Universidade Federal de Pernambuco
Centro de Informática (Cin-UFPE)

Bacharelado em Engenharia da Computação

Redução de alarmes falsos em UTIs para diagnósticos de arritmias cardíacas utilizando redes neurais convolucionais

Gabriel Bezerra Lima

Trabalho de Graduação

Recife - PE
11 de dezembro de 2018

Universidade Federal de Pernambuco
Centro de Informática (Cin-UFPE)

Gabriel Bezerra Lima

Redução de alarmes falsos em UTIs para diagnósticos de arritmias cardíacas utilizando redes neurais convolucionais

Monografia apresentada ao Programa de Bacharelado em Engenharia da Computação do Centro de Informática (Cin-UFPE) da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Engenharia da Computação.

Orientador: *Prof. Dr. Germano Crispim Vasconcelos*

Recife - PE
11 de dezembro de 2018

Resumo

Anualmente milhões de pessoas são internadas em Unidades de Terapia Intensiva (UTIs). Uma característica dessas unidades é a presença de equipamentos de cabeceira utilizados para monitoramento constante de pacientes graves. Em geral esses equipamentos podem gerar alarmes de até 80dB (mesma intensidade encontrada em fábricas) para alertar a equipe médica sobre algum problema no paciente. Porém estudos mostram que o contato frequente com esses alarmes dificulta a recuperação de todos os pacientes afetados devido a privação de sono e leva a dessensibilização a equipe médica. Um agravante destes problemas é que cerca de 90% dos alarmes em UTIs são falsos. O objetivo desse trabalho é melhorar a identificação dos alarmes falsos através do uso de redes neurais convolucionais. Para isso foram comparadas duas abordagens na representação dos sinais: gráfico de linha e *recurrence plots*. Como resultados foram obtidos modelos com performance melhor que a identificada em enfermeiros experientes (i.e. 38% de acerto na classificação dos alarmes) independente do tipo de entrada e com taxas de falsos positivos menores que 30%. Finalmente, foi identificado que não há diferença significativa no uso dos tipos diferentes de entrada.

Palavras-chave: deep learning, transfer learning, arritmia, UTI, recurrence plots, redes neurais, vgg

Abstract

Annually millions of people are hospitalized in Intensive Care Unit (ICU). A characteristic of this units is the presence of bedside equipments used for constant monitoring of severe pacients. At large these equipments can generate alarms up to 80dB (same value encountered in factories) to alert the medical team about any pacient's problems. However studies shows that the frequent exposure to these alarms complicate recovery of all the affected patients due to sleep privation and lead to a dessenbilization of the medical team. An aggravating factor is that about 90% of ICU alarms are false. The goal of this work is to improve the identification of false alarms through the use of Convoluitional Neural Networks (CNN). To achieve the objective this work compares two approaches on the signals representation: line graphs and recurrence plots. As results we obtained models that performed better on alarms classification than experienced nurses (i.e. 38%) with any of the input types and with false positive rate less than 30%. Finally it was discovered that there is no significative difference on the use of any of the input types.

<DIGITE O ABSTRACT AQUI>

Keywords: deep learning, transfer learning, arrhythmia, ICU, recurrence plots, neural networks, vgg

Sumário

Lista de Figuras

Lista de Tabelas

CAPÍTULO 1

Introdução

1.1 Motivação

Anualmente milhões de pessoas são internadas em Unidades de Terapia Intensiva (UTIs). Segundo o Conselho Regional de Medicina de São Paulo (CREMESP) [Bra95] essas unidades se caracterizam pelo acompanhamento através de monitoramento constante à pacientes graves ou com alto risco de recuperação. Em geral, esse tipo de monitoramento é feito através de vários equipamentos de cabeceira que tem uma característica em comum: a geração de alarmes. Esses alarmes tem como objetivo avisar a equipe médica sobre riscos à vida dos pacientes internados.

Segundo Chambrin [Cha01] em uma UTI é possível encontrar mais de 40 fontes diferentes de alarmes (e.g. eletrocardiógrafos, medidores de pressão arterial, respiradores). Porém, apesar do alto potencial desses alarmes em salvar vidas, estudos indicam que o contato constante com eles podem trazer sérios problemas na recuperação de pacientes e no dia-a-dia da equipe médica.

Um equipamento de monitoramento hospitalar pode produzir alarmes de até 80dB, esse intensidade é a mesma encontrada em fábricas e o dobro da encontrada em um quarto comum [PT09]. Uma consequência direta disso é a privação do sono não só do paciente monitorado mas de todos os pacientes afetados pelo barulho. Um problema ainda mais grave desse contato constante com alarmes é a dessensibilização da equipe médica. Após um certo tempo foi observado que a equipe aumentava bastante os limites para ativação dos alarmes de tal forma que mesmo problemas sérios não seriam detectados pelo alarme [BSWI11]. Além disso alguns enfermeiros começaram a atrasar o atendimento aos pacientes tentando reconhecer se o alarme era verdadeiro ou falso apenas pelos sons (mesmo enfermeiros experientes tinham uma taxa de acerto de 38%) [Cha01].

Outro problema bastante encontrado em UTIs em relação a alarmes falsos é a quantidade de falso positivos (i.e. quando um alarme indica um problema que na realidade não está ocorrendo). Segundo Borowski [BSWI11] cerca de 90% dos alarmes são falsos positivos. Esses tipo de alarme geralmente é causado por problemas de equipamento, movimento dos pacientes e até conexões mal feitas.

Em geral, o diagnóstico de um paciente em um leito hospitalar é feito majoritariamente através de inspeção visual do paciente e/ou dos equipamentos que o monitoram. Porém esse tipo de análise tem um alto fator interpretativo que é muito suscetível a erros humanos na identificação da natureza dos alarmes. Em virtude disso muitos trabalhos têm surgido com o objetivo de maximizar a detecção de alarmes falsos em UTIs.

1.2 Estado da Arte

Para resolver esse problema de identificação de alarmes falsos muitos autores consideraram a abordagem de extração de características dos sinais estudados [LC12, ANS+08, BSWI11, LRC14]. Essa abordagem consiste em definir e extrair valores derivados dos dados originais de forma manual (e.g. quantidade de batidas em um eletrocardiograma, número de picos, valor máximo) para serem utilizados na classificação das entradas.

Em Aboukhalil [ANS+08] e Borowski [BSWI11] as características são extraídas para a construção de filtros que detectam se os alarmes gerados pelos equipamentos são verdadeiros ou falsos. No primeiro, é criado um filtro posterior que analisa sinais de pressão sanguínea arterial (ABP) e eletrocardiogramas (ECG) para 5 tipos de problemas cardíacos diferentes. No segundo é desenvolvido um novo algoritmo para aplicação em equipamentos de eletrocardiograma.

Já em Li [LC12, LRC14] as características extraídas alimentam algoritmos de aprendizagem de máquina para resolver os problemas propostos. Em [LC12] é feita a análise de fotopletismografias (PPG) para classificação de qualidade dos sinais usando *Multilayer Perceptrons (MLP)*. O mesmo autor em [LRC14] extrai 14 características diferentes para análise de ECGs utilizando *Support Vector Machines (SVMs)*.

O problema do uso desse tipo de abordagem é o tempo necessário para encontrar e definir as características manualmente (que nem sempre refletem nas melhores opções que definem o problema). Para séries temporais (i.e. informações que variam com o tempo) a extração de características ainda é um desafio maior devido a ruído, alta dimensionalidade e características (como média e variância) que variam ao longo do tempo [LKL14].

