz de confusão multirrótulos
μ_β	Média do *mini-batch*
m_t	Média móvel exponencial do gradiente
v_t	Média móvel exponencial do gradiente ao quadrado
C	Número de classes
n_l	Número de unidades da camada l
N_c	Número total de rótulos preditos
θ_t	Parâmetro atualizado
θ_{t-1}	Parâmetro da iteração anterior
$\mathcal{F}(x)$	Resíduo
$\hat{y}_c$	Rótulo predito da c-ésima classe
y_c	Rótulo verdadeiro da c-ésima classe
$\mathcal{H}(x)$	Saída do bloco residual
α	Taxa de aprendizagem
β_1	Taxa de decaimento exponencial atribuído à média dos gradientes
β_2	Taxa de decaimento exponencial atribuído à média dos quadrados dos gradientes
σ_β^2	Variância do *mini-batch*

SUMÁRIO

1 INTRODUÇÃO

No decorrer das últimas décadas, doenças cardiovasculares vêm figurando dentre as principais causas de morte registradas no mundo. Estima-se que, apenas em 2019, 17,9 milhões de pessoas morreram por decorrência de doenças cardiovasculares, representando 32% das mortes registradas ao redor do mundo (WORLD HEALTH ORGANIZATION, 2021). Essas doenças, muitas vezes, podem ser detectadas/diagnosticadas por um profissional de saúde (por exemplo, um médico cardiologista) através de um exame cardíaco não invasivo, denominado eletrocardiograma (ECG). O ECG consiste em uma forma de representação da atividade elétrica do coração em um dado intervalo, contendo assim informações essenciais para a análise do ritmo cardíaco bem como da morfologia dos batimentos (SANNINO; PIETRO, 2018). Todavia, a interpretação de sinais de ECG não é uma tarefa trivial e depende da experiência do profissional de saúde para identificar (visualmente) alterações de padrão (mesmo as mais sutis); especificamente, cardiologistas costumam identificar de 53% a 96% das anomalias, enquanto médicos não-cardiologistas de 36% a 96% (SALERNO *et al.*, 2003; MELE, 2008; CAIRNS *et al.*, 2016). Tal dificuldade na interpretação adequada do ECG se deve, sobretudo, à semelhança de morfologias encontradas em diferentes condições cardíacas, o que pode resultar em importantes erros de diagnóstico (seja por falta de experiência do profissional ou por fadiga humana) (SALERNO; ALGUIRE; WAXMAN, 2003; ACHARYA *et al.*, 2017; SOMANI *et al.*, 2021). Vale destacar que cerca de 33% das interpretações de ECG contêm erros de diagnóstico de grande relevância, conforme apontado por (SALERNO; ALGUIRE; WAXMAN, 2003).

Visando auxiliar os profissionais de saúde e melhorar a frequência de acertos dos diagnósticos das patologias cardíacas, ferramentas computacionais vêm sendo continuamente aprimoradas e cada vez mais utilizadas em análises clínicas para superar as dificuldades existentes (MINCHOLÉ *et al.*, 2019). Em particular, na interpretação de sinais de ECG, técnicas de aprendizado de máquina (especificamente, aprendizado profundo[1]) vêm sendo amplamente empregadas na área de eletrofisiologia clínica para construir redes neurais (isto é, modelos matemáticos) capazes de detectar automaticamente padrões e identificar alterações morfológicas (SHAMEER *et al.*, 2018; MINCHOLÉ *et al.*, 2019). Isso se deve, sobretudo, aos avanços tecnológicos (em termos de *hardware* e *software*) ocorridos nos últimos anos, possibilitando a concepção e implementação de arquiteturas de redes neurais mais complexas, assim como aos grandes conjuntos de dados disponíveis, obtidos a partir de registros eletrônicos de saúde (MINCHOLÉ *et al.*, 2019). Como resultado, um importante número de trabalhos de pesquisa (RAJPURKAR *et al.*, 2017; XIONG; STILES; ZHAO, 2017; GALLOWAY *et al.*, 2019; ATTIA *et al.*, 2019; LEUR *et al.*, 2020; RIBEIRO *et al.*, 2020), utilizando diferentes arquiteturas, pode ser encontrado na literatura versando sobre classificação de patologias cardíacas envolvendo sinais de ECG; para

[1] O aprendizado profundo surgiu como uma solução para contornar as limitações do aprendizado de máquina no processamento de dados na forma bruta; em resumo, o aprendizado de máquina (convencional) requer um extrator de atributos para reconhecer padrões a partir de dados brutos, enquanto no aprendizado profundo tais atributos são aprendidos durante o próprio treinamento (LECUN; BENGIO; HINTON, 2015).

mais detalhes, veja a extensa revisão apresentada em (SOMANI *et al.*, 2021). Dentre tais trabalhos, dois considerados relevantes são destacados e discutidos a seguir.

Uma arquitetura de rede neural profunda (DNN) capaz de detectar/classificar 12 tipos distintos de arritmias em sinais de ECG (brutos) é descrita em (RAJPURKAR *et al.*, 2017). Tal arquitetura conta com 34 camadas, sendo 33 camadas convolucionais (do inglês, *convolutional*) seguidas por uma camada totalmente conectada (do inglês, *fully connected*); por fim, a função de ativação *softmax* é usada para indicar uma dentre as diferentes classes possíveis. Como forma de otimizar o treinamento, as camadas da rede foram dispostas em 16 blocos residuais com conexões de atalho (do inglês, *residual blocks*), contendo 2 camadas convolucionais por bloco. As métricas de desempenho consideradas correspondem à precisão, sensibilidade e F1-*score*, alcançando valores de 80,9%, 82,7% e 80,9%, respectivamente. Vale mencionar que, posteriormente, essa mesma arquitetura foi aplicada na identificação de arritmias em tempo real (HANNUN *et al.*, 2019). Por outro lado, (RIBEIRO *et al.*, 2020) apresenta uma arquitetura de DNN que permite detectar 6 tipos de anormalidades (de ritmo e morfologia) consideradas representativas pelos autores. Essa arquitetura possui apenas 10 camadas, consistindo de 1 camada convolucional de entrada, seguida de 4 blocos residuais, contendo 2 camadas convolucionais por bloco e conexões de atalho, e 1 camada totalmente conectada; ainda, na camada de saída, a função de ativação *sigmoid* é utilizada, dado que as diferentes classes não são assumidas mutuamente exclusivas (isto é, um paciente pode apresentar mais de uma patologia cardíaca). O desempenho alcançado está acima de 80% para o F1-*score* e acima de 99% para a especificidade. Contudo, apesar do bom desempenho alcançado, os conjuntos de dados utilizados no treinamento das arquiteturas apresentadas em (RAJPURKAR *et al.*, 2017) e (RIBEIRO *et al.*, 2020) não são públicos, o que dificulta comparações de desempenho assim como a proposição de melhorias para as arquiteturas.

1.1 Objetivos

Neste contexto, o presente trabalho de conclusão de curso trata da implementação de uma ferramenta computacional para detecção e classificação (automática) multirrótulo de 5 superclasses de patologias cardíacas a partir de sinais (brutos) de ECG, visando auxiliar os profissionais de saúde quanto ao diagnóstico de doenças e/ou condições cardiovasculares. Especificamente, busca-se aqui

i) implementar a arquitetura introduzida em (RAJPURKAR *et al.*, 2017) adaptada para o caso multirrótulo bem como aquela descrita em (RIBEIRO *et al.*, 2020), utilizando a linguagem Python (ROSSUM; DRAKE, 2009), juntamente as bibliotecas Tensorflow (ABADI *et al.*, 2015) e Keras (CHOLLET *et al.*, 2015);

ii) realizar o treinamento das arquiteturas implementadas, considerando (exclusivamente) o conjunto de dados, público e já rotulado por cardiologistas, disponível em (WAGNER *et al.*, 2020); e

iii) verificar o desempenho das arquiteturas implementadas por meio de métricas de avaliação adequadas para o caso de classificação multirrótulo, conforme apropriadamente descrito em (HEYDARIAN; DOYLE; SAMAVI, 2022).

1.2 Organização do documento

O presente trabalho de conclusão de curso está organizado como segue. A Seção 2 traz uma revisão breve acerca da interpretação de sinais de ECG e revisita alguns conceitos fundamentais sobre redes neurais convolucionais e residuais. A Seção 3 descreve as metodologias adotadas para o desenvolvimento do classificador de patologias cardíacas, abordando o processamento do conjunto de dados, as arquiteturas de referência implementadas e as métricas de avaliação utilizadas para avaliação de desempenho. Por sua vez, a Seção 4 apresenta e discute os resultados de simulação obtidos na tarefa considerada, contemplando a apresentação da matriz de confusão multirrótulos, o desempenho frente às métricas consideradas e uma análise do custo computacional envolvido no treinamento. Para encerrar, a Seção 5 traz algumas considerações finais deste trabalho, destacando as principais conclusões alcançadas, a lista dos trabalhos publicados e/ou em vias de publicação, assim como algumas sugestões para futuros trabalhos de pesquisa na área.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção, alguns aspectos importantes são brevemente discutidos visando fornecer uma base teórica para o entendimento do problema em questão. Primeiramente, a Seção 2.1 relembra alguns conceitos relacionados a atividade elétrica do coração bem como sobre a interpretação de sinais de ECG. Em seguida, na Seção 2.2, conceitos fundamentais sobre redes neurais convolucionais (CNNs), necessários para o desenvolvimento do presente trabalho, são revisitados. Por último, a Seção 2.3 apresenta uma discussão sobre o conceito de rede neural residual (ResNet).

2.1 Sobre a atividade elétrica do coração e o eletrocardiograma

O coração é formado por dois átrios e dois ventrículos, sendo responsável por bombear o sangue venoso para os pulmões e o sangue oxigenado para os órgãos periféricos. Essa atividade acontece por meio da contração e relaxamento do músculo cardíaco, caracterizando o ciclo cardíaco. No período de relaxamento, denominado diástole, as células cardíacas são polarizadas (carregadas) eletricamente, enchendo assim os átrios de sangue. Em seguida, as células se despolarizam, tratando-se do período de contração do músculo cardíaco, chamado de sístole, propelindo então o sangue dos ventrículos para a circulação pulmonar ou periférica (DUBIN, 1996; GUYTON; HALL, 2006). Essa atividade (elétrica) do coração, quando capturada por sensores cutâneos (eletrodos), costuma ser representada/registrada através de um ECG (DUBIN, 1996); geralmente, a aquisição desse sinal de ECG se dá através de sensores dispostos em pontos específicos do corpo, compondo usualmente 12 derivações (6 periféricas e 6 precordiais)[1]. Um sinal de ECG considerado normal (tal como o ilustrado na Figura 1) é composto pela onda P, complexo QRS e onda T, cujos traçados caracterizam os eventos associados ao ciclo cardíaco. Especificamente, o segmento P representa a despolarização nos átrios, registrando o potencial elétrico gerado pela contração atrial. Por sua vez, o segmento QRS é o registro do potencial elétrico gerado pela despolarização dos ventrículos, representando a contração ventricular. Por fim, o segmento T representa a repolarização dos ventrículos, passando ao estado de relaxamento e encerrando o clico cardíaco (GUYTON; HALL, 2006; DUBIN, 1996). Vale mencionar que o intervalo RR é utilizado para mensurar a frequência cardíaca. Diante do exposto, torna-se evidente que um sinal de ECG serve como uma base importante para identificar a ausência ou presença de diferentes patologias cardíacas que (de alguma forma) provocam alterações na frequência/ritmo e/ou na morfologia do sinal (veja exemplos na Seção 3.1 à frente).

