DAS Departamento de Automação e Sistemas
CTC Centro Tecnológico
UFSC Universidade Federal de Santa Catarina

Classificação de Laudos e Traçados de Exames de Eletrocardiogama Utilizando Redes Neurais

Relatório submetido à Universidade Federal de Santa Catarina
como requisito para a aprovação da disciplina:
DAS 5511: Projeto de Fim de Curso

Nathaniel Salvador de Oliveira

Florianópolis, novembro de 2018

Classificação de Laudos e Traçados de Exames de Eletrocardiogama Utilizando Redes Neurais

Nathaniel Salvador de Oliveira

Esta monografia foi julgada no contexto da disciplina

DAS 5511: Projeto de Fim de Curso

e aprovada na sua forma final pelo

Curso de Engenharia de Controle e Automação

Prof. Eduardo Camponogara

Banca Examinadora:

Gabriel Paim
Orientador na Empresa

Prof. Eduardo Camponogara
Orientador no Curso

Prof. Hector Bessa Silveira
Responsável pela disciplina

Alexandre Reeberg de Melo, Avaliador

Débora Gardinal de Sousa, Debatedor

Volnei Fontana Junior, Debatedor

Agradecimentos

Gostaria de primeiramente agradecer Gabriel Paim, Lucas Neves e Jonatas Pavei pela oportunidade de trabalhar no problema proposto e por receberem-me de braços abertos na InPulse Animal Health.

Presto meus agradecimentos ao professor Eduardo Camponogara pela suas orientação fundamental para o encaminhamento, desenvolvimento e finalização do trabalho.

Gostaria também de agradecer ao médico veterinário Luis Felipe dos Santos pela sua prestatividade e ajuda na confecção da lista de cardiopatias utilizada no trabalho.

Agradeço também a Paulo Curado pela sua ajuda com o ponta-pé inicial na implementação do sistema web de classificação de frases.

Por último, mas não menos importante, agradeço aos meus pais, amigos e namorada, por estarem sempre presentes prestando o apoio moral necessário para uma boa saúde mental nessa etapa.

Resumo

A InPulse Animal Health fornece um sistema de telemedicina juntamente com software e dispositivos voltados para a área veterinária. Utilizando o sistema e dispositivos InCardio, é possível capturar um exame de ECG de um paciente e enviá-lo, através da internet, para realização de um laudo por um médico cardiologista. Até o presente momento existem mais de 100 mil exames e laudos no banco de dados da empresa. O trabalho proposto consiste em criar uma ferramenta, que, utilizando os dados de exames capturados, juntamente com os laudos emitidos pelos médicos veterinários especializados em cardiologia, seja capaz de determinar a probabilidade de certas cardiopatias estarem presentes no exame sem intervenção humana. Utilizando uma ferramenta desenvolvida customizada para auxiliar na classificação das conclusões de laudos de exame ECG, juntamente com redes neurais treinadas para classificar novos exames automaticamente, foi desenvolvida uma rede neural com objetivo de, recebendo como entrada apenas os sinais ECG extraídos utilizando aparelho eletrocardiógrafo, tentar classificar estes sinais de forma tal que sejam selecionados entre as categorias Normal (não há cardiopatia presente) e Anormal (há cardiopatia presente).

Palavras-chave: aprendizado de máquina, redes neurais, doc2vec, embeddings, ecg, classificação, patologias, cardiopatias.

Abstract

The company InPulse Animal Health provides a telemedicine system paired with devices and software targeted at the area of veterinary. Using the InCardio device together with the telemedicine system, it is possible to acquire an ECG exam from a patient and send it, through the internet, to be reported by a veterinarian specialized in cardiology. On this day, there are over 100 thousand exams and reports in the company's database. The proposed study aims to develop a tool, which, using the exam data and reports made by cardiologists, is capable of determining the probability of certain cardiopathies to be present in the exam without human intervention. Making use of a customized web service to aid in the classification of report conclusions of ECG exams, toguether with neural networks with the objective of, upon receiving as input only the ECG signals extracted by the eletrocardiographer, the tool attempts to classify these signals in a way such that they fall into two distinct categories: Normal (no cardiopathy present) or Abnormal (cardiopathy present).

Keywords: machine learning, neural networks, doc2vec, embeddings, ecg, classification, cardiopathies.

Lista de ilustrações

Lista de Listas

Lista de tabelas

Lista de abreviaturas e siglas

MVC	Model-View-Controller
PCA	Principal components analysis
ECG	Eletrocardiograma
NLTK	Natural Language Toolkit
HTTP	HyperText Transfer Protocol
API	Application Programming Interface
REST	Representational State Transfer
JPA	Java Persistence API
SGBD	Sistema Gerenciador de Banco de Dados

Sumário

1 Introdução

Desde a introdução do Sistema de Telemedicina, do Software InCardio e dos dispositivos eletrocardiógrafos InCardio da InPulse Animal Health ao mercado da medicina veterinária, esses produtos e tecnologias vêm sendo utilizados por médicos veterinários de todo o Brasil para disponibilizar em suas clínicas e hospitais o exame de eletrocardiograma. Por meio deste, cardiologistas e veterinários sem especialização, permitiu-se aos veterinários requisitar aos laudistas (cardiologistas) a laudagem remota de exames realizados em seus próprios consultórios, realizando todo o processo de requisição e entrega de laudo via internet.

Até o momento da escrita deste trabalho, já passaram pelo sistema de telemedicina da empresa mais de 100 mil exames e laudos, permanecendo estes registrados no banco de dados da InPulse. Propôs-se a criação de uma ferramente que fizesse uso de todo esse conhecimento armazenado incorporando-o em um sistema automatizado com o objetivo de classificar sinais provenientes da atividade elétrica do coração dos pacientes. A classificação desejada espera selecionar os sinais em Normais e Anormais, onde denominam-se Normais aqueles que não apresentam cardiopatias e Anormais aqueles que apresentam.

Por se tratar de um problema de classificação e após investigação de técnicas e tecnologias modernas para a resolução do problema apresentado, optou-se pela utilização de Redes Neurais pois estas são aplicadas com sucesso nos mais diversos tipos de problemas e áreas do conhecimento.

Será visto durante o trabalho o conjunto de aparelhos eletrocardiógrafos e sistema de telemedicina da InPulse Animal Health juntamente uma breve explicação sobre os mesmos.

No trabalho também será apresentado um conhecimento superficial sobre o funcionamento do coração e o processo de extração dos sinais de atividade elétrica do coração, conhecido como exame de ECG.

Serão abordados dois problemas de classificação. O primeiro é a classificação de textos oriundos da conclusão dos laudos de exames ECG. Este problema surge da necessidade de existir rótulos dos sinais de ECG provenientes dos exames contidos no banco de dados. Esses rótulos (labels) são necessários para o treinamento da rede neural cuja entrada é o sinal da atividade elétrica do coração. O segundo problema abordado no trabalho será a classificação desses sinais de ECG utilizando redes neurais.

Será explorado neste trabalho ferramentas e tecnologias voltadas ao funcionamento, treinamento e implementação de redes neurais artificiais, as quais serão utilizadas no

decorrer do projeto para realizar a classificação de textos e sinais de exames ECG. Essas incluem uma ferramenta de auxílio à classificação de frases, um sistema capaz de classificar exames de eletrocardiograma e por último uma rede neural cujo objetivo é classificar diretamente o sinal de ECG. Tais ferramentas foram utilizadas para classificar 31 mil exames de ECG cujos respectivos sinais fizeram parte do conjunto de treinamento da rede neural. Será feita uma análise crítica dos resultados obtidos durante o trabalho, procurando entender e justificar o sucesso ou falha dos mesmos. Por fim, serão apresentadas perspectivas de trabalhos futuros e conclusões variadas sobre os tópicos tratados no trabalho.

2 A InPulse Animal Health

Fundada em 2014, a InPulse Animal Health nasceu com a missão de ajudar na melhoria e manutenção da saúde dos animais através do desenvolvimento de tecnologias inovadoras no campo da medicina veterinária.

Hoje no Brasil existe um número muito grande de animais de estimação cujos donos preocupam-se com a saúde e bem-estar dos mesmos porém não existe um número suficiente de médicos veterinários especializados em cardiologia, capazes de realizar laudos de exames eletrocardiográficos para atender todos esses pacientes nas suas próprias clínicas ou hospitais.

A InPulse desenvolveu um sistema capaz de levar o acesso a exames de eletrocardiograma à clínicas veterinárias onde não há médico veterinário especializado em cardiologia, através de seus aparelhos eletrocardiógrafos da linha InCardio e Sistema de Telemedicina.

2.1 Eletrocardiógrafos InCardio

A linha de eletrocardiógrafos InCardio da InPulse é utilizada, em conjunto com Software InCardio e aplicativo android InCardio, para capturar e armazenar os sinais do coração de um paciente, ou seja, que são utilizados para realizar um exame de eletrocardiograma (ECG).

Os aparelhos da linha serão descritos brevemente a seguir.

2.1.1 InCardio

O primeiro produto apresentado ao mercado apresenta diversas características que o diferenciam dos seus competidores, como a possibilidade de utilização sem fios de alimentação e comunicação, por possuir bateria interna e conexão *Bluetooth* ®, necessitando apenas do posicionamento dos eletrodos no corpo do paciente.

O eletrocardiógrafo InCardio captura até com 12 derivações, possui precisão de 24 bits e taxa de amostragem de 500 Hz.

Na Figura 1 vemos um aparelho InCardio em utilização na captura dos sinais da atividade elétrica do coração e a colocação dos eletrodos sobre o corpo do paciente.

2.1.2 InCardio CS

O InCardio CS diferencia-se do InCardio em suas características apenas na falta da bateria interna e comunicação *Bluetooth* ®, podendo este ser utilizado apenas através

Figura 1 – InCardio sendo utilizado na captura de exame ECG em gato mourisco.

da interface USB de comunicação. Todas as outras características permanecem iguais às do InCardio.

2.1.3 InCardio Agile

Possui como diferencial *pads* localizados na parte externa do aparelho que permitem, apenas pelo contato com a pele de um paciente, ou seja, sem conexão de eletrodos, extrair os sinais de ECG do coração do paciente. Essa forma de uso permite que seja feita uma avaliação mais ágil pois não há necessidade de colocação de vários eletrodos sobre o corpo do paciente para se obter o sinal de ECG.

O InCardio Agile possui suporte para 6 derivações, quando conectado aos eletrodos. Também possui precisão de 24 bits e taxa de amostragem de 500 Hz, assim como os outros aparelhos da linha InCardio.

Na Figura 2 temos o aparelho InCardio Agile na sua maleta de transporte. Podemos ver que o aparelho é compacto e, pareado com um dispositivo móvel Android, torna-se prático quando o médico veterinário encontra a necessidade de avaliação rápida do estado de saúde cardíaco do paciente.

Figura 2 – InCardio Agile, eletrodos e maleta para transporte

2.2 Sistema de Telemedicina

O Software InCardio, para *Windows* e *Mac OS*, o aplicativo InCardio, para *Android*, e o sistema InCloud oferecem uma solução de Telemedicina voltada para exames de Eletrocardiograma.

O termo "Telemedicina" refere-se a qualquer serviço médico que pode ser prestado à distância. No caso do sistema de telemedicina da InPulse, o serviço prestado é a realização de laudos de exames de ECG à distância e funciona da seguinte maneira: o médico veterinário em posse de um eletrocardiógrafo da linha InCardio, em conjunto com software ou aplicativo InCardio, realiza o exame de ECG, gravando os sinais capturados do paciente em seu computador ou dispositivo móvel. Após a gravação do exame, o médico veterinário pode solicitar um laudo, requisitando que este exame seja laudado por um médico veterinário especializado em cardiologia. Através do software InCardio, o cardiologista produz o laudo e envia-o de volta para o médico veterinário que solicitou o mesmo, como mostrado na Figura 3.

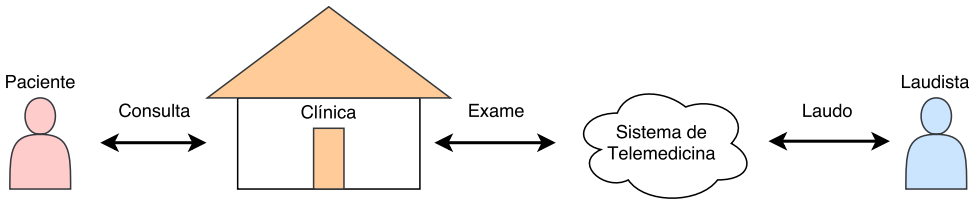


Figura 3 – Fluxo de Telemedicina

Até a data da escrita deste documento já foram laudados, através do sistema de telemedicina, mais de 100 mil exames de ECG.

Mais informações sobre a empresa, aparelhos eletrocardiógrafos, software e aplicativo InCardio e sistema de telemedicina podem ser encontradas em [1].

3 Problema e Ferramentas

O problema a ser estudado neste trabalho pode ser descrito como um problema de classificação. A ideia principal, em poucas palavras, é sugerir ao médico veterinário sem treinamento em cardiologia, que este solicite um laudo de um exame via Telemedicina ou não, usando-se da análise e processamento dos sinais de Eletrocardiograma (ECG) obtidos através dos aparelhos InCardio e softwares InCardio. Tal sistema teria a capacidade de avaliar, de forma simplificada, a qualidade geral de um exame eletrocardiográfico, classificando este em *Normal* e *Anormal*. O termo *Normal* será utilizado para designar exames os quais não há necessidade de nova avaliação por médico cardiologista para realização de laudo, e o termo *Anormal* para designar exames onde há a necessidade de nova avaliação.

3.1 Eletrocardiograma

Os sinais de ECG são utilizados por médicos cardiologistas na identificação de problemas cardíacos. Estes sinais retratam a atividade elétrica do coração. Para entendermos um pouco mais sobre os sinais de ECG é necessário que se entenda um pouco sobre o funcionamento mecânico e elétrico do coração, começando a partir do ciclo cardíaco.

Conteúdo descrito nesta seção foi extraído de [2].

3.1.1 Eventos do Ciclo Cardíaco

O coração é composto por quatro câmaras separadas por válvulas e apresenta um comportamento cíclico. As câmaras são Átrio esquerdo, Átrio direito, Ventrículo esquerdo e Ventrículo direito, sendo cada câmara responsável por armazenar por um curto período de tempo o sangue vindo de outra câmara, dos pulmões ou do resto do corpo.

O funcionamento normal do coração depende da ordem de acionamento dos músculos responsáveis por expulsar o sangue armazenado em cada cavidade. Em um coração saudável o ciclo de eventos é:

1. Contração Ventricular Isovolumétrica
2. Ejeção Ventricular
3. Relaxamento Isovolumétrico
4. Preenchimento dos Ventrículos

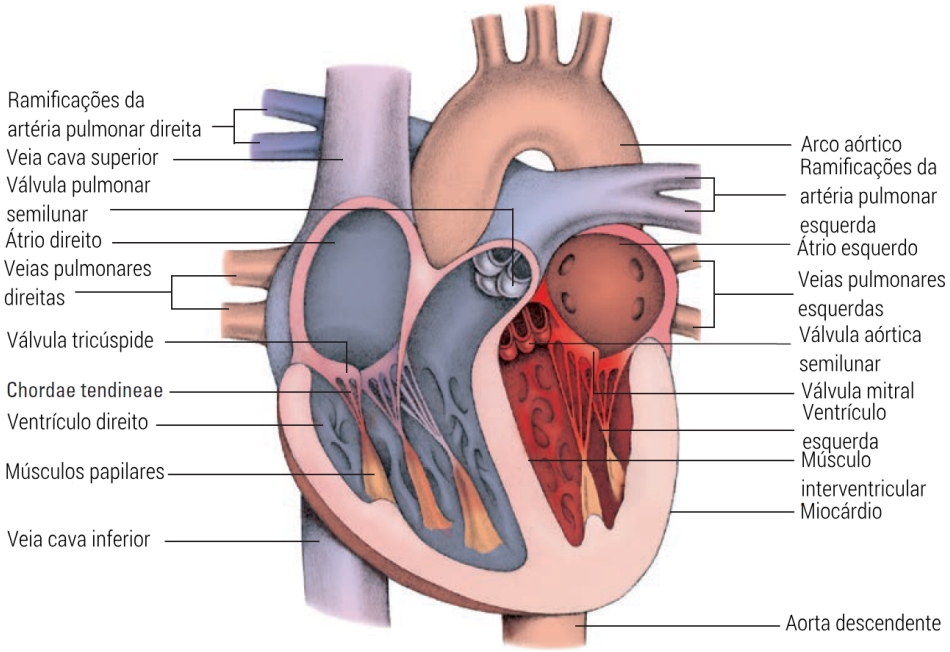


Figura 4 – Câmaras, válvulas e músculos do coração

5. Sístole Atrial

Segue uma breve explicação sobre cada fase do ciclo, representado na Figura 5.

3.1.1.1 Contração Ventricular Isovolumétrica

Em resposta à despolarização ventricular, a tensão nos ventrículos aumenta. O aumento de pressão interna dos ventrículos leva ao fechamento das válvulas mitrais e tricúspides. As artérias pulmonares e aórticas permanecem fechadas durante esta fase.

3.1.1.2 Ejeção Ventricular

Quando a pressão ventricular excede as pressões das artérias aorta e pulmonar, as válvulas aórticas e pulmonares abrem e os ventrículos ejetam sangue.

3.1.1.3 Relaxamento Isovolumétrico

Quando a pressão ventricular cai abaixo das pressões das artérias aorta e pulmonar, as válvulas aórticas e pulmonares se fecham. Todas as válvulas estão fechadas nessa fase. A diástole atrial ocorre e sangue preenche os átrios.

3.1.1.4 Preenchimento dos Ventrículos

A pressão atrial excede a pressão ventricular, o que causa as válvulas mitral e tricúspide a abrirem. O sangue então escorre passivamente para dentro dos ventrículos. Por volta de 70% do preenchimento dos ventrículos ocorre durante esta fase.

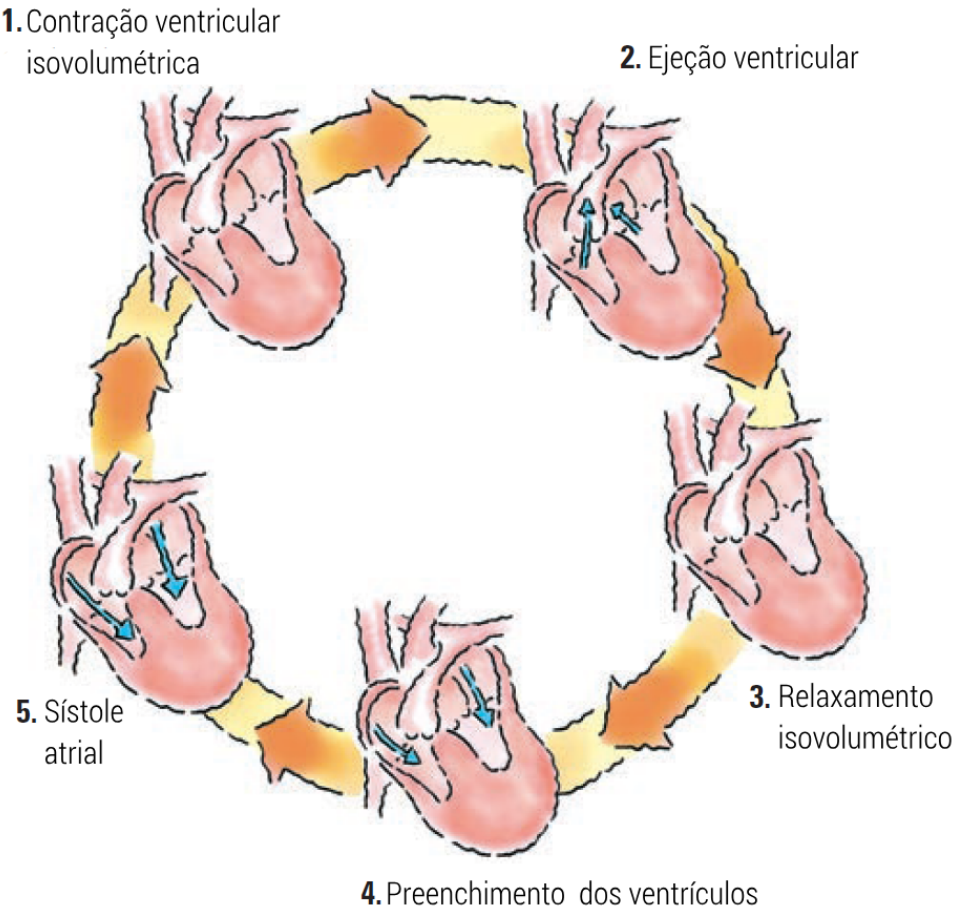


Figura 5 – Ciclo cardíaco

3.1.1.5 Sístole Atrial

Conhecido como "Chute Atrial", a sístole atrial (que coincide com o final da diástole ventricular) provê aos ventrículos o resto dos 30% de sangue para cada batimento do coração.

3.1.2 Ritmo do Coração

O ritmo (batimentos por minuto) do coração é controlado por dois ramos do sistema nervoso autônomo: o Simpático e Parassimpático. O sistema nervoso simpático pode ser descrito como uma espécie de "acelerador" do coração. Essas substâncias aumentam a frequência cardíaca: a Norepinefrina (Noradrenalina) e a Epinefrina (Adrenalina).

Já o sistema parassimpático seria uma espécie de "freio" do coração. Um dos nervos desse sistema, o nervo Vago, carrega impulsos que desaceleram o ritmo cardíaco. Estimular este sistema libera a substância química Acetilcolina, reduzindo a frequência de batimentos do coração.

3.1.3 Transmissão de impulsos elétricos

O coração é incapaz de bombear sangue sem que primeiramente ocorram estímulos elétricos. A geração e transmissão desses impulsos dependem de quatro características das células do coração:

- Automaticidade: a habilidade de uma célula de espontaneamente disparar um impulso elétrico.
- Excitabilidade: indica a capacidade de resposta de uma célula a impulsos elétricos.
- Condutibilidade: habilidade de uma célula de transmitir um impulso à outras células cardíacas.
- Contratibilidade: refere-se à habilidade de contração de uma célula após o recebimento de um estímulo.

À medida que os impulsos são transmitidos, as células cardíacas passam por ciclos de despolarização e repolarização. As células cardíacas em repouso são consideradas polarizadas, ou seja, sem atividade elétrica. Quando uma célula está completamente despolarizada, esta tenta retornar ao seu estado de repouso, em um processo chamado de repolarização. As cargas elétricas em uma célula são revertidas e retornam ao normal.

Após a ocorrência de despolarização e repolarização, o impulso elétrico resultante viaja através do coração por um caminho chamado de "sistema de condução". Impulsos partem do nodo Sinoatrial e são transmitidos ao resto do coração.

O nodo Sinoatrial está localizado no canto superior direito do átrio, onde a veia cava superior se encontra com o tecido atrial. É o principal regulador de frequência do coração, gerando impulsos em frequências de 60 a 100 impulsos por minuto. Usualmente esses impulsos não fluem no sentido contrário pois as células não conseguem responder à estímulos imediatamente após despolarização.

O nodo Atrioventricular, localizado no canto inferior direito do átrio direito, é responsável por atrasar os impulsos que o alcançam. Embora esse nodo não possua células marcapasso, o tecido que o envolve contém células que disparam na frequência de 40 a 60 impulsos por minuto.

Danos às células marcapasso do coração podem ocasionar diversos tipos de problemas cardíacos, tanto envolvendo o ritmo de batimentos, como arritmias, pausas e paradas sinusais, quanto problemas no bombeamento do sangue através das cavidades do coração.

3.1.4 Obtenção do ECG

O sinais de ECG são obtidos através de um aparelho eletrocardiógrafo. O aparelho possui cabos que se conectam aos eletrodos que são posicionados em contato com a pele do paciente. Esses eletrodos são colocados em locais padronizados no corpo do paciente, de modo que os diferentes sinais obtidos sigam um padrão para facilitar o estudo e comparação entre sinais obtidos de pacientes diferentes. Sinais obtidos a partir de diferentes eletrodos fazem com que exista um destaque maior em partes específicas da onda cardíaca.