Como forma de minimizar os erros inerentes a extração de características trabalhos mais recentes têm usado modelos de *deep learning* para realizar as tarefas que propõem [AFO+17, RHH+17, PKM17, HGD18]. A principal vantagem do uso de *deep learning* é a capacidade de extrair essas características dos dados automaticamente, reduzindo os custos associados a extração e permitindo até encontrar características relevantes que um ser humano não seja capaz de encontrar.

Nos trabalhos citados são usadas diferentes arquiteturas de Redes Neurais Convolucionais (CNN) devido a sua especialidade no processamento de dados em grade (i.e. séries temporais, imagens, vídeos, entre outros). Em Acharya [AFO+17] são utilizados dados de 12 eletrodos de eletrocardiograma (de 200 pacientes) para detecção automática de infarto no miocárdio. Em Rajpurkar [RHH+17] são utilizados 64121 gravações de ECGs para classificação de 12 tipos de arritmias cardíacas. Já em Pyakillya [PKM17] são utilizadas 8528 gravações para identificação do ritmo cardíaco dos pacientes.

Finalmente em Hatami [HGD18] são utilizados *recurrence plots* de dimensões 28x28 para a criação de uma arquitetura de rede neural convolucional em dois estágios capaz de classificar séries temporais.

A dificuldade no uso de algoritmos de *deep learning* é o custo computacional para se encontrar uma arquitetura apropriada que obtenha uma boa performance. Além disso há uma necessidade muito maior de dados se comparado a algoritmos de aprendizagem de máquina mais comuns. Dados que muitas vezes não estão disponíveis ou dependem de esforço humano na sua categorização.

1.3 Objetivos

O objetivo deste trabalho é aplicar os conceitos de *deep learning* para encontrar um modelo que contribua na redução de alarmes falsos em 5 tipos de arritmias cardíacas graves: Assistolia, bradicardia extrema, taquicardia extrema, taquicardia ventricular e fibrilação ventricular. Pra atingir o objetivo foi utilizado sinais provenientes de 750 gravações de 4 equipamentos de monitoramento hospitalar usados em leitos de UTIs: Eletrocardiógrafo, oxímetro de pulso, respirador e transdutor de pressão. Os dados podem ser encontrados no desafio *"Reducing False Arrhythmia Alarms in the ICU: the PhysioNet/Computing in Cardiology Challenge 2015"* da Physionet [GAG+13].

1.3.1 Objetivos Específicos

- Utilizar uma abordagem de *transfer learning* para detecção de alarmes falsos em uma arquitetura de rede neural convolucional VGG16.
- Analisar o impacto do uso de gráficos de linha e *recurrence plots* na classificação de séries temporais.

CAPÍTULO 2

Conceitos Básicos

Nesse capítulo será feita a introdução de conceitos básicos necessários para o entendimento deste trabalho. Inicialmente serão explicadas as arritmias cardíacas e os aparelhos hospitalares utilizados pelo banco de dados. Logo depois será explicado os *recurrence plots* utilizado para geração de um dos tipos de entrada do modelo abordado no projeto. Por fim, serão introduzidos os conceitos relacionados a aprendizagem de máquina, redes neurais e *deep learning* utilizados no projeto. Um aprofundamento teórico sobre a forma de aplicação desses conceitos pode ser encontrado no capítulo 3.

2.1 Arritmias Cardíacas

Segundo o *National Heart, Lung and Blood Institute* [NHN12] arritmias (ou disritmias) cardíacas são alterações no taxa ou no ritmo dos batimentos cardíacos em relação à um valor normal (i.e. 60 a 100 bpm em adultos).

A causa das arritmias são mudanças no tecido do coração, sejam elas por ingestão de medicamentos, estresse, procedimentos cirúrgicos e até problemas nas células que enviam sinais elétricos pro coração. Dentre os fatores de risco estão idade, exposição à poluentes, histórico familiar, entre outros.

Na maioria dos casos, arritmias cardíacas não apresentam sintomas graves além de cansaço, fraqueza, dificuldade de respiração e outros sintomas menores. Porém em casos mais graves podem trazer risco à vida dos pacientes a partir de paradas cardíacas e a longo prazo está associado até a doença de Alzheimer.

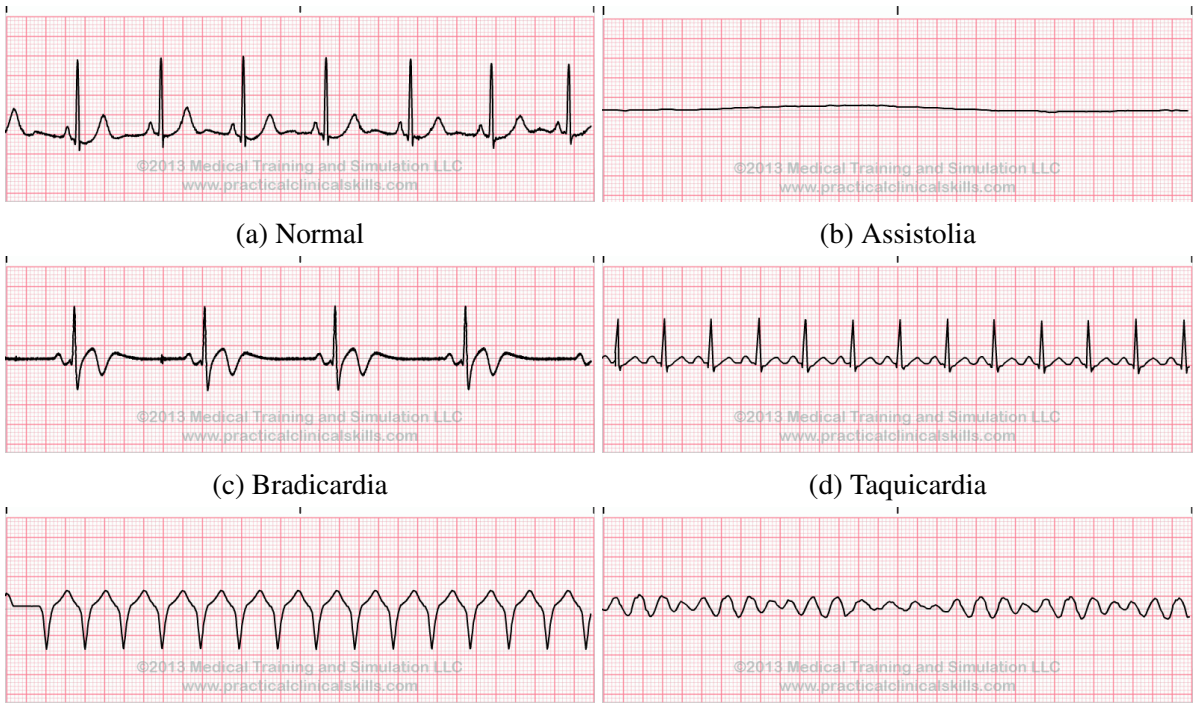
Tabela 2.1: Descrição dos tipos de arritmias encontradas no banco de dados utilizados no projeto. Fonte: *Reducing False Arrhythmia Alarms in the ICU: the PhysioNet/Computing in Cardiology Challenge 2015*, Physionet, Acessado Nov 2018.