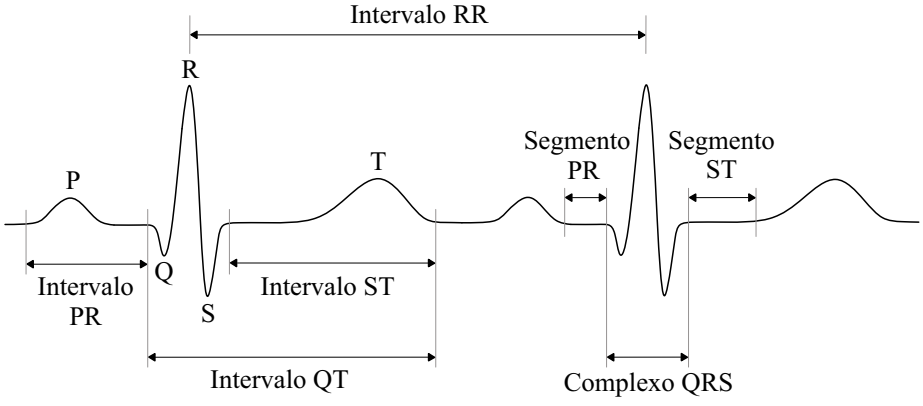
[1] As derivações periféricas são obtidas por eletrodos posicionados sobre os braços direito e esquerdo e sobre a perna esquerda, formando 6 eixos de referência cruzando o tórax do paciente e provendo uma visualização da atividade do coração em um plano vertical. Já as derivações precordiais são obtidas dispondo eletrodos, de forma sequencial, em 6 diferentes posições do tórax do paciente, visando fornecer uma visualização da atividade do coração em um plano horizontal (DUBIN, 1996).

2.2 Conceitos gerais sobre redes neurais convolucionais

Uma rede neural convolucional (CNN) costuma ser utilizada para processar dados com informação espacial, tal como sinais e sequências em uma dimensão (1D), imagens e espectrogramas de áudio em duas dimensões (2D) ou, ainda, vídeos e imagens volumétricas em três dimensões (3D) (LECUN; BENGIO; HINTON, 2015). Tais redes são frequentemente utilizadas em tarefas de classificação, buscando aprender, a partir de um conjunto de dados já rotulado, um modelo capaz de predizer rótulos de exemplos/instâncias/amostras fora do conjunto de dados (HERRERA *et al.*, 2016). A tarefa de classificação se dá em função da forma dos atributos de saída, de modo que na classificação multirrótulos, cada exemplo do conjunto de dados é associado a um vetor de saída, cujo tamanho é fixo e corresponde ao número de classes do conjunto de dados. Os elementos desse vetor assumem valores binários, indicando se uma determinada classe corresponde ao exemplo. Dessa forma, na classificação multirrótulos, é possível que o exemplo seja classificado em mais de uma classe, isto é, as diferentes classes não são assumidas mutuamente exclusivas (HERRERA *et al.*, 2016).

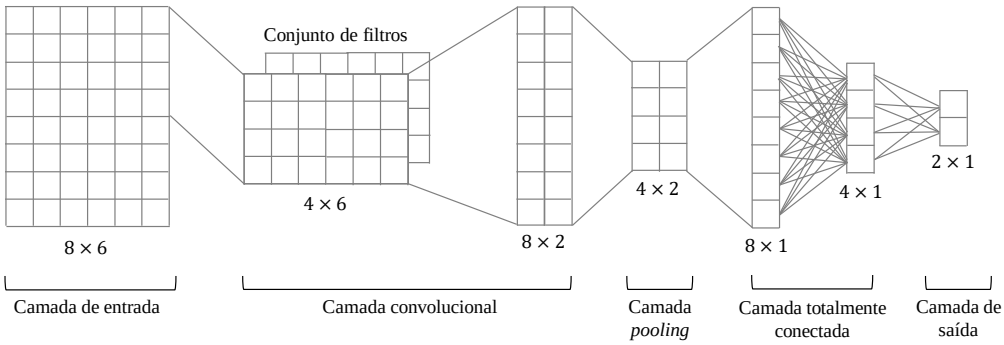
A Figura 2 mostra uma arquitetura típica de uma CNN utilizada em tarefas de classificação, sendo a camada de entrada uma matriz (tensor 2D) contendo o sinal e a camada de saída dada por um vetor (tensor 1D) cujos elementos correspondem às classes distintas consideradas. Note que a dimensão da entrada é dada em função do comprimento da sequência do sinal e do número de canais (PASZKE *et al.*, 2019); por exemplo, o sinal de ECG possui uma dimensão temporal, equivalente ao número de amostras obtido da amostragem, e uma dimensão espacial, equivalente às diferentes derivações. Entre as camadas de entrada e saída, tem-se comumente três tipos, a saber: convolucional, *pooling* e totalmente conectada (GU *et al.*, 2018). A camada convolucional (Conv) é composta por um conjunto de filtros, cujos coeficientes (também denominados pesos) são otimizados durante o processo de aprendizagem. Essa camada desempenha uma operação de filtragem linear, visando reconhecer o mesmo padrão em diferentes partes da

Figura 1 – Exemplo de um sinal de ECG considerado dentro da normalidade, com destaque aos diferentes eventos e/ou segmentos que compreendem um ciclo cardíaco.



Fonte: Adaptado de (GUYTON; HALL, 2006).

Figura 2 – Arquitetura típica de rede neural convolucional.



Fonte: Adaptado de (KIM, 2014).

entrada (LECUN; BENGIO; HINTON, 2015). Isso ocorre através da operação de correlação cruzada[2] entre a entrada e os coeficientes dos filtros. Essa operação pode ser controlada por um determinado passo (do inglês, *stride*) (FAWAZ *et al.*, 2019), o qual pode ser aumentado para reduzir o custo computacional decorrente do treinamento provocando assim uma subamostragem da entrada. Eventualmente, a técnica de *zero padding* no modo *same* pode ser aplicada, preenchendo a entrada com uma quantidade de zeros de forma a manter o tamanho da saída igual ao tamanho da entrada (GOODFELLOW; BENGIO; COURVILLE, 2016). Os atributos resultantes da operação são organizados em uma matriz, denominada mapa de atributos, a qual serve como entrada para a camada seguinte da rede. Por sua vez, a camada *pooling* introduz a invariância do modelo ao deslocamento espacial da entrada; geralmente, são utilizadas após a camada convolucional objetivando a redução da quantidade de coeficientes das camadas seguintes através da subamostragem do mapa de atributos. Uma função de *pooling* comumente utilizada é a *max pooling*, retornando o valor máximo de regiões delimitadas e espaçadas, por um determinado passo, do mapa de atributos (GOODFELLOW; BENGIO; COURVILLE, 2016). Através da operação de *pooling*, é possível minimizar o *overfitting*[3] e melhorar a eficiência computacional. Por fim, a camada totalmente conectada, ou densa (do inglês, *dense*), consiste na arquitetura perceptron multicamadas (MLP), onde cada unidade da camada é conectada, através de coeficientes, a todas as unidades da camada seguinte. A última camada totalmente conectada corresponde à camada de saída da CNN, gerando uma distribuição de probabilidades para cada rótulo, ou classe, do conjunto de dados (FAWAZ *et al.*, 2019). Vale salientar ainda que melhorias, tais como a escolha de diferentes funções de ativação e de funções custo, inclusão de regularização e otimização (discutidas a seguir), vêm sendo incorporadas nas CNNs no decorrer dos anos a fim de aprimorar o desempenho.

[2] Matematicamente, a correlação cruzada pode ser vista como uma convolução discreta sem a inversão da ordem dos coeficientes.

[3] O *overfitting* ocorre quando o modelo se ajusta exatamente aos seus dados de treinamento, perdendo sua capacidade de generalização (GOODFELLOW; BENGIO; COURVILLE, 2016).

2.2.1 Função de ativação

A função de ativação é utilizada para melhorar a capacidade representativa da rede neural, introduzindo um componente não linear (WANG *et al.*, 2020). A função *Rectified Linear Unit* (ReLU) consiste em uma das funções de ativação mais utilizadas, sendo definida como (WANG *et al.*, 2020)

$$f(x) = \max(0, x) \tag{1}$$

onde x denota a entrada da função de ativação. A função ReLU, para valores de entrada menores ou iguais a zero, resulta em uma saída igual a zero; caso contrário, a saída é igual ao valor de entrada. Essa operação de forçar o resultado a zero para determinados valores de entrada aumenta a velocidade do treinamento, calculando seu valor mais rapidamente em comparação com outras funções de ativação (WANG *et al.*, 2020).

2.2.2 Função custo

No processo de aprendizagem, busca-se determinar os parâmetros ótimos para a tarefa desempenhada pela rede. Tais valores são obtidos minimizando uma função custo apropriada para a tarefa, considerando as características dos dados de entrada. Em tarefas de classificação multirrótulos, onde cada exemplo do conjunto de dados pode enquadrar-se em um ou mais rótulos, a função custo de entropia cruzada binária (do inglês, *binary cross-entropy*) pode ser utilizada, a qual é definida por (CAI *et al.*, 2020)

$$\mathcal{L} = -\sum_{c}^{C} \left[y_c \log(\hat{y}_c) + (1 - y_c)(1 - \log(\hat{y}_c)) \right] \tag{2}$$

onde C denota o número de classes, $\hat{y}_c$ consiste no rótulo predito e y_c no rótulo verdadeiro da c-ésima classe. O resultado da função custo corresponde à soma das probabilidades de cada rótulo na camada de saída da rede neural.

2.2.3 Dropout

A técnica de regularização *dropout* tem por objetivo reduzir os efeitos de *overfitting* no treinamento. Tal técnica consiste na remoção temporária de unidades da rede, diminuindo a sua dependência à coeficientes específicos. Com isso, o *dropout* força a rede a desempenhar sua tarefa com parte das informações removidas, visando melhorar a sua capacidade de generalização. A seleção das unidades a serem removidas é aleatória, sendo determinada através do hiperparâmetro de taxa de *dropout* que consiste na probabilidade de retenção de cada unidade (SRIVASTAVA *et al.*, 2014).

2.2.4 Inicialização

A inicialização adequada dos parâmetros é um aspecto crucial para assegurar a convergência da rede neural (sobretudo, quando composta por muitas camadas) (GU *et al.*, 2018). Nesse sentido, o método de inicialização dos coeficientes introduzido em (HE *et al.*, 2015b) foi desenvolvido para redes neurais com função de ativação ReLU, cuja inicialização é dada por variáveis aleatórias de uma distribuição gaussiana de média zero, isto é,

$$w_l \sim \mathcal{N}\left(0, \frac{2}{n_l}\right) \tag{3}$$

onde w_l denota o coeficiente da camada de índice l a ser inicializado, $\mathcal{N}$ indica a distribuição gaussiana e n_l é o número de unidades daquela camada. Os elementos de viés de cada camada são inicializados em zero. Essa técnica de inicialização busca evitar a redução ou ampliação exponencial da magnitude dos sinais de entrada, como forma de evitar o problema de gradientes explosivos.