3.1.4.1 Posicionamento dos Eletrodos

O posicionamento dos eletrodos no corpo do paciente seguem um padrão e as posições dos eletrodos são:

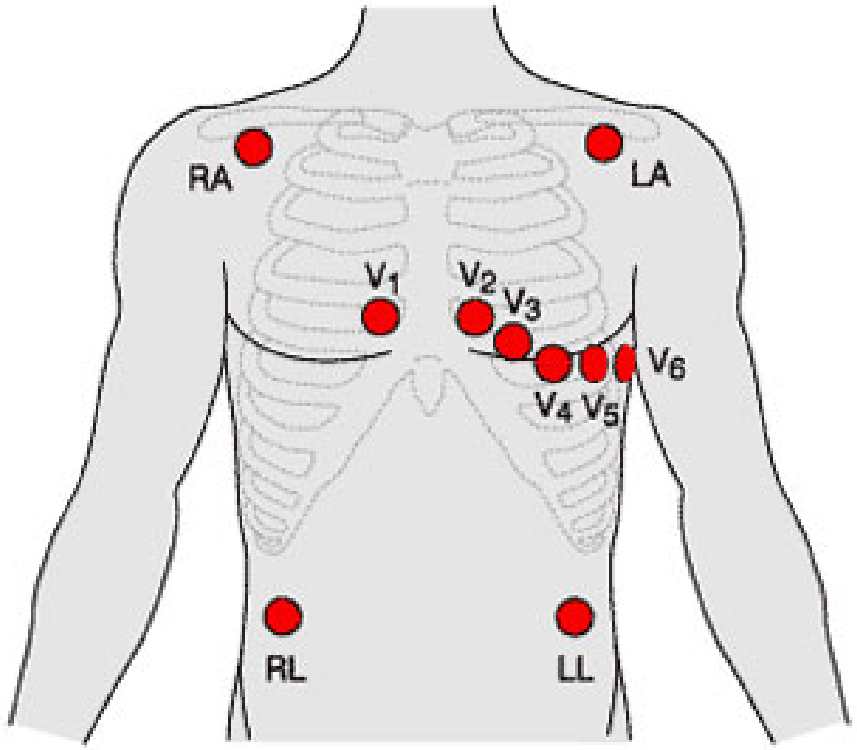


Figura 6 – Posicionamento dos eletrodos

- RL: *Right Leg*, no tórax, próximo à perna direita.
- LL: *Left Leg*, próximo à perna esquerda.
- RA: *Right Arm*, próximo ao braço direito.
- LA: *Left Arm*, próximo ao braço esquerdo.
- V1: lado direito do externo, no quarto espaço intercostal da costela.

- V2: posicionado à esquerda do externo, no quarto espaço intercostal da costela.
- V3: entre V2 e V4.
- V4: no quinto espaço intercostal na linha central clavicular.
- V5: posicionado no quinto espaço intercostal na linha auxiliar anterior.
- V6: mesmo plano de V4 na linha média auxiliar.

3.1.4.2 Derivações

Os sinais obtidos através dos eletrodos são chamados de derivações. Uma derivação nada mais é do que a diferença de potencial entre dois eletrodos, utilizando-se um terceiro como referência. Normalmente utiliza-se o eletrodo RL como referência. Os tipos de derivações são:

- Bi-polares

  As derivações bi-polares apresentam uma visão da atividade elétrica no coração no plano vertical. As derivações bi-polares e seus respectivos eletrodos utilizados na medição da diferença de potencial, todos utilizando RL como referência, como representado na Figura 7, são:

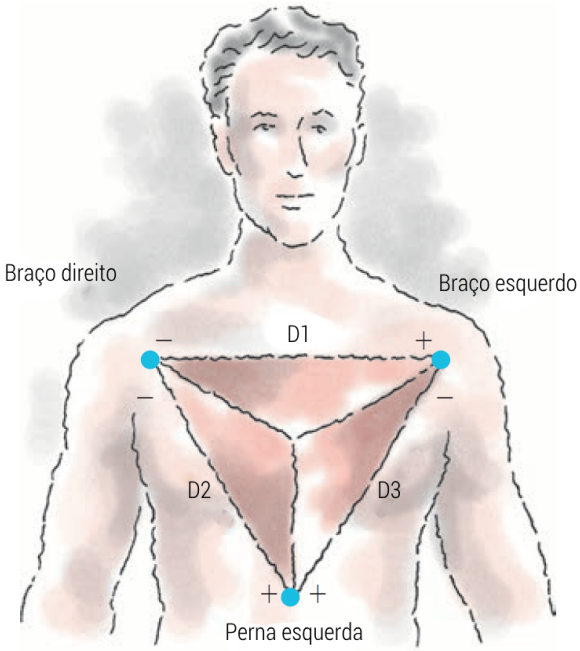


Figura 7 – Derivações Bipolares

  – D1: *ddp* entre RA e LA
  – D2: *ddp* entre RA e LL

– D3: *ddp* entre LA e LL

Normalmente a derivação D2 é utilizada para medir o ritmo cardíaco por apresentar a maior amplitude de onda durante a sístole ventricular, facilitando a detecção desses picos de atividade elétrica.

- Aumentadas

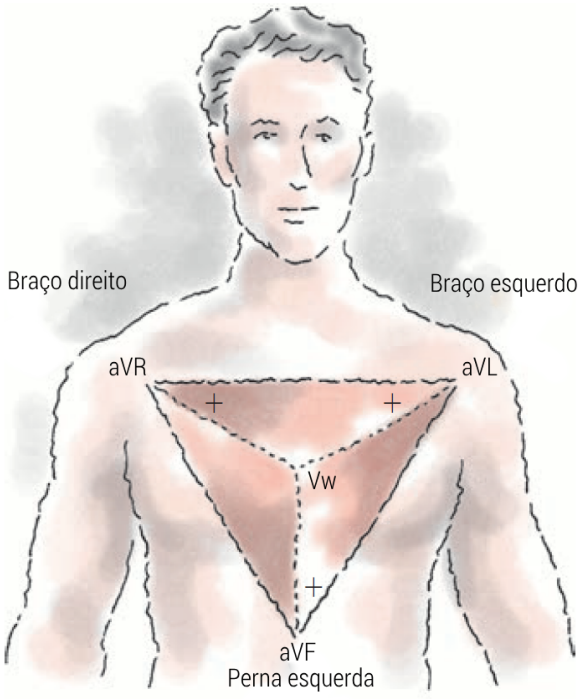


Figura 8 – Derivações Aumentadas

As derivações aumentadas, aVR, aVL e aVF, representadas na Figura ??, também mostram a atividade elétrica do plano frontal do coração. Os sinais dessas derivações podem ser obtidas através de simples cálculos matemáticos a partir dos sinais D1, D2 e D3, e utilizando um eletrodo virtual, chamado de Eletrodo Virtual Comum. O potencial do Eletrodo Virtual Comum é calculado através da seguinte equação:

$$V_w = \frac{1}{3}(RA + LA + LL) \tag{3.1}$$

Utilizando o valor do potencial V_w calculado, podemos calcular os potenciais das derivações aumentadas:

$$aVR = \frac{3}{2}(RA - V_w) \tag{3.2}$$

$$aVL = \frac{3}{2}(LA - V_w) \tag{3.3}$$

$$aVF = \frac{3}{2}(LL - V_w) \tag{3.4}$$

- Pré-cordiais

  As derivações pré-cordiais apresentam uma visão da atividade elétrica no coração no plano horizontal. Utilizando o eletrodo RL como referência, as pré-cordiais recebem o mesmo nome que seus respectivos eletrodos, de V1 a V6.

  Os sinais obtidos a partir dessas derivações ajudam o médico cardiologista a identificar problemas cardíacos que dizem respeito à lateralidade, por exemplo, identificação da localização: átrio/ventrículo esquerdo ou direito.

3.1.5 Sinal do ECG

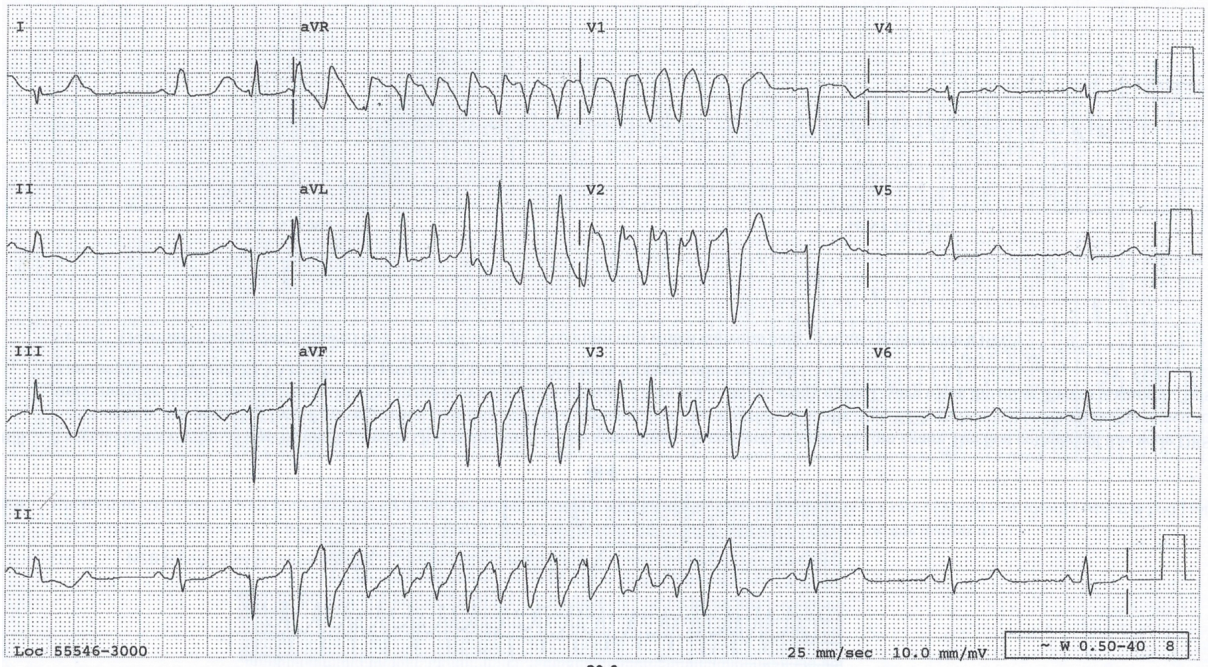


Figura 9 – Exemplo de exame eletrocardiográfico

Na Figura 9 temos um exemplo de exame eletrocardiográfico. Nesse exame, os sinais denominados *I*, *II* e *III* representam os sinais das derivações D1, D2 e D3, respectivamente. Também na figura, os sinais denominados V1 a V6 representam as derivações de mesmo nome, da mesma forma que aVR, aVL e aVF. Na última linha do gráfico é mostrada uma captura estendida da derivação D2, com objetivo de análise de ritmo cardíaco especificamente.

O sinal de ECG pode ser digitalizado através de um conversor analógico-digital, transmitido e armazenado para análise posterior. São aplicados diversos filtros sobre o sinal adquirido, com o objetivo de retirar ruídos e interferências presentes. Algumas fontes de ruído e interferência existentes são a rede elétrica (tipicamente 60 Hz), lâmpadas

fluorescentes (tipicamente 120 Hz), atividade muscular que não o coração (de 10 a 40 Hz), entre outros. A faixa de frequência de interesse do sinal de ECG é de 10 a 40 Hz, como mostrado na Figura 10.

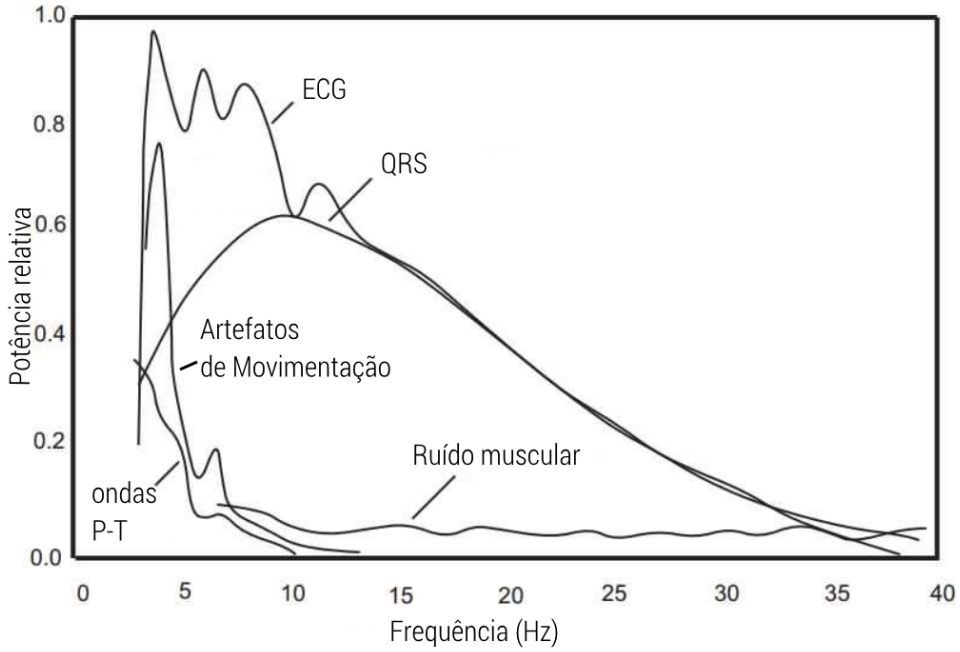


Figura 10 – Espectro de um sinal de ECG típico com ritmo de 150 bpm

3.2 Classificação de Frases

Até o início do presente trabalho, havia disponível cerca de 31 mil exames de eletrocardiograma cujos laudos foram realizados por médicos cardiologistas. Nesses exames, nas conclusões dos laudos, parte mais importante do laudo, estão descritos a existência ou não existência de cardiopatias, distúrbios de ritmo, bloqueios, alterações, entre outros possíveis problemas associados ao coração. Visto que o conteúdo da conclusão dos laudos está armazenado sem formatação, ou seja, em texto livre, escrito por médicos cardiologistas, se faz necessário encontrar uma maneira de separar os exames através da escrita contida nas conclusões dos laudos.

Nós humanos possuímos a habilidade de extrair significados e relações a partir de texto escrito, utilizando a leitura. Para um computador, isso se torna um tanto quanto difícil pois para o computador os textos são apenas uma sequência de números que, à princípio, não carregam nenhum valor semântico. Um dos objetivos deste trabalho é desenvolver um sistema capaz de extrair o significado dos textos das conclusões de laudos de exames de ECG de forma que seja possível para um computador "entender"que determinadas sequências de caracteres possuem um significado especial.

O objetivo final de obter um sistema capaz de "entender" as conclusões de laudos é tornar possível a classificação dos 31 mil exames com intuito de separar os exames em conjuntos que possuam as mesmas características distintas, buscando a geração de um conjunto de exames para treinamento e validação de um sistema de classificação dos traçados de ECG.

Podemos imaginar o esforço necessário para classificar estas conclusões manualmente. Estima-se que uma pessoa, em média, levaria 30 segundos para classificar um laudo. Não são necessários muitos cálculos para observarmos que é inviável realizar a classificação manual de todo o conjunto de laudos em um tempo hábil. Por esse motivo, optou-se pelo desenvolvimento do sistema de classificação de frases.

Basicamente, o objetivo desse sistema é receber como entrada um texto não-formatado e disponibilizar como saída quais as classificações, quanto a cardiopatias, distúrbios, etc., estão contidos nele. Há alguns detalhes que devemos nos preocupar para obtenção de boa performance para o sistema: presença de erros gramaticais, erros de pontuação, diferenças entre letras maiúsculas e minúsculas, conteúdos que não possuem relação com problemas cardíacos (nomes, números, endereços de e-mail, etc.), entre outros. Na presença desses "problemas", seria muito difícil implementar, utilizando programação clássica, um algoritmo que considerasse todos os erros gramaticais, conteúdo, etc., de forma que não houvessem erros de identificação, pois seria necessário inserir nesse algoritmo, todas as combinações possíveis de erros e acertos. Por esse motivo foi escolhido a utilização de redes neurais para resolver esse problema.

3.3 Classificação de Traçado

Objetivo principal deste trabalho, a identificação e classificação de necessidade ou não de realização de laudo por especialista, a ser aplicada diretamente sobre o traçado obtido através do exame ECG, busca diferenciar entre exames que apresentam condições *Normais* e *Anormais* utilizando o traçado obtido através do eletrocardiógrafo.

Existem técnicas parecidas em utilização na medicina humana como por exemplo a utilização de redes neurais para a classificação de câncer de mama através de imagens. Também existem outros trabalhos utilizando o sinal de ECG de humanos com objetivo de classificar entre categorias de problemas distintos.

3.4 Redes Neurais

Redes neurais são um modelo computacional inspirado nos neurônios e suas conexões (sistema nervoso) dos seres vivos. Essa "rede" de células possui a capacidade de aprendizado e manutenção desse conhecimento.

Conhecimento e figuras desta seção foram extraídos de [3].

3.4.1 Neurônio Biológico

O neurônio biológico é uma célula especializada que possui características marcantes, as quais propiciam a interconexão com os neurônios adjacentes. Também são capazes de responder à estímulos elétricos e, em certas condições, modificar o estímulo recebido antes de passá-lo adiante.

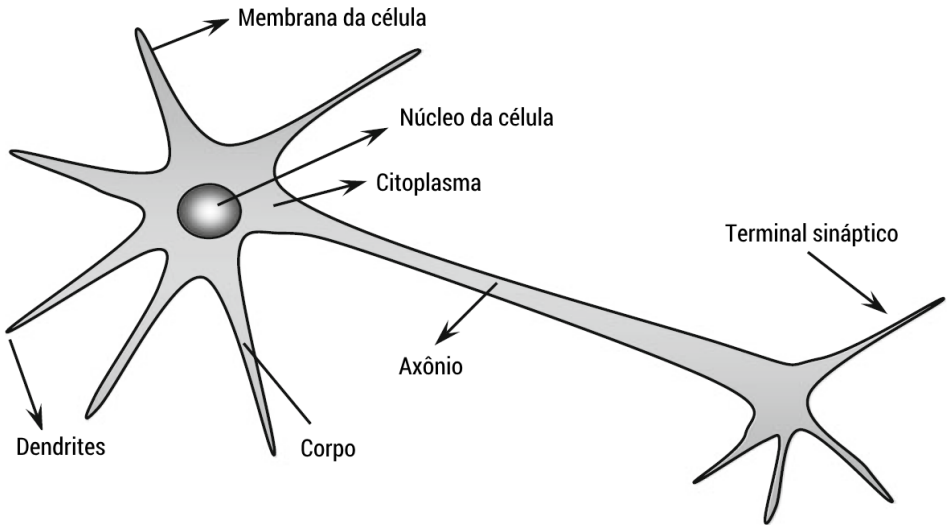


Figura 11 – Neurônio biológico simplificado

Vemos na Figura 11 a representação simplificada de um neurônio biológico. O neurônio pode ser divido em três partes distintas: dendrites, corpo e axônio.

O dendrito é composto por diversas extensões, chamadas de dendrites. O propósito principal das dendrites é receber estímulos de outros neurônios ou do ambiente externo (neurônios sensoriais).

O corpo do neurônio é responsável pelo processamento dos estímulos recebidos das dendrites e produção de um potencial de ativação, que indica se o neurônio pode disparar um impulso elétrico através do seu axônio.

O axônio é composto por uma única extensão cujo objetivo é guiar os impulsos elétricos disparados pelo corpo até outros neurônios. A terminação do axônio é composta por diversas extensões chamadas de sinapses. As sinapses são as interconexões que ocorrem entre neurônios, mais precisamente as ligações entre axônio e dendrites dos neurônios adjacentes.

3.4.2 Neurônio Artificial

O neurônio artificial é uma tentativa de modelar matematicamente o comportamento dos neurônios biológicos. Sua modelagem surgiu da análise dos comportamentos dos neurônios na geração e transmissão dos estímulos elétricos. Na Figura 12 vemos um diagrama de blocos representando o modelo de um neurônio artificial.

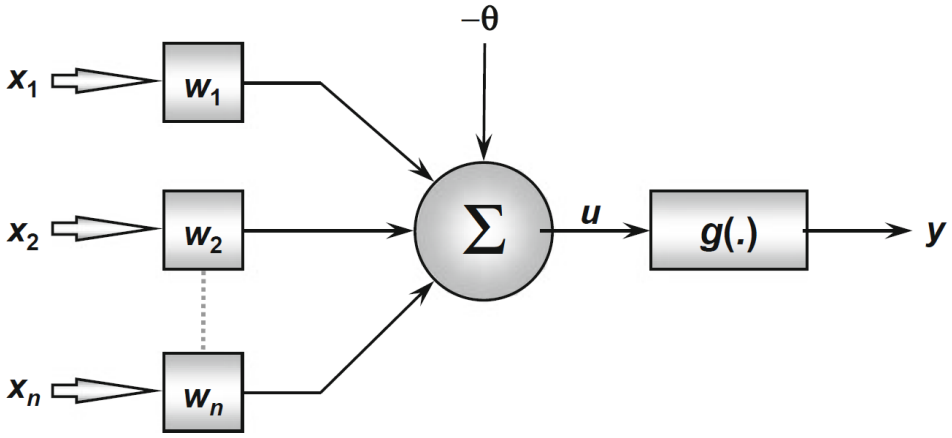


Figura 12 – Neurônio artificial

Os neurônios artificiais são implementados utilizando multiplicações, somas e uma função de ativação potencialmente não-linear. No modelo clássico de neurônio artificial, este recebe valores de entrada, multiplica por "pesos", opcionalmente agrega um valor de *bias*, soma o resultado e aplica a função de ativação para obter o valor de saída.

As equações que descrevem o funcionamento de um neurônio artificial clássico são:

$$u = \sum_{i=1}^{n} w_i x_i - \theta \tag{3.5}$$

$$y = g(u) \tag{3.6}$$

onde, n é o número de entradas, x_i a entrada de índice i, w_i o peso associado à entrada x_i, θ é o valor de *bias*, u é o somatório parcial, $g(u)$ é a função de ativação e y a saída final do neurônio.

3.4.2.1 Funções de Ativação

As funções de ativação podem ser classificadas em basicamente dois grupos: as *diferenciáveis* e as *parcialmente diferenciáveis*. As funções diferenciáveis são aquelas cujas primeiras derivadas são contínuas em $\mathbb{R}$. As parcialmente diferenciáveis são aquelas cujas primeiras derivadas não são contínuas em $\mathbb{R}$.

Algumas funções de ativação diferenciáveis seguem abaixo

- Função Logística:

$$g(u) = \frac{1}{1 + e^{-\beta u}} \tag{3.7}$$

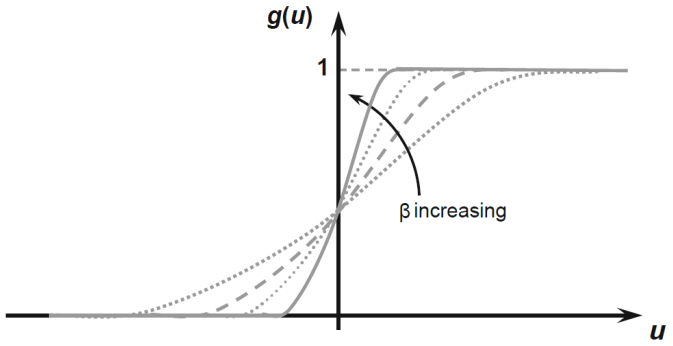


Figura 13 – Função Logística

A função logística recebe valores de todos os $\mathbb{R}$ e mapeia-os no intervalo $(0, 1)$. O parâmetro β determina o declive da rampa no ponto de inflexão da curva. Na Figura 13 está mostrado o gráfico da função logística e a influência do parâmetro β. Valores de β maiores fazem com que a função aproxime a Função Degrau.

- Função Linear:

$$g(u) = u \tag{3.8}$$

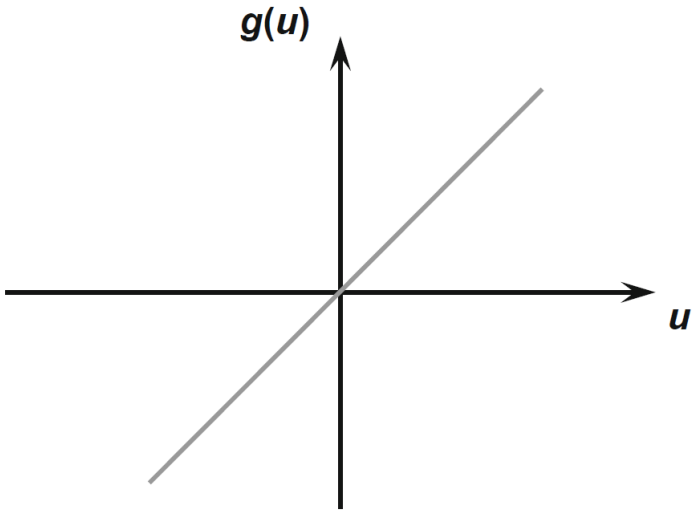


Figura 14 – Função Linear

A saída da função linear é seu próprio argumento, ou seja, não há transformação no seu resultado. Na Figura 14 vemos o seu gráfico.