Tipo	Descrição
Assistolia	Ausência de QRS por pelo menos 4 segundos
Bradicardia Extrema	Ritmo cardíaco menor que 50bpm por 5 batimentos consecutivos
Taquicardia Extrema	Ritmo cardíaco maior que 140 bpm por 17 batimenos consecutivos
Taquicardia Ventricular	5 ou mais batimentos ventriculares com ritmo cardíaco maior que 100 bpm
Flutter/Fibrilação ventricular	Forma de onda fibrilatória, flutter ou oscilatória por pelo menos 4 segundos

2.1.1 Tipos de arritmias

Arritmias cardíacas podem ser divididas em duas categorias principais dependendo da frequência dos batimentos em relação aos valores considerados normais. Nas taquicardias o paciente pode ter batimentos maiores que 100 bpms. Já nas bradicardias esses batimentos são menores que 60 bpms.

As seções a seguir explicam os 5 tipos de arritmias encontradas no banco de dados encontrado na [GAG+13] utilizados nesse projeto como descritos na tabela 2.1. A figura 2.1 ilustra as arritmias descritas na tabela bem como um caso de ritmo cardíaco normal.



(a) Normal (b) Assistolia

(c) Bradicardia (d) Taquicardia

(e) Taquicardia Ventricular (f) Fibrilação Ventricular

Figura 2.1: Exemplos de eletrocardiogramas correspondentes as arritmias descritas na tabela 2.1. A figura (a) representa um ritmo normal. Note que as figuras (b) e (c) são correspondentes aos problemas de bradicardia e taquicardia em sua forma mais comum (i.e. não extrema). Fonte: Practical Clinical Skills, acessado em Nov 2018, https://www.practicalclinicalskills.com/ekg-reference.

2.1.1.1 Assistolia

A assistolia é a ausência total de batimentos cardíacos do paciente por um período de pelo menos 4 segundos [GAG+13]. Em eletrocardiogramas esse problema é caracterizado pela presença de uma linha reta (figura 2.1b).

Esse problema é muito grave pois indica a parada total do coração e consequentemente ausência de bombeamento de sangue em órgãos vitais como o cérebro. Muitas vezes ela é usada como critério para determinação de óbito de um paciente. O problema é combatido através do uso de ressuscitação cardiopulmonar (RCP) e/ou a aplicação de vasopressores como a adrenalina.

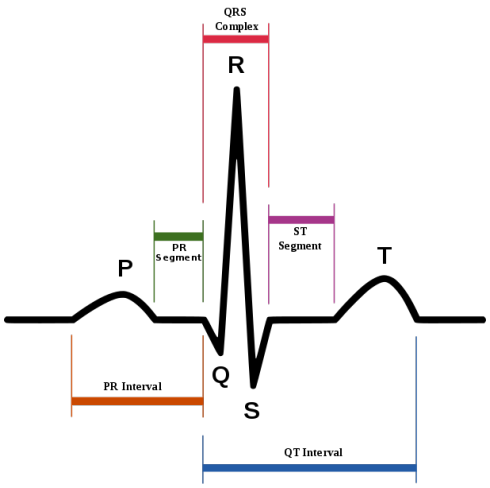


Figura 2.2: Representação de um batimento cardíaco em um ECG com destaque para suas formas de onda. O complexo QRS é destacado em vermelho. Fonte: Anthony Atkielski, domínio público, acessado em Nov 2018, https://commons.wikimedia.org/w/index.php?curid=1560893

2.1.1.2 Bradicardia Extrema

Um paciente é dito com bradicardia (figura 2.1c) quando a velocidade dos seus batimentos é menor que 60 bpms [NHN12]. Para o caso extremo esse valor deve ser menor que 50 bpms [GAG+13].

Cansaço, fadiga, confusão, tontura, baixa na pressão, entre outros são os sintomas mais comuns em uma bradicardia normal devido a falta de oxigenação do cérebro. Em casos mais extremos pode levar a desmaios e até a uma parada cardíaca [hea].

2.1.1.3 Taquicardia Extrema e Ventricular

A figura 2.1d ilustra um exemplo de taquicardia comum. Esse problema é caracterizado por um ritmo cardíaco acima de 100 bpms [NHN12]. No caso extremo o ritmo passa de 140 bpms [GAG+13].

A taquicardia ventricular também é caracterizada pelo ritmo acelerado, porém originados das câmaras mais baixas do coração resultando em um formato de onda 2.1e. Essa sintoma impede que as câmaras do coração se encham completamente entre os batimentos, comprometendo o fluxo de sangue [hea].

Os sintomas causados por esse problema são similares aos de bradicardia. Em casos mais severos, desmaios, falta de ar e náuseas são comuns. Já no caso ventricular, a exposição por um longo período de tempo pode evoluir para uma fibrilação ventricular [hea].

2.1.1.4 *Flutter*/Fibrilação ventricular

A fibrilação ventricular (figura 2.1f) é a desordem cardíaca mais grave entre as arritmias. Nesse problema, o coração ao invés de bombear o sangue normalmente, o coração treme em

várias partes. Esse comportamento pode levar o paciente a uma parada cardíaca súbita.

2.2 Aparelhos de Monitoramento Hospitalar

Leitos hospitalares de UTIs são repletos de equipamentos para monitoramento dos sinais vitais dos pacientes. O objetivo do uso desses equipamentos é alertar a equipe médica em caso de problemas de saúde dos pacientes monitorados. Em geral esse monitoramento é feito através de alertas sonoros.

As seções a seguir descrevem brevemente o funcionamento dos aparelhos de monitoramento hospitalar encontrados utilizados para criação do banco de dados e seus respectivos exames.

2.2.1 Eletrocardiógrafo

Um eletrocardiógrafo (figura 2.3) é um equipamento que captura as diferenças de potencial elétrico (chamados *leads*) do corpo através do uso de 10 eletrodos fixados no corpo do paciente. No total 12 *leads* são formados através da disposição correta dos eletrodos como mostrado na figura 2.4.

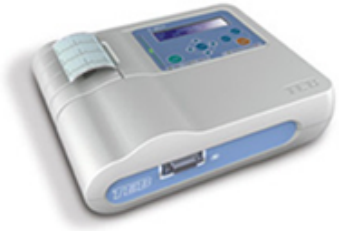
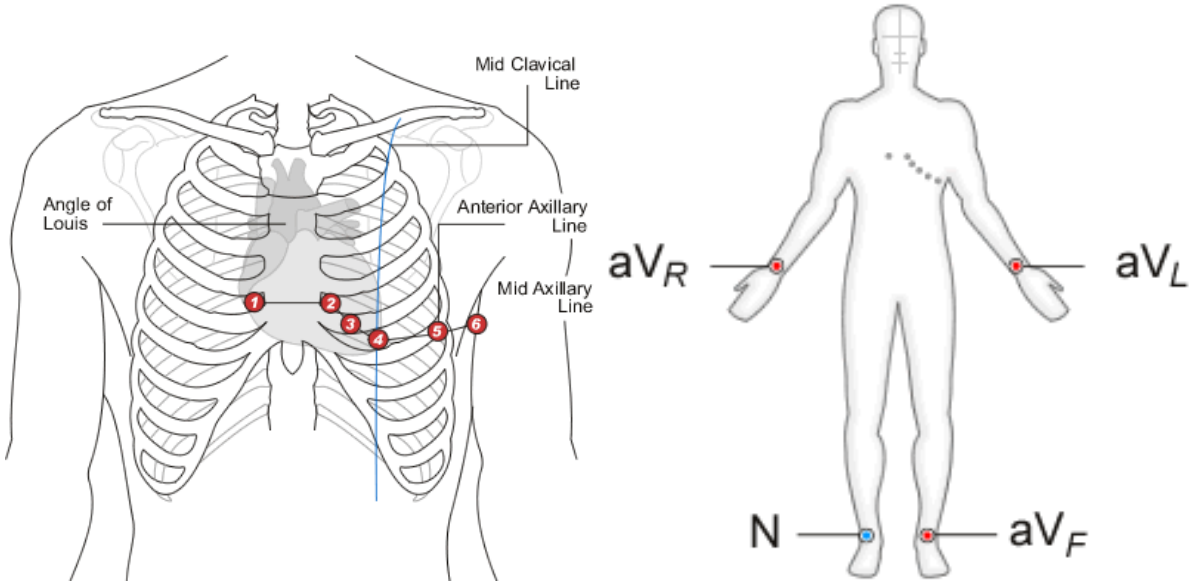