2.2.5 Batch normalization

A camada *batch normalization* (BN) lida com os efeitos de perda gradual da capacidade de aprendizado e acurácia, decorrentes da mudança nas características dos dados conforme fluem pelas diferentes camadas da rede (IOFFE; SZEGEDY, 2015; GU *et al.*, 2018). Em particular, tal camada introduz uma etapa de normalização responsável por ajustar a média e a variância dos dados de entrada das diferentes camadas. Esse ajuste é realizado a cada *mini-batch* e pode ser expresso matematicamente através da seguinte relação (IOFFE; SZEGEDY, 2015):

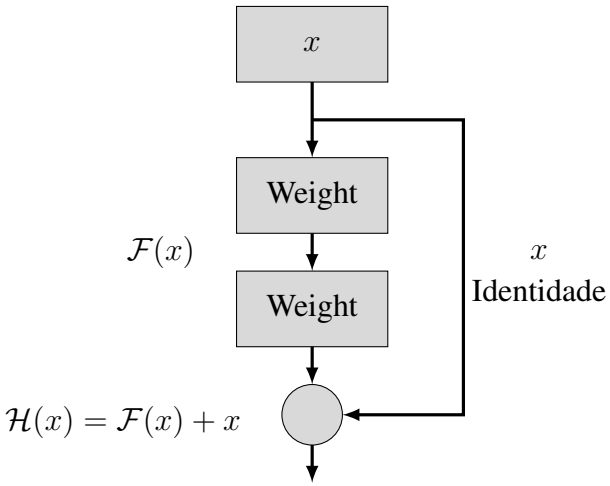
$$\tilde{x}_i = \frac{x_i - \mu_\beta}{\sqrt{\sigma_\beta^2 + \epsilon}} \tag{4}$$

onde x_i denota o i-ésimo dado de entrada da camada, μ_β e σ_β^2 são a média e variância do *mini-batch*, respectivamente, e ϵ é uma constante adicionada à variância do *mini-batch* para garantir estabilidade numérica.

2.2.6 Otimizador Adam

O otimizador Adam consiste em um método para ajuste dos coeficientes da rede. Tal método utiliza estimativas do primeiro e segundo momento do gradiente para ajustar a taxa de aprendizagem para cada coeficiente da rede. O primeiro momento corresponde à média e o segundo momento é a variância não centralizada, calculando uma média móvel exponencial do gradiente m_t e do gradiente ao quadrado v_t, enquanto que os parâmetros β_1 e β_2 controlam a taxa

Figura 3 – Exemplo de um bloco residual.



Fonte: Adaptado de (HE *et al.*, 2015a).

de decaimento dessas médias móveis (KINGMA; BA, 2017). A atualização dos coeficientes é dada por (KINGMA; BA, 2017)

$$\theta_t = \theta_{t-1} - \alpha \cdot \frac{\widehat{m}_t}{\sqrt{\widehat{v}_t} + \epsilon} \tag{5}$$

onde θ_t é o parâmetro atualizado, θ_{t-1} é o parâmetro da iteração anterior, α consiste na taxa de aprendizagem, enquanto $\hat{m}_t$ e $\hat{v}$ são as estimativas não polarizadas do primeiro e segundo momento, respectivamente.

2.3 Redes neurais residuais

Redes neurais residuais (ResNets) são estruturas profundas modularizadas compostas por blocos residuais, as quais emergem como uma solução aos problemas de degradação causados pela adição de muitas camadas na rede; geralmente, levando a uma saturação da acurácia e elevado erro de treinamento (HE *et al.*, 2015a). Os blocos residuais são conectados através de "atalhos" (do inglês, *shortcuts*), passando assim a informação mais "rasa" diretamente às camadas mais profundas a fim de evitar problemas de degradação. A Figura 3 apresenta um exemplo de bloco residual com conexão de atalho, onde x representa a entrada do bloco, $\mathcal{H}(x) = \mathcal{F}(x) + x$ denota a saída do bloco residual e $\mathcal{F}(x)$ representa o resíduo determinado pelas camadas do bloco residual. A conexão de atalho representada na Figura 3 desempenha a função de mapeamento de identidade, não adicionando parâmetros extras ou complexidade computacional (HE *et al.*, 2015a). Versões modificadas desse bloco residual são encontradas e utilizadas em (RAJPURKAR *et al.*, 2017) e (RIBEIRO *et al.*, 2020).

3 METODOLOGIA

Nesta seção, a metodologia utilizada para o desenvolvimento da ferramenta computacional para classificação multirrótulo de 5 superclasses de patologias cardíacas é apresentada. Especificamente, a Seção 3.1 apresenta uma descrição do conjunto de dados utilizado para realizar o treinamento das arquiteturas implementadas e a validação dos modelos obtidos. Por sua vez, as Seções 3.2 e 3.3 trazem breves descrições das arquiteturas propostas em (RAJPURKAR *et al.*, 2017) e (RIBEIRO *et al.*, 2020), respectivamente. A Seção 3.4 descreve os parâmetros adotados para o processo de treinamento de ambas as arquiteturas. Por fim, a Seção 3.5 revisita as métricas de avaliação utilizadas para a análise de desempenho dos modelos obtidos na tarefa de classificação multirrótulos.

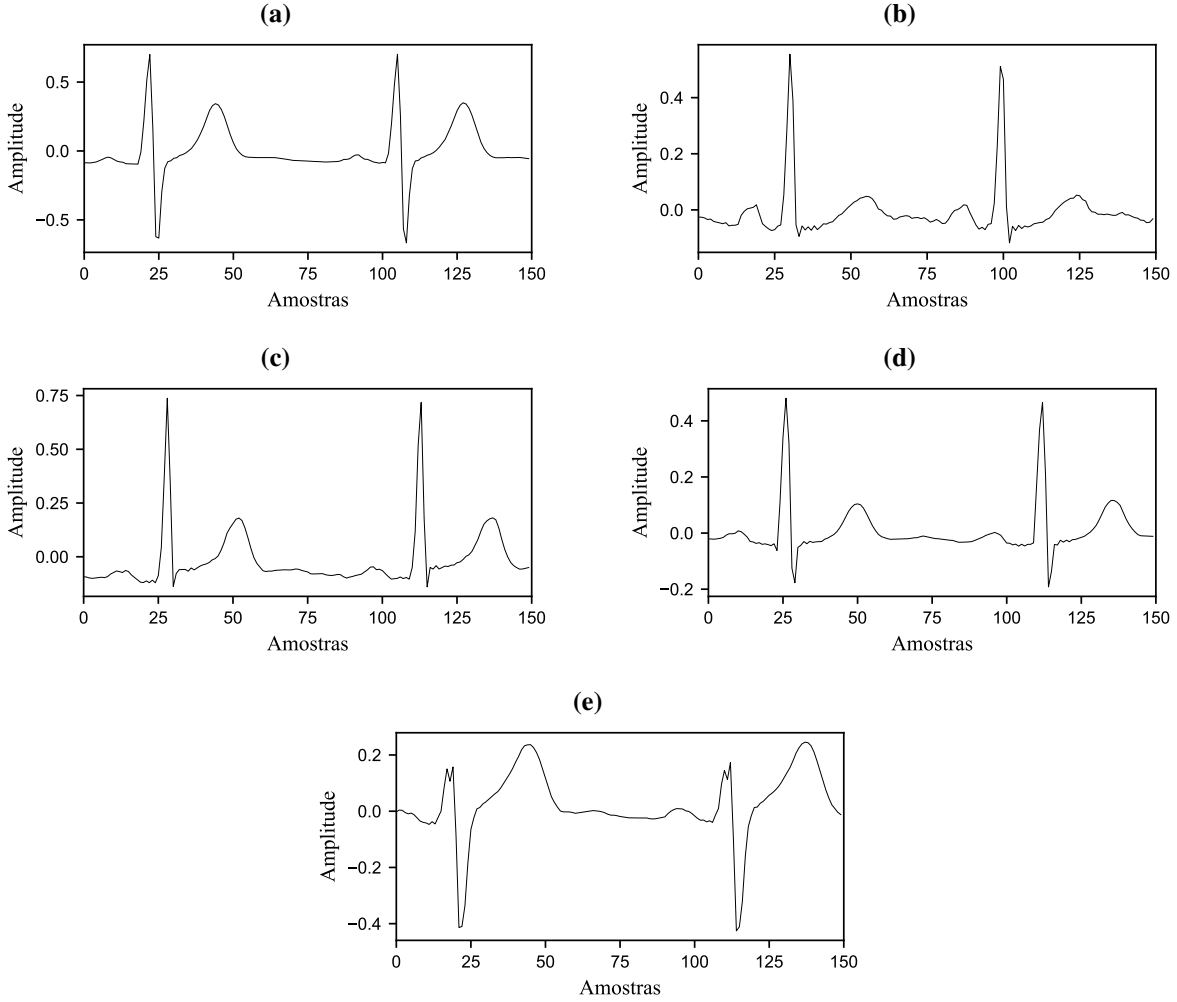
3.1 Conjunto de dados considerado

O conjunto de dados considerado, disponível em (WAGNER *et al.*, 2020), consiste de um grande número de registros de sinais de ECG. Esse conjunto público (denominado PTB-XL) é composto por 21.837 registros de sinais de ECG, de 12 derivações, com 10 segundos de duração, gravados com 18.885 pacientes (52% homens e 48% mulheres), bem como frequência de amostragem de 100 Hz (usada aqui) e 500 Hz. Os dados são compostos por sinais de ECG rotulados por cardiologistas e por metadados contendo declarações de diagnóstico de acordo com o padrão descrito na Norma ISO 11073-91064:2009 (ISO Central Secretary, 2009), podendo as 44 declarações de diagnósticos possíveis serem agrupadas (conforme indicado no Quadro 1) em 5 superclasses distintas, a saber: normal (NORM), alteração do segmentos ST e onda T (STTC), distúrbios de condução (CD), infarto do miocárdio (MI) e hipertrofia (HYP) (veja exemplos na Figura 4). Tais registros, denominados exemplos, são divididos em três conjuntos, a saber: treinamento, validação e teste, seguindo a recomendação dada em (WAGNER *et al.*, 2020); especificamente, os *folds* de 1 a 8 são utilizados para o treinamento, o *fold* 9 é utilizado para a validação e o *fold* 10 é utilizado para o teste. Vale mencionar que os exemplos que não possuem declarações de diagnóstico são eliminados do conjunto de dados, restando assim 20.373 exemplos (isto é, 16.289 exemplos no conjunto de treinamento, 2.034 exemplos no conjunto de validação e 2.050 exemplos no conjunto de teste).

A partir dos exemplos de sinais de ECG com diagnóstico, a rotulagem considerada para o treinamento é realizada de acordo com a probabilidade de diagnóstico constante nos metadados, isto é, declarações de diagnóstico com probabilidade maior que 50% estabelecem a superclasse correspondente como um possível rótulo do exemplo. Vale salientar que cada exemplo pode conter mais de uma declaração de diagnóstico com probabilidade maior que 50%, dado que diferentes patologias podem ocorrer de forma simultânea; por isso, os rótulos de cada exemplo são unidos de forma a produzir um conjunto de dados multirrótulos. Dessa forma, o conjunto de treinamento conta com 20.631 rótulos, o conjunto de validação conta com 2.575

rótulos e o conjunto de teste conta com 2.593 rótulos, cujos diagnósticos enquadram-se em (ao menos) uma das 5 superclasses. A distribuição da quantidade de rótulos estabelecidos dentre as 5 superclasses para os conjuntos de treinamento, validação e teste é apresentada na Figura 5, na qual observa-se que a maioria dos rótulos estabelecidos correspondem à superclasse NORM (ausência de patologias cardíacas). Verifica-se também, a partir de tal figura, um desbalanceamento na quantidade de rótulos estabelecidos dentre as superclasses; por exemplo, a superclasse NORM representa 36,5% da quantidade total de rótulos do conjunto de treinamento (36,6% do conjunto de validação e 36,8% do conjunto de teste), enquanto a superclasse HYP conta com apenas 8,8% de rótulos do conjunto de treinamento (8,4% do conjunto de validação e 8,6% do conjunto de teste). Todavia, qualquer tratamento relativo ao desbalanceamento das superclasses está além do escopo do presente trabalho.