- Função Tangente Hiperbólica (*tanh*)

$$g(u) = \frac{1 - e^{-\beta u}}{1 + e^{-\beta u}} \tag{3.9}$$

A função tangente hiperbólica mapeia valores de $\mathbb{R}$ ao intervalo $(-1, 1)$. Da mesma forma que a função logística, com o aumento do valor de β, há aumento do declive no ponto de inflexão. É interessante lembrar que tanto a função logística quanto a função tangente hiperbólica fazem parte de um grupo de funções chamado de sigmóides. Vemos na Figura 15 o gráfico da função e a influência do parâmetro β.

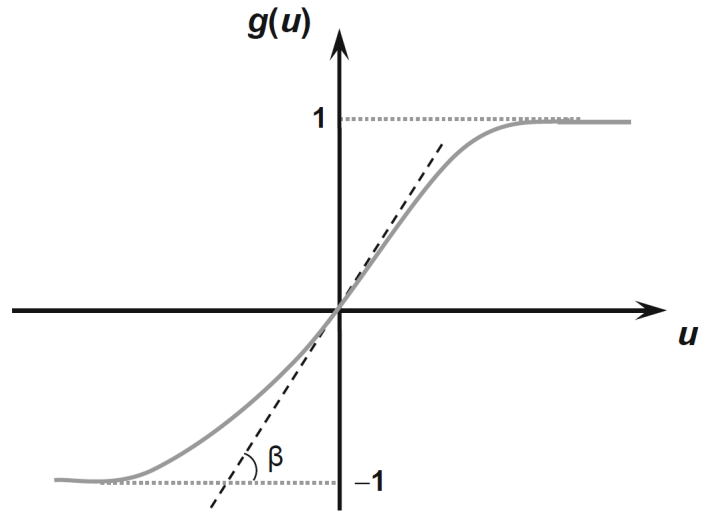


Figura 15 – Função Hiperbólica

Algumas funções de ativação parcialmente diferenciáveis

- Função Degrau

$$g(u) = \begin{cases} 1, se\ u \geq 0 \\ 0, se\ u < 0 \end{cases} \tag{3.10}$$

A função degrau é avaliada em 1 caso o seu argumento seja maior ou igual a zero e em 0 quando seu argumento é menor que zero. Há também variações da função degrau como a função sinal. Vemos na Figura 16 o seu gráfico.

- Função Linear Retificada (ReLU - *Rectified Linear Unit*)

$$g(u) = \begin{cases} u, se\ u \geq 0 \\ 0, se\ u < 0 \end{cases} = max(u, 0) \tag{3.11}$$

A função $ReLU$ é muito utilizada em redes neurais para classificação de imagens. O gráfico da função está representado na Figura 17. Possui a vantagem de ser uma

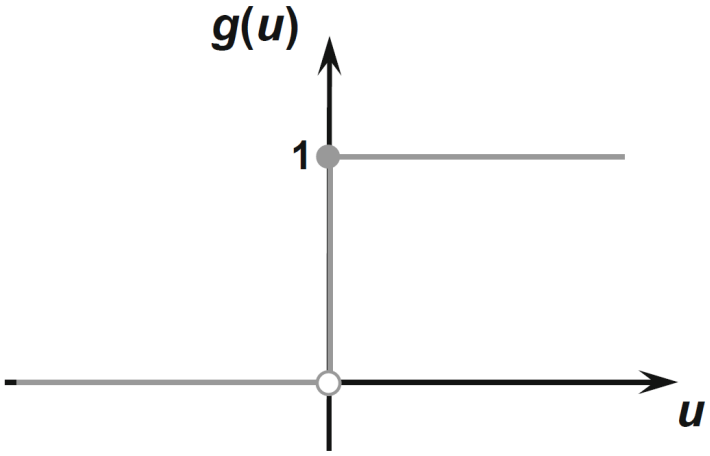


Figura 16 – Função Degrau

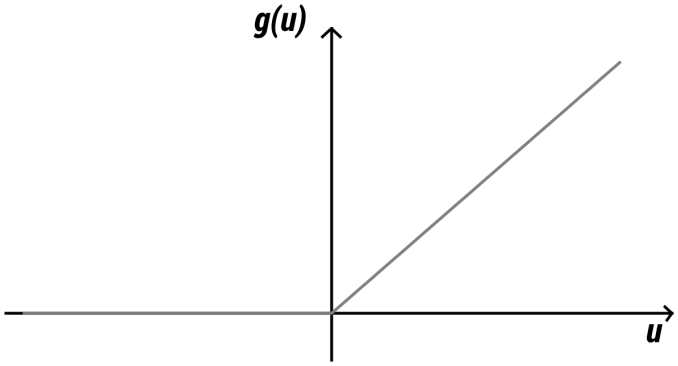


Figura 17 – Função ReLU

função não-linear, porém com baixo custo computacional, comparada às funções tangente hiperbólica e logística, o que acelera o processo de treinamento da rede ao mesmo tempo que propicia praticamente a mesma acurácia final.

- Função Linear Retificada com Vazamento (LeakyReLU)

$$g(u) = \begin{cases} u, se\ u \geq 0 \\ \alpha\ u,\ \ se\ u < 0 \end{cases} \tag{3.12}$$

Essa função permite um pequeno gradiente quando o valor de u for negativo. Vemos o seu gráfico na Figura 18.

Pode-se encontrar análises mais aprofundadas sobre as diferentes funções de ativação e as consequências dos seus usos no desempenho de redes neurais em [4] e [5].

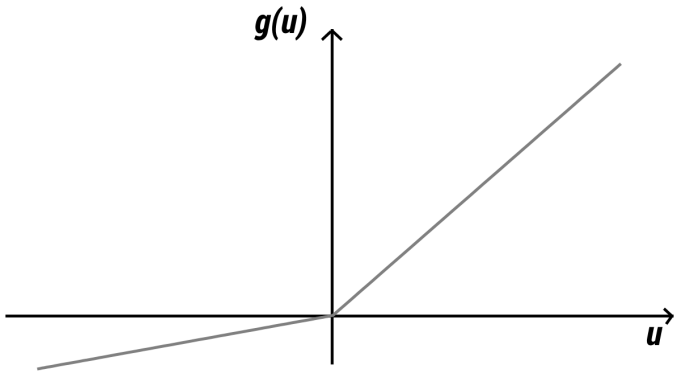


Figura 18 – Função LeakyReLU

3.4.3 Algumas Arquiteturas

Nesta seção serão apresentadas algumas das principais arquiteturas utilizadas juntamente com breves discussões no que diz respeito às vantagens e desvantagens de cada arquitetura. A arquitetura de uma rede é definida pelo arranjo dos neurônios, suas funções de ativação, quantidade de neurônios e número de conexões entre os mesmos.

Normalmente neurônios são organizados em camadas (layers). Podem ser divididas em basicamente três tipos de camadas. São indispensáveis para um modelo de redes neurais: camada de entrada (input layer) e camada de saída. Também existe outro tipo chamado de camada escondida (hidden layer).

A camada de entrada é responsável por receber a informação, ou seja, dados, sinais, *features* ou medições do ambiente externo. Dá-se preferência a entradas normalizadas, dentro dos intervalos de saída das funções de ativação escolhidas. A normalização facilita a precisão numérica das operações matemáticas realizadas na rede.

Camadas de saída são camadas cujos neurônios apresentam o que podemos considerar como saídas da rede neural. Normalmente usa-se um neurônio com função de ativação logística para classificadores binários ou outra função de ativação para redes que realizam regressões.

Chamamos de *hidden layers* todas as camadas que encontram-se entre as camadas de entrada e saída. Essas camadas são as principais responsáveis pelas propriedades de aprendizado de uma rede neural.

3.4.3.1 Redes Feed-Forward (dense)

É conhecida como a arquitetura clássica de redes neurais. Consiste em realizar a ligação de diversos neurônios artificiais, todos conectados entre si, daí o nome "densa", pela grande densidade de conexões entre neurônios. São utilizadas na resolução de diversos problemas, como: aproximação de funções, classificação de padrões, identificação de

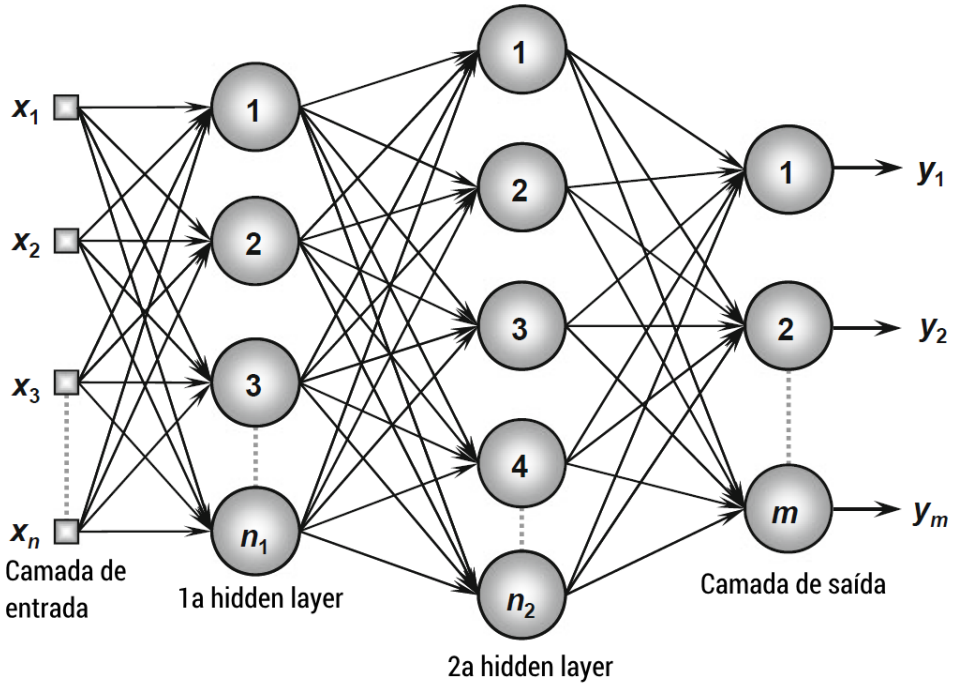


Figura 19 – Rede Feed-Forward de múltiplas camadas

sistemas, controle de processos, otimização, entre outros.

Temos representada na Figura 19 um exemplo de rede do tipo feed-forward, com camada de entrada, *hidden layers* e camada de saída. Vemos que todos os neurônios de uma camada em conjunto com a próxima estão conectados entre si. Essa característica faz com que redes desse tipo apresentem um aumento do número de parâmetros com o aumento da quantidade de neurônios em cada camada.

Como vantagens pode-se destacar a alta capacidade de aprendizado desse tipo de arquitetura. Devemos observar que alta capacidade de aprendizado nem sempre corresponde à alta capacidade de generalização do conhecimento, por isso deve-se tomar cuidado no momento do treinamento deste tipo de rede. Como desvantagens pode-se citar o alto custo computacional, visto que para cada camada adicionada, são inseridas novas conexões entre todos os neurônios da camada anterior e a nova.

3.4.3.2 Redes Convolucionais (CNNs, conv-nets)

Essa é uma arquitetura mais moderna e diferencia-se da convencional por utilizar conjuntos de neurônios chamados de filtros. Entradas para esse tipo de rede podem ser sequencias de dados 1D, 2D, 3D, com múltiplos canais, se necessário. São chamadas de redes convolucionais pois aplicam-se os filtros da rede sobre toda a sequencia de entrada da rede, de forma parecida com a convolução clássica da matemática, daí o nome. A operação realizada é o produto escalar entre o campo receptivo e os pesos do filtro.

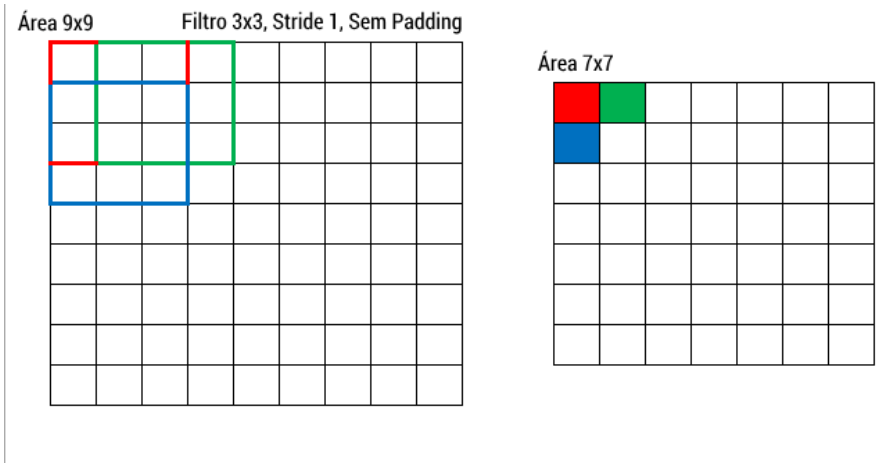


Figura 20 – Operação de convolução 2D utilizando único filtro de dimensões 3x3

Na figura 20 está representado um volume de dados onde um filtro de tamanho 3x3 percorre nas duas dimensões aplicando seus pesos e função de ativação. Na figura da esquerda temos as áreas dos campos receptivos dos filtros destacadas nas diferentes cores e na figura da direita a saída respectiva de cada aplicação do filtro sobre o volume de entrada. Essa operação ocorre movendo-se o campo receptivo em uma unidade de distância por vez. A distância percorrida pelo filtro entre duas operações consecutivas é chamada de *stride*.

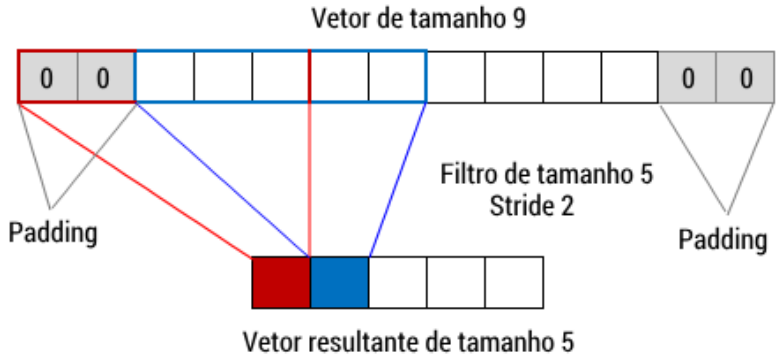


Figura 21 – Operação de convolução 1D utilizando único filtro de dimensões 5x5, Stride 2 e Padding

Na figura 21 temos agora a representação da operação de convolução em uma dimensão. Temos também a adição de *padding*. Padding é quando ocorre a adição de dados nulos no começo e no final do volume de dados de forma que podemos manipular a dimensão do vetor de saída mais livremente. Comumente é escolhido valores de padding com o objetivo de dimensionar o volume de saída para que este mantenha-se igual ao de entrada.

Temos que a formula para determinar a dimensão de saída de uma dada camada

de convolução é:

$$O = \frac{(W - K + 2P)}{S} + 1 \tag{3.13}$$

onde O é a dimensão de saída, W é a dimensão de entrada, K é a dimensão do filtro, P é dimensão do padding e S o valor de stride.

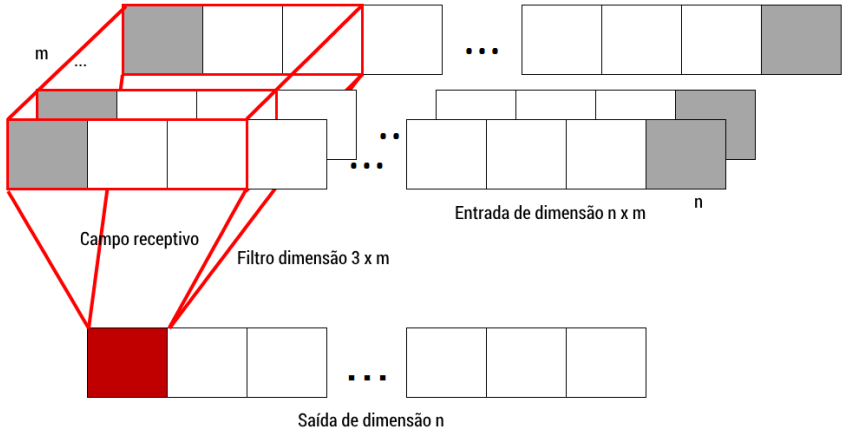


Figura 22 – Operação de convolução 1D com único filtro de dimensão 3 x m e Padding em entrada de múltiplos canais

Na figura 22 temos o exemplo de aplicação de operação de convolução 1D em entrada com múltiplos canais com stride 1 e padding. Observemos que o filtro apresenta segunda dimensão de modo à, após operação de produto escalar e função de ativação, obtermos resultado escalar. Dessa maneira chegamos em um filtro que é capaz de considerar diferentes visualizações de um mesmo sinal. Analogamente para o caso 2D com múltiplos canais (ex: imagens com canais RGB), é possível apresentar mais informação para um mesmo filtro de uma vez.

Como vantagem destaca-se o baixo custo computacional, comparando-se à redes feed-forward com o mesmo tamanho de vetor de entrada. Como desvantagem, apresenta-se a necessidade de um volume de dados para treinamento muito maior do que a das redes feed-forward, na ordem de milhares até mesmo milhões de exemplos, dependendo do tipo de problema.

3.4.3.3 Redes Recorrentes (RNNs)

Na figura 23 temos a representação do fluxo de entrada e saída de uma rede neural recorrente. Na figura, h representa o estado interno, x a nova entrada, o a saída, U a função de entrada, W a função de saída e V a função de estado. Temos que é inserido no modelo da rede recorrente uma sequência de entradas x e obtidos como saída uma sequência de saídas o.

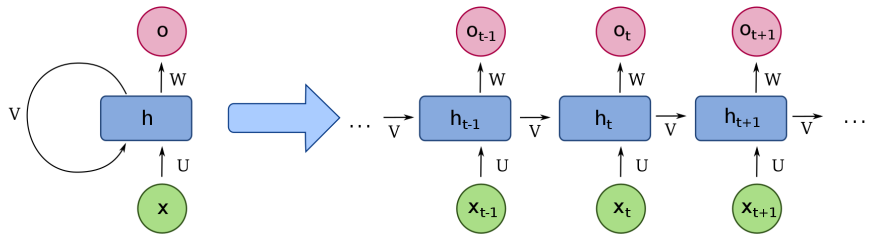


Figura 23 – Rede Neural Recorrente

Diferentemente das arquiteturas mostradas até agora, a saída de uma rede recorrente pode ser uma sequência sem limite de tamanho. Normalmente as redes recorrentes trabalham de forma sequencial, ou seja, cada nova entrada é inserida após o processamento da entrada anterior.

Como vantagem, destaca-se a possibilidade de processar sequencias arbitrárias de dados. Como desvantagem, custo computacional elevado e dificuldade de elaboração do conjunto de dados para treinamento.

3.4.4 Treinamento

O ajuste de pesos de um neurônio é chamado de "treinamento". É realizado através de um algoritmo chamado de *back-propagation* (comumente chamado de *backprop*). O treinamento é um processo iterativo, onde a cada iteração, os pesos dos neurônios são ajustados de forma que, expondo uma entrada de exemplo, cuja saída já é conhecida, são calculados os gradientes de erro (de cada neurônio e através de todas as camadas), e conhecido o gradiente de erro, são feitos ajustes nos pesos de cada neurônio de forma que expondo a mesma entrada de exemplo novamente, o erro de saída seja minimizado iterativamente.

Um conjunto de dados de treinamento é comumente chamado de *dataset*. Define-se pelo nome de *epoch* quando, durante o treinamento, é apresentado o conjunto de treinamento em sua totalidade.

3.4.4.1 Treinamento Supervisionado

O treinamento supervisionado é a forma padrão de treinamento de redes neurais. Consiste em compilar um conjunto de dados categorizados de alguma maneira, onde entradas e saídas esperadas (rótulos) são conhecidas. Após esse passo, separa-se o conjunto de dados em dois novos conjuntos: treinamento e teste. O conjunto de dados de treinamento será o conjunto de dados utilizado para treinamento efetivo da rede e o conjunto de testes será utilizado para avaliar a performance da rede. Usa-se comumente a proporção de 50% a

90% do conjunto de dados para treinamento e o restante para teste. É importante também que haja seleção randômica dos exemplos que vão para cada conjunto, de forma a evitar a ocorrência de padrões nas sequências de exemplos a serem apresentados à rede.

É importante que haja separação clara entre os conjuntos de treinamento e teste, para podermos distinguir uma boa performance de "overfitting". Overfitting é quando uma rede passa a "decorar" os exemplos de treinamento sem conseguir generalizar o conhecimento obtido. Caracteriza-se quando a acurácia da rede, utilizando-se o conjunto de treinamento para avaliação, aproxima-se de 100%, mas, ao mesmo tempo, a acurácia, utilizando-se o conjunto de teste para avaliação, mantém-se constante ou variando aleatoriamente.

Também é importante que o conjunto de treinamento seja balanceado, ou seja, exista um número parecido de exemplos para cada categoria. Dessa maneira, evita-se que a acurácia da rede aumente em casos onde a rede identifica apenas a classe com o maior número de exemplos do conjunto. Por exemplo: tem-se um conjunto de dados onde 90% dos elementos são da categoria A e 10% da categoria B. Caso a rede neural classifique todas as entradas como A, teremos 90% de acurácia, mesmo errando 100% dos exemplos categorizados como B, nos levando a crer que a rede está respondendo ao treinamento quando na verdade esta não consegue diferenciar entre as categorias. Mais análises sobre a importância do balanceamento e como tratar esse problema em [6].

3.4.4.2 Treinamento Não-Supervisionado

O treinamento não-supervisionado se dá quando não há a necessidade de conhecimento prévio na categorização do conjunto de dados. Portanto, o sistema deverá ser capaz de automaticamente identificar padrões e agrupar os elementos de todo o conjunto de treinamento, ajustando pesos dos neurônios da rede de forma que esses conjuntos sejam refletidos no modelo.

3.4.4.3 Funções de Perda

As funções de perda (loss functions) são utilizadas para avaliar, durante o treinamento e teste, o erro entre a saída esperada e a saída obtida pela rede. Recebem como entrada o vetor de saída da rede e o vetor de saída esperada. Algumas comparações entre performance de funções de perda são feitas em [7].

São alguns exemplos de loss functions:

- Binary Crossentropy

  Utilizada principalmente para classificadores binários.

- Categorical Crossentropy

Utilizada principalmente para classificadores multi-classe.

- Mean Squared Error

  Utilizada principalmente para redes que realizam regressão.

3.4.4.4 Algoritmos de treinamento

Existem vários algoritmos de treinamento para redes neurais. Todos, à princípio, utilizam os gradientes de erro na rede para ajustar os ganhos e buscam minimizar a função de perda escolhida. Diferem na forma com que é feita a iteração sobre o conjunto de dados de treinamento e alguns parâmetros que afetam a velocidade de convergência.

Alguns algoritmos de treinamento:

- Gradient Descent - GD
- Stochastic Gradient Descent - SGD

  Apresentado em [8]. Comparações entre diferentes algoritmos (GD, SGD, etc) estão presentes em [9].

- Adam

  Algoritmo apresentado em [10].

3.4.5 Outros tipos de camadas e operações

Existem outros tipos de operações que podem ser realizadas em redes neurais que objetivam diminuir o número de parâmetros, melhorar a capacidade de generalização de uma rede, normalizar os pesos dos neurônios, entre outros. Serão apresentadas algumas operações e camadas utilizadas neste trabalho.

3.4.5.1 Dropout

É uma operação que busca, durante a fase de treinamento, anular a saída de neurônios escolhidos aleatoriamente da camada anterior. O objetivo de anular alguns neurônios é aumentar a robustez da rede fazendo com que nem todos os neurônios de determinada camada sejam necessários para a classificação ou regressão de certas entradas, como mostrado em [11]. Normalmente é escolhido anular a saída de 20% a 70% dos neurônios de uma camada. Outro efeito da utilização desse tipo de camada é a melhoria na capacidade de generalização de uma rede pois tem o efeito de diminuição do poder de uma rede "decorar"as entradas.