Figura 2.3: Exemplo de um eletrocardiógrafo comumente encontrado em leitos de UTIs. Fonte: CardioNetworks, CC BY-SA 3.0, Acessado em Nov 2018, https://commons.wikimedia.org/w/index.php?curid=24370995.

O resultado da aplicação desse equipamento são os eletrocardiogramas (ECG). Eles consistem em um gráfico de linha disposto em uma grade de células quadradas de 1mm de lado. No eixo x cada unidade representa um tempo de 0.04 segundos e no eixo y a mesma unidade representa uma tensão de 0.1 mV [Bec06]. A figura 2.5 ilustra essa configuração.

Para entender como interpretar um eletrocardiograma, é necessário entender como ele funciona. Quando o coração se contrai ou se expande ele gera correntes elétricas no corpo que são captados pelos eletrodos. Juntando essas diferenças de potenciais captadas em todos os eletrodos temos então um padrão como o mostrado na figura 2.2. Esse padrão é divido em várias partes onde cada um representa algum estágio do bombeamento do coração. O objetivo da equipe médica então é de analisar o quanto o ECG do paciente difere desse padrão normal (2.1a) e qual a abordagem necessária para ser tomada. Por exemplo, para medir o ritmo cardíaco do paciente, mede-se a distancia entre os complexos QRS e divide o valor por 300 para termos os resultados em segundos [Bec06].



(a) Eletrodos Precordiais (b) Eletrodos Periféricos

Figura 2.4: Localização correta de aplicação dos eletrodos de um eletrocardiograma. Fonte: University of Nottingham, School of Health Sciences, Acessado em Nov 2018, https://www.nottingham.ac.uk/nursing/practice/resources/cardiology/function/placement_of_leads.php

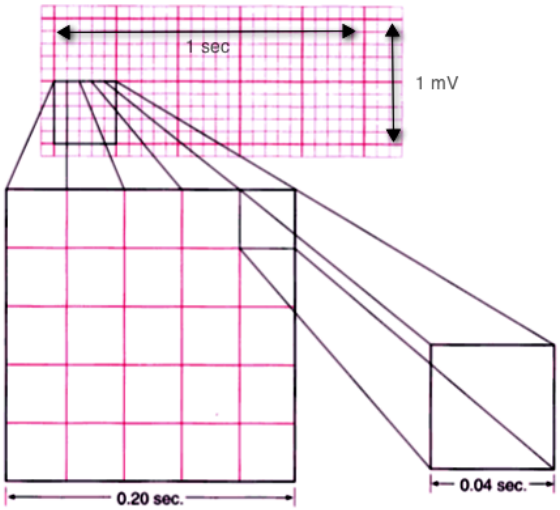


Figura 2.5: Exemplo de configuração da saída de um eletrocardiograma. Na figura a menor subdivisão de célula de dimensões 1mm x 1mm representam 0.04s no eixo do tempo X e 0.1 no eixo de tensão Y. As maiores subdivisões são compostas de 5 células em cada lado formando assim uma grade de 0.2s x 0.5mV. Fonte: Practical Clinical Skills, Acessado em Nov 2018, https://www.practicalclinicalskills.com/ekg-lesson?coursecaseorder=1&courseid=301.

2.2.2 Oxímetro de Pulso

Um oxímetro de pulso (figura 2.6) é um equipamento que consiste em 2 diodos emissores de luz de comprimento de onda 660nm (vermelho) e 940nm (infravermelho) e fotodiodos aplicados em uma parte translúcida do corpo como orelha ou ponta dos dedos. Esse equipamento, a partir da diferença de luminosidade entre os diodos emissores e receptores, consegue captar a saturação de oxigênio (i.e. fração da hemoglobina saturada por oxigênio em relação ao valor total) no eixo Y ao longo do tempo no eixo X. O resultado é o chamado fotopetismografia (PPG).[SS01]

Figura 2.6: Exemplo de um oxímetro de pulso. O nome pulso é dado pela medição do pulso sanguíneo e não da localização do equipamento. Fonte: UusiAjaja, CC0, Acessado em Nov 2018, https://commons.wikimedia.org/w/index.php?curid=15840996

Segundo Shelley [SS01] existem várias aplicações para esse tipo de exame, entre eles medição de tônus muscular, variabilidade respiratória e pressão sanguínea. Nesse projeto o oxímetro de pulso foi utilizado no monitoramento de ritmo cardíaco dos pacientes. Ainda de acordo com o trabalho citado, esse monitoramento em conjunto com os eletrocardiogramas podem melhorar muito na análise de ritmo cardíaco dos pacientes em casos de movimento do paciente ou problemas elétricos. A figura 2.7 mostra um exemplo de uma onda de ECG normal e uma de PPG.

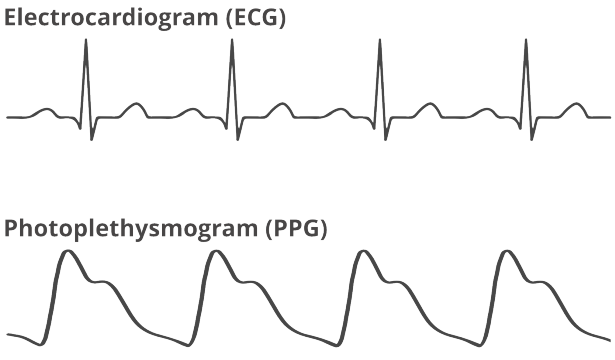


Figura 2.7: Comparação de um ritmo cardíaco normal medidos por um eletrocardiógrafo e um oxímetro de pulso. Fonte: Byteflies, Acessado em Nov 2018, https://www.byteflies.com/science.

2.2.3 Respirador

Além de auxiliar na respiração, um respirador tem como objetivo monitorar as atividades pulmonares do paciente através da aplicação de uma máscara. Assim como nos exames citados anteriormente, a interpretação desse tipo de equipamento é baseado em análise de padrões. Esse exame 2.8 resulta no monitoramento de pressão (cmH_2O), fluxo (L/min) e volume (mL) no eixo y de acordo com sua variação no tempo no eixo x.