Figura 4 – Exemplos de sinais de ECG das 5 superclasses presentes no conjunto de dados (WAGNER *et al.*, 2020). (a) Normal. (b) Alterações ST/T. (c) Distúrbios de condução. (d) Infarto do miocárdio. (e) Hipertrofia.



Fonte: Adaptado de (WAGNER *et al.*, 2020).

Quadro 1 – Categorização das declarações de diagnósticos do conjunto de dados (WAGNER *et al.*, 2020).

Superclasses	Subclasses	Declarações de diagnósticos
NORM	NORM	Eletrocardiograma normal
MI	AMI	Infarto do miocárdio anterior
	IMI	Infarto do miocárdio inferior
	LMI	Infarto do miocárdio lateral
	PMI	Infarto do miocárdio posterior
CD	LAFB/LPFB	Bloqueio fascicular anterior esquerdo/posterior esquerdo
	IRBBB	Bloqueio incompleto de ramo direito
	ILBBB	Bloqueio incompleto de ramo esquerdo
	CLBBB	Bloqueio completo do ramo esquerdo
	CRBBB	Bloqueio completo do ramo direito
	_AVB	Bloqueio atrioventricular
	IVCB	Distúrbio de condução intraventricular não específico (bloqueio)
	WPW	Síndrome de Wolff-Parkinson-White
STTC	ISCA	Isquemia em derivações anteriores
	ISCI	Isquemia em derivações inferiores
	ISC_	Isquemia não específica
	STTC	Alterações ST-T
	NST_	Alterações ST não específicas
HYP	LVH	Hipertrofia ventricular esquerda
	RVH	Hipertrofia ventricular direita
	LAO/LAE	Sobrecarga/aumento do átrio esquerdo
	RAO/RAE	Sobrecarga/aumento do átrio direito
	SEHYP	Hipertrofia septal

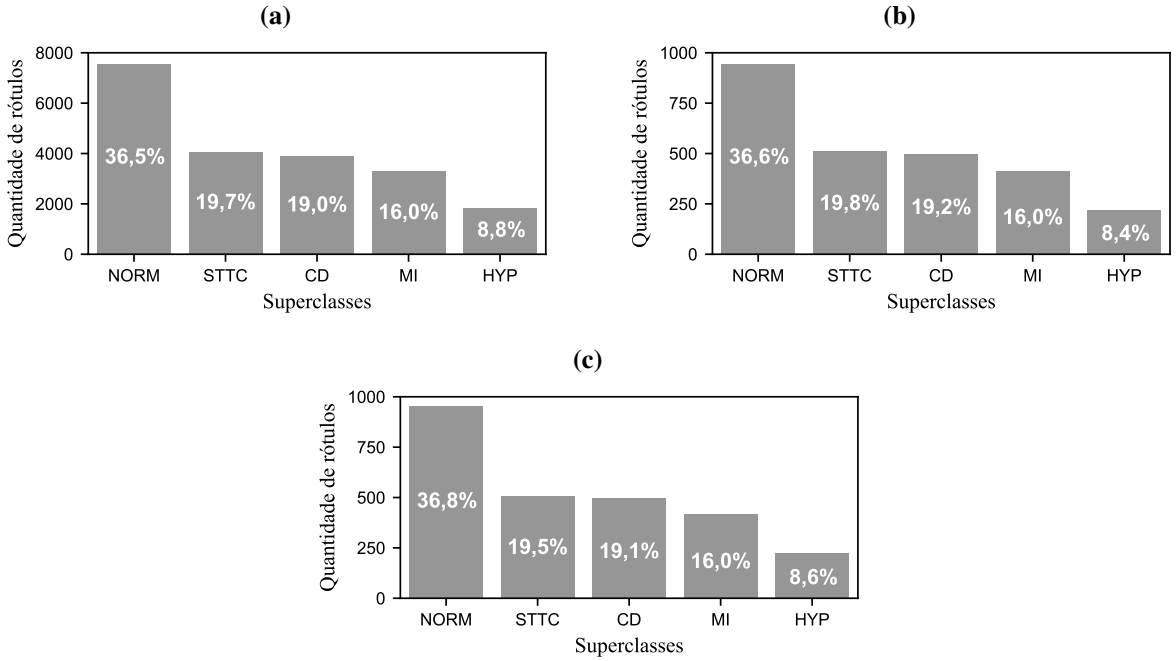
Fonte: Adaptado de (WAGNER *et al.*, 2020).

3.2 Revisitando a arquitetura de (RAJPURKAR *et al.*, 2017)

A Figura 6 traz um diagrama de blocos da arquitetura introduzida em (RAJPURKAR *et al.*, 2017), já considerando a alteração[1] necessária na camada de saída para acomodar o caso de classificação multirrótulos (MAXWELL *et al.*, 2017). Especificamente, essa arquitetura consiste de 34 camadas (33 camadas convolucionais e 1 camada totalmente conectada), organizadas em 1 bloco de entrada, 16 blocos residuais e 1 bloco de saída. A composição dos blocos é dada como segue. O bloco de entrada é constituído por uma camada convolucional, seguida por *batch normalization* e ReLU. Os blocos residuais contêm uma sequência de camadas convolucional, *batch normalization*, ReLU e *dropout*, além de conexões de atalho formadas por uma camada *max pooling*, utilizada para adequar a dimensão da entrada de cada bloco. Observe as modifica-

[1] A função de ativação *softmax* foi substituída pela função de ativação *sigmoid*.

Figura 5 – Quantidade de rótulos para cada superclasse. (a) Conjunto de treinamento. (b) Conjunto de validação. (c) Conjunto de teste.



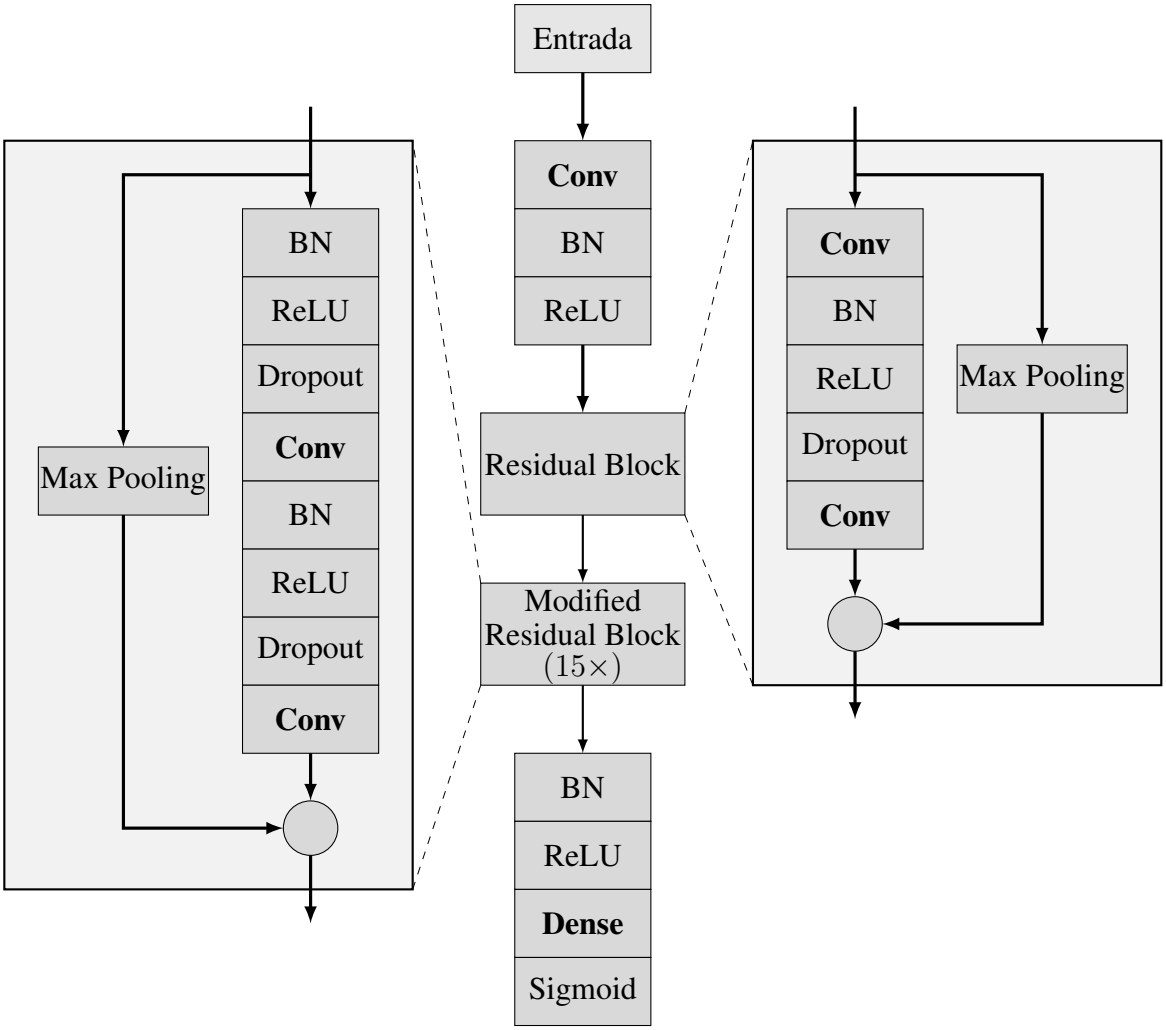
Fonte: Autoria própria.

ções introduzidas em 15 desses blocos residuais, adicionando camadas de *batch normalization*, ReLU e *dropout* ao bloco residual inicial. Por fim, o bloco de saída conta com camadas de *batch normalization*, ReLU, *dense* e função de ativação *sigmoid*. Na composição das camadas convolucionais, o número de filtros aumenta a cada 4 blocos residuais, alternando entre os valores 64, 128, 192 e 256. Esses filtros possuem, para todas as camadas convolucionais, tamanho igual a 16. Ainda, os valores dos passos (isto é, *stride*) são determinados de forma a subamostrar a entrada em um fator de 2 a cada 2 blocos residuais. Desse modo, a entrada original é subamostrada por um fator de 2^8 ao final da arquitetura. Vale mencionar que os coeficientes das diferentes camadas são inicializados conforme (HE *et al.*, 2015b).