3.4.5.2 MaxPooling

A operação de MaxPooling consiste em, utilizando janelas de tamanho parametrizável, comparar os elementos da janela e extrair apenas o elemento com o maior valor numérico presente. É muito aplicada na sequência de camadas de convolução, com o intuito de diminuir a dimensão das camadas posteriores. Análises mais aprofundadas podem ser encontradas em [12].

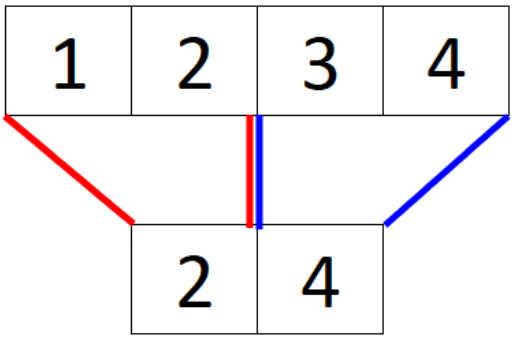


Figura 24 – Operação de MaxPooling

Na figura 24 está mostrada a operação realizada em um vetor de entrada de dimensão 4 utilizando valor de stride 2. A operação de MaxPooling tem como objetivo reduzir a dimensão dos vetores de camadas superiores porém preservando as features mais relevantes.

3.4.5.3 Softmax

Softmax é uma camada que substitui as funções de ativação da camada anterior. É utilizada em classificadores multi-categoria. É uma função que estende a função logística de forma que considera todas as saídas da camada anterior no cálculo da sua saída. A saída da função softmax é a distribuição de probabilidade sobre n categorias.

$$x'_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{3.14}$$

onde x é o vetor de saídas, x_i é a saída do neurônio de índice i e n é o número de saídas.

3.4.6 NLP (Natural Language Processing) utilizando redes neurais

Redes neurais também podem ser aplicadas em problemas de processamento de texto, como tradução de idiomas, análise de sentimento, chatbots, sugestão de palavras, entre outros. Dos tipos de redes existentes, as CNNs e RNNs são as mais empregadas na resolução desse tipo de problema. Mais análises podem ser encontradas em [13].

Na maioria dos casos de NLP, existe a necessidade de se fazer um pré-processamento do texto. Etapas de pré-processamento incluem, sem ordem particular: remoção de acentos e caracteres especiais, normalização (maiúscula ou minúsucla), *stemming* (retirada de sufixos como -ando, -assem, -avam, etc.), *tokenization* (separar as palavras de uma frase criando uma sequência de palavras), entre outros.

Não é necessário a aplicação de todas as técnicas de pré-processamento textual para todo problema de NLP. Isso depende do problema em particular e diferentes combinações de processamentos devem ser testadas com o intuito de conseguir os melhores resultados, como mostrado em [14].

Um dos requisitos de entrada para uma rede neural é de que as entradas sejam (na maioria dos casos) números reais ou inteiros (float ou integer), porém, a representação natural de textos em um computador (chamada de encoding) não é um bom formato, pois as letras são representadas utilizando (no caso de ASCII ou UTF-8) números inteiros de 65 a 122 (A-Z,a-z). Já foi demonstrado em diversos estudos que redes neurais obtém melhores resultados quando são utilizados dados normalizados (números reais de 0 a 1) como entrada da rede.

Para resolver esse problema, utiliza-se uma técnica de transformação chamada de *embedding*.

3.4.7 Embeddings

Embeddings são basicamente vetores de tamanho arbitrário que, de alguma maneira, mapeiam um objeto (palavra, letra, imagem) à um vetor do $\mathbb{R}^n$. Muitas vezes, essa transformação não segue nenhuma lógica, sendo apenas uma maneira de transformar entradas que seriam ditas "ruins" para a rede em novas entradas, desta vez, mais apropriadas para o processamento por uma rede neural.

Os embeddings, ou seja, o mapeamento objeto→vetor de $\mathbb{R}^n$, podem ser aprendidos em conjunto com os pesos dos neurônios durante o processo de treinamento de uma rede neural.

3.4.7.1 Word2vec

Word2Vec é uma família de redes neurais desenvolvida especialmente para o treinamento de embeddings. O treinamento dessa rede é feito de forma não-supervisionada, recebendo como entrada uma sequência de *tokens* (palavras). O conjunto de textos utilizado para treinamento é chamado de *corpus*.

O principal objetivo de utilização do Word2Cec é mapear as palavras a um vetor no $\mathbb{R}^n$ de tal maneira que palavras com significados semelhantes disponham-se em lugares próximos (distância euclidiana) no espaço vetorial resultante, como mostrado em [15]

e [16]. Outro fenômeno que emerge do treinamento dos embeddings Word2Vec é a relação entre palavras. Por exemplo, se realizarmos operações com os vetores correspondentes das palavras "Paris", "França", Itália", obtemos um novo vetor cujo vetor mais próximo corresponde à palavra "Roma". Desta forma:

$$Paris - França + Itália = Roma \tag{3.15}$$

Outro exemplo de relação encontrada é: $cobre - Cu + zinco = Zn$

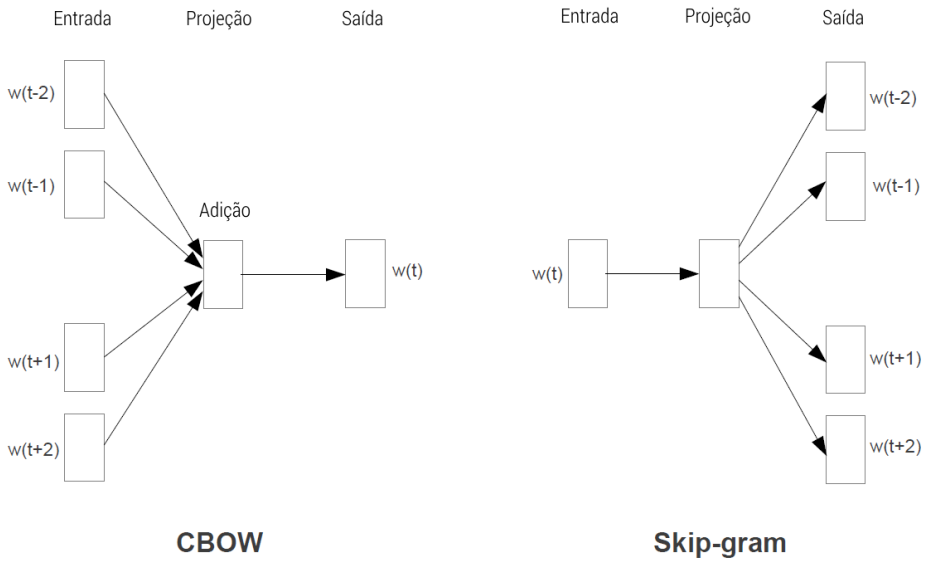


Figura 25 – Modelos CBOW e Skip-gram

3.4.7.1.1 Continuous Bag of Words - CBOW

No modelo CBOW de treinamento dos embeddings, o treinamento transcorre da seguinte maneira: uma janela de palavras é processada, onde a palavra do meio é definida como a saída esperada e a soma dos vetores correspondentes às palavras no entorno da palavra do meio é definida como a entrada exemplo, como mostrado na figura 25.

3.4.7.1.2 Skip-gram

No modelo Skip-gram, temos o contrário do modelo CBOW. Tenta-se, utilizando a palavra do meio, prever as palavras do entorno.

3.4.7.2 Doc2vec

O Doc2Vec é uma extensão dos conceitos Word2Vec para "documentos". O termo documentos é empregado para descrever qualquer tipo de texto, seja uma frase, parágrafo, um artigo, e até mesmo um livro. Analogamente ao Word2Vec, a rede Doc2Vec tem por

objetivo a obtenção de embeddings para documentos e foi apresentado à comunidade científica através da publicação [17].

A diferença principal entre o Doc2Vec e o Word2Vec é a adição de um vetor extra específico que carrega informação sobre o documento, além dos vetores de palavras. A informação embutida no vetor do documento dá contexto ao restante dos vetores.

3.5 Bibliotecas

Nesta seção serão apresentadas algumas bibliotecas utilizadas para auxiliar na implementação de diversas etapas deste projeto.

3.5.1 NLTK

Biblioteca sem custo e open-source dedicada ao desenvolvimento de softwares em Python capazes de processar dados da linguagem humana. Provê uma gama de ferramentas de processamento de texto incluindo classificação, tokenização, stemming, entre outros. Também facilita o acesso a diversos bancos de dados textuais de modo que o desenvolvedor consiga facilmente utilizá-los em sua aplicação. Acesso à mais informações em [18].

3.5.2 TensorFlow

TensorFlow é uma biblioteca open-source para computação numérica de alta performance. Com sua arquitetura flexível permite a execução em diversas plataformas (CPUs, GPUs), desktops, servidores e até dispositivos móveis. Foi desenvolvida originalmente por pesquisadores e engenheiros da Google.

Seu principal uso é no campo de machine learning e deep learning mas também pode ser utilizada em vários outros campos da ciência.

Mais informações em [19].

3.5.3 Keras

Keras é uma biblioteca voltada à redes neurais. Provê uma API de alto-nível na linguagem Python para construção de arquiteturas e modelos. O foco da biblioteca é permitir experimentação rápida.

É capaz de utilizar a biblioteca TensorFlow, CNTK ou Theano como sua base. Mais informações em [20].

3.5.4 TensorBoard

A ferramenta TensorBoard foi desenvolvida pela Google com o objetivo de auxiliar a análise de redes neurais durante a fase de treinamento. É possível visualizar gráficos mostrando a evolução da acurácia e perda ao longo das *epochs* de treinamento além de gráficos que mostram a distribuição numérica dos pesos dos neurônio de cada camada. Também é possível visualizar grafos representando a arquitetura da rede. Além disso, é possível realizar a comparação dos índices de desempenho de diferentes redes comparando-as através dos gráficos. Mais informações em [21].

4 Desenvolvimento

Neste capitulo serão explorados as motivações e requisitos funcionais e não-funcionais dos diferentes sistemas implementados durante este projeto.

Como explanado no começo deste documento, o banco de dados de laudos de exames de ECG da empresa InPulse conta com mais de 31 mil exames elegíveis para estudo neste projeto. Os requisitos para seleção dos exames foram:

1. Possuir laudo
2. Possuir dados completos (derivações D1 e D2 presentes)
3. Laudo respectivo possuir conteúdo no campo conclusão (um parágrafo de texto não-formatado).

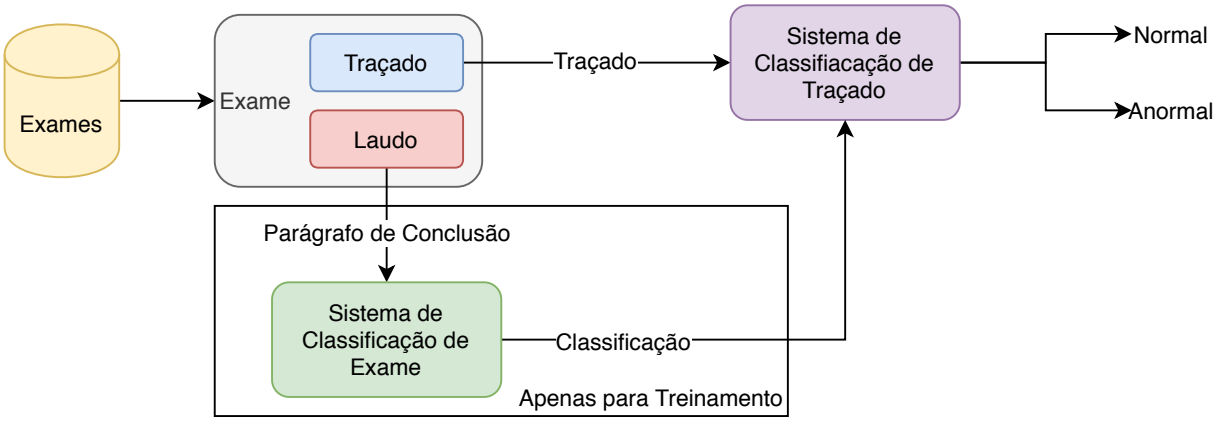


Figura 26 – Visão Geral dos Objetivos do Trabalho

Na Figura 26 temos uma representação da disposição dos sistemas envolvidos neste trabalho. Temos um banco de dados contendo exames, fazendo parte deste o traçado e seu respectivo laudo. Para atingirmos o objetivo final deste trabalho, a classificação do traçado de um exame em *Normal* e *Anormal*, é necessário, primeiramente, a rotulação do exame em si. Só é possível realizar a classificação de um exame se o mesmo possuir um laudo, caso contrário, não há nenhuma informação sobre o traçado contido neste exame. Todas as características de um exame serão retiradas da conclusão de seu laudo, que é um parágrafo de texto. É necessário existir essa classificação para a criação do conjunto de treinamento da rede neural classificadora de traçados.

Iremos analisar agora um exemplo de conclusão de laudo de exame de ECG:

Avaliação eletrocardiográfica revela ritmo sinusal predominante, alternando para momentos de arritmia sinusal.

Ondas P de amplitude preservada e duração aumentada, sugestivo de sorbecarga atrial esquerda. Complexos QRS de amplitude e duração preservadas. Intervalos PR e QT preservados. Segmento ST e Onda T preservados.

Eixo elétrico médio desviado a esquerda do plano frontal, sugestivo de sobrecarga ventricular esquerda.

Sugere-se, a critério clínico, avaliação ecodopplercardiográfica para análise morfológica e funcional cardíaca.

Observa-se que podem existir diversas categorias dentro de um parágrafo. Portanto, não será possível criar uma única classificação para cada conclusão. Sendo assim, a solução será dividir o parágrafo em frases, onde cada frase terá uma única classificação.

Neste momento faz-se necessário a separação das frases em categorias. Com ajuda de um médico cardiologista, compilou-se a seguinte lista de categorias para classificação das frases de um laudo:

Lista 4.1 – Lista de Classificações de Frases

1. Ritmo sinusal
2. Arritmia sinusal
3. Bradicardia sinusal
4. Taquicardia sinusal
5. Marcapasso migratório
6. Parada sinusal
7. Pausa sinusal
8. Distúrbio de repolarização
9. Alta penetração ventricular
10. Alta resposta ventricular
11. Atraso de condução interatrial
12. Atrial standstill
13. Aumento atrial direito
14. Aumento atrial esquerdo
15. Aumento ventricular direito
16. Aumento ventricular esquerdo
17. Baixa penetração ventricular
18. Baixa resposta ventricular
19. Bigeminismo atrial
20. Bigeminismo ventricular
21. Bloqueio atrioventricular
22. Bloqueio atrioventricular de grau 1
23. Bloqueio atrioventricular de grau 2
24. Bloqueio atrioventricular de grau 3
25. Bloqueio atrioventricular completo
26. Bloqueio de ramo direito
27. Bloqueio de ramo direito completo
28. Bloqueio de ramo direito incompleto

29. Bloqueio de ramo esquerdo
30. Bloqueio de ramo esquerdo completo
31. Bloqueio de ramo esquerdo incompleto
32. Bloqueio fascicular anterior esquerdo
33. Complexo atrial prematuro
34. Complexos ventriculares prematuros monomórficos
35. Complexos ventriculares prematuros polimórficos
36. Complexo ventricular prematuro
37. Condução fibrilatória
38. Condução oculta
39. Couplet
40. Dissociação atrioventricular
41. Ectopia atrial
42. Ectopia atrial isolada
43. Ectopia ventricular
44. Ectopia ventricular isolada
45. Ectopias ventriculares monomórficas
46. Ectopias ventriculares pareadas
47. Ectopias ventriculares polimórficas
48. Extrassístole atrial
49. Extrassístole ventricular
50. Extrassístoles ventriculares em pares
51. Extrassístoles ventriculares monomórficas
52. Extrassístoles ventriculares pareadas
53. Fibrilação atrial
54. Fibrilação ventricular
55. Flutter ventricular
56. Intervalo PR curto
57. Pré-excitação ventricular
58. Ritmo ectópico atrial
59. Ritmo idioventricular
60. Ritmo idioventricular acelerado
61. Ritmo juncional
62. Silêncio atrial
63. Sobrecarga atrial
64. Sobrecarga ventricular
65. Sobrecarga atrial direita
66. Sobrecarga atrial esquerda
67. Sobrecarga ventricular direita
68. Sobrecarga ventricular esquerda
69. Sobrecarga biatrial
70. Sobrecarga biventricular
71. Sobrecarga atrioventricular esquerda
72. Sobrecarga atrioventricular direita
73. Taquicardia atrial focal
74. Taquicardia atrial multifocal
75. Taquicardia atrioventricular ortodrômica
76. Taquicardia juncional
77. Taquicardia sinusal inapropriada
78. Taquicardia supraventricular
79. Taquicardia ventricular
80. Taquicardia ventricular nao sustentada
81. Trigeminismo ventricular
82. Triplet
83. Via acessória
84. Flutter atrial
85. Disfunção sinusal
86. Aberrância de condução

A lista 4.1 está ordenada pela frequência de aparição de algumas classes, que foram inseridas por primeiro, sendo o restante em ordem alfabética.

Mais informações sobre as cardiopatias presentes na lista em [22].

Um dos maiores problemas em realizar a classificação é o tempo necessário para fazê-la e há obrigatoriedade de possuirmos essa categorização para proceder com o treinamento de uma rede neural capaz de automatizar o processo de classificação, visto que a classificação dos 31 mil exames manualmente é inviável.

Chegou-se na conclusão de que seria necessário utilizar algum tipo de ferramenta para ajudar no processo de classificação das frases. Essas frases classificadas formam um conjunto de dados para treinamento da rede neural e desenvolvimento de sistema capaz de automatizar a classificação dos exames.

Para acelerar o processo mais ainda, tal ferramente deveria possibilitar a classificação simultânea por usuários distintos. Além disso, essa ferramenta deveria poder ser distribuída de forma rápida e via internet.

Para atender a todos os requisitos citados acima, foram avaliadas opções de sistemas online prontos para uso. Tais sistemas objetivam auxiliar no *labeling* (classificação, etiquetagem) de dados para treinamento de redes neurais, porém não foi encontrado nenhum serviço que atendesse aos requisitos e ao mesmo tempo fosse gratuito.

Levando em consideração todos os motivos acima citados, optou-se pelo desenvolvimento de uma ferramenta web customizada para auxiliar na classificação das frases.

4.1 Sistema Web de Classificação de Frases

A ferramenta web, têm como principal objetivo a agilização do *labeling* das frases extraídas das conclusões dos laudos de exames ECG. Os dados serão importados para o banco de dados do sistema após a realização de um pré-processamento dos textos. O passo de pré-processamento será descrito na seção 4.2.3.

Depois de realizar o pré-processamento das conclusões de laudos, se faz necessário um meio de armazenar essas frases em um banco de dados. Esse banco de dados também armazenará os *labels* de cada frase. Para interfacear com o banco de dados, haverá necessidade de um *backend* que será encarregado de comunicar-se com o banco de dados e expor o seu conteúdo via uma API REST.

O objetivo principal de expor uma API é de fazer uma separação clara entre "cliente" e "servidor". O cliente, que será acessado via internet, não deve ocupar muitos recursos computacionais e não deverá carregar as mesmas informações contidas no servidor, acessando apenas o necessário para classificar uma frase por vez.

O cliente será utilizado pelo usuário que desejar participar da classificação de frases de determinado *dataset*. Este se comunica exclusivamente via a API REST disponibilizada pelo servidor.

4.1.1 Modelagem do Banco de Dados

O banco de dados do sistema de classificação de frases deve incluir apenas as informações importantes para o treinamento da rede e funcionamento do *backend*.

De modo a permitir que existam simultaneamente a classificação de dois conjuntos de dados utilizando o mesmo sistema, introduziremos a noção de *dataset*. Um dataset é um conjunto de dados para treinamento. Como o requisito do trabalho até este momento é a classificação de frases, o banco de dados deve dar suporte apenas a datasets do tipo texto. Outro dado importante que deve ser armazenado no banco de dados é a listagem de opções de classificação de cada elemento contido em um determinado dataset.

Portanto a definição de um dataset inclui os seguintes campos:

1. Identificador

   Utilizado pelo sistema para identificar os diferentes datasets.

2. Nome

   Nome para facilitar a identificação por humanos.

3. Lista de Opções

   Classes a serem mostradas aos usuários do cliente web para classificação de um dataset.

Para englobar a definição de dataset também iremos definir um tipo de dado chamado de "Projeto". Um projeto nada mais é do que um conjunto de datasets acrescidos de identificador e nome. Segue sua definição:

1. Identificador

   Utilizado para identificar os diferentes projetos.

2. Nome

   Utilizado para identificação por humanos.

Além da definição de dataset, é necessário também definirmos os campos para os elementos (entrada e *label* associado). São estes:

1. Identificador de elemento

   Este campo é necessário para diferenciar um elemento de outro, pois é possível que existam frases idênticas em um mesmo dataset para classificação, servindo, assim, este campo para desambiguar entre os mesmos.

2. Identificador do dataset a qual este elemento pertence

   Diferencia um elemento de um dataset de outro elemento pertencente a outro dataset. Da mesma forma que é possível existir textos idênticos em um mesmo dataset, também podem existir em diferentes datasets, porém, nem sempre os mesmos textos farão parte de um mesmo contexto, podendo ser classificados de forma diferente dependendo de qual dataset estes pertencem.

3. Texto a ser classificado

   A frase a ser classificada.

4. Classificação

   Campo que guarda a classificação escolhida pelo usuário através do cliente web, podendo ser nulo indicando que este elemento ainda não foi classificado.

As definições de campos de dados apresentados acima poderão ser modificadas para atender a eventuais requisitos não-funcionais do sistema, ou para satisfazer algum requerimento de implementação não especificado até o momento, adicionando ou removendo campos conforme o necessário. Serão discutidas alterações nas especificações quando necessário no capítulo de implementação.

4.1.2 Backend

O backend do sistema será responsável por comunicar-se com o banco de dados, gerenciando o acesso às informações contidas no mesmo, expondo-as através de uma interface. Essa separação permite que o backend possa ser implementado e instalado separadamente do cliente. É importante ressaltar que é necessário que exista uma infraestrutura (servidores) onde será instalado o backend.

Será descrita agora a interface disponibilizada por este serviço ao cliente. Parâmetros a ser passados aos métodos da interface estarão inseridos na URL na forma "{parâmetro}". Serão colocados GET e POST para especificar os métodos HTTP a serem utilizados pelos clientes para acesso ao respectivo método da interface. O tipo de retorno do método será colocado após seta à direita ($\rightarrow$).

Lista 4.2 – Lista de Métodos da API REST

1. GET /projects → Lista<Projeto>

   Lista os projetos contidos no banco de dados.

2. GET /projects/{id} → Projeto

   Retorna dados de um projeto.

3. POST /projects → Projeto

   Cria um novo projeto com as informações contidas no corpo da requisição HTTP e as retorna.

4. GET /projects/{projectId}/datasets → Lista<Dataset>

   Lista os datasets do projeto com identificador "projectId".

5. GET /datasets/{id} → Dataset

   Retorna informações de um dataset.

6. POST /datasets → Dataset

   Cria um novo dataset com as informações contidas no corpo da requisição HTTP e as retorna.

7. GET /datasets/{datasetId}/entries → Lista<Elemento>

   Retorna todos os elementos, classificados ou não, do dataset com identificador "datasetId".

8. GET /datasets/{datasetId}/{id} → Elemento

   Retorna dados do elemento de "id" do dataset com identificador "datasetId".

9. GET /nextlabel/{datasetId}/ → Elemento

   Retorna o próximo elemento a ser classificado do dataset com identificador "datasetId". O elemento a ser classificado é escolhido aleatoriamente entre os elementos que ainda não foram classificados.

10. GET /unlock/{datasetId}/{id} → Nulo

    Desbloqueia o elemento de identificador "id" do dataset com identificador "datasetId"para ser classificado. O mecanismo de bloqueio será explicado posteriormente no capítulo de implementação.

11. POST /datasets/{datasetId}/{id} → Elemento

    Envio de um elemento podendo este estar classificado ou não. Modifica o elemento persistido no banco de dados com as informações enviadas no corpo da requisição HTTP. Caso o cliente insira o "label" escolhido, a classificação será persistida no banco de dados do sistema.