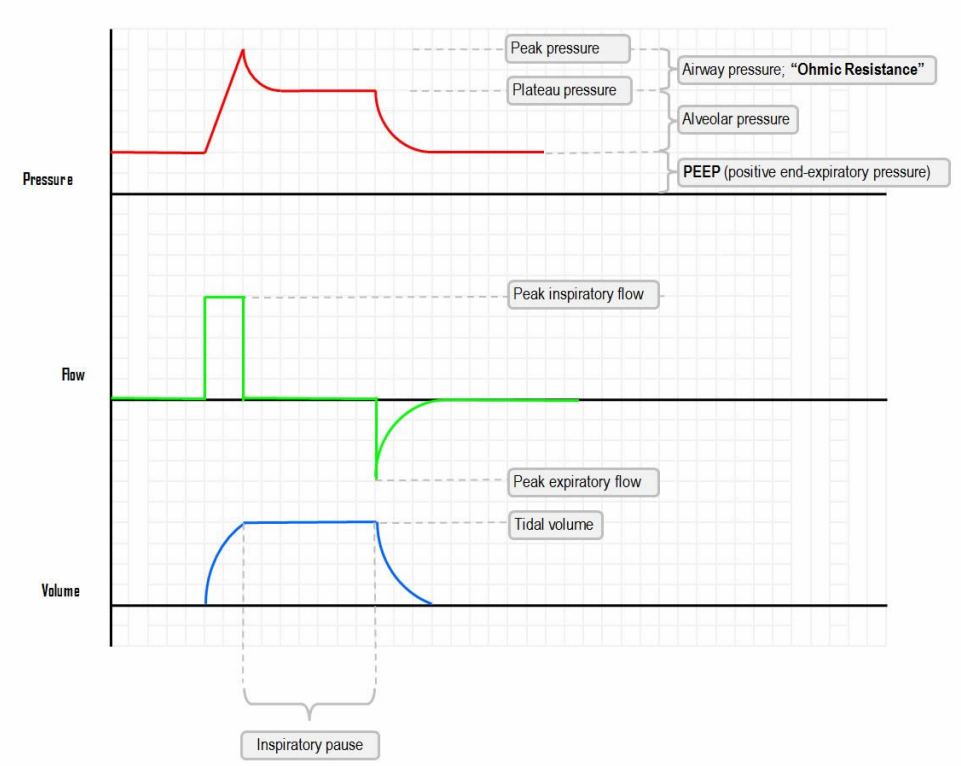


Figura 2.8: Exemplo simplificado de formas de onda medidas através de um respirador. De cima para baixo as ondas são de pressão, fluxo e volume em um paciente saudável. Fonte: Alex Yartsev, Deranged Physiology, acessado em Nov 2018, https://derangedphysiology.com/main/required-reading/respiratory-medicine-and-ventilation/Chapter%202.5.2/analysis-ventilator-waveforms.

2.2.4 Transdutor de Pressão

O transdutor de pressão arterial é um dispositivo utilizado em conjunto com um cateter e uma coluna líquida para monitoramento contínuo da pressão sanguínea em um paciente. A forma de onda (figura 2.9) gerada pelo equipamento é a pressão sanguínea arterial medidas em mmHg (eixo y) e tempo em segundos (eixo x).

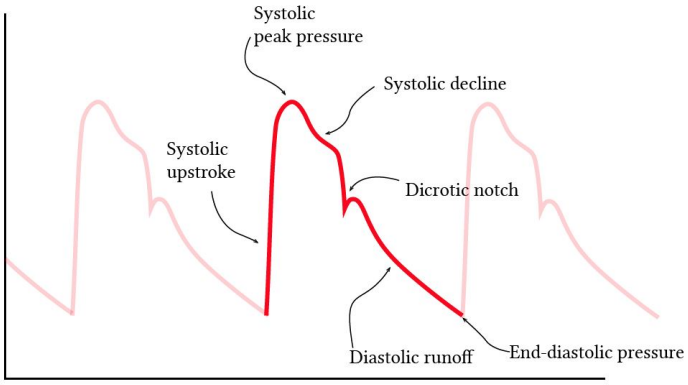
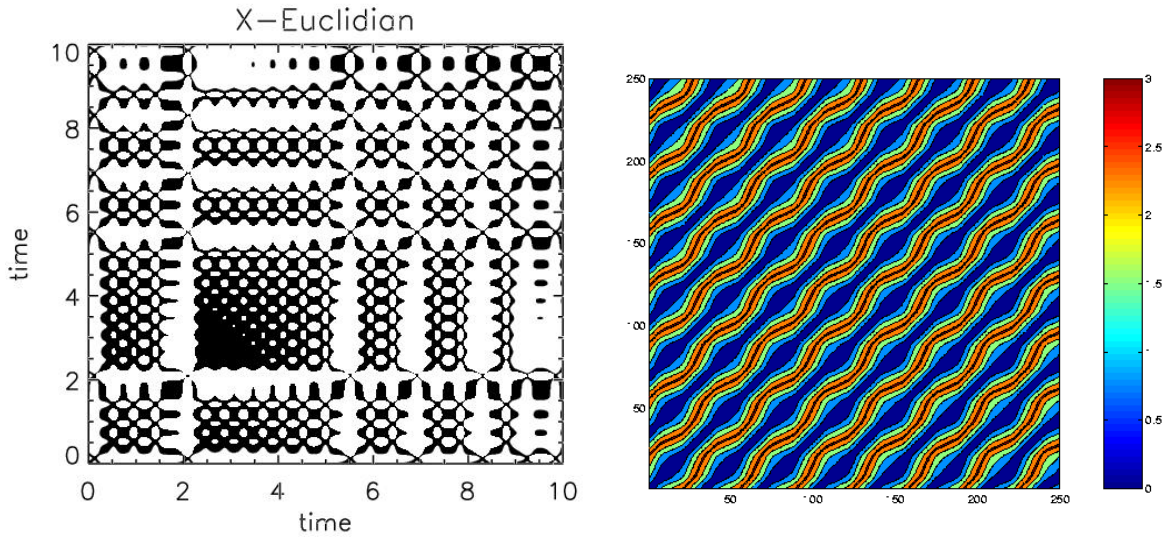


Figura 2.9: Exemplo de forma de onda encontrada na aferição de pressão sanguínea arterial. A parte em destaque representa um pulso. Fonte: Alex Yartsev, Deranged Physiology, acessado em Nov 2018, https://derangedphysiology.com/main/cicm-primary-exam/required-reading/cardiovascular-system/Chapter%207.6.0/normal-arterial-line-waveforms

2.3 Recurrence Plots

Um *recurrence plot* é uma forma de visualização da periodicidade de séries temporais introduzidos por Eckmann [EKR87] em 1987. As figuras 2.10b e 2.10a representam respectivamente, um *recurrence plot* colorido e um preto e branco.



(a) (b)

Figura 2.10: Exemplos de *recurrence plots*. Em (a) cada ponto representa a repetição da trajetória em uma série temporal. Em (b) a intensidade da cor determina a distância euclideana entre os pontos da trajetória. Fonte: Department of Computer Science, University of Colorado, Acessado em Nov 2018, https://www.cs.colorado.edu/~lizb/rps.html.

Seja $\vec{x}(i)$ o i-ésimo ponto que descreve uma trajetória de um sistema dinâmico em um espaço de d-dimensões para $i = \{1, ..., N\}$. Um *recurrence plot* (equação 2.1) é uma matriz quadrada $N \times N$ onde um ponto é colocado em (i, j) sempre que $\vec{x}(i)$ é suficientemente perto de $\vec{x}(j)$ [EKR87].

$$R(i,j) = \begin{cases} 1, & \text{se} ||\vec{x}(i) = \vec{x}(j)|| \leq \varepsilon \\ 0, & \text{caso contrário} \end{cases} \tag{2.1}$$

A equação acima descreve um *recurrence plot* em preto e branco como na figura 2.10a onde o ponto (i, j) é marcado quando $R(i, j) = 1$. No caso colorido, o valor de $R(i, j)$ é substituiodo pela distância euclideana entre os pontos da trajetória (i.e. valor de ε não se faz mais necessário).