3.3 Revisitando a arquitetura de (RIBEIRO *et al.*, 2020)

A Figura 7 apresenta um diagrama de blocos com a arquitetura descrita em (RIBEIRO *et al.*, 2020). Especificamente, essa arquitetura contém 10 camadas (9 camadas convolucionais e 1 camada totalmente conectada), organizadas em 1 bloco de entrada, 4 blocos residuais e 1 bloco de saída. Tais blocos podem ser descritos como segue. O bloco de entrada consiste de 1 camada convolucional, seguida de *batch normalization* e ReLU. Cada bloco residual é composto por camadas convolucional, *batch normalization*, ReLU e *dropout*. Uma conexão de atalho contendo camadas *max pooling* e convolucional é adicionada à segunda camada convolucional e outra conexão de atalho, com uma função identidade, parte dessa adição. Para melhor entendimento, a primeira conexão de atalho é denominada conexão A e a segunda, conexão B. No primeiro

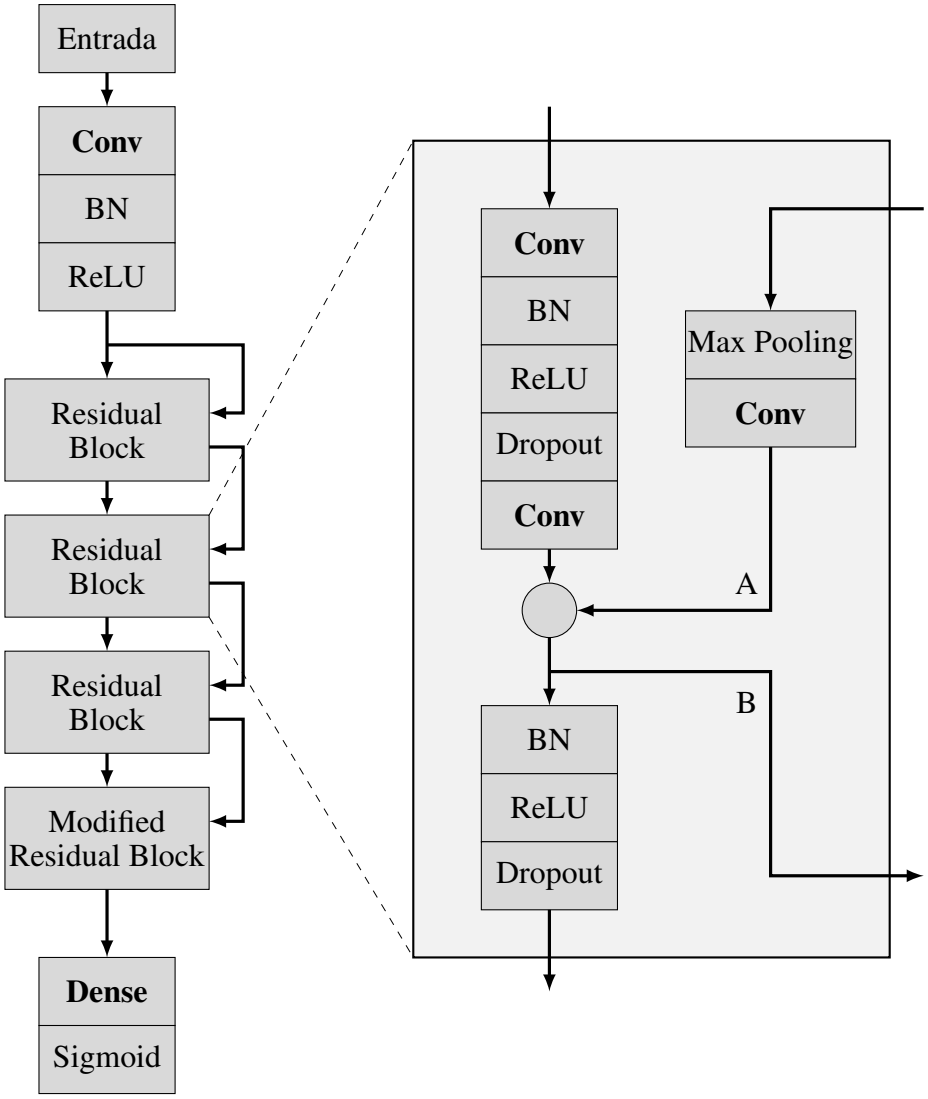
Figura 6 – Arquitetura proposta por (RAJPURKAR *et al.*, 2017).



Fonte: Adaptado de (RAJPURKAR *et al.*, 2017).

bloco residual, a conexão A tem como entrada a saída do bloco de entrada da arquitetura, enquanto que nos outros blocos residuais, a entrada da conexão A consiste na saída da conexão B do bloco residual anterior. Por consequência, em cada bloco residual, a saída da conexão B equivale à entrada da conexão A do próximo bloco residual, com exceção do quarto bloco em que a conexão de atalho B não se faz presente. Por fim, o bloco de saída é composto por uma camada totalmente conectada, seguida da função de ativação *sigmoid*. Vale mencionar que, na composição das camadas convolucionais, os números de filtros são alterados no decorrer das camadas, de modo que na primeira camada convolucional o número de filtros equivale a 64 e nas camadas posteriores, o número de filtros é acrescido de 64 a cada 2 blocos residuais. Esses filtros, para todas as camadas convolucionais, possuem um tamanho igual a 16. Por outro lado, as camadas *max pooling* e convolucional que compõem a primeira conexão de atalho dos blocos residuais possuem filtros com tamanho igual a 1, visando adequar a dimensão dos dados. Ainda, o passo é determinado de forma a subamostrar a entrada em um fator de 4 a cada bloco residual. Os coeficientes das diferentes camadas da rede são inicializados conforme (HE *et al.*, 2015b).

Figura 7 – Arquitetura proposta por (RIBEIRO *et al.*, 2020).



Fonte: Adaptado de (RIBEIRO *et al.*, 2020).

3.4 Quanto à implementação, treinamento e escolha dos hiperparâmetros

Quanto aos recursos de *software* necessários na implementação das arquiteturas de (RAJPURKAR *et al.*, 2017) e (RIBEIRO *et al.*, 2020), considera-se aqui o uso da linguagem Python (versão 3.9.15) juntamente das bibliotecas Keras / TensorFlow (versão 2.10.0) e Scikit-learn (versão 1.1.3), dentre outras. A IDE Visual Studio Code (versão 1.73.1) da Microsoft é utilizada para o desenvolvimento e teste do código, dada a sua integração e interatividade com *notebooks* Jupyter. Quanto aos recursos computacionais necessários, destaca-se que a concepção, treinamento preliminar e verificação do código se deu em um computador Acer Nitro 5, com processador Intel Core i7, 16 GB de memória RAM e 1 GPU (do inglês, *graphics processing unit*) NVIDIA GeForce GTX 1650 com 12 GB de memória RAM, no sistema operacional Windows 10, enquanto o treinamento efetivo dos modelos ocorreu em um computador dedi-

cado com processador AMD Ryzen 5800x, 32 GB de memória e 1 GPU NVIDIA RTX 3080 com 10 GB de memória RAM, no sistema operacional Ubuntu 22.04. Por fim, com respeito ao processo de treinamento das arquiteturas consideradas, a otimização da função custo (entropia cruzada binária) é realizada através do otimizador Adam com uma taxa de aprendizagem de 0,1. Essa taxa de aprendizagem é reduzida por um fator de 2 se o custo de validação não apresentar melhora nas últimas 10 épocas. O modelo é treinado por no máximo 100 épocas com um tamanho de lote (do inglês, *batch size*) de 256 exemplos, sendo o treinamento interrompido caso o custo no conjunto de validação não apresente melhora nas últimas 15 épocas. Vale salientar que a adequação da taxa de aprendizagem e a interrupção do treinamento são realizadas, respectivamente, pelos *callbacks* `ReduceLROnPlateau` e `EarlyStopping` da biblioteca Keras (CHOLLET *et al.*, 2015). Ainda, uma taxa de *dropout* de 0,8 é utilizada aqui.

3.5 Métricas de desempenho

Visando avaliar o desempenho das arquiteturas implementadas, a metodologia introduzida recentemente em (HEYDARIAN; DOYLE; SAMAVI, 2022) para o caso de classificação multirrótulo é considerada aqui, diferindo assim da abordagem convencional utilizada até então na literatura. A partir dessa metodologia, é construída uma matriz de confusão expandida $\mathbf{M}$ (veja a Figura 13 adiante), de dimensão $(C+1) \times (C+1)$ em que C representa o número de classes, a qual é denominada matriz de confusão multirrótulos (MLCM). Nessa matriz $\mathbf{M}$, a linha $C+1$ corresponde à inexistência de rótulo tanto verdadeiro quanto predito (do inglês, *no true label*) (NTL) para um dado exemplo, enquanto a coluna $C+1$ corresponde à situação em que um ou mais rótulos verdadeiros do exemplo não são preditos (do inglês, *no predicted label*) (NPL). A partir de tal matriz, torna-se possível determinar então o número de ocorrências[2] verdadeiro positivo (TP), verdadeiro negativo (TN), falso positivo (FP) e falso negativo (FN) no caso multirrótulo; especificamente, a partir da MLCM, para uma determinada classe c_k com $k \in [0,\ C]$,

$$\mathrm{TP}_k = [\mathbf{M}]_{(k,k)} \tag{6}$$

$$\mathrm{TN}_k = \sum_{c=0}^{C} [\mathbf{M}]_{(c,c)} - [\mathbf{M}]_{(k,k)} \tag{7}$$

$$\mathrm{FP}_k = \sum_{c=0}^{C} [\mathbf{M}]_{(c,k)} - [\mathbf{M}]_{(k,k)} \tag{8}$$

[2] Quanto à terminologia, tem-se que i) TP diz respeito ao resultado de um teste que detectou corretamente a presença de uma condição ou característica; ii) TN se refere ao resultado de um teste que indicou corretamente a ausência de uma condição ou característica; iii) FP trata da situação em que o resultado de um teste detectou erroneamente uma condição ou característica; e v) FN se refere ao resultado de um teste que indicou erroneamente a ausência de uma condição ou característica.

e

$$\text{FN}_k = \sum_{c=0}^{C} [\mathbf{M}]_{(k,c)} - [\mathbf{M}]_{(k,k)}. \tag{9}$$

Consequentemente, as métricas precisão, sensibilidade e F1-*score*[3], que relacionam o número de ocorrências TP, TN, FP e FN, para uma dada classe c, podem ser computadas da seguinte forma:

$$\text{Precisão}_c = \frac{\text{TP}_c}{\text{TP}_c + \text{FP}_c} \tag{10}$$

$$\text{Sensibilidade}_c = \frac{\text{TP}_c}{\text{TP}_c + \text{FN}_c} \tag{11}$$

e

$$\begin{aligned} \text{F1-score}_c &= 2\frac{\text{Precisão}_c \times \text{Sensibilidade}_c}{\text{Precisão}_c + \text{Sensibilidade}_c} \\ &= \frac{2\text{TP}_c}{2\text{TP}_c + \text{FN}_c + \text{FP}_c}. \end{aligned} \tag{12}$$

Ainda, a partir de (10)-(12), torna-se possível obter a média das métricas precisão, sensibilidade e F1-*score* levando em conta todas as classes (de forma a avaliar o desempenho geral das arquiteturas), o que resulta então nas métricas médias *micro*, *macro* e *weighted*. Para exemplificar, considerando a métrica F1-*score* [dada em (12)], as médias *micro*, *macro* e *weighted* são obtidas, respectivamente, através de

$$\text{F1-score}_{\text{micro}} = \frac{2\sum_{c=1}^{C} \text{TP}_c}{2\sum_{c=1}^{C} \text{TP}_c + \sum_{c=1}^{C} \text{FN}_c + \sum_{c=1}^{C} \text{FP}_c} \tag{13}$$

$$\text{F1-score}_{\text{macro}} = \frac{1}{C}\sum_{c=1}^{C} \text{F1-score}_c \tag{14}$$

e

$$\text{F1-score}_{\text{weighted}} = \frac{\sum_{c=1}^{C} \text{F1-score}_c \, \text{N}_c}{\sum_{c=1}^{C} \text{N}_c} \tag{15}$$

onde N_c consiste no número total de rótulos preditos[4] da classe c no conjunto de dados. Dessa forma, definida a metodologia de avaliação e as métricas consideradas, pode-se proceder para a análise e discussão dos resultados.