12. POST /addentry → Elemento

    Adiciona um único elemento ao dataset. A escolha do dataset estará contida nas informações enviadas através do corpo da requisição HTTP.

13. POST /addentries/{datasetId} → Nulo

    Adiciona a Lista<String> contida no corpo da requisição HTTP enviada, criando e inserindo novos elementos ao dataset com identificador "datasetId".

Ter uma interface clara e bem definida é importante para agilizar a implementação do cliente web. Idealmente, cada método de uma interface deve ser auto-descritivo, seu nome e parâmetros auto-explicativos, assim reduzindo a necessidade de documentação formal direcionada ao desenvolvedor.

4.1.3 Cliente Web

O cliente web é uma ferramente especialmente desenvolvida para tornar o processo manual de classificação ágil e descomplicado, de forma que seja possível aumentar a taxa de classificações por minuto. É muito importante que a classificação seja feita o mais rápido possível para evitar que o usuário classificando acabe perdendo o foco no seu trabalho e classifique de forma errônea algum elemento, o que pode afetar a performance da rede após o treinamento. É importante também que a classificação seja feita rapidamente para permitir iterações mais rápidas tanto na arquitetura de rede quanto nos datasets.

Um dos motivos principais para a utilização de um cliente web é a dispensa de necessidade de instalação da ferramenta no computador do usuário. Este necessita apenas acessar um *site* utilizando a internet via um navegador e o identificador do dataset que o usuário deseja classificar.

No que diz respeito aos usuários iniciais do sistema, estes terão formação em cardiologia e não necessariamente familiaridade com a instalação de *softwares* customizados em seu computador, mais um motivo para a escolha da utilização de um cliente web. Com o entendimento de que nem todas as frases serão claramente classificáveis e que nem todos os usuários terão formação especializada, foi previsto a inclusão de um botão para ignorar o presente elemento a ser classificado, com o título de "Não Sei". Esse botão permite ao usuário não classificar a frase que está aparecendo no momento e fazendo com que o cliente web faça o pedido de uma nova frase ao backend.

4.2 Sistema de Classificação de Exames

Será descrito nesta seção os objetivos do sistema e também apresentada a definição da entrada e saída do mesmo.

4.2.1 Objetivos

Como definido anteriormente, não é possível classificar a conclusão de um laudo (parágrafo) como um todo em apenas uma categoria pois este pode conter diversas cardiopatias em um mesmo parágrafo, sendo necessário separar o parágrafo em frases, estas contendo apenas uma classificação.

Tendo as frases retiradas das conclusões dos laudos de exames ECG classificadas, o próximo passo será utilizá-las na confecção de um dataset de treinamento da rede neural contida no sistema.

O classificador de frases será de extrema importância para a separação dos traçados de exames em categorias distintas, e a seguir, possibilitar o treinamento de uma nova rede que utilizará os traçados de exames no seu treinamento.

Lembremo-nos de que não havia, no banco de dados da empresa, nenhum campo de dado estruturado indicando a existência ou não e quais tipos de cardiopatias estariam presentes em determinado exame, fazendo-se necessário, como primeiro passo, a classificação dos laudos dos mesmos, realizados por médicos cardiologistas.

4.2.2 Entrada e Saída

Detalhando melhor a entrada do sistema, temos que esta será um texto não formatado, com erros gramaticais, palavras que não possuem relação com cardiopatias como emails, telefones, entre outros, acentos, letras maiúsculas e minúsculas, caracteres especiais, etc.

O papel do sistema será de pré-processar o texto recebido, como descrito na seção 4.2.3 de modo que o parágrafo torne-se uma sequência de frases, as quais passarão uma por vez pela classificação via rede neural, da mesma forma que ocorre o treinamento. Uma vez classificadas todas as frases, define-se a classificação de um exame como a concatenação das classificações das frases retiradas da conclusão de seu laudo.

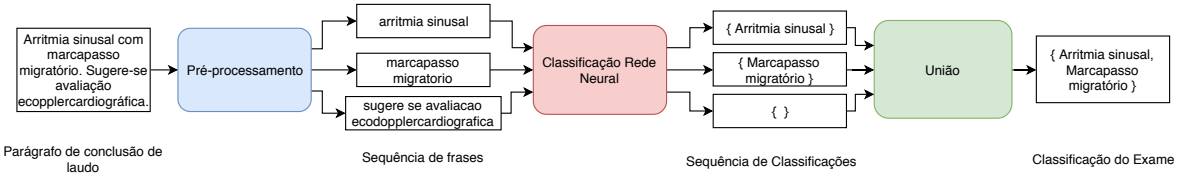


Figura 27 – Processo de classificação de exame

Na Figura 27 está mostrado todo o processo de classificação de um exame, desde o pré-processamento da conclusão do laudo, passando pela classificação de frases através da rede neural e por fim a saída esperada do sistema, a classificação do exame.

4.2.3 Pré-Processamento da Conclusão de Laudos

Será descrito agora o processo ao qual o texto da conclusão de um laudo é submetido para prepará-lo para classificação. Podemos dividir este pré-processamento em duas etapas: separação em frases e normalização.

O texto é dividido, criando duas novas sequências de caracteres, na presença das seguintes palavras:

- " e "
- " com "
- " e\n "
- "na presença de "
- "alternando para"
- ". "
- ".\n"
- "?\n"

Nos pontos acima, o caractere especial "\n"é o caractere que indica "nova linha".

Após a separação em novas frases, o texto para por um processo de normalização. Os passos de normalização são:

- Letras maiúsculas substituídas por minúsculas. Ex.: "Palavra" torna-se "palavra".
- Acentos são substituídos por caracteres similares sem acento. Ex.: "presença" torna-se "presenca".
- Numerais são substituídos pela palavra "numero". Ex.: "com 120 bpm" torna-se "com numero bpm".

É importante lembrarmos que mesmo após o pré-processamento do texto ainda existirão erros gramaticais, entre outros tipos de erros, por esse motivo é necessário que no conjunto de treinamento da rede neural existam exemplos de frases contendo erros de modo a aumentar a robustez do sistema a erros.

Aplicando-se os passos de pré-processamento de texto obtemos o exemplo apresentado anteriormente (ver página 55) pós-processamento (sequência de frases):

- "avaliacao eletrocardiografica revela ritmo sinusal predominante,"

- "momentos de arritmia sinusal."
- "ondas p de amplitude preservada"
- "duracao aumentada, sugestivo de sobrecarga atrial esquerda."
- "complexos qrs de amplitude e duracao preservadas."
- "intervalos pr e qt preservados."
- "segmento st e onda t preservados."
- "eixo eletrico medio desviado a esquerda do plano frontal, sugestivo de sobrecarga ventricular esquerda."
- "sugere-se, a criterio clinico, avaliacao ecodopplercardiografica para analise morfologica e funcional cardiaca."

Seguindo o exemplo mostrado anteriormente, está disposto na Tabela 1 as frases com suas classificações esperadas como saída do sistema.

Tabela 1 – Exemplos de frases de laudos e suas classificações

Frase	Classificação
avaliacao eletrocardiografica revela ritmo sinusal predominante,	Ritmo sinusal
momentos de arritmia sinusal.	Arritmia sinusal
ondas p de amplitude preservada	Inválido
duracao aumentada, sugestivo de sobrecarga atrial esquerda.	Sobrecarga atrial esquerda
complexos qrs de amplitude e duracao preservadas.	Inválido
intervalos pr e qt preservados.	Inválido
segmento st e onda t preservados.	Inválido
eixo eletrico medio desviado a esquerda do plano frontal, sugestivo de sobrecarga ventricular esquerda.	Sobrecarga ventricular esquerda
sugere-se, a criterio clinico, avaliacao ecodopplercardiografica para analise morfologica e funcional cardiaca.	Inválido

Este passo deverá ser executado em todos os 31 mil exames disponíveis. Somente após a realização da classificação de todos esses exames é que poderá iniciar-se a compilação dos datasets de treinamento da rede neural que analisará os traçados.

4.3 Rede Neural para Classificação de Traçados

Depois de classificados os exames, o próximo passo é utilizar os dados dos traçados destes, adquiridos através de eletrocardiógrafo e armazenados no sistema InCloud, serviço de sincronização de exames e laudos da InPulse.

4.3.1 Objetivos

Espera-se obter taxa de acerto de 60% ou mais, se possível, na classificação entre exames classificados como *Normal* e *Anormal*. Supondo que 50% de taxa de acerto seria obtido caso o sistema apenas sorteasse a classificação, no caso em que a acurácia após treinamento seja maior do que 60% poderíamos dizer que houve aprendizado durante o treinamento.

4.3.2 Entrada e Saída

A linha de eletrocardiógrafos InCardio, utilizada para extrair o sinal dos pacientes, utiliza taxa de amostragem de 500 Hz e captura, simultaneamente 12 derivações, D1, D2 e D3, aVR, aVL e aVF e V1 a V6), porém, as derivações D3, aVR, aVL e aVF e V1 a V6 não são obrigatórias para a solicitação de um laudo, somente D1 e D2, por esse motivo não serão utilizados os traçados das derivações opcionais.

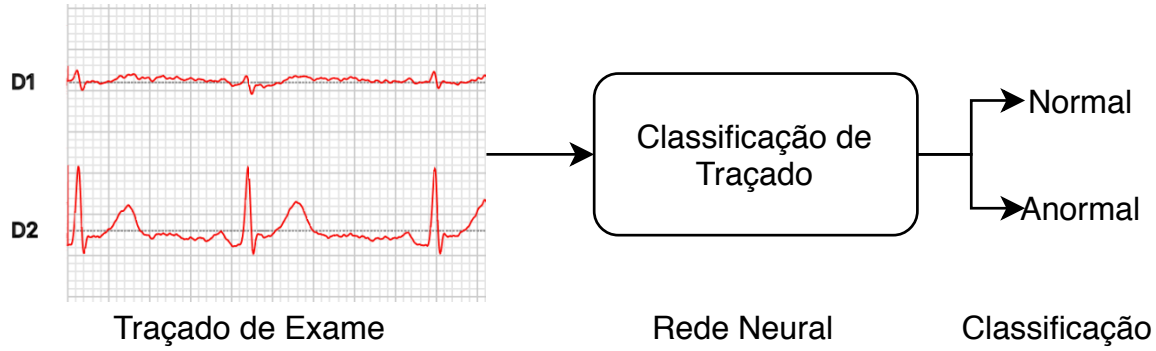


Figura 28 – Processo de Classificação de Traçado

Na Figura 28 temos a representação da entrada e saída do sistema de classificação de traçado. Como entrada temos os sinais das derivações D1 e D2 e como saída a classificação *Normal* ou *Anormal*.

O sinal obtido do eletrocardiógrafo e armazenado no banco de dados não contém nenhum tipo de filtro aplicado sobre o mesmo, de forma que, antes de poder ser utilizado para análise, este deverá passar por um pré-processamento, que inclui, além da aplicação de filtros digitais, o truncamento ou repetição dos sinais, visto que estes não possuem dimensões bem definidas. Segundo o médico veterinário especializado em cardiologia, Luis Felipe dos Santos, são necessários apenas 2 minutos de captura para determinarmos a grande maioria de cardiopatias que aparecem apenas no traçado de ECG. Outras cardiopatias necessitam, ou de uma captura de 24 (ou mais) horas, chamada Holter, ou de um exame chamado de ecodopplercardiograma para serem detectadas, pois um exame de ECG curto não é capaz de revelar tais anomalias. Pelo motivo citado acima, será definido que os vetores utilizados para treinamento possuirão dimensão de $500[\text{Hz}] \times 120[\text{s}] = 60.000$ amostras.

Sinais que possuírem dimensão maior que 60.000 amostras serão truncados, ou seja, serão considerados apenas as primeiras 60.000 amostras. Sinais que possuírem menos que 60.000 amostras serão repetidos até alcançarem 60.000.

É importante para redes neurais que entrada e saída tenham dimensões bem definidas, no caso das redes neurais feed-forward e convolucionais.

O formato de armazenamento dos traçados é um arquivo cujo conteúdo é um vetor de bytes onde a cada quatro bytes estão dispostos, em conjuntos, os bits de um número de ponto flutuante seguindo a norma IEEE-754, codificados em big-endian, ou seja, com os bytes mais significativos primeiro.

5 Implementação

A implementação dos sistemas será abordada em três partes distintas, uma para o sistema web de classificação de frases, outra para o sistema e redes neurais para classificação de exames e a última para sistema e rede neural de classificação de traçado.

5.1 Sistema Web para Classificação de Frases

Nesta seção serão abordadas as escolhas de linguagem, frameworks, arquiteturas e suas respectivas justificativas de escolha.

5.1.1 Linguagens, Bibliotecas e Frameworks Utilizados

Por ser um trabalho relativamente extenso, foram escolhidas as linguagens mais familiares para desenvolvimento. Estas são: Java, Kotlin e JavaScript. Java e Kotlin foram utilizados no backend e JavaScript para desenvolver o cliente web, chamado de frontend. O backend é um software executado em um servidor e acessado via internet. Já o frontend, é um software executado no computador cliente, via um *browser*. O segundo comunica-se com o backend através de uma API REST. A API REST nada mais é do que uma interface com funções, parâmetros e retorno bem definidos. Os métodos da API estão descritos na seção 4.1.2.

Mais informações sobre a linguagem Java podem ser obtidas em [23]. Mais informações sobre Kotlin em [24] e sobre JavaScript em [25].

5.1.2 Banco de Dados

Para armazenamento do banco de dados foi escolhido o Sistema Gerenciador de Banco de Dados (SGBD) PostgreSQL, pelos seguintes motivos: o banco de dados é do tipo relacional, é um projeto open-source, sem custo e possui suporte para todos os tipos de dados que serão usados no projeto, extensa documentação online e por ser familiar ao desenvolvedor. Mais informações sobre o PostgreSQL em [26].

Houve pouca necessidade de trabalhar diretamente na criação de tabelas e campos via SQL pois, como será mostrado em seguida, existem bibliotecas disponíveis que encarregam-se de transformar objetos (código) em tabelas e relações entendidas pelo SGBD.

5.1.3 Backend

Arquiteturalmente, a implementação do backend segue o padrão Model-View-Controller (MVC), sendo o principal objetivo de seguir padrões de software obter uma clara separação entre atribuições de função entre cada parte de um código-fonte, além de auxiliar na manutenção futura, caso necessário, pois evita que os componentes tornem-se interconectados (quando a implementação de dois componentes diferentes são afetadas por uma mudança no código-fonte).

No padrão MVC, o Model refere-se às definições de dados (objetos Java ou Kotlin), no caso deste trabalho as definições de Dataset, Entry e Project. O Controller representa o componente que media a interação entre a View e o Model. Por exemplo: fica no Controller a parte do código que encarrega-se de carregar informações do banco de dados e apresentá-las à view. Neste trabalho a View (componente que interage diretamente com o usuário) é representada pela API REST, mas poderia ser uma página web, por exemplo.

Nos dias de hoje, existem frameworks web muito maduros, testados de funcionamento comprovado por inúmeros usuários, por esse motivo não há a necessidade de implementação completa. É possível, utilizando um destes frameworks, acelerar o desenvolvimento de uma backend e API REST.

O framework web escolhido para desenvolver o sistema foi o Spring Boot. Mais informações sobre o Spring Boot em [27]. Este encarrega-se de receber e processar requisições HTTP e HTTPS, tornando o trabalho do desenvolvedor apenas a implementação do que é conhecido por "lógica de negócio".

Lógica de negócio no campo da programação é a definição do código responsável por incorporar ao software as regras que determinam como são criados, modificados e acessados os dados de um sistema. Também coordena a relação entre diferentes segmentos de um software.

O framework Spring Boot dá suporte a um ecossistema de bibliotecas que implementam diversas funcionalidades. Uma destas, a interação com um SGBD relacional. Existe um padrão chamado de JPA (Java Persistence API) que consiste em uma definição de uma série de interfaces as quais podem ser implementadas por bibliotecas que desejam interagir, segundo a especificação da interface, com softwares de banco de dados. O Spring Boot provê uma implementação do padrão JPA chamado de Spring Data. Ver [28] para mais informações.

A vantagem de basear a própria implementação em uma interface padronizada é a de que, na teoria (nem sempre a transição é tão suave quanto proposta inicialmente), é possível, sem afetar o código que a utiliza, modificar a implementação que está sendo utilizada. Isso implica que seria, sem muito esforço, possível trocar o SGBD sem afetar o código do serviço backend.

A especificação JPA permite, com anotações simples, ao desenvolvedor fazer um mapeamento lógico entre um objeto em código Java e Kotlin a uma tabela, campos e relações de um banco de dados relacional.

Fazendo uso desta especificação foram definidos os modelos para Dataset e Entry. Dataset representa o conjunto de dados para treinamento e Entry representa um elemento de um dataset. Project representa um projeto, ou seja, um conjunto de datasets.

A implementação dos modelos em Kotlin encontra-se no anexo A.

Uma vez definidos os modelos, parte-se para a implementação dos Controllers (gerenciamento e processamento da lógica de negócio). O módulo de Controllers é o que implementa efetivamente o funcionamento do backend, recebendo requisições, através do framework Spring Boot, e processando-as antes de responder ao cliente.

São esses componentes os encarregados de receber do cliente os dados para criação de um novo dataset, criação dos elementos de um dataset, da classificação de um elemento de um dataset e assim por diante.

Por último, temos os componentes responsáveis pela View. Neste caso, a View representa a API REST, ou seja, para implementar a view, será necessário o uso de componentes do framework chamados de Roteadores. Os roteadores indicam ao framework quais requisições à URLs (ex.: */datasets/{id}*) serão processadas por quais controladores (Controllers). Dessa maneira, é feita a conexão entre a View e Controller.

Após o desenvolvimento desta parte do sistema, é necessário preparar um ambiente com o objetivo de colocá-lo em execução e disponibilizar o acesso a este via internet. Para isso, será utilizado a infraestrutura já existente da empresa e estabelecida na Digital Ocean, um provedor de servidores virtuais. Mais informações sobre a Digital Ocean em [29].

Para facilitar a implantação do sistema na infrastrutura da empresa, foi utilizada uma plataforma chamada de Docker (ver [30]). Esta plataforma permite ao desenvolvedor criar imagens executáveis com todas as bibliotecas, código-fonte, ferramentas necessárias para o funcionamento de um software. Essas imagens podem ser colocadas em execução em um ambiente diferente do que foi utilizado em sua confecção.

5.1.4 Frontend

Para o frontend a escolha de linguagem foi JavaScript. Um dos frameworks para desenvolvimento de aplicações web mais reconhecido e utilizado é o framework React. A grande motivação para a escolha do React foi a experiência de desenvolvimento, número de bibliotecas disponíveis e extensa documentação para desenvolvedores.

Arquiteturalmente, foi escolhido um padrão de software conhecido por Redux. Mais informações sobre Redux em [31].

O cliente web implementa a interface com o usuário. Esta carrega, através da API, um elemento de um dataset por vez, apresentando-o ao usuário para que este realize a classificação. Uma vez classificado o elemento, o usuário, clicando em um botão na interface gráfica, envia-o de volta ao backend, que por sua vez, é responsável pela persistência em banco de dados da classificação escolhida pelo usuário.

Como falado anteriormente, não há necessidade do usuário instalar a ferramenta em seu computador, sendo apenas necessário acessar uma URL através de um navegador.

Apesar de não necessitar de instalação para utilizar a ferramenta, ainda assim é necessário que o usuário realize o download da mesma. Isso faz com que exista a necessidade de um servidor que disponibilize tal download via internet. Esse passo foi implementado utilizando o mesmo princípio do backend, a plataforma Docker. Pode-se encontrar mais informações sobre Docker em [30].

Fazendo uso de uma imagem pré-definida do web server NGINX, este foi configurado para servir os arquivos necessários para o funcionamento do frontend web. Mais informações sobre NGINX em [32].

Por fim, para fazer a importação das frases retiradas da conclusão de laudos, implementou-se um script utilizando a linguagem Python. Esse script carrega os laudos a partir de um arquivo de texto, o qual foi exportado do banco de dados da empresa, pré-processa o seu conteúdo, separando as conclusões em frases, etc., e após processamento, envia-as, através de requisições HTTP, para o backend, onde este prepara-as para a classificação, em seguida. Mais informações sobre a linguagem Python em [33].

5.2 Redes Neurais para Classificação de Frases

Nesta seção serão apresentadas as escolhas técnicas para a implementação do sistema de classificação de frases.

5.2.1 Linguagens, Bibliotecas e Frameworks Utilizados

Para a grande maioria do sistema, utilizou-se a linguagem Python para desenvolvimento de ferramentas. Por ser uma linguagem de alto nível, é possível desenvolver ferramentas, com diversas funcionalidades complexas, em relativamente pouco tempo. Outro motivo para a escolha da linguagem foram as escolhas de bibliotecas e frameworks para desenvolvimento das redes neurais.

Keras, uma biblioteca para implementação de redes neurais possui uma API simples, de alto nível. Dá suporte aos mais diferentes tipos de operações, arquiteturas de rede, camadas, funções de ativação, etc. O Keras, por sua vez, é apenas uma "casca". Por baixo dos panos, a implementação das funções para treinamento é feita através da biblioteca

TensorFlow. Publicada pela Google, o TensorFlow é um dos backends suportados pelo Keras. Ver seção 3.5.3.

A biblioteca TensorFlow permite a utilização dos recursos do computador, como a placa de vídeo, para acelerar o processo de treinamento e inferência das redes neurais, sendo possível um aumento de ordens de magnitude na velocidade de processamento, comparado à processamento utilizando o processador, como mostrado em [34].

Para observar o progresso no treinamento de diferentes modelos foi utilizada a ferramenta TensorBoard. Esta é publicada pela Google para ser utilizada juntamente com a biblioteca TensorFlow. Com esta ferramenta é possível visualizar gráficos de performance versus tempo e efetuar a comparação entre diferentes parâmetros ou arquiteturas de rede. Também é possível visualizar um gráfico em 3D que representa, após análise de PCA ou T-SNE a posição tridimensional dos vetores respectivos às frases. Essa visualização permite-nos ter uma ideia melhor quanto à performance dos modelos verificando a localidade dos vetores resultantes. Ver seção 3.5.4.

Outra biblioteca utilizada nas primeiras iterações deste sistema foi a biblioteca Gensim (ver [35]). Esta é uma biblioteca que implementa uma família de redes neurais chamadas de Word2Vec e Doc2Vec. Ver itens 3.4.7.1 e 3.4.7.2. O Word2Vec e Doc2vec são responsáveis por gerar os embeddings das frases.

A biblioteca Python chamada de NLTK (Natural Language Toolkit) foi utilizada para facilitar a tokenização das frases. Mais informações na seção 3.5.1.

Pandas, biblioteca Python que dedica-se à implementação da leitura e manipulação de dados em forma de tabela, foi utilizada para ler e gravar os dados utilizados no treinamento da rede. Mais informações sobre a biblioteca Pandas em [36].

Para o treinamento, foram desenvolvidos scripts na linguagem Python. Tais scripts recebem como parâmetros os dados para treinamento, por quantas *epochs* deve-se treinar, tamanhos de vetor para representação via embeddings, entre outros.

5.2.2 Utilizando Doc2Vec

Nas primeiras versões do sistema, utilizou-se a ferramenta Doc2Vec da biblioteca Gensim para treinar os embeddings das frases. Vemos na Figura 29 o processo de obtenção do modelo Doc2Vec.

O treinamento do modelo Doc2Vec ocorre da seguinte maneira:

1. Geração de um corpus para treinamento

   O corpus define-se por todos os textos para treinamento, sem classificação.