A figura 2.11 ilustra o processo de construção desse tipo de visualização para uma série temporal de 12 pontos em 2 dimensões usando uma distância de 1 segundo. Na figura da esquerda estão destacados os pontos que irão definir a trajetória. Já na figura do meio cada ponto no espaço de fases é defido por $s_n = (x_n, x_{n+1})$ onde $x(n)$ representa o ponto onde x está no tempo n. A figura da direita é a matriz $R(i, j) = dist(s_i, s_j)$.

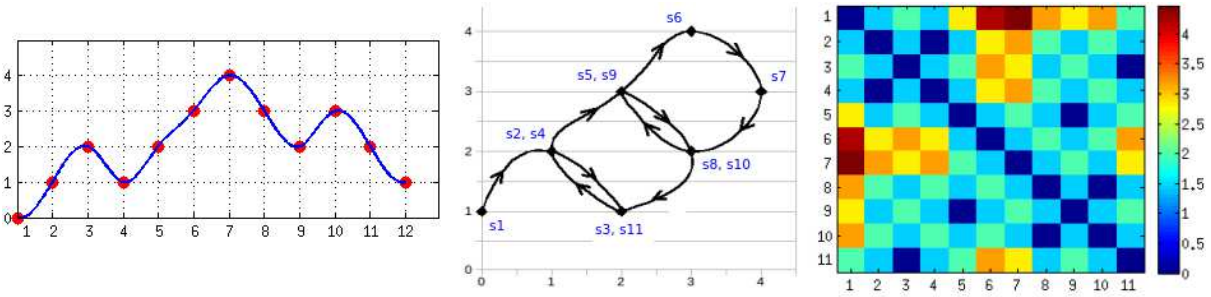


Figura 2.11: Exemplo de construção de um *recurrence plot* colorido (direita) a partir do espaço de fases (meio) gerado de acordo com a série temporal (esquerda). Fonte: Retirado de Hatami [HGD18]

2.4 Aprendizagem de Máquina

A aprendizagem de máquina é uma subárea da inteligência artificial que aprende o comportamento de um fenômeno observado através da utilização de dados. De acordo com Mitchell [Mit97] aprendizado no contexto computacional é (tradução livre):

> Um programa de computador é dito ter aprendido uma experiência E em respeito a uma classe de tarefas T e uma medida de performance P, se a sua performance em tarefas de T, medidas por P, melhoram com a experiência E.

Na definição acima $t \in T$ é uma função que modela o fenômeno, P é uma métrica que avalia quão boa é essa aproximação e E é o conjunto de dados utilizado no treinamento. Usando este trabalho novamente como exemplo, temos:

- **Tarefa (T):** Diminuir a quantidade de alarmes falsos em UTIs.
- **Métrica (P):** AUROC.
- **Experiência (E):** Sequência de imagens geradas a partir do monitoramento de pacientes em UTIs.

A aprendizagem de máquina pode ser dividida em 3 categorias principais: Aprendizagem supervisionada, não-supervisionada e semi-supervisionada. Em linhas gerais, na aprendizagem supervisionada há a presenta dos dados para treinamento e suas respectivas classificações. Na abordagem não-supervisionada (ou *clustering*) os dados não tem as suas classificações pré-definidas, o objetivo é tentar encontrar mais informações a respeito dos exemplos. A semi-supervisionada encontra-se entre as duas abordagens citadas anteriormente. Nela, somente alguns exemplos são categorizados previamente.

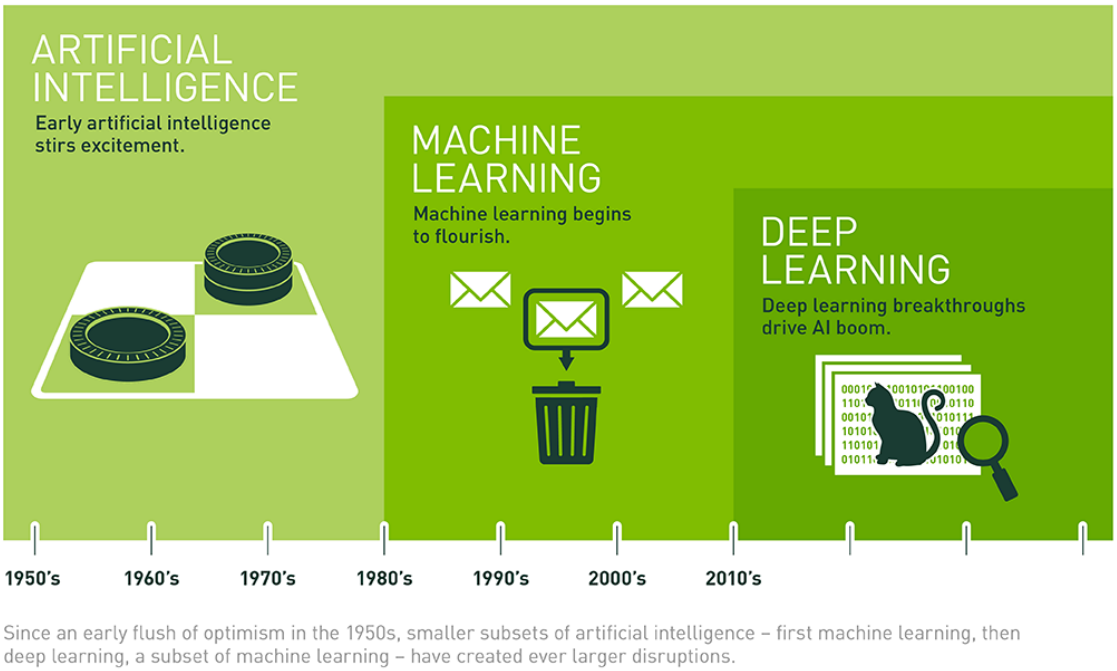


Figura 2.12: Relação das áreas de inteligência artificial, aprendizagem de máquina e *deep learning* em relação aos seus campos de estudo e tempo. Fonte: Michael Copeland, NVidia Blog, Acessado em Nov 2018, https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/.

2.4.1 Aprendizagem Supervisionada

Segundo Russel [RN16] a tarefa de um algoritmo de aprendizagem supervisionada é:

> Dado um conjunto de treinamento formado por N exemplos de tuplas entrada-saída (x_1, y_1), (x_2, y_2), ..., (x_N, y_N). Onde cada y_j é gerado por uma função desconhecida $y = f(x)$. A tarefa da aprendizagem supervisionada é descobrir a função h que mais se aproxima da verdadeira função f.

Na definição data x e y podem ser qualquer valor (não necessariamente números). Quando y é um conjunto discreto finito o problema é chamado de classificação e quando é um valor contínuo é chamado de regressão. A função h é chamada de hipótese.

Utilizando a definição acima para esse projeto temos um conjunto com N=750 exemplos de tupla entrada-saída onde x_i são as imagens contendo os sinais e y_i suas respectivas classificações entre verdadeiros e falsos. f é a função que identifica todos os alarmes verdadeiros e falsos a partir de imagens no formato escolhido e h é o algoritmo usado e seus respectivos parâmetros.