[3] Vale observar que a métrica F1-*score* [dada por (12)] resulta da média harmônica entre as métricas precisão [dada por (10)] e sensibilidade [dada por (11)].

[4] O número total de rótulos preditos consiste na soma das ocorrências TP e FN de cada classe.

4 RESULTADOS E DISCUSSÃO

Nesta seção, os principais resultados obtidos durante o desenvolvimento do presente trabalho são apresentados. Em particular, a Seção 4.1 mostra fragmentos de códigos em linguagem Python relacionados ao tratamento do conjunto de dados multirrótulos, assim como a implementação e treinamento das arquiteturas de (RAJPURKAR *et al.*, 2017) e (RIBEIRO *et al.*, 2020). Por sua vez, a Seção 4.2 ilustra a MLCM gerada a partir do conjunto de teste para cada modelo (isto é, após o treinamento das arquiteturas), enquanto a Seção 4.3 trata de comparar o desempenho alcançado pelos modelos com respeito às métricas consideradas. Por fim, as Seções 4.4 e 4.5 trazem discussões sobre a evolução do custo sobre os conjuntos de treinamento e de validação, assim como um comparativo da complexidade computacional relacionada às arquiteturas consideradas.

4.1 Implementação realizada em linguagem Python

As Figuras 8–12 apresentam fragmentos de código necessários para a implementação da ferramenta computacional proposta para a classificação multirrótulo de 5 superclasses de patologias cardíacas. Especificamente, a Figura 8 mostra um fragmento do código relacionado à manipulação e ao tratamento do conjunto de dados utilizado, destacando a forma como as anotações e os diagnósticos são agregados, o processo de rotulagem aplicado, assim como a divisão do conjunto de dados entre treinamento, validação e teste (como descrito na Seção 3.1). Por sua vez, as Figuras 9 e 10 apresentam, respectivamente, fragmentos de código desenvolvidos para a implementação das arquiteturas de (RAJPURKAR *et al.*, 2017) (conforme a Figura 6) e (RIBEIRO *et al.*, 2020) (conforme a Figura 7), identificando a composição dos blocos (e camadas) juntamente da parametrização envolvida. Já a Figura 11, traz um fragmento do código para o treinamento das arquiteturas, no qual se evidencia a função custo e o otimizador utilizados, assim como os valores considerados para os parâmetros tanto dos *callbacks* quanto do otimizador (conforme Seção 3.4). Por fim, a Figura 12 apresenta fragmentos do código utilizados para a implementação da MLCM e dos cálculos das métricas de desempenho consideradas, utilizando a metodologia introduzida recentemente em (HEYDARIAN; DOYLE; SAMAVI, 2022) (veja detalhes na Seção 3.5). Vale mencionar que todos os códigos desenvolvidos bem como resultados obtidos estão disponíveis em (MEURER; ZANCO; KUHN, 2022).

Figura 8 – Fragmento de código desenvolvido para tratamento do conjunto de dados.

```
    # Converte os dados de anotação
    metadata = pd.read_csv(path / 'ptbxl_database.csv')
    metadata.scp_codes = metadata.scp_codes.apply(
                                        lambda x: ast.literal_eval(x))

    # Agrega os diagnósticos
    meta_scp = pd.read_csv(path / 'scp_statements.csv', index_col=0)
    meta_scp = meta_scp[meta_scp.diagnostic == 1]

    # Sequência das superclasses
    super_classes = ['NORM','STTC','CD','MI','HYP']

    ...

    # Processo de rotulagem
    def simple_diagnostic(scp_codes):
        for key, item in scp_codes.items():
            if key in met_scp.index:
                diag_class = subdiag_dict[key]
                if item >= 50:
                    vec[super_classes.index(diag_class)] = 1
        # Sem diagnóstico presente
        if vec.sum() == 0:
            return '???'
        return vec

    ...

    # Conjunto de treino (folds 1-8)
    X_train = X[metadata.strat_fold.values < 9]
    y_train = Y[metadata.strat_fold.values < 9]
    # Conjunto de validação (fold 9)
    X_val = X[metadata.strat_fold.values == 9]
    y_val = Y[metadata.strat_fold.values == 9]
    # Conjunto de teste (fold 10)
    X_test = X[metadata.strat_fold.values == 10]
    y_test = Y[metadata.strat_fold.values == 10]
```

Fonte: Autoria própria.

Figura 9 – Fragmento de código referente a implementação da arquitetura de (RAJPURKAR *et al.*, 2017).

```
    # Implementação da arquitetura
    def get_model(input_layer, model_name):
    ...
        conv_config = dict(kernel_size=16,
                           filters=64,
                           padding='same',
                           kernel_initializer=initializer)

        # Bloco de entrada
        layers = Conv1D(strides=1, **conv_config)(input_layer)
        layers = BatchNormalization()(layers)
        layers = ReLU()(layers)

        # Conexão de atalho
        skip = MaxPooling1D(pool_size=1, strides=2)(layers)

        # Segundo bloco
        layers = Conv1D(strides=1, **conv_config)(layers)
        layers = BatchNormalization()(layers)
        layers = ReLU()(layers)
        layers = Dropout(rate_drop)(layers)
        layers = Conv1D(strides=2, **conv_config)(layers)

        # Adição de camadas
        layers = Add()([layers, skip])
        num_filter = [64, 64, 64,
                      128, 128, 128, 128,
                      192, 192, 192, 192,
                      256, 256, 256, 256]

        # Blocos residuais modificados
        for i in range(15):
            layers = residual_blocks_rajpurkar(layers, i=i,
                                               stride=(i%2)+1,
                                               num_filter=num_filter[i],
                                               rate_drop=rate_drop,
                                               initializer=initializer)

        # Bloco de saída
        layers = BatchNormalization()(layers)
        layers = ReLU()(layers)
        layers = Flatten()(layers)
        layers = Dense(32)(layers)
        classification = Dense(5, activation='sigmoid')(layers)

        # Realiza a construção do modelo
        model = Model(inputs=input_layer, outputs=classification)

        return model
```

Fonte: Autoria própria.

Figura 10 – Fragmento de código referente a implementação da arquitetura de (RIBEIRO *et al.*, 2020).

```
    # Implementação da arquitetura
    def get_model(input_layer, model_name):
    ...
        # Bloco de entrada
        layers = Conv1D(kernel_size=16, filters=64,
                        strides=1, padding='same',
                        kernel_initializer=initializer)(input_layer)
        layers = BatchNormalization()(layers)
        layers = ReLU()(layers)

        # Número de filtros
        num_filter = np.array([128, 192, 256, 320])

        # Manipulação das camadas
        layer, skip = layers, layers

        # Blocos residuais
        for i in range(4):
            layer, skip = residual_blocks_ribeiro([layer, skip],
                                                  num_filter=num_filter[i],
                                                  rate_drop=rate_drop,
                                                  initializer=initializer,
                                                  downsample=downsample)

        # Bloco de saída
        layer = Flatten()(layer)
        classification = Dense(5, activation='sigmoid',
                               kernel_initializer=initializer)(layer)

        # Realiza a construção do modelo
        model = Model(inputs=input_layer, outputs=classification)

        return model
```

Fonte: Autoria própria.

4.2 Análise de desempenho a partir da matriz de confusão multirrótulo

A Figura 13 apresenta as matrizes de confusão multirrótulos obtidas a partir das predições do conjunto de teste com os modelos resultantes. Tais matrizes são normalizadas com base no número total de ocorrências TP e FN correspondente a cada classe; especificamente, a Figura 13(a) ilustra a MLCM obtida para o modelo de (RAJPURKAR *et al.*, 2017) e a Figura 13(b) para o modelo de (RIBEIRO *et al.*, 2020). Observa-se a partir dos valores dispostos sobre a diagonal principal das matrizes que as classes com mais rótulos estabelecidos (isto é, quando se tem um número maior de exemplos) exibem um desempenho superior na predição das diferentes condições cardíacas, conforme constatado, por exemplo, com a classe NORM em que mais de 90% dos rótulos são preditos corretamente. Por outro lado, classes que possuem menor número de rótulos estabelecidos apresentam (proporcionalmente) menos predições cor-

Figura 11 – Fragmento de código usado para o treinamento das arquiteturas.

```
    # Treinando o modelo
    def training(model, X_train, y_train, X_val, y_val,
                 model_name, save_parameters,
                 learning_rate=0.1, epochs=100):

        # Define e parametriza os callbacks
        callbacks = [ReduceLROnPlateau(monitor='val_loss',
                                       factor=0.5, patience=10,
                                       mode='min', min_lr=1e-6),
                     ModelCheckpoint(model_path, monitor='val_loss',
                                     mode='auto', verbose=1,
                                     save_best_only=True),
                     CSVLogger(csv_path, separator=',', append=True),
                     EarlyStopping(monitor='val_loss', mode='auto',
                                   verbose=1, patience=15)]

        # Compilando o modelo
        model.compile(loss='binary_crossentropy',
                      optimizer=kopt.Adam(learning_rate),
                      metrics=['accuracy'])

        # Treinando o modelo
        history = model.fit(X_train, y_train,
                            validation_data=(X_val, y_val),
                            batch_size=256,
                            epochs=epochs,
                            callbacks=callbacks)

        return history
```

Fonte: Autoria própria.

retas (independente do modelo utilizado), como é o caso da classe HYP que teve apenas 44,4% das predições corretas usando o modelo de (RAJPURKAR *et al.*, 2017) e 36,1% através do modelo de (RIBEIRO *et al.*, 2020). [Logo, fica ratificada a necessidade de se investigar metodologias para o tratamento do desbalanceamento entre as classes (conforme sugerido na Seção 5.3).] Também, verifica-se que, mesmo usando a arquitetura de (RAJPURKAR *et al.*, 2017) que possui um número significativamente maior de camadas, a capacidade do modelo de predizer corretamente rótulos pertencentes às classes CD e MI é inferior à capacidade alcançada pelo modelo de (RIBEIRO *et al.*, 2020). Ainda, é possível constatar que, quando ocorre uma predição errônea de um dado exemplo, ambos os modelos tendem a atribuir o rótulo NORM para aquele dado exemplo (como se constata ao longo das linhas da primeira coluna) ou a não realizar predição alguma daquele rótulo (como indicado nas linhas da última coluna intitulada NPL); na verdade, grande parte dos rótulos preditos incorretamente são decorrentes da situação em que aquele dado rótulo não é se quer predito, o que se torna mais evidente quando se tem um menor número de exemplos. Vale mencionar que a última linha (intitulada NTL) se encontra zerada dado que, no tratamento do conjunto de dados (conforme descrito na Seção 3.1), os

Figura 12 – Fragmento de código relacionado ao cálculo das métricas de desempenho.

```
    # Plota a matriz de confusão
    def plot_confusion_matrix(y_test, y_pred, model_name,
                              target_names, plot_path='results',
                              print_note='false'):

        # Função que implementa a MLCM
        cm, _ = mlcm.cm(y_test, y_pred)
        target_names = np.array([*target_names, 'NoC'])

        # Cálculo da normalização da matriz de confusão
        divide = cm.sum(axis=1, dtype='int64')
        divide[divide == 0] = 1
        cm_norm = 100 * cm / divide[:, None]
        ...
        return cm

    # Computa as métricas de desempenho do modelo
    def get_metrics(y_test, prediction, prediction_bin,
                    target_names, model_name, cm):
        ...
        report_mlcm = mlcm.stats(cm, False)
        ...
        return
```

Fonte: Autoria própria.