2. Tokenização

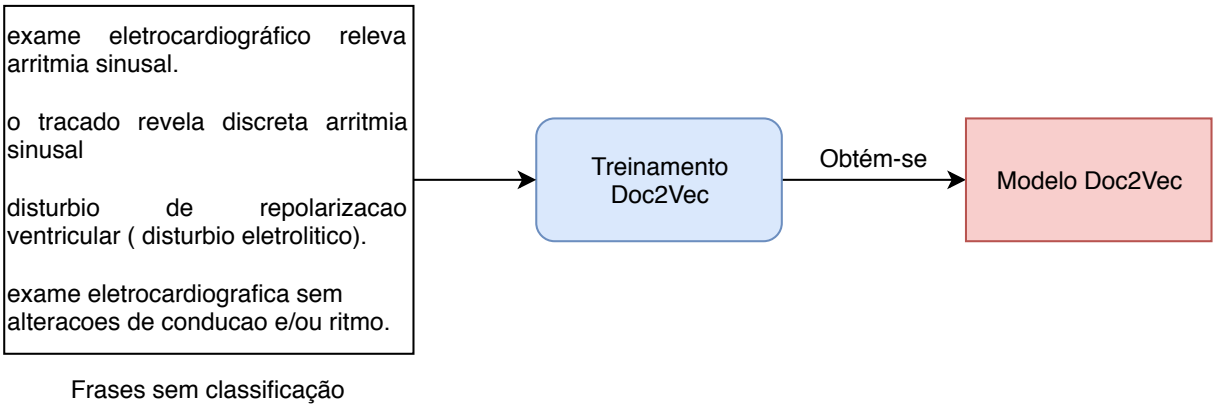


Figura 29 – Visão Geral do Treinamento do modelo Doc2Vec

Divisão de uma frase em tokens. Tokens são palavras e caracteres especiais. Uma frase torna-se uma sequência de tokens.

3. Separação em documentos

   Para treinar a rede Doc2Vec, usa-se uma *tag* e uma sequência de tokens para formar um "documento". A tag utilizada foi um número inteiro de 1 ao número total de frases do conjunto de dados.

4. Treinamento efetivo do Doc2Vec

   Após o treinamento dos embeddings, é possível inserir uma frase e obter um vetor do $\mathbb{R}^n$ que mapeia-a em um espaço vetorial.

5. Treinamento da classificação de frases

   Utilizando o modelo Doc2Vec obtido, as frases classificadas anteriormente passam por inferência, assim obtendo embeddings. Esses embeddings são utilizados como vetor de entrada para a rede neural em treinamento.

Além das métricas executadas sobre o conjunto de dados de teste, foram feitos testes empíricos do resultado do modelo. Utilizando um script em Python, foi desenvolvido uma espécie de console, onde é possível inserir parágrafos para serem processados pela rede e obter a classificação após processamento.

Vemos na Figura 30 o fluxo de dados no processo de treinamento da rede neural de classificação de frases utilizando os embeddings obtidos através do treinamento prévio do modelo Doc2Vec sobre o conjunto de frases. Temos como entrada para a rede o vetor do $\mathbb{R}^n$ e como saída esperada a classificação da frase ("Arritmia sinusal", "Marcapasso migratório", etc).

Na Figura 31 temos a arquitetura proposta para a rede de classificação de frases. Como entrada temos o embedding resultante do processamento de uma frase através do

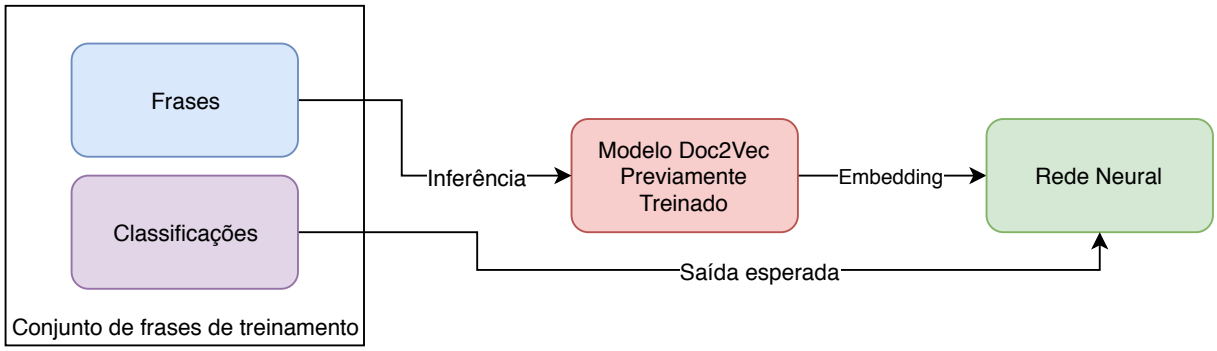


Figura 30 – Processo de Treinamento da Rede Neural com Embeddings obtidos via modelo Doc2Vec

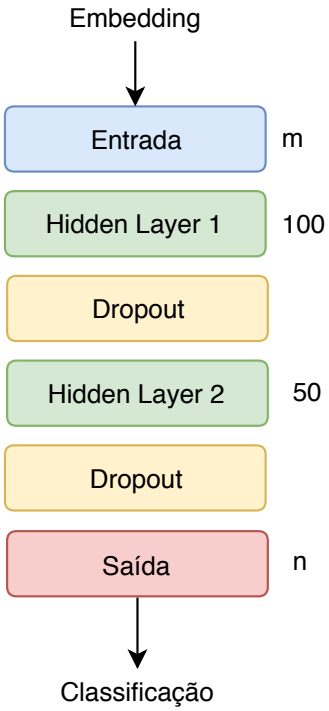


Figura 31 – Arquitetura da Rede Utilizando Word2Vec

modelo Doc2Vec treinado previamente. Para as hidden layers, foram utilizadas camadas feed-forward simples com função de ativação ReLU (ver página 40). Na sequência de ambas, foram colocadas camadas Dropout (ver item 3.4.5.1) com parâmetro de anulação de 20%. Para a camada de saída foi escolhida a função de ativação Softmax (ver página 40). Na figura está escrito à direita de cada camada a sua dimensão de saída, sendo m a dimensão escolhida para os embeddings e n o número de classes. Como função de perda para treinamento foi escolhida a função "categorical crossentropy"(ver item 3.4.4.3).

O número de classes foi determinado após a classificação manual. Classes não escolhidas nenhuma vez foram retiradas do conjunto (ver lista 4.1). A dimensão dos vetores de embedding foi variada utilizando os valores 50, 100 e 200.

Após análises do resultado (ver seção 6.2.3), este não foi considerado suficiente,

partiu-se, então, para um novo tipo de rede utilizando embeddings treináveis juntamente com a classificação das frases, sem utilização do Doc2Vec.

5.2.3 Utilizando Embeddings Treináveis

Nesta seção será tratada a nova arquitetura desenvolvida para tentar melhorar o poder de generalização da rede. Desta vez usou-se a camada Embedding da biblioteca Keras. Outra mudança feita foi que, agora, a rede recebe uma sequência de tokens, assim como na arquitetura anterior, porém, cada token é convertido em um embedding independente, ou seja, antes uma frase inteira era transformada em vetor e agora cada palavra é transformada em um vetor. Os vetores resultantes são treinados juntamente com a classificação final, ou seja, não temos mais as relações entre palavras obtidas como no caso do Word2Vec, porém temos que as distâncias entre os vetores resultantes de mapeamento de palavras tentam maximizar as suas diferenças de acordo com a classificação final de uma frase, dessa maneira, quando antes tínhamos palavras semanticamente próximas na representação via Doc2Vec, ou seja, vetores resultantes próximos, agora isto não se repete. Por exemplo: na representação Doc2Vec as palavras "esquerda" e "direita" localizam-se próximas no espaço enquanto na nova arquitetura, durante o treinamento essas duas palavras se localizarão mais distantes caso seja favorável à melhoria da acurácia da rede.

O dataset para treinamento contou originalmente com 1460 frases classificadas que, modificadas, acrescentando-se erros gramaticais, resultaram em 317.187 frases. Constatou-se, em análise das frases que estas possuíam muitos erros de digitação. Esses erros prejudicavam de forma considerável a acurácia de predição da rede, por esse motivo, utilizou-se da inserção de erros propositais nas frases de treinamento. O novo conjunto de treinamento contendo erros foi gerado da seguinte maneira:

1. Escolha de palavras-chave.

   Para evitar a explosão do número de elementos do conjunto de treinamento, escolheu-se gerar erros apenas para algumas palavras.

2. Percorrer o conjunto de frases buscando pelas palavras-chave.

3. Adicionar uma nova frase para cada permutação possível de erro sobre as palavras-chave encontradas nesta frase.

As palavras escolhidas como palavras-chave foram:

Lista 5.1 – Lista de Palavras-chave para Geração de Erros Gramaticais

1. arritmia
2. sinusal
3. ritmo
4. taquicardia
5. marcapasso
6. migratorio

7. bloqueio
8. atrioventricular
9. ventriculo
10. esquerdo
11. direito
12. atrio
13. bradicardia
14. disturbio
15. repolarizacao
16. aumento
17. complexo
18. ectopia
19. extrassistole
20. extrassistoles
21. sobrecarga

A escolha das palavras-chave foi feita realizando uma análise empírica das palavras que mais apareciam com erros no conjunto de treinamento original.

Os erros gramaticais foram gerados de duas maneiras distintas: remoção de letras e substituição de letras. Todas as combinações possíveis de remoção e substituição são geradas. Para exemplificar o processo de remoção com a palavra "arritmia", temos alguns possíveis erros via remoção de letras: "arrtmia", "aritimia", "arrimia" e assim por diante. Para exemplificar a substituição de letras utilizando a palavra "arritmia", temos: "artitimia", "arritnia", "arrutmia", etc.

Outra diferença é que agora foi adicionada uma camada de Convolução 1D da biblioteca Keras. Dessa forma os vetores obtidos através da camada de Embeddings são processados por filtros que utilizam uma janela para calcular a saída (ver seção 3.4.3.2). Após a camada de convolução é adicionada uma camada de MaxPooling, para reduzir a dimensão dos vetores de saída.

Na Figura 32 temos a representação da arquitetura da nova rede. Como entrada da rede, esta recebe um vetor com 40 tokens. Para cada token recebido do vetor, a camada Embedding mapeia-os à um vetor de dimensão 50, obtemos assim dimensões de saída da camada Embedding 40x50. Essa matriz é repassada à camada de convolução Conv1D (ver seção 3.4.3.2). Esta foi configurada para utilizar 128 filtros de dimensões 3x50, com utilização de padding. Para esses filtros a função de ativação escolhida foi ReLU. Após a camada Conv1D, foi incluída uma camada para realizar a operação de MaxPooling configurada com janela de tamanho 2 e stride 2, assim reduzindo para a metade a dimensão de saída desta (20x128). Como a camada feed-forward não aceita entradas matriciais, é necessário inserir a camada Flatten. Esta realiza a operação de concatenação dos vetores, transformando uma matriz de dimensões n por m em um vetor com dimensão nm. Após a camada Flatten, temos a camada feed-forward com 128 neurônios e função de ativação ReLU. É adicionado a camada Dropout após a camada Dense com parâmetro de dropout com valor de 50%. Por fim temos a camada de saída onde foram colocados n neurônios e escolhida a função de ativação Softmax. Como saída da rede temos a classificação das frases.

Para treinamento foi utilizado novamente a função de perda *categorical crossentropy*

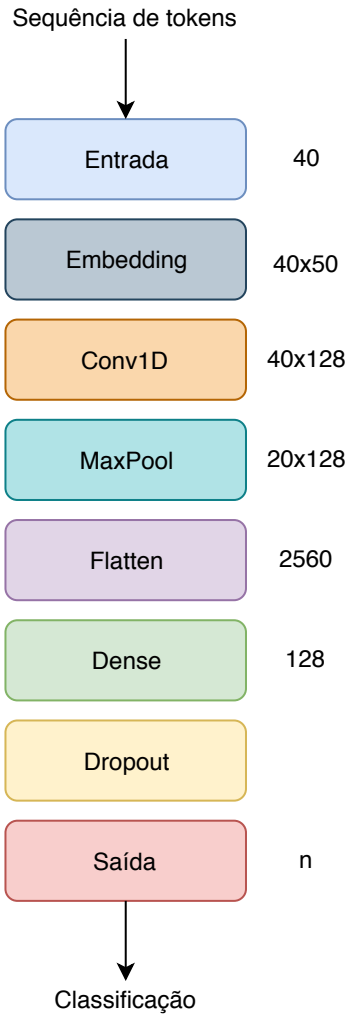


Figura 32 – Arquitetura da Rede Utilizando Embeddings Treináveis

e algoritmo Adam de otimização. O treinamento foi realizado utilizando uma GPU nVidia GTX 970.

Devemos lembrar que os valores escolhidos para as dimensões de saída das camadas foram escolhidos arbitrariamente. Não existe metodologia definida para a escolha das dimensões. Foi utilizado um processo de tentativa e erro, até encontrar uma arquitetura que respondeu positivamente ao treinamento.

Todo o pré-processamento do texto de entrada transcorre da mesma maneira que a rede apresentada anteriormente com a adição de uma operação de mapeamento de palavras à tokens. Para realizar essa operação processa-se os textos do conjunto de treinamento coletando todas as palavras que aparecem e adicionando-as em um dicionário. Utilizando o dicionário é feita a conversão de uma palavra da frase para token. Os tokens são números inteiros de 1 a k, sendo k o número total de palavras contidas no dicionário. O token 0 é reservado para espaços e palavras fora do dicionário.

Para a entrada temos que essa é uma sequência de tokens de tamanho 40, porém

existem frases com menos do que 40 palavras. No caso de frases desse tipo, são prefixados tokens 0 nas primeiras $40 - len$ posições do vetor de tokens, sendo len o número de palavras da frase em questão.

5.3 Redes Neurais para Classificação de Traçados

Assim como a rede neural de classificação de frases, foram utilizadas camadas de convolução 1D em conjunto com camadas de *maxpooling* nas arquiteturas testadas, porém com a remoção da camada de embedding na entrada da rede.

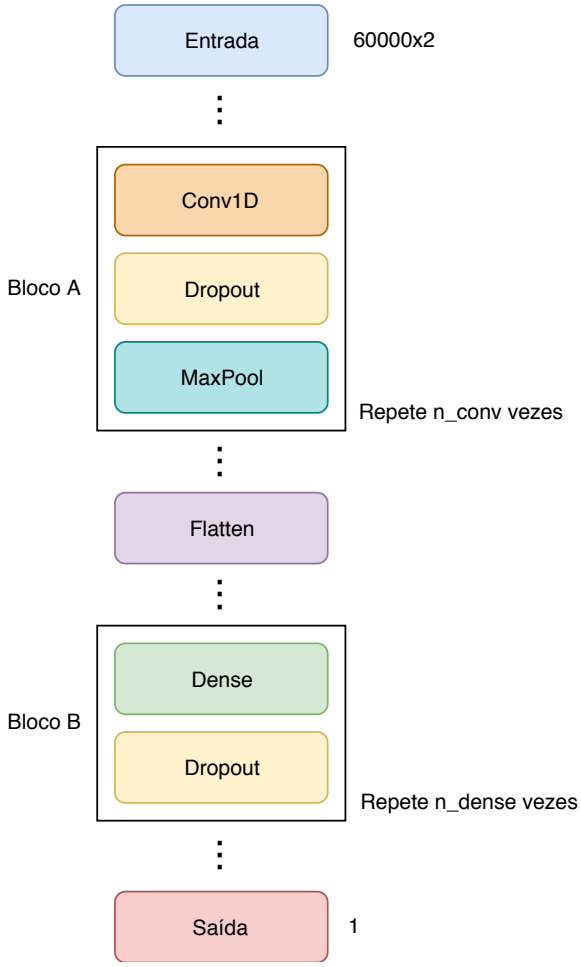


Figura 33 – Arquitetura da Rede Neural de Classificação de Traçados

Na Figura 33 está mostrada a arquitetura da rede neural. Temos que a entrada da rede é de dimensão (60000, 2), onde a primeira dimensão são as amostras do sinal de ECG e segunda dimensão são os canais D1 e D2. Analogamente às redes convolucionais para classificação de imagem, a inserção de dois canais de entrada provê mais informação sobre um traçado, assim como no caso de imagens onde os canais utilizados são as cores em RGB de uma imagem. Foram omitidas as dimensões das saídas de algumas camadas pois esse valor varia de acordo com os parâmetros de n_{conv}, n_{dense}, tamanho da janela e stride do maxpooling, etc. Os valores do parâmetro de dropout também foram variados.

Foram testadas uma grande quantidade de combinações envolvendo o número de camadas de convolução + maxpooling (Bloco A da Figura 33), número da camadas densas (Bloco B da Figura 33), número de neurônios nas camadas densas, janela dos filtros de convolução, entre outros parâmetros. Será explorado melhor no capítulo de resultados.

Quanto ao dataset de treinamento, foi utilizado um script Python que processa um arquivo de texto contendo as informações dos laudos dos exames classificados em formato CSV, filtra-os de acordo com alguns critérios, carrega os dados de traçados a partir do sistema de arquivos, filtra-os e exporta-os para um local indicado. Os dados de traçados não possuem nenhum tipo de processamento, são apenas a gravação dos dados crus obtidos do eletrocardiógrafo, por esse motivo estes dados precisam passar por filtros digitais para prepará-los ao treinamento. Os filtros aplicados são filtros de resposta infinita ao impulso (IIR), responsáveis por atenuar a flutuação da linha de base, ruídos e interferências no sinal. A implementação de tais filtros faz parte de biblioteca desenvolvida na empresa utilizando a linguagem C. Fez parte do desenvolvimento deste projeto o desenvolvimento de uma interface para permitir a utilização da biblioteca a partir da linguagem Python fazendo uso de uma ferramenta conhecida por SWIG. Mais informações sobre SWIG em [37].

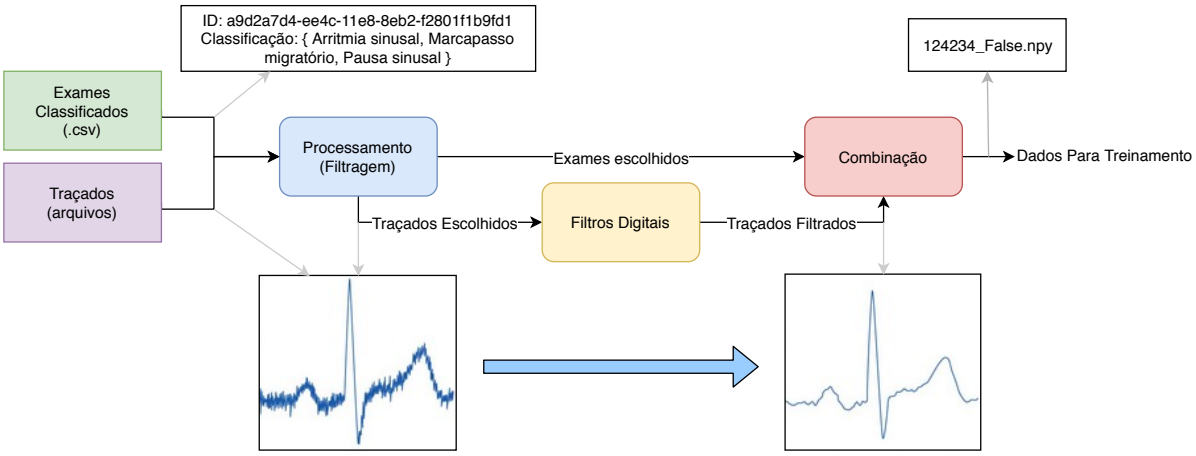


Figura 34 – Processo de Geração do Conjunto de Dados de Treinamento

Na Figura 34 temos a representação do processo de geração do conjunto de dados. A figura mostra os passos realizados pelo script Python de geração. Primeiramente é carregado um arquivo CSV contendo os exames já classificados pelo sistema de classificação de exame (ver seção 4.2), após a filtragem seguindo alguns critérios, são carregados os dados dos traçados correspondentes aos exames selecionados, que são filtrados em seguida. Essas informações (exames selecionados e seus respectivos traçados filtrados) são combinadas da seguinte maneira: grava-se um arquivo no sistema de arquivos do computador cujo nome é a concatenação do identificador daquele traçado (um exame pode ter mais de um traçado, todos são utilizados) com o resultado da seleção (True para exames tidos como *Anormais* e False para *Normais*). O conteúdo do arquivo são as derivações D1 e D2 do traçado em formato utilizado pela biblioteca Numpy (ver [38]), para facilitar o

carregamento posteriormente, na fase de treinamento.

Para o treinamento foi desenvolvido mais um script Python que carrega os exame pré-filtrados e classificados de acordo com o filtro especificado no script de preparação de dados para treinamento e apresenta-os à rede. Serão explorados os datasets utilizados no treinamento e testes com mais detalhe no capítulo de resultados.

Quanto à classificação dos elementos pertencentes aos datasets testados utilizou-se uma váriável Booleana para indicar a classificação escolhida. Essa variável, foi embutida no nome do arquivo utilizado para persistir os dados de exames pré-processados, juntamente com o identificador da aquisição de exame.

Definiu-se que a saída esperada da rede para uma classificação "False" seria 0 e 1 para "True". Fez-se uso da função de perda "binary crossentropy" como função objetivo a ser minimizada pelo processo de treinamento da rede. O algoritmo de otimização Adam foi escolhido. Além disso foram empregados 4 valores de limiares diferentes para calcular as métricas de acurácia, com o objetivo de melhor verificar o comportamento da rede. Os limiares escolhidos foram 0.5, 0.6, 0.7 e 0.8. Os valores de limiar afetam o resultado da seguinte maneira: após o processamento de um elemento através da rede temos como saída um número de ponto flutuante que varia de 0 a 1. Caso tal valor ultrapasse o limiar, considera-se que o elemento de entrada foi classificado como "True", caso contrário como "False".

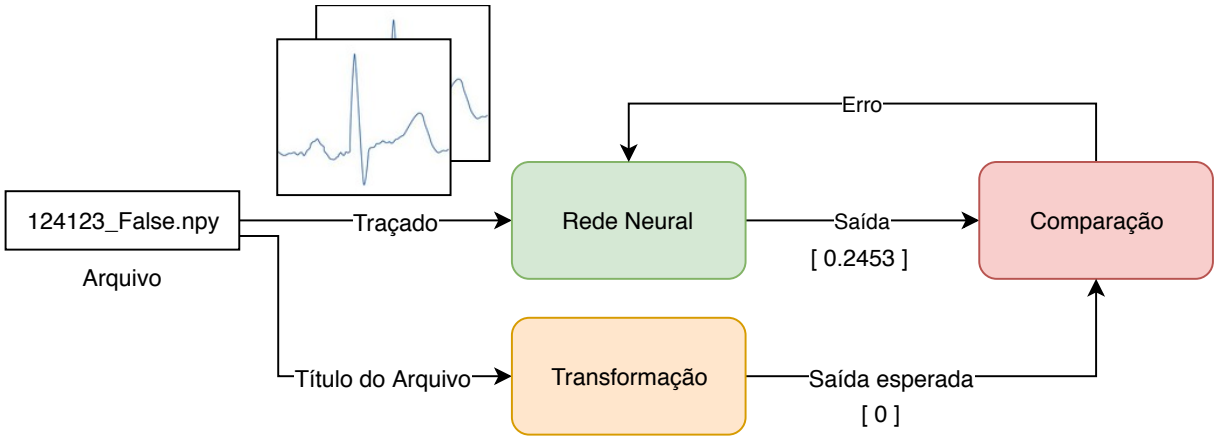


Figura 35 – Processo de Treinamento Da Rede Neural de Traçados

Na Figura 35 é mostrado o processo de treinamento da rede. São carregados os exemplos (arquivos .npy) a partir do sistema de arquivos. De acordo com o título do arquivo de exemplo é obtida a saída esperada. Após comparação da saída da rede e saída esperada é feito o ajuste dos pesos da rede.

Para observar a performance dos diferentes modelos obtidos também foi utilizada a ferramenta TensorBoard.

6 Resultados

Neste capítulo serão analisados mais profundamente os resultados do desenvolvimento de ferramentas e modelos de redes neurais obtidos ao decorrer do trabalho.

6.1 Sistema Web para Classificação de Frases

Nas seguintes imagens observaremos o resultado obtido do desenvolvimento do sistema classificação de frases online.

Utilizando o sistema foram classificados dois datasets, que, somados, possuem 2.892 elementos classificados, de um total de 43.064 elementos.