2.5 Redes Neurais Artificiais

Uma rede neural artificial (ANN) é um algoritmo de aprendizagem de máquina baseado na estrutura do cérebro humano. A área de pesquisa se deu início a partir do surgimento do conceito de neurônios artificiais feito por McCulloch e Pitts em 1943 no artigo *A logical calculus of the ideas immanent in nervous activity*. De acordo com Haykin [HHHH09] uma rede neural é (tradução livre):

> Uma rede neural é um processador massivo distribuído paralelamente feito a partir de unidades de processamento simples que são naturalmente propensas a guardar conhecimento experimental e torná-lo disponível para uso. Ela se assemelha ao cérebro em dois aspectos:
>
> 1. O conhecimento é adquirido pela rede a partir do seu ambiente através de um processo de aprendizado.
> 2. Forças de conexão interneurais, conhecidos como pesos sinápticos são usados para guardar o conhecimento adquirido.

As seções a seguir tratarão de vários conceitos de redes neurais como *perceptrons*, *multilayer perceptrons*, funções de ativação, funções de perda, entre outros. Na seção seguinte (seção 2.6) esses conceitos serão aprofundados através da introdução de *deep learning*.

2.5.1 Perceptron

Estima-se que um cérebro humano é composto por aproximadamente 100 bilhões de neurônios. Essas células são especializadas pela passagem de informação entre cérebro e o resto do corpo de forma eficiente. Essa comunicação neuronal, de forma simplificada acontece através

de estímulos excitatórios e inibitórios ocorridos nos dendritos e corpo célular. Um neurônio fica ativo (i.e. envia um impulso elétrico) quando a resultante desses estímulos ultrapassa um valor pré-definido chamado (*threshold*) [Ins17].

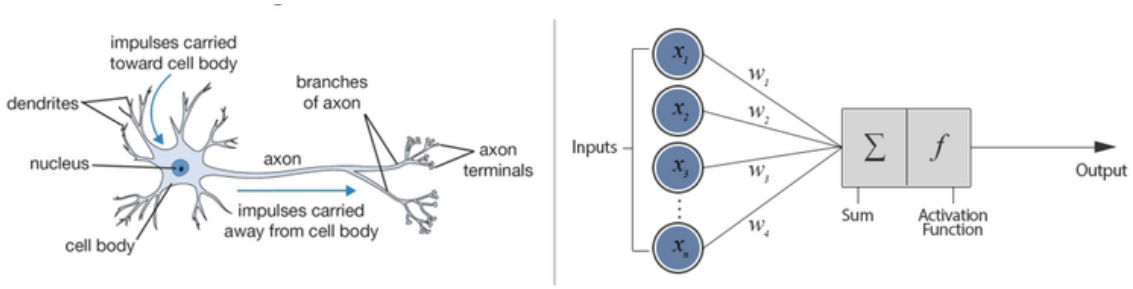


Figura 2.13: Neurônio real (esquerda) e artificial (direita). Em um neurônio artificial as entradas correspondem aos dendritos. Fonte: Jason Roell, Towards Data Science, Acessado em Nov 2018, https://towardsdatascience.com/from-fiction-to-reality-a-beginners-guide-to-artificial-neural-networks-d0411777571b.

Um neurônio artificial (ou perceptron) apresenta um comportamento análogo na transmissão de informação. Essa estrutura se baseia no modelo proposto por Rosenblat (1958). Como pode ser visto na figura 2.13 um neurônio artificial é composto por um conjunto de entradas $\vec{X} = [x_0, ..., x_N]$ e um conjunto de pesos $\vec{W} = [w_0, ..., w_N]$ e uma saída z (equação 2.2).

Na imagem são omitidos os valores de w_0 e x_0. Esses valores tem um valor pré-determinado tal que $w_0 = -1$ e $x_0 = b$ e representam a função de ativação f. O valor b é chamado *bias*. A equação 2.3 descreve o classificador linear proposto por Rosenblat, onde θ é o *threshold*.

$$z = \sum_{i=1}^{N} x_i \times w_i + b = \vec{X} \cdot \vec{W} + b \tag{2.2}$$

$$y = f(\vec{x}) = \begin{cases} 1, & z \geq \theta \\ 0, & \text{caso contrário} \end{cases} \tag{2.3}$$

Um problema do uso do modelo de Rosenblat é que devido a utilização de uma função degrau (equação 2.3) que é uma equação linear, problemas de natureza não-linear não conseguem ser resolvidas por esse modelo (e.g. função xor). Para se comportar melhor diante do mundo real é então proposta a inclusão de funções não-lineares. Esse tipo de função torna as entradas anteriormente discretas em contínuas. A figura 2.14 ilustra as duas funções de ativação utilizadas nesse projeto: *sigmoid* e *rectified linear unit (ReLU)*.

A função *sigmoid* (equação 2.4) é uma função contínua em formato de "s"muito utilizada quando se deseja prever probabilidades devido à sua característica de estar compreendida no intervalo $[0, 1]$. A principal desvantagem dessa função é que para valores maiores que 1 ou menores que 0 as saídas são reduzidas para esses extremos, ou seja, há perda de informação.

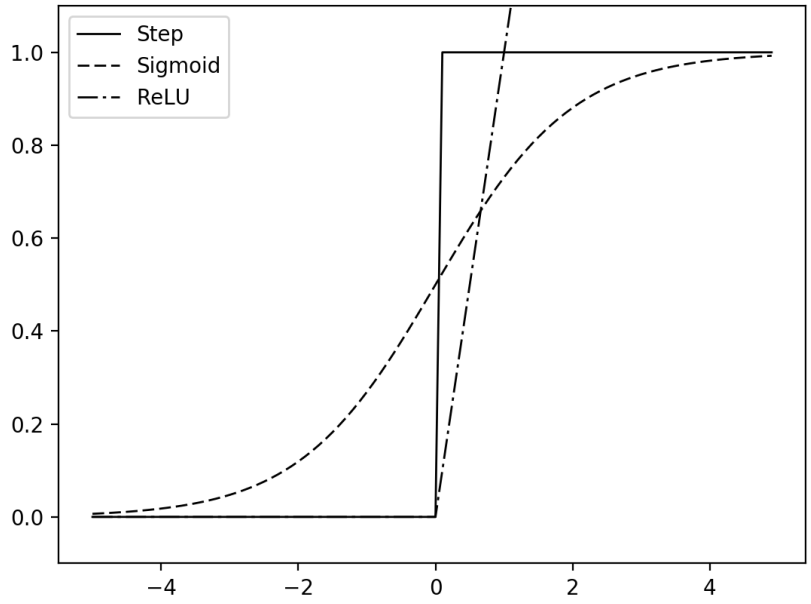


Figura 2.14: Compararação entre as funções degrau (*step*), *sigmoid* e *ReLU*) Fonte: Kyohei Sahara, Thinkage, Acessado em Nov 2018, https://schwalbe10.github.io/thinkage/2017/02/12/activation.html.

$$f(x) = \frac{1}{1+e^{-z}} \tag{2.4}$$

A ReLU (equação 2.5) é uma das funções mais utilizadas em redes neurais. Ela consiste em uma função cujo valor é igual ao valor z exceto quando menor ou igual a 0. Nessa função então não há a perda de informação como na *sigmoid*. Outra vantagem é a velocidade em se computar seu valor.

$$f(x) = \max(0,x) \tag{2.5}$$

2.5.2 Multilayer Perceptron (MLP)

Mesmo com adição de não-linearidade em um modelo perceptron ainda há a dificuldade de modelar problemas mais complexos. Assim como nosso cérebro é formado por bilhões de neurônios, os *multilayer perceptrons* (figura 2.15) são formados pela combinação linear de vários perceptrons dispostos em camadas.