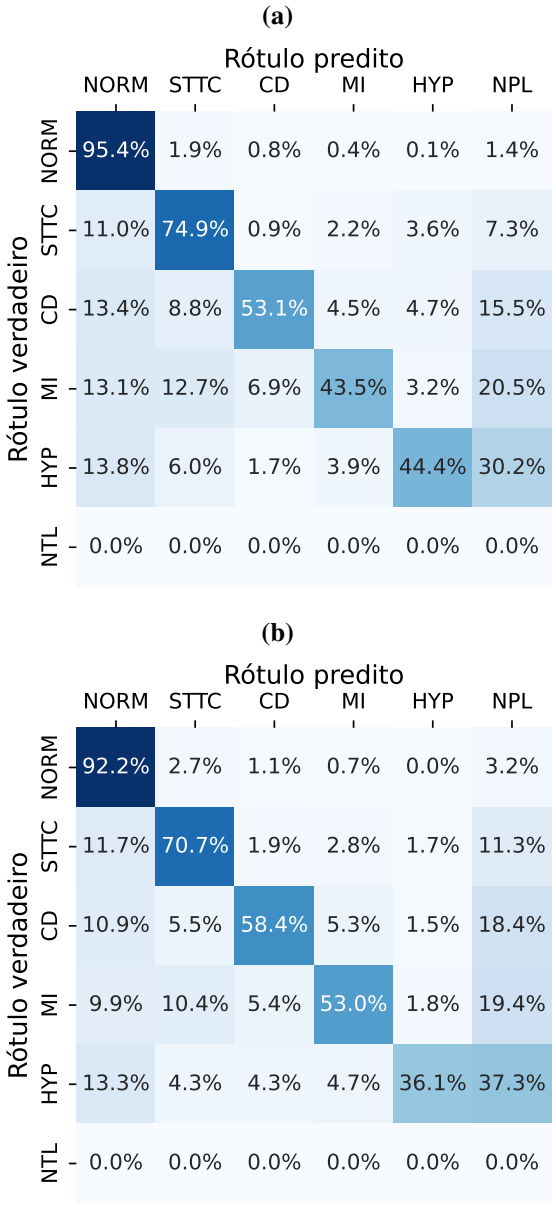
exemplos não rotulados foram removidos; em outras palavras, todos os exemplos possuem ao menos 1 rótulo estabelecido, sendo tal rótulo predito corretamente ou não. Portanto, apesar das pequenas diferenças, observa-se que os modelos geradas pelas arquiteturas de (RAJPURKAR *et al.*, 2017) e de (RIBEIRO *et al.*, 2020) apresentam um desempenho bastante similar na tarefa de classificação considerada.

4.3 Avaliação de desempenho dos modelos frente às métricas consideradas

A Tabela 1 apresenta uma análise comparativa do desempenho dos modelos, para as diferentes classes, a partir das métricas precisão, sensibilidade e F1-*score* bem como dos valores médios *micro*, *macro* e *weighted*[1] (veja a Seção 3.5). Dessa tabela, verifica-se que, com exceção de algumas classes específicas (devidamente destacados para facilitar a interpretação), o modelo de (RIBEIRO *et al.*, 2020) exibe um desempenho levemente superior ao alcançado pelo modelo de (RAJPURKAR *et al.*, 2017); sobretudo, em se tratando das métricas médias *micro*, *macro* e *weighted* onde se observa desempenho similar e/ou superior em termos de precisão e F1-*score*. Vale destacar que os resultados obtidos não podem ser diretamente comparados com aqueles descritos em (RAJPURKAR *et al.*, 2017) e (RIBEIRO *et al.*, 2020), devido, por exemplo, i) a

[1] A coluna *Weight* da Tabela 1, usada para computar a média ponderada (*weighted*), apresenta a soma das ocorrências TP, FN, NTL e NPL para cada classe.

Figura 13 – Matriz de confusão multirrótulos. (a) Arquitetura de (RAJPURKAR *et al.*, 2017). (b) Arquitetura de (RIBEIRO *et al.*, 2020).



Fonte: Autoria própria.

diferença na tarefa de classificação considerada (veja a Seção 1); ii) a metodologia utilizada para computar as métricas (veja a Seção 3.5); ou iii) a utilização de um conjunto de dados diferente (veja a Seção 3.1). Todavia, ainda assim, constata-se que os resultados atingidos pelos modelos obtidos aqui (a partir do conjunto de dados PTB-XL) são próximos daqueles fornecidos por (RAJPURKAR *et al.*, 2017) e (RIBEIRO *et al.*, 2020); em particular, (RAJPURKAR *et al.*, 2017) indica ter atingido uma precisão de $0{,}81$ (contra $0{,}79$), uma sensibilidade de $0{,}82$ (contra $0{,}70$) e um F1-*score* de $0{,}81$ (contra $0{,}72$), enquanto (RIBEIRO *et al.*, 2020) indica ter atingido um F1-*score* maior do que $0{,}80$ (contra $0{,}74$). Diante do exposto, é possível concluir que ambas as arquiteturas foram implementadas e treinadas com sucesso, resultando em modelos que apresentam resultados compatíveis com aqueles da literatura.

Tabela 1 – Desempenho das arquiteturas de (RAJPURKAR *et al.*, 2017) e de (RIBEIRO *et al.*, 2020) frente às métricas consideradas.

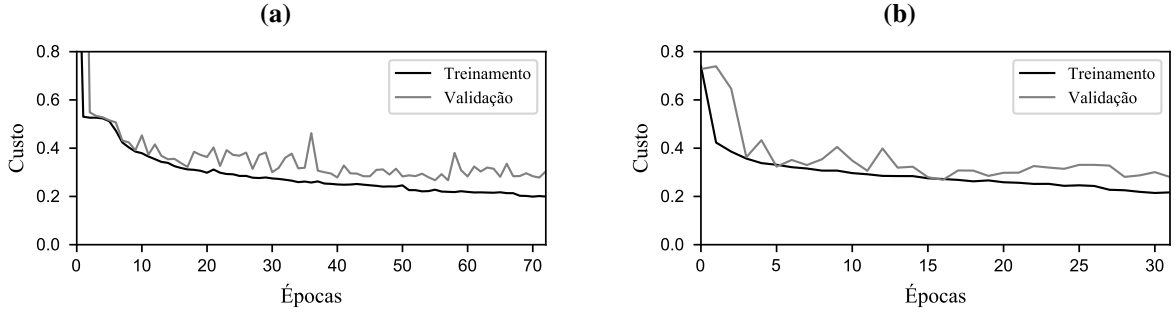
	Precisão		**Sensibilidade**		**F1-*score***		***Weight***	
	Rajpurkar	Ribeiro	Rajpurkar	Ribeiro	Rajpurkar	Ribeiro	Rajpurkar	Ribeiro
NORM	0,80	0,82	0,95	0,92	0,87	0,87	961	964
STTC	0,74	0,77	0,75	0,71	0,74	0,74	534	540
CD	0,86	0,85	0,53	0,58	0,66	0,69	580	543
MI	0,80	0,79	0,44	0,53	0,56	0,64	464	443
HYP	0,62	0,77	0,44	0,36	0,52	0,49	232	233
Micro avg	0,70	0,70	0,70	0,70	0,70	0,70	2771	2723
Macro avg	0,76	0,80	0,62	0,62	0,67	0,69	2771	2723
Weighted avg	0,79	0,81	0,70	0,70	0,72	0,74	2771	2723

Fonte: Autoria própria.

4.4 Evolução do custo sobre os conjuntos de treinamento e de validação

A Figura 14 apresenta a evolução do custo sobre os conjuntos de treinamento e de validação durante o treinamento das arquiteturas de (RAJPURKAR *et al.*, 2017) e de (RIBEIRO *et al.*, 2020). A partir de tal figura, observa-se certa similaridade na evolução do custo ao longo do treinamento de ambas as arquiteturas; em outras palavras, verifica-se custo elevado no conjunto de treinamento e validação no início do processo de treinamento, o que decresce gradualmente até atingir a condição de parada (isto é, quando não se observa melhoria no custo sobre o conjunto de validação após 15 épocas). Ainda, pode-se inferir a partir das curvas apresentadas a ocorrência de um *overfitting* pouco significativo, devido sobretudo a interrupção do treinamento de ambas as arquiteturas (conforme definido na Seção 3.4). Consequentemente, conclui-se que os modelos gerados a partir das arquiteturas de (RAJPURKAR *et al.*, 2017) e de (RIBEIRO

Figura 14 – Evolução do custo sobre os conjuntos de treinamento e de validação durante o treinamento das arquiteturas. (a) Arquitetura de (RAJPURKAR *et al.*, 2017). (b) Arquitetura de (RIBEIRO *et al.*, 2020).



Fonte: Autoria própria.

et al., 2020) apresentam uma boa capacidade de generalização na classificação de patologias cardíacas em exemplos não vistos até então (novos).

4.5 Considerações sobre o custo computacional

A Tabela 2 apresenta um comparativo do custo computacional relativo ao treinamento das arquiteturas de (RAJPURKAR *et al.*, 2017) e (RIBEIRO *et al.*, 2020), destacando em especial o número total de parâmetros, assim como o número de épocas e o tempo transcorrido até o encerramento do treinamento. Especificamente, o número total de parâmetros compreende os parâmetros treináveis (gerados pelas camadas convolucionais e totalmente conectada) e não treináveis (geradas com as camadas *batch normalization*). Nesse sentido, a arquitetura de (RAJPURKAR *et al.*, 2017) possui 16.521.285 parâmetros, dos quais 16.510.917 são treináveis e 10.368 são não treináveis, enquanto a arquitetura de (RIBEIRO *et al.*, 2020) conta com 6.353.093 parâmetros, sendo 6.349.381 treináveis e 3.712 não treináveis. Essa diferença (de quase 3 vezes) no número de parâmetros decorre da quantidade de camadas treináveis (compare as Figuras 6 e 7) presentes na arquitetura de (RAJPURKAR *et al.*, 2017), levando assim a um maior custo computacional para o treinamento. Isso se verifica, por exemplo, pela diferença significativa no tempo requerido para o treinamento de cada uma das arquiteturas (3h32min contra 3min) até que a condição de parada seja atingida, isto é, quando o custo no conjunto de validação não apresenta melhora após 15 épocas (como definido na Seção 3.4). Ainda, destaca-se que o treinamento da arquitetura de (RAJPURKAR *et al.*, 2017) demanda pouco mais de 2 vezes (73 contra 32) a quantidade de épocas necessárias para o treinamento da arquitetura de (RIBEIRO *et al.*, 2020) (dado o limite de 100 definido na Seção 3.4). Diante disso, torna-se evidente que o significativo número de parâmetros da arquitetura de (RAJPURKAR *et al.*, 2017) implica em um aumento do custo computacional para treinamento, o que, contudo, não se traduz em acréscimo expressivo no desempenho do modelo.

Tabela 2 – Detalhes do treinamento das arquiteturas de (RAJPURKAR *et al.*, 2017) e (RIBEIRO *et al.*, 2020).

	Rajpurkar	**Ribeiro**
Número (total) de parâmetros	16.521.285	6.353.093
Tempo transcorrido de treinamento	3h32min	3min
Número de épocas de treinamento	73	32

Fonte: Autoria própria.

5 CONSIDERAÇÕES FINAIS

Nesta seção, as considerações finais deste trabalho de conclusão de curso são apresentadas. Primeiro, os resultados mais importantes alcançados são destacados e discutidos na Seção 5.1. Em seguida, uma lista de trabalhos publicados e/ou em vias de publicação é fornecida na Seção 5.2. Por fim, na Seção 5.3, algumas sugestões para trabalhos futuros na área são elencadas.