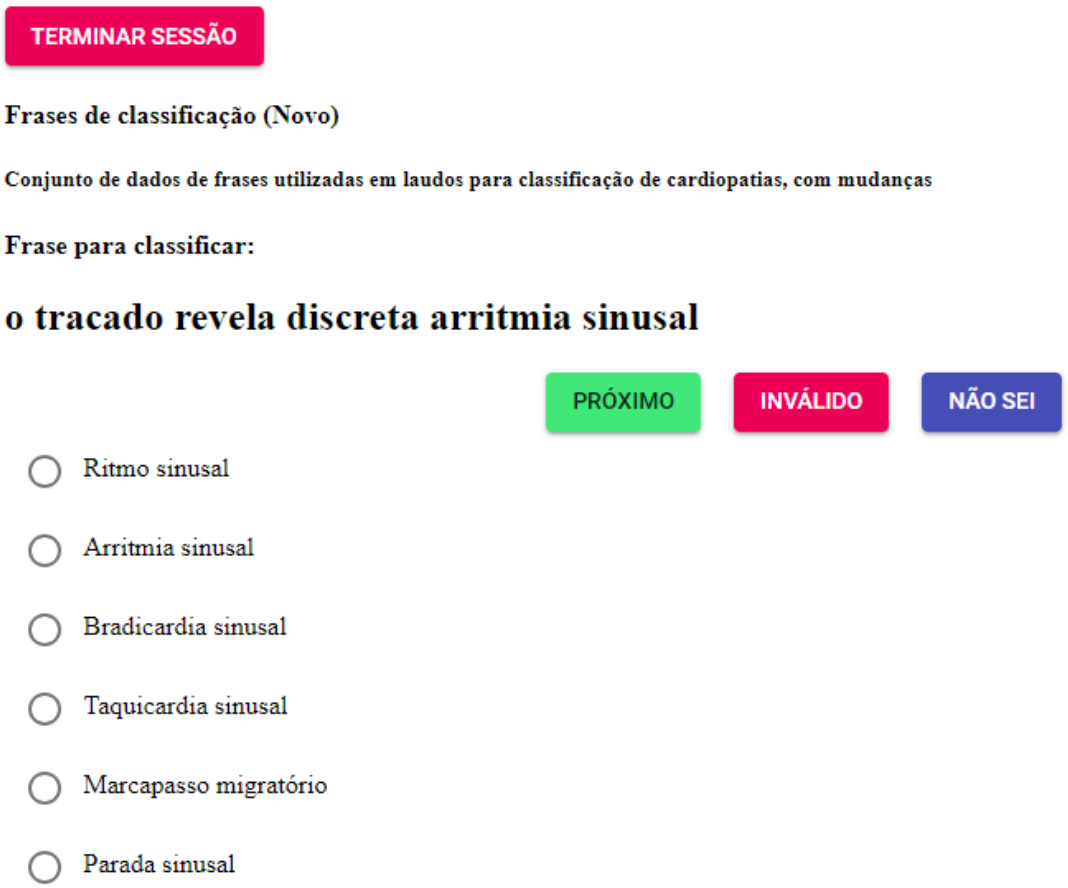


Figura 36 – Interface de Usuário do Cliente Web

Na Figura 36 vemos a interface com o usuário resultante da implementação do cliente web. Devemos lembrar que o restante das opções de classificações não estão retratadas na figura pois não há espaço porém estas podem ser acessadas utilizando a rolagem da tela.

O cliente foi implementado utilizando pouco menos de 1.000 linhas de código JavaScript em um período total de 20 dias, sendo que nem todos os dias do período foram utilizados para o desenvolvimento.

Na fase de classificação do segundo dataset de treinamento foi mensurado a classificação de 1.400 elementos em um período de 4 horas, o que equivale à velocidade de $\approx 5,83$ classificações por minuto. Deve-se destacar que as maiores contribuições para obtenção desta velocidade de classificação foram a adição dos botões "Nao sei" e "Inválido" e a ordem em que as opções aparecem para o usuário.

Quanto à experiência do usuário, esta poderia ser melhorada oferecendo uma ferramenta de busca para filtrar as opções disponíveis de forma que bastaria digitar um trecho da opção para encontrar a desejada, diferentemente do modo atual, onde é necessário que o usuário "arraste" a página até localizar a opção que procura.

Outra maneira de acelerar ainda mais o processo de classificação seria oferecer atalhos de teclado para as opções "Salvar", "Não Sei" e "Inválido". Dessa maneira não haveria necessidade de realizar certas movimentações com o mouse, caso o usuário estivesse acessando o sistema através de um computador desktop. Essa modificação não afetaria o uso no caso de aparelhos móveis.

Além dessas modificações, poderia-se, ainda, modificar a interface de forma que a frase a ser classificada acompanhasse a rolagem da página. Dessa maneira seria possível a realização de uma conferência rápida por parte do usuário do sistema, diferentemente da implementação atual, onde o usuário necessita rolar a página até o topo para consultar a frase a ser classificada novamente.

Essas melhorias não foram implementadas pois julgou-se que apresentariam pouco aumento na velocidade de classificação e atrasariam o andamento do projeto como um todo, além de terem sido descobertas após a compilação do segundo dataset de treinamento, ou seja, não havia mais necessidade de classificar outro dataset naquele momento no tempo.

6.2 Redes Neurais para Classificação de Frases

Nesta seção exploraremos os resultados obtidos através do treinamento da rede de classificação de frases de laudos e será feito um comparativo entre os dois tipos de arquitetura.

6.2.1 Análise dos Embeddings Doc2Vec

De forma a entendermos um pouco melhor o funcionamento do mapeamento de frases à um vetor do $\mathbb{R}^n$ foi realizada uma Análise de Componentes Principais (PCA) de modo à reduzir a dimensão dos vetores resultantes do embedding das frases para 3

dimensões, possibilitando plotar um gráfico representando uma visualização do espaço vetorial resultante.

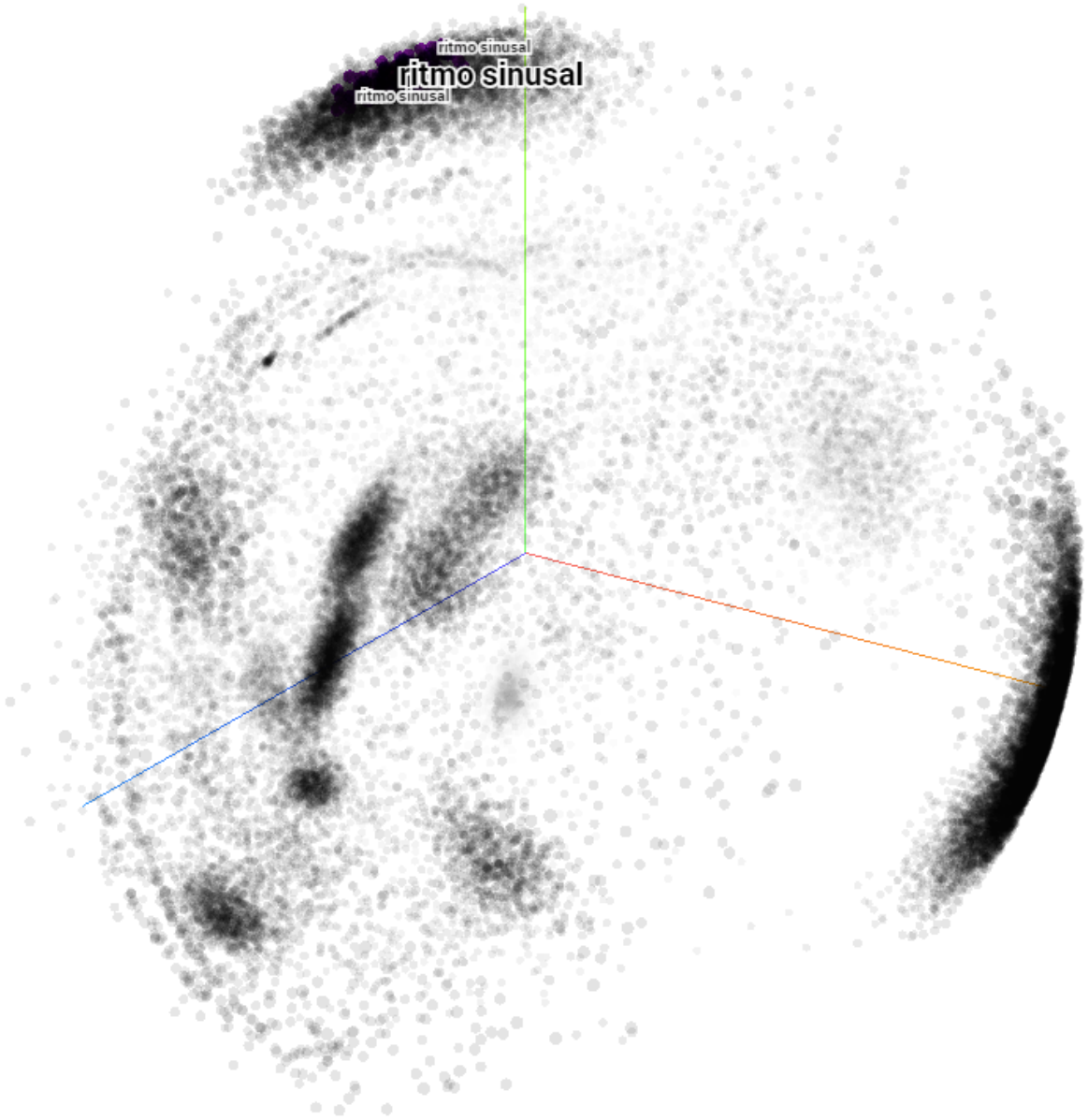


Figura 37 – Visualização dos embeddings após PCA

Na Figura 37 estão posicionados em um gráfico 3D os vetores resultantes da aplicação de PCA sobre os embeddings treinados pela ferramenta Doc2Vec. Observa-se a formação de *clusters* (aglomerações) de frases semelhantes, como mostrado na parte superior da figura com a frase "ritmo sinusal".

A existência de clusters é um bom indício de que será possível treinar uma rede neural para classificar os embeddings derivados das frases.

6.2.2 Treinamento dos Modelos Doc2Vec e ConvNet

Os dois tipos de arquitetura foram treinados utilizando o mesmo dataset de treinamento e validação. O de treinamento conta com 36.812 frases classificadas e no dataset de validação encontram-se 1460 frases. O dataset de treinamento foi obtido fazendo a redistribuição de classificações de frases repetidas pertencentes ao conjunto de dados base (sem classificação, contendo todas as frases extraídas das conclusões de laudos) cuja classificação já foi realizada, através do sistema web de classificação de frases. Dessa maneira foi possível obter um número significativamente maior de exemplos para serem utilizados no treinamento, com a desvantagem de ocorrer repetições entre os elementos.

A partir daqui chamar-se-á de ConvNet o modelo cuja arquitetura utiliza camadas de convolução e de Doc2Vec a família de modelos que utilizam os embeddings previamente treinados pela ferramenta Doc2Vec.

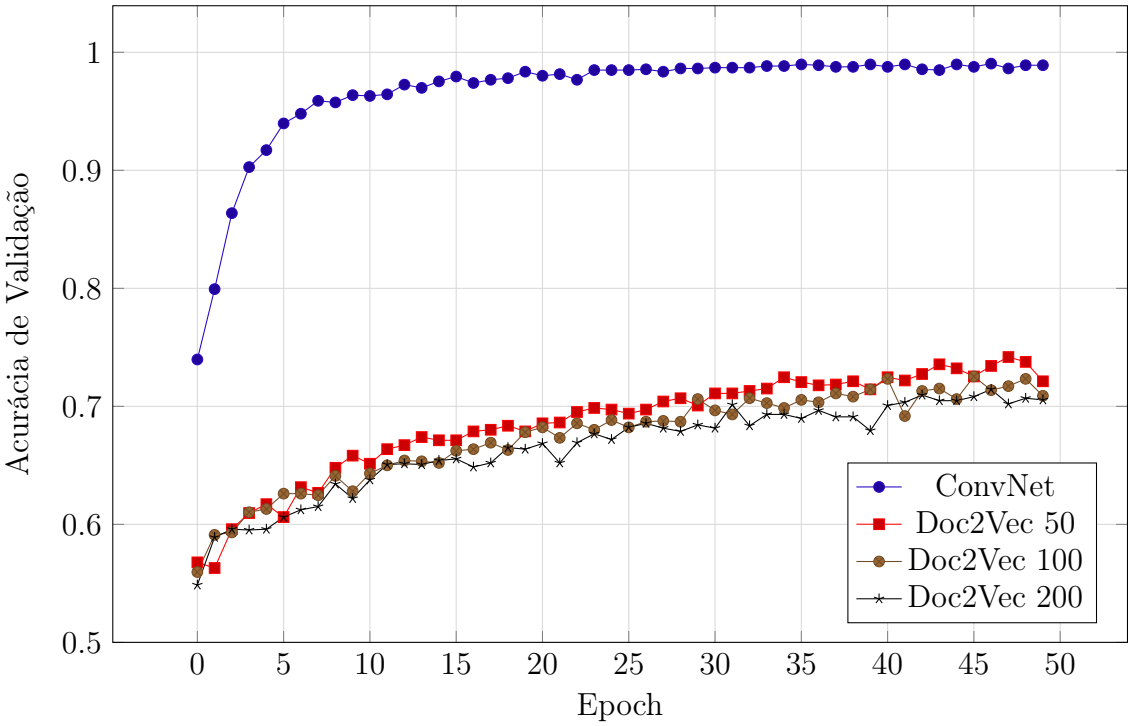


Figura 38 – Acurácia de validação dos modelos ao longo do treinamento

Na Figura 38 temos o gráfico da métrica de acurácia de validação (parcela do conjunto de dados de validação contendo os elementos classificados corretamente pela rede neural). Observa-se que a acurácia dos modelos do tipo Doc2Vec permaneceram com valores próximos uns dos outros durante todo o período de treinamento, enquanto que a acurácia do modelo ConvNet cresceu rapidamente, permanecendo praticamente estável a partir da epoch 25.

6.2.3 Desempenho dos Modelos

Será comparado agora o resultado do treinamento das diferentes redes neurais sobre o mesmo dataset. Os valores apresentados na tabela originam da avaliação das métricas de perda e acurácia finais sobre o conjunto de dados de validação. O número que acompanha o nome dos modelos Doc2Vec significa o tamanho do vetor de embedding escolhido. Foi escolhido um embedding de dimensão 50 para o modelo ConvNet.

Tabela 2 – Tabela de comparação do desempenho das redes neurais de classificação de frases

Modelo	Perda	Acurácia ao final de 50 *epochs*
Doc2Vec 50	1.0407	0.7212
Doc2Vec 100	1.1179	0.7089
Doc2Vec 200	1.1515	0.7055
ConvNet	0.0818	0.9890

Na Tabela 2 pode-se ver que houve pouca diferença de desempenho com a variação do tamanho do embedding, tanto nos valores de perda quanto de acurácia. Vemos também que a arquitetura utilizando camadas de convolução atingiu um patamar de acurácia muito superior às redes Doc2Vec.

Serão analisados agora alguns exemplos de frases classificadas pelos modelos comparando as suas respectivas classificações para tentarmos entender essa discrepância entre os tipos de arquitetura. Por apresentarem resultados muito similares, os modelos utlizando Doc2Vec serão representados apenas pelo modelo Doc2Vec 50, evitando-se assim redundâncias e por este ter apresentando o melhor resultado entre os 3 testados.

Para analisar mais detalhadamente o comportamento dos modelos, foi compilada a Tabela 3, onde estão dispostas as saídas obtidas através do processamento da frase na primeira coluna através das redes neurais. Foi adicionado um ✓ para simbolizar o acerto e X para o erro de classificação.

Olhando atentamente às diferenças entre as frases da Tabela 3, nota-se que o modelo Doc2Vec tem dificuldade de diferenciar palavras como "esquerdo" e "direito", assim como adição de palavras no começo de uma frase. O modelo também classificou erroneamente a frase "ritmo sinual". Isso mostra que esta tem problemas ao encontrar erros gramaticais. Houveram erros por parte do modelo ConvNet, porém este se mostrou um poucos mais resistente a erros gramaticais, adição de palavras no começo da frase e substituição de palavras próximas.

Sabendo que a qualidade da geração dos embeddings afeta diretamente o desempenho de ambos os modelos, fez-se uma investigação mais aprofundada para tentar encontrar a fonte dos problemas observados.

Na Tabela 4 temos o resultado do cálculo da similaridade entre os embeddings

Tabela 3 – Tabela de comparação entre classificações de frases obtida dos modelos Doc2Vec e ConvNet

Frase	Doc2Vec 50	ConvNet
amplitude aumentada, sugestivo de sobrecarga atrial direita.	Sobrecarga atrial direita ✓	Sobrecarga atrial direita ✓
amplitude aumentada, sugestivo de sobrecarga atrial esquerda.	Sobrecarga ventricular esquerda X	Sobrecarga atrial esquerda ✓
amplitude aumentada, sugestivo de sobrecarga ventricular direita	Sobrecarga atrial direita X	Sobrecarga ventricular esquerda X
aumento ventricular esquerdo.	Aumento ventricular esquerdo ✓	Aumento ventricular esquerdo ✓
aumento ventricular direito.	Sobrecarga ventricular esquerda X	Inválido X
ritmo sinusal	Ritmo sinusal ✓	Ritmo sinusal ✓
ritmo sinual	Inválido X	Ritmo sinusal ✓
arritmia sinusal	Arritmia sinusal ✓	Arritmia sinusal ✓
o exame revelou presenca de arritmia sinusal.	Inválido X	Arritmia sinusal ✓
arritmia sinusal foi encontrada no exame.	Inválido X	Inválido X
Total de acertos	4/10	7/10

Tabela 4 – Similaridade entre Embeddings de Palavras Doc2Vec vs ConvNet

Palavras	Similaridade Doc2Vec	Similaridade ConvNet
"esquerdo" e "direito"	0.8434753	-0.33746848
"ventricular" e "atrial"	0.7794222	-0.10196117
"sinusal" e "sinual"	0.45716077	0.77395505

das palavras da coluna "Palavras". A similaridade foi calculada à partir do cosseno do ângulo entre os vetores provenientes do embedding das palavras (vetores de $\mathbb{R}^{50}$) nos dois diferentes modelos.

Após o cálculo da similaridade, fica mais evidente o porquê das diferenças de performance entre os modelos. Trata-se de uma consequência do treinamento, realizado diferentemente entre as duas arquiteturas. Na primeira, Doc2Vec, os embeddings são treinados de forma separada às classificações de frases. Apesar da rede neural do modelo Doc2Vec utilizar os embeddings gerados a partir de uma frase, e não de palavras, a geração dos embeddings de frases depende diretamente dos embeddings de palavras, consequentemente, se os embeddings de duas palavras distintas forem muito similares, no momento da geração do embedding de uma frase contendo uma destas, o embedding resultante desta frase será similar ao do embedding resultante da frase contendo a palavra

substituída por sua similar. O contrário também é valido.

Tabela 5 – Similaridade entre Embeddings de Frases Doc2Vec

Frase 1	Frase 2	Similaridade
aumento atrial esquerdo	aumento atrial direito	0.468314
ritmo sinusal	ritmo sinual	0.3661409
sobrecarga atrial direita	sobrecarga ventricular direita	0.69439983

Temos na Tabela 5 as similaridade entre as frases das colunas Frase 1 e Frase 2. Observa-se que há uma relativa similaridade entre os embeddings das frases "aumento atrial esquerdo" e "aumento atrial direito" ao mesmo tempo que há uma relativa baixa similaridade entre os embeddings das frases "ritmo sinusal" e "ritmo sinual". Essa constatação, juntamente com os valores mostrados na Tabela 4, indica que há de fato uma dependência direta da similaridade das frases com a similaridade das palavras. Fato, este, que afeta diretamente o desempenho do modelo Doc2Vec visto que este utiliza os embeddings de frases para realizar a classificação.

Para o modelo ConvNet, o mesmo não se observa, pois os embeddings das palavras são treinados juntamente com as classificações das frases, consequentemente, o mapeamento resultante procura ressaltar a diferença, ou seja, reduzir a similaridade entre as palavras, maximizando, assim, a capacidade de diferenciação entre as palavras que mais afetam as classificações das frases, como as palavras "esquerdo"e "direito". Esse fenômeno pode ser observado verificando a Tabela 4, onde mostrou-se que há uma baixa similaridade entre os embeddings de palavras de alta importância na classificação e alta similaridade entre os embeddings de palavras que pouco afetam a classificação, caso de "sinusal"e "sinual".

Considerando-se todos resultados mostrados até agora, escolheu-se o modelo ConvNet para realizar a classificação dos 31 mil exames disponíveis para utilização no trabalho haja vista que este apresentou o melhor desempenho entre as arquiteturas de redes neurais testadas (consultar Tabela 2).

6.3 Redes Neurais para Classificação de Traçados

Nesta seção abordaremos os resultados obtidos do treinamento das redes neurais que recebem como entrada diretamente o traçado de um exame de ECG e tentam classificar em *Normal* e *Anormal*

6.3.1 Treinamento

Na fase do treinamento foram confeccionados 4 dataset diferentes. Cada dataset define de maneira distinta o que é considerado *Normal* e *Anormal*. Segue o nome destes e as suas respectivas definições:

1. Dogs

   Este dataset seleciona somente exames provenientes de animais da espécie canina. Sua definição de *Normal* é: exame apresenta apenas as classificações pertencentes a um subconjunto de { Ritmo sinusal, Arritmia sinusal }, *Anormal* para todos os outros.

2. Cats

   Este dataset seleciona apenas de pacientes da espécie felina. Suas definições de *Normal* e *Anormal* são identicas às do dataset Dogs.

3. Marc

   Este dataset seleciona apenas exames de pacientes das espécie canina. Sua definição de *Normal* é: exame apresenta apenas classificação { Ritmo sinusal }. Sua definição de *Anormal* é: apenas classificações { Ritmo sinusal, Marcapasso migratório }.

4. Ritm

   Este dataset também utiliza apenas exames de cães. Sua definição de *Normal* é: exame possui apenas classificação Ritmo sinusal . De *Anormal*: apenas { Arritmia sinusal }.

Deu-se preferência para exames da espécie canina pois, em sua grande maioria, o sinal de ECG aparece de forma mais proeminente frente a ruídos e interferências quando comparado ao sinal obtido de pacientes felinos.

Na Figura 39 vemos uma pequena parte dos modelos treinados para classificar os traçados de exames ECG diretamente. No eixo horizontal temos a *epoch* de treinamento. No eixo vertical temos a acurácia de validação. Observa-se que nenhum modelo conseguiu convergir para um valor alto (maior que 60%) de acurácia, permanecendo sempre no entorno de 50%, o que poderia ser considerado como se a rede estivesse escolhendo ao acaso as classificações. Apesar de todos os modelos treinados não constarem no mesmo gráfico, essa figura retrata bem o comportamento de todos os modelos treinados, isto é, não responderam ao treinamento.

Na busca para encontrar um modelo que conseguisse alcançar aprendizado, criou-se diversos hiperparâmetros que modificam a arquitetura da rede em si. Fazendo a variação destes hiperparâmetros (número de camadas de convolução, dimensão dos filtros de convolução, janela e stride de maxpooling, dropout, número de neurônios das camadas de decisão, entre outros), foram testadas mais de 60 versões de arquiteturas da rede neural através de *grid search*.

Também tentou-se variar o número de elementos dos diferentes datasets utilizados em treinamento, sem sucesso. Adicionalmente, variou-se a dimensão do sinal de entrada.

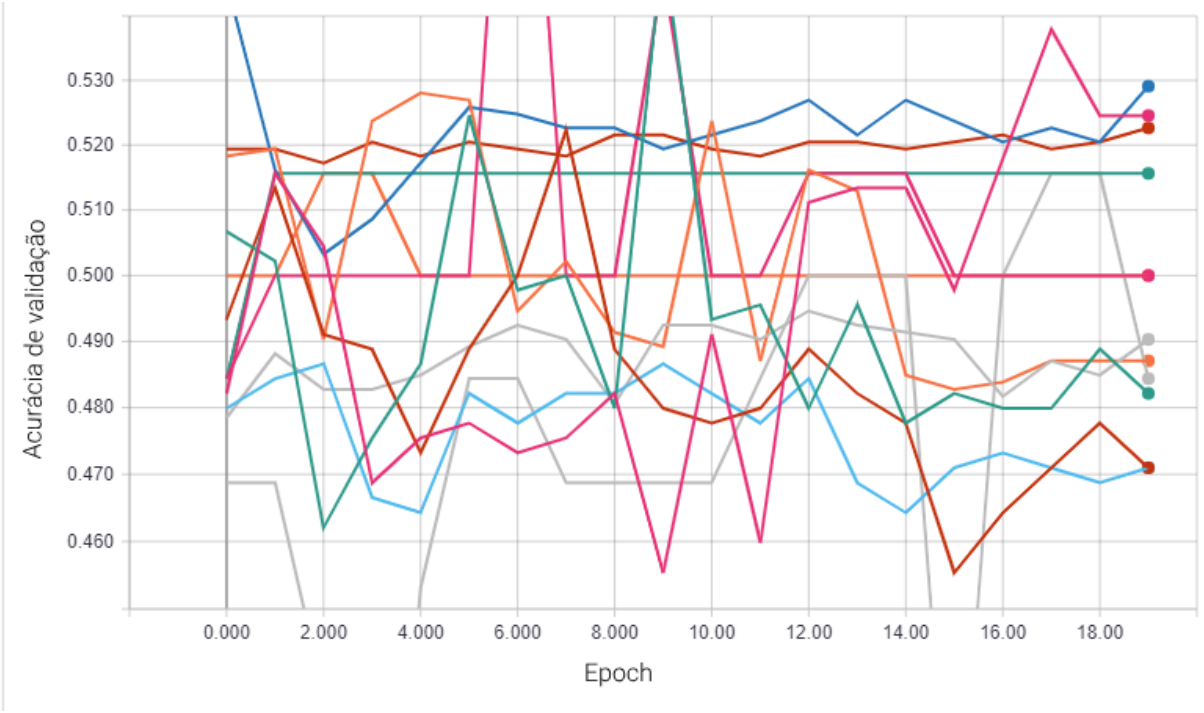


Figura 39 – Acurácia de validação dos modelos para classificação de traçado

Foram testadas versões da rede com dimensões de entrada de 1000×2, 2000×2, 6000×2 e 30000×2. Nenhuma dessas variações obteve sucesso.

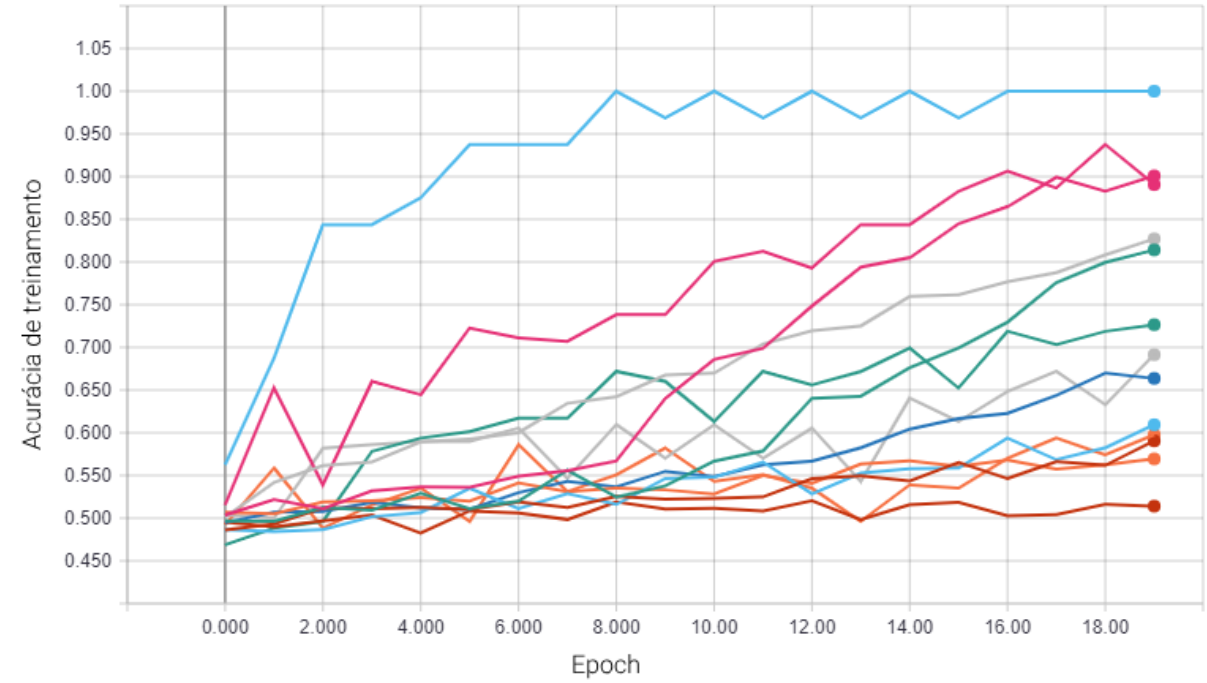


Figura 40 – Acurácia de treinamento dos modelos para classificação de traçado

Na Figura 40 temos o gráfico da acurácia de treinamento (eixo vertical) ao longo das epochs de treinamento (eixo horizontal). A acurácia de treinamento é o mesmo teste de acerto aplicado ao conjunto de validação, porém aplicado ao conjunto de treinamento,

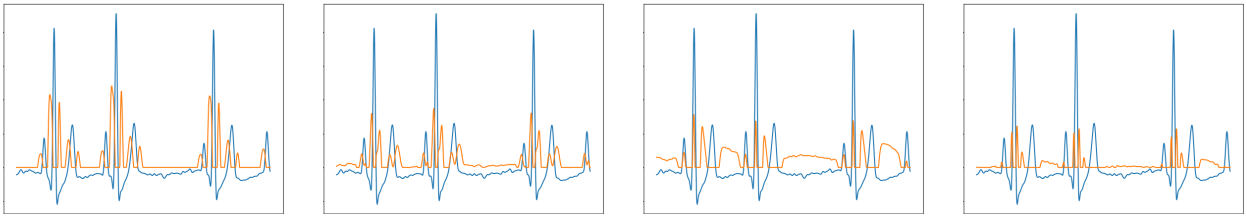
ou seja, não testa o poder de generalização dos modelos. Porém, observou-se que há resposta positiva dos modelos ao treinamento, visto que existe a convergência dos valores de acurácia para 100%.

Esse resultado acabou criando a necessidade de uma análise mais aprofundada sobre os motivos do péssimo desempenho no quesito generalização apresentado pelos modelos.

6.3.2 Análise interna dos modelos

Para tentarmos entender um pouco mais sobre os pontos de falha dos modelos, serão analisadas as saídas das camadas intermediárias do modelo na tentativa de encontrar precisamente a causa.

Iremos analisar agora a saída intermediária da primeira camada de convolução de um dos modelos treinados, isto é, a saída dos seus filtros.



(a) Filtro 1 (b) Filtro 2 (c) Filtro 3 (d) Filtro 4

(e) Filtro 5 (f) Filtro 6 (g) Filtro 7 (h) Filtro 8

Figura 41 – Saída dos filtros da primeira camada de convolução

Na Figura 41 temos as saídas dos 8 filtros de convolução utilizados em laranja, com o sinal de ECG original em azul ao fundo. Pode-se ver que os filtros modificam o sinal de forma tal a destacar mais ou menos partes específicas do traçado. Esta operação é chamada de "extração de *features*". Features são informações específicas, as vezes sem correspondência com as nossas expectativas, sobre os dados de entrada.

As primeiras camadas de uma rede neural de convolução realizam operações ditas de "baixo nível", isto é, trabalham de forma mais próxima com a fonte da informação. Do outro lado da rede, na saída, temos apenas um único escalar para representar toda a informação contida no sinal de entrada. Analisaremos agora a entrada que maximiza a classificação *Anormal*.

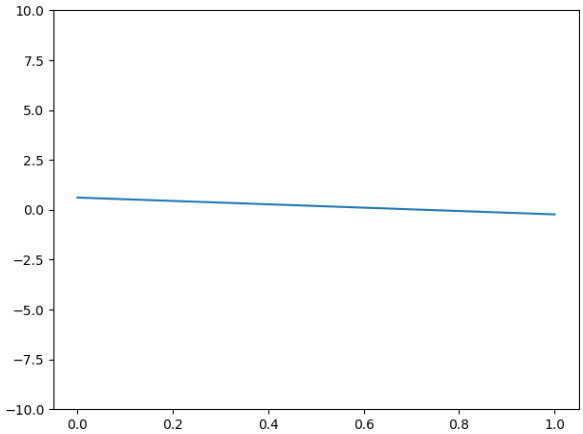


Figura 42 – Entrada que maximiza a categoria *Anormal*

Na Figura 42 observou-se que o gráfico do sinal hipotético que maximizaria a classificação do sinal de entrada como *Anormal* (saída 1.0 da rede) é uma reta com pequeno declive ao longo de toda a a extensão do sinal.

Com esse resultado pode-se supor dois problemas. Primeiro: não há similaridade suficiente entre os exemplos de treinamento para as classes *Normal* e *Anormal*. Segundo: a rede não é capaz de propagar a informação até o final da mesma. É importante ressaltar que o mesmo comportamento manifestou-se após o treinamento dos modelos sobre todos os datasets.

A segunda hipótese é parcialmente descartada pois analisando o comportamento da acurácia de treinamento (Figura 40) estas convergem para 100%, de todas as arquiteturas, ou seja, de alguma forma a rede é capaz de distinguir entre os exemplos de treinamento.

A primeira hipótese é mais provável de ser precisamente o problema enfrentado neste trabalho. Os sinais de ECG dos pacientes diferem muito em amplitude máxima e mínima, nível de ruído e interferência, variação da linha de base e diferenças morfológicas do próprio sinal.

Para melhorar a correlação entre diferentes sinais poderia ser empregado normalização, porém isso afetaria de maneira desproporcional traçados cujos sinais apresentam mudanças bruscas na linha de base, desconexão dos eletrodos ou atividade muscular repentina. Nesses casos a normalização diminuiria ainda mais a qualidade dos traçados. Esperava-se que, com o treinamento em grande quantidade de dados, a rede neural conseguiria tolerar esses tipos de problemas.

Pode-se argumentar que a rede não foi treinada utilizando um número de exemplos suficientes para que ocorra o aprendizado. Não há como este fato: na área de conhecimento das redes neurais, possuir mais dados disponíveis é, na maioria dos casos, melhor. A quantidade de dados necessária é o maior empecilho para que empresas pequenas, pesquisadores e entusiastas sejam capazes de implementar sistemas de inteligência artificial aplicando

redes neurais. Para adquirir mais dados de exames e laudos é necessário que mais médicos veterinários façam uso do sistema de telemedicina da InPulse continuamente. Assim, com o decorrer do tempo, em um momento haverá volume de dados suficiente para que o treinamento dos modelos ocorra de maneira efetiva.

Pode-se argumentar também que as categorias binárias escolhidas como saída do sistema são pouco granulares e não permitem que o treinamento da rede neural encontre correlações relevantes a todos os exemplos de sinais do conjunto de dados. Este é um ponto a ser testado, caso o projeto continue em andamento: verificar se a adição de classes mais granulares beneficiam ou prejudicam o desempenho dos modelos.

Outro ponto a ser verificado é se a escolha de utilizar os sinais diretamente no domínio do tempo realmente faz sentido. Pode-se estudar como alternativa, no futuro, a transformação do sinal para o domínio da frequência utilizando transformadas de Fourier ou wavelets. Essa abordagem permitiria robustez maior nos casos onde o sinal apresenta ruído e interferência pois permite que a rede neural processe diretamente as componentes frequenciais relevantes do sinal, ignorando, se possível, as componentes que não afetam diretamente a classificação.

Por fim temos que a experiência obtida através do desenvolvimento deste trabalho é de suma importância para a continuação das tentativas de resolução da problemática principal deste estudo: a classificação automática de sinais de ECG de pacientes animais.

7 Conclusões e Perspectivas

Espera-se que o presente trabalho acrescente um pouco de conhecimento e experiência ao campo do aprendizado de máquina aplicado na saúde animal.

Embora o objetivo principal do trabalho não tenha sido alcançado como esperado, o estudo e aplicação de redes neurais em dois problemas diferentes foi de grande valia para a consolidação do conhecimento empregado no desenvolvimento do projeto como um todo.

Um dos objetivo parciais do trabalho foi atingido com sucesso. Este era a classificação dos laudos de exames através do texto contido na conclusão do mesmo. Utilizando duas arquiteturas diferentes, uma com resultado não satisfatório e outra que excedeu as expectativas, de redes neurais foram classificados cerca de 31 mil exames de ECG do banco de dados da empresa InPulse Animal Health. Tais classificações foram de extrema importância para a continuação do trabalho pois serviram de base para a confecção dos conjuntos de dados para treinamento das redes neurais em fase posterior do trabalho.

Quanto à classificação de frases, acredito que seria possível melhorar ainda mais a acurácia de predição caso houvessem mais frases classificadas para treinamento. Devemos observar que várias das categorias previstas não constaram em nenhum exemplo classificado que fizeram parte do conjunto de treinamento e por esse motivo tais categorias foram ignoradas no momento de classificar o conjunto das conclusões de laudos de exames. Esse fato pode ter prejudicado a classificação dos laudos, porém no momento não há nenhuma maneira de fazer essa verificação.

Visto que as redes neurais treinadas durante o trabalho não obtiveram sucesso na classificação dos sinais, em próximos estudos seria pertinente melhorar de alguma forma o pré-processamento dos sinais de ECG, realizando análise no domínio da frequência ou aplicando filtros diferentes dos aplicados neste trabalho e ao mesmo tempo melhorar a qualidade e quantidade de traçados. Há estudos onde foi aplicado, sobre o sinal de entrada, transformadas de Fourier, transformando assim, a entrada efetiva da rede em uma sequência de transformadas de Fourier, com bons resultados.

Caso o objetivo principal do trabalho tivesse obtido sucesso, este seria o primeiro passo para a automatização da laudagem de exames eletrocardiográficos no campo da veterinária. Tal serviço seria capaz de mudar o estado atual da disponibilidade de realização exames de ECG em pacientes animais, trazendo acesso rápido para a emissão de laudos em todo o Brasil e até no mundo, bastando simplesmente que o requisitante possua um equipamento capaz de capturar a atividade elétrica do coração do paciente.

Referências

1 InPulse Animal Health. *InPulse*. 2018. <https://inpulse.vet.br/>. [Online; accessed 20-July-2018]. Citado na página 26.

2 WILKINS, L. W. . *ECG Interpretation Made Incredibly Easy!*. Wolters Kluwer/Lippincott Williams & Wilkins Health, 2011. (Made Incredibly Easy). ISBN 9781608312894. Disponível em: <https://books.google.com.br/books?id=qTaKQgAACAAJ>. Citado na página 27.

3 Silva, I. N. d. et al. *Artificial Neural Networks : A Practical Course*. 1. ed. [S.l.]: Springer International Publishing, 2017. ISBN 978-3-319-43162-8,978-3-319-43161-1. Citado na página 37.

4 Basirat, M.; Roth, P. M. The Quest for the Golden Activation Function. *ArXiv e-prints*, ago. 2018. Citado na página 41.

5 Pedamonti, D. Comparison of non-linear activation functions for deep neural networks on MNIST classification task. *ArXiv e-prints*, abr. 2018. Citado na página 41.

6 Dong, Q.; Gong, S.; Zhu, X. Imbalanced Deep Learning by Minority Class Incremental Rectification. *ArXiv e-prints*, abr. 2018. Citado na página 47.

7 Janocha, K.; Czarnecki, W. M. On Loss Functions for Deep Neural Networks in Classification. *ArXiv e-prints*, fev. 2017. Citado na página 47.

8 KIEFER, J.; WOLFOWITZ, J. Stochastic estimation of the maximum of a regression function. *Ann. Math. Statist.*, The Institute of Mathematical Statistics, v. 23, n. 3, p. 462–466, 09 1952. Disponível em: <https://doi.org/10.1214/aoms/1177729392>. Citado na página 48.

9 Ruder, S. An overview of gradient descent optimization algorithms. *ArXiv e-prints*, set. 2016. Citado na página 48.

10 Kingma, D. P.; Ba, J. Adam: A Method for Stochastic Optimization. *ArXiv e-prints*, dez. 2014. Citado na página 48.

11 Hinton, G. E. et al. Improving neural networks by preventing co-adaptation of feature detectors. *ArXiv e-prints*, jul. 2012. Citado na página 48.

12 Saeedan, F. et al. Detail-Preserving Pooling in Deep Networks. *ArXiv e-prints*, abr. 2018. Citado na página 49.

13 Khurana, D. et al. Natural Language Processing: State of The Art, Current Trends and Challenges. *ArXiv e-prints*, ago. 2017. Citado na página 49.

14 Tixier, A. J.-P. Notes on Deep Learning for NLP. *ArXiv e-prints*, ago. 2018. Citado na página 50.

15 Mikolov, T. et al. Efficient Estimation of Word Representations in Vector Space. *ArXiv e-prints*, jan. 2013. Citado na página 50.

16 Mikolov, T. et al. Distributed Representations of Words and Phrases and their Compositionality. *ArXiv e-prints*, out. 2013. Citado na página 51.

17 Le, Q. V.; Mikolov, T. Distributed Representations of Sentences and Documents. *ArXiv e-prints*, maio 2014. Citado na página 52.

18 Bird, S.; Loper, E.; Klein, E. *Natural Language Toolkit*. 2014. <https://www.nltk.org/>. [Online; accessed 19-July-2018]. Citado na página 52.

19 Google. *TensorFlow*. 2015. <https://www.tensorflow.org/>. [Online; accessed 20-July-2018]. Citado na página 52.

20 CHOLLET, F. *Keras: The Python Deep Learning library*. 2015. <https://keras.io/>. [Online; accessed 19-July-2018]. Citado na página 52.

21 Google. *TensorBoard: Visualizing Learning*. 2017. <https://www.tensorflow.org/guide/summaries_and_tensorboard>. [Online; accessed 19-July-2018]. Citado na página 53.

22 Sociedade Brasileira de Cardiologia. *III DIRETRIZES DA SOCIEDADE BRASILEIRA DE CARDIOLOGIA SOBRE ANÁLISE E EMISSÃO DE LAUDOS ELETROCARDIOGRÁFICOS*. [S.l.: s.n.], 2016. Citado na página 58.

23 Sun Microsystems. *Java Programming Language*. 1995. <https://www.oracle.com/java/>. [Online; accessed 19-July-2018]. Citado na página 69.

24 JetBrains s.r.o. *Kotlin Programming Language*. 2016. <https://kotlinlang.org/>. [Online; accessed 19-July-2018]. Citado na página 69.

25 EICH, B. *JavaScript Programming Language*. 1995. <https://www.javascript.com/>. [Online; accessed 19-July-2018]. Citado na página 69.

26 PostgreSQL Global Development Group. *PostgreSQL: The World's Most Advanced Open Source Relational Database*. 1989. <https://www.postgresql.org/>. [Online; accessed 19-July-2018]. Citado na página 69.

27 Pivotal Software. *Spring Framework*. 2002. <https://spring.io/>. [Online; accessed 19-July-2018]. Citado na página 70.

28 Pivotal Software. *Spring Data*. 2017. <http://spring.io/projects/spring-data>. [Online; accessed 19-July-2018]. Citado na página 70.

29 Digital Ocean. *Digital Ocean*. 2018. <https://www.digitalocean.com/>. [Online; accessed 19-July-2018]. Citado na página 71.

30 HYKES, S. *Docker*. 2013. <https://www.docker.com/>. [Online; accessed 19-July-2018]. Citado 2 vezes nas páginas 71 e 72.

31 ABRAMOV, D.; CLARK, A. *Redux*. 2015. <https://redux.js.org/>. [Online; accessed 19-July-2018]. Citado na página 71.

32 SYSOEV, I. *NGINX: High Performance Load Balancer, Web Server and Reverse Proxy*. 2004. <https://nginx.org/>. [Online; accessed 19-July-2018]. Citado na página 72.

33 ROSSUM, G. van. *Python Programming Language*. 1991. <https://www.python.org/>. [Online; accessed 19-July-2018]. Citado na página 72.

34 Kothapalli, K. et al. CPU and/or GPU: Revisiting the GPU Vs. CPU Myth. *ArXiv e-prints*, mar. 2013. Citado na página 73.

35 ŘEHůřEK, R. *gensim: Topic Modelling For Humans*. 2009. <https://radimrehurek.com/gensim/>. [Online; accessed 19-July-2018]. Citado na página 73.

36 MCKINNEY, W. *Pandas*. 2008. <https://pandas.pydata.org/>. [Online; accessed 19-July-2018]. Citado na página 73.

37 BEAZLEY, D. M. *SWIG: An Easy to Use Tool for Integrating Scripting Languages with C and C++*. 1996. <http://www.swig.org/>. [Online; accessed 19-July-2018]. Citado na página 80.

38 OLIPHANT, T. *Numpy*. 2006. <https://www.numpy.org/>. [Online; accessed 19-July-2018]. Citado na página 80.

Anexos

ANEXO A – Código-fonte das entidades do sistema web

Lista A.1 – Definição de Entry

```
package io.pittacium.domain

import org.hibernate.annotations.Type
import java.io.Serializable
import java.util.*
import javax.persistence.*

data class EntryDto(val datasetId: UUID?, val id: Int, val value: String, val label:
    ↪ String? = null)

data class CreateEntryDto(val datasetId: UUID, val value: String)

data class LabelEntryDto(val datasetId: UUID, val id: Int, val label: String)

class EntryId(
        var datasetId: UUID? = null,
        var id: Int? = null

) : Serializable {

    companion object {
        private const val serialVersionUID = 1293193719379123L
    }

    override fun equals(other: Any?): Boolean {
        if (this === other) return true
        if (javaClass != other?.javaClass) return false

        other as EntryId

        if (datasetId != other.datasetId) return false
        if (id != other.id) return false

        return true
    }

    override fun hashCode(): Int {
        var result = datasetId?.hashCode() ?: 0
        result = 31 * result + (id ?: 0)
        return result
    }


}

@Entity
@IdClass(EntryId::class)
@Table(name = "entry")
data class EntryEntity(
```

```
        @Id
        @Column(name = "datasetId")
        var datasetId: UUID? = null,

        @Id
        @GeneratedValue(strategy = GenerationType.SEQUENCE)
        var id: Int? = null,

        var locked: Boolean = false,

        @Column(columnDefinition = "TEXT")
        val value: String = "",

        @Column(columnDefinition = "TEXT")
        var label: String? = null
) {

    fun toDto() = EntryDto(datasetId, id!!, value, label)

    companion object {

        fun fromDto(create: CreateEntryDto) = EntryEntity(datasetId = create.datasetId,
            ↪ id = null, value = create.value)

        fun fromDto(label: LabelEntryDto) = EntryEntity(label.datasetId, id = label.id,
            ↪ label = label.label)

    }

}
```

Lista A.2 – Definição de Project

```kotlin
package io.pittacium.domain

import java.util.*
import javax.persistence.*
import kotlin.collections.HashSet

data class ProjectDto(val id: UUID, val name: String, val description: String?)

data class CreateProjectDto(val name: String, val description: String?)

@Entity
@Table(name = "project")
data class ProjectEntity(
        @Id @GeneratedValue
        var id: UUID? = null,
        val name: String = "",
        val description: String? = null
) {

    fun toDto() = ProjectDto(id!!, name, description)

    companion object {

        fun fromDto(dto: ProjectDto) = ProjectEntity(dto.id, dto.name, dto.description)

        fun fromDto(dto: CreateProjectDto) = ProjectEntity(name = dto.name, description =
            ↪ dto.description)

    }

}
```

Lista A.3 – Definição de Dataset

```
package io.pittacium.domain

import org.hibernate.annotations.Cascade
import java.util.*
import javax.persistence.*
import javax.transaction.Transactional

data class CreateDatasetDto(val projectId: UUID, val name: String, val description:
    ↪ String?, val options: Collection<String>)

data class DatasetDto(val projectId: UUID, val id: UUID, val name: String, val
    ↪ description: String?, val options: Collection<String>)

@Entity
@Table(name = "dataset")
@Transactional
data class DatasetEntity(
        @Id @GeneratedValue
        var id: UUID? = null,

        @ManyToOne(targetEntity = ProjectEntity::class, fetch = FetchType.LAZY, cascade =
            ↪ [CascadeType.REMOVE])
        @JoinColumn(name = "projectId")
        var project : ProjectEntity? = null,

        @Column(name = "projectId", updatable = false, insertable = false)
        val projectId: UUID? = null,

        @ElementCollection(fetch = FetchType.EAGER)
        @Cascade
        val options: Collection<String> = listOf(),

        val name: String = "",
        val description: String? = null
) {

    fun toDto() = DatasetDto(projectId!!, id!!, name, description, options)

    companion object {

        fun fromDto(create: CreateDatasetDto) = DatasetEntity(projectId = create.
            ↪ projectId, name = create.name, description = create.description, options =
            ↪ create.options)

        fun fromDto(dto: DatasetDto) = DatasetEntity(dto.id, projectId = dto.projectId,
            ↪ name = dto.name, description = dto.description, options = dto.options)

    }
}
```