5.1 Conclusões

Neste trabalho, a implementação de uma ferramenta computacional foi conduzida visando a detecção e classificação multirrótulo de 5 superclasses de patologias cardíacas a partir de sinais de ECG. Essa ferramenta foi baseada nas arquiteturas de redes neurais profundas introduzidas em (RAJPURKAR *et al.*, 2017) e (RIBEIRO *et al.*, 2020), as quais foram treinadas aqui utilizando o conjunto de dados multirrótulo disponibilizado por (WAGNER *et al.*, 2020). Tal implementação foi realizada utilizando a linguagem Python (ROSSUM; DRAKE, 2009) juntamente as bibliotecas Tensorflow (ABADI *et al.*, 2015) e Keras (CHOLLET *et al.*, 2015), por oferecerem recursos importantes para o desenvolvimento do presente trabalho. O desempenho dos modelos obtidos (após o treinamento) foi avaliado por meio de uma metodologia padronizada, trazida recentemente por (HEYDARIAN; DOYLE; SAMAVI, 2022), para o caso de classificação multirrótulos; em particular, a partir da matriz de confusão desenvolvida para o caso multirrótulos, a forma de computar as métricas precisão, sensibilidade e F1-*score*, bem como os valores médios *micro*, *macro* e *weighted*, foi então revisada. Adicionalmente, foi realizada uma análise do custo computacional demandado para o treinamento de cada arquitetura, a fim de avaliar a viabilidade e necessidade de um número maior de parâmetros no modelo.

A partir dos resultados obtidos aqui, infere-se que ambas arquiteturas apresentaram um bom desempenho na tarefa de classificação multirrótulos (quando o conjunto de dados e as métricas são padronizadas), atingindo uma precisão média de $0{,}80$, sensibilidade média de $0{,}70$ e F1-*score* médio acima de $0{,}70$. Vale destacar que o modelo de (RIBEIRO *et al.*, 2020) apresentou um desempenho similar, ou ainda ligeiramente melhor, para as diferentes métricas consideradas do que aquele obtido a partir da arquitetura de (RAJPURKAR *et al.*, 2017). Outra vantagem a favor da arquitetura de (RIBEIRO *et al.*, 2020) diz respeito ao custo computacional requerido para o treinamento, visto que a arquitetura de (RIBEIRO *et al.*, 2020) possui um número menor de camadas, isto é, 10 contra 34 camadas de (RAJPURKAR *et al.*, 2017). Com isso, um número significativamente menor de parâmetros é gerado e o tempo transcorrido para o treinamento de tal arquitetura foi de apenas 3min [contra 3h32min para o treinamento da arquitetura de (RAJPURKAR *et al.*, 2017)].

Portanto, para o problema de detecção e classificação considerado (detalhado na Seção 3) como também levando em conta os aspectos discutidos, conclui-se que a arquitetura de

(RIBEIRO *et al.*, 2020) resulta em um modelo com melhor relação de compromisso entre custo computacional e desempenho atingido.

5.2 Trabalhos publicados e/ou em vias de publicação

Baseado nos resultados obtidos com o desenvolvimento deste trabalho de conclusão de curso, os seguintes trabalhos científicos foram publicados e/ou estão em vias de publicação:

- LEANDRO, R. M.; MEURER, S. M.; ZANCO, D. G. de P.; KUHN, E. V. Classificação de patologias cardíacas usando sinais de eletrocardiograma através de redes neurais residuais. **Encontro de Iniciação Científica (ENDICT)**, Toledo, PR, 2022.

- MEURER, S. M.; ZANCO, D. G. de P.; KUHN, E. V. On the performance of the (RAJPURKAR *et al.*, 2017) and (RIBEIRO *et al.*, 2020) models for classification of cardiac pathologies based on electrocardiograms. **Journal of the Franklin Institute**, 2023 (em redação).

5.3 Sugestões para trabalhos de pesquisa futuros

Como ideias para trabalhos de pesquisa futuros na área, sugere-se:

- Investigar e aplicar estratégias para lidar com os aspectos de desbalanceamento observados no conjunto de dados considerado.

- Estender as comparações de desempenho realizadas frente a outras arquiteturas descritas na literatura.

- Conduzir um número maior de experimentos, de forma metódica e sistemática, destinados à seleção e escolha apropriada dos hiperparâmetros.

- Implementar as arquiteturas aqui descritas utilizando a biblioteca Pytorch (PASZKE *et al.*, 2019).

- Avaliar o desempenho das arquiteturas considerando sinais de ECG amostrados a 500 Hz e/ou frente a outros conjuntos de dados da literatura.

- Realizar a adequação das arquiteturas visando a classificação das 24 subclasses de patologias descritas no conjunto de dados considerado.

REFERÊNCIAS

ABADI, M. *et al.* **TensorFlow: Large-scale machine learning on heterogeneous systems**. 2015. Software available at tensorflow.org. Disponível em: http://tensorflow.org/.

ACHARYA, U. R. *et al.* A deep convolutional neural network model to classify heartbeats. **Computers in Biology and Medicine**, v. 89, p. 389–396, 2017.

ATTIA, Z. I. *et al.* An artificial intelligence-enabled ECG algorithm for the identification of patients with atrial fibrillation during sinus rhythm: A retrospective analysis of outcome prediction. **The Lancet**, v. 394, p. 861–867, 2019.

CAI, J. *et al.* Multi-ECGNet for ECG arrythmia multi-label classification. **IEEE Access**, v. 8, p. 110848–110858, 2020.

CAIRNS, A. W. *et al.* A computer-human interaction model to improve the diagnostic accuracy and clinical decision-making during 12-lead electrocardiogram interpretation. **Journal of Biomedical Informatics**, v. 64, p. 93–107, 2016.

CHOLLET, F. *et al.* **Keras**. GitHub, 2015. Disponível em: https://github.com/fchollet/keras.

DUBIN, D. **Interpretação rápida do ECG**. 3. ed. [*S.l.*]: Publicações Científicas, 1996. ISBN 0912912006.

FAWAZ, H. I. *et al.* Deep learning for time series classification: A review. **Data Mining and Knowledge Discovery**, v. 33, p. 917–963, 2019.

GALLOWAY, C. D. *et al.* Development and validation of a deep-learning model to screen for hyperkalemia from the eletrocardiogram. **JAMA Cardiology**, v. 5, p. 428–436, 2019.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep learning**. [*S.l.*]: MIT Press, 2016.

GU, J. *et al.* Recent advances in convolutional neural networks. **Pattern Recognition**, v. 77, p. 354–377, 2018.

GUYTON, A. C.; HALL, J. E. **Tratado de fisiologia médica**. 11. ed. [*S.l.*]: Elsevier, 2006. ISBN 9788535216417.

HANNUN, A. Y. *et al.* Cardiologist-level arrhythmia detection and classification in ambulatory electrocardiograms using a deep neural network. **Nature Medicine**, v. 25, p. 65–69, 2019.

HE, K. *et al.* Deep residual learning for image recognition. **arXiv**, 2015.

HE, K. *et al.* Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. **arXiv**, 2015.

HERRERA, F. *et al.* **Multilabel classification: Problem analysis, metrics and techniques**. 1. ed. [*S.l.*]: Springer Cham, 2016. ISBN 978-3-319-41110-1.

HEYDARIAN, M.; DOYLE, T. E.; SAMAVI, R. MLCM: Multi-label confusion matrix. **IEEE Access**, v. 10, p. 19083–19095, 2022.

IOFFE, S.; SZEGEDY, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. **arXiv**, 2015.

ISO Central Secretary. **Health informatics – Standard communication protocol – Part 91064: Computer-assisted electrocardiography**. Geneva, CH, 2009. Disponível em: https://www.iso.org/standard/46493.html.

KIM, Y. Convolutional neural nerworks for sentence classification. **arXiv**, 2014.

KINGMA, D. P.; BA, J. L. Adam: a method for stochastic optimization. **arXiv**, 2017.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. **Nature**, v. 521, p. 436–444, 2015.

LEUR, R. R. van de *et al.* Automatic triage of 12-lead ECGs using deep convolutional neural networks. **Journal of the American Heart Association**, 2020.

MAXWELL, A. *et al.* Deep learning architectures for multi-label classification of intelligent health risk prediction. **BMC Bioinformatics**, v. 18, 2017.

MELE, P. Improving electrocardiogram interpretation in the clinical setting. **Journal of Electrocardiology**, v. 41, p. 438–439, 2008.

MEURER, S. M.; ZANCO, D. G. de P.; KUHN, E. V. **Detecção e classificação de patologias cardíacas em eletrocardiogramas utilizando redes neurais profundas**. [*S.l.*]: GitHub, 2022. https://github.com/SarahMeurer/ECG-classification.

MINCHOLÉ, A. *et al.* Machine learning in the electrocardiogram. **Journal of Electrocardiology**, v. 57, p. S61–S64, 2019.

PASZKE, A. *et al.* Pytorch: An imperative style, high-performance deep learning library. In: **Advances in Neural Information Processing Systems 32**. [*S.l.*]: Curran Associates, Inc., 2019. p. 8024–8035.

RAJPURKAR, P. *et al.* Cardiologist-level arrhythmia detection with convolutional neural networks. **ArXiv**, abs/1707.01836, 2017.

RIBEIRO, A. H. *et al.* Automatic diagnosis of the 12-lead ECG using a deep neural network. **Nature Communications**, v. 11, 2020.

ROSSUM, G. V.; DRAKE, F. L. **Python 3 reference manual**. Scotts Valley, CA: CreateSpace, 2009. ISBN 1441412697.

SALERNO, S. M.; ALGUIRE, P. C.; WAXMAN, H. S. Competency in interpretation of 12-lead electrocardiograms: A summary and appraisal of published evidence. **Annals of Internal Medicine**, v. 138, p. 751–760, 2003.

SALERNO, S. M. *et al.* Training and competency evaluation for interpretation of 12-lead electrocardiograms: recommendations from the american college of physicians. **Annals of Internal Medicine**, v. 138, p. 747–750, 2003.

SANNINO, G.; PIETRO, G. D. A deep learning approach for ECG-based heartbeat classification for arrhythmia detection. **Future Generation Computer Systems**, v. 86, p. 446–455, 2018.

SHAMEER, K. *et al.* Machine learning in cardiovascular medicine: Are we there yet? **Heart**, p. 1–9, 2018.

SOMANI, S. *et al.* Deep learning and the electrocardiogram: A review of the current state-of-the-art. **European Society of Cardiology**, v. 23, p. 1179–1191, 2021.

SRIVASTAVA, N. *et al.* Dropout: a simple way to prevent neural networks from overfitting. **Journal of Machine Learning Research**, v. 15, p. 1929–1958, 2014.

WAGNER, P. *et al.* PTB-XL, a large publicly available electrocardiography dataset. **Sci Data**, v. 7, 2020.

WANG, Y. *et al.* The influence of the activation function in a convolution neural network model of facial expression recognition. **Applied Sciences**, v. 10, 2020.

WORLD HEALTH ORGANIZATION. **Cardiovascular diseases (CVDs)**. 2021. Disponível em: https://www.who.int/en/news-room/fact-sheets/detail/cardiovascular-diseases-(cvds). Acesso em: 09 de abril de 2022.

XIONG, Z.; STILES, M. K.; ZHAO, J. Robust ECG signal classification for detection of atrial fibrillation using a novel neural network. **Computing in Cardiology (CinC)**, p. 1–4, 2017.