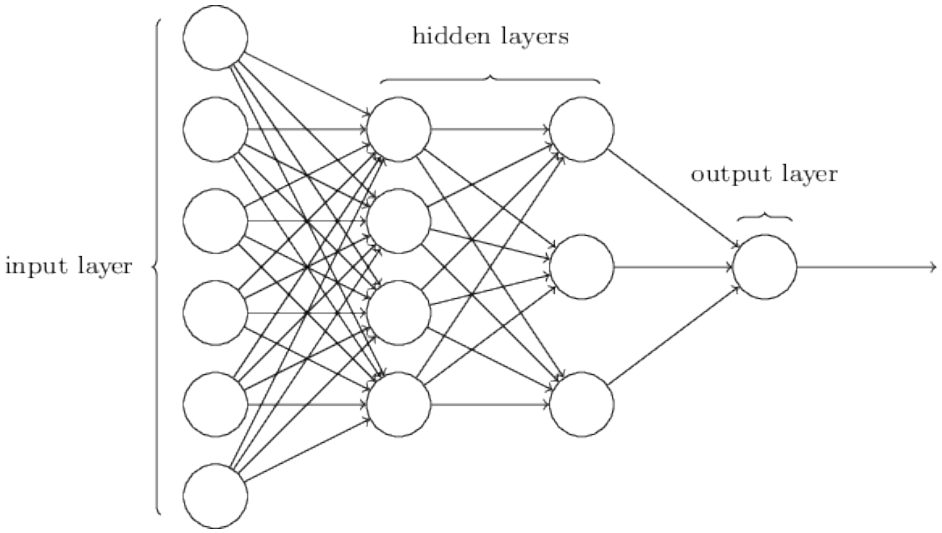


Figura 2.15: Representação de um *Multilayer Perceptron Feedfoward* contendo 6 neurônios na sua camada de entrada, 2 camadas intermediárias com respectivamente 4 e 3 neurônios e uma camada de saída com 1 neurônio. Fonte: Raymundo Cassani, Acessado em Nov 2018, https://github.com/rcassani/mlp-example.

A forma mais comum de um MLP é a rede *feedfoward*. Nesse tipo de rede a camada de entrada recebe o conjunto de dados como entrada (como estímulos do cérebro) e retorna um conjunto de saída que irá alimentar a próxima camada (i.e. camada intermediária). Cada camada intermediária pode alimentar a próxima até que se chegue na última (i.e camada de saída) que é responsável pela classificação das entradas de acordo com suas classes.

Assim como todo modelo de aprendizagem de máquina o objetivo é encontrar a função que melhor descreve o fenômeno. Para esse treinamento um dos algoritmos mais famosos é o gradiente descendente (ou *backpropagation*). A utilização desse algoritmo permite que as camadas que estão mais distantes da saída consigam receber um estímulo inibitório, podendo assim fazer a correção dos seus pesos e se adaptando a cada novo exemplo de treinamento.

Para efetuar a correção é necessário saber o grau de erro das saídas de cada um dos perceptrons. A função de perda tem como objetivo quantificar esse erro. A equação 2.6 define a função *binary cross-entropy* utilizada neste trabalho. Na equação $a = f(z)$ onde f é a função de ativação e z é definida na equação 2.2.

$$C = -\frac{1}{n} \times \sum_{x} [y \ln a + (1-y) \ln (1-a)] \tag{2.6}$$

2.6 Deep Learning

Como explicado anteriormente, a fundação básica da aprendizagem de máquina é extrair conhecimento através do uso de dados adquiridos através da observação de um fenômeno. A performance desse tipo de abordagem depende muito da forma em que esses dados serão

representados. Essa representação geralmente se dá através de características das entradas [GBCB16]. Por exemplo, para resolver um problema de fraude bancária pode-se usar como caraterísticas o saldo bancário do cliente e as categorias que ele mais consome.

Porém alguns problemas são complexos o bastante para que essa representação através de características não sejam tão simples ou viáveis. Por exemplo, em um problema de identificar caracteres escritos a mão, extrair características que descrevem o fenômeno é altamente complexo pois cada pessoa tem sua grafia e sua forma de escrever certos caracteres.

Para solucionar esses problemas uma abordagem utilizada é a de aprendizagem por representação. Nesse tipo de abordagem o algoritmo não só aprende o mapeamento entre entrada e saída, mas aprende também a melhor forma de representar a entrada.

Contudo, mesmo essa representação pode ser influenciada por diversos fatores (e.g. incidência de luz em uma iagem). Esse problema é resolvido através do uso de *deep learning*. Nessa abordagem, representações mais complexas são expressas em função de representações mais simples (e.g. bordas, cor e curvas em imagens). De forma bastante simplificada uma rede de *deep learning* é um MLP com inúmeras camadas intermediárias.

2.6.1 Redes Neurais Convolucionais

Uma rede neural convolucional (CNN ou ConvNet) é um conceito introduzido por Yann LeCun em 1989 no artigo *Backpropagation Applied to Handwritten Zip Code Recognition* (figura 2.16). Essa arquitetura é um modelo especializado em dados com uma topologia em grade (e.g. séries temporais e imagens).

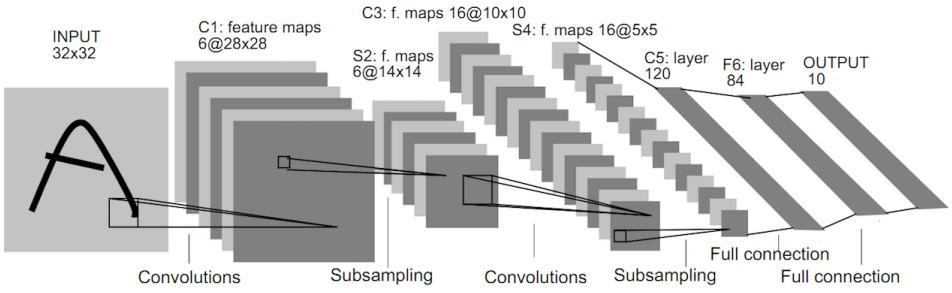


Figura 2.16: Representação da arquitetura LeNet proposta por LeCun em 1989. Fonte: Retirado do artigo original *Backpropagation Applied to Handwritten Zip Code Recognition*

Assim como muitos conceitos de redes neurais essa arquitetura também tem uma inspiração biológica: a visão. De forma simplificada no sistema visual dos mamíferos após a imagem ser formada na retina ela passa por diversas estruturas até chegar no córtex visual primário (V1) localizado atrás da cabeça. Essa região é a responsável pela primeira parte do processamento significativo das imagens. Para esse fim essa região é formada por várias células simples dispostas em uma estrutura bidimensional onde cada uma delas responde somente a estímulos gerados na sua região (e.g. Presença de um círculo no canto superior esquerdo). Além disso apresenta células mais complexas que além de responder aos estímulos locais tem uma propriedade de invariância a pequenas transições de posição das imagens. Acredita-se que essa

estrutura vá se repetindo em todos os estágios posteriores da visão onde respondem á estímulos cada vez mais específicos e são cada vez mais invariantes a transformações das imagens [GBCB16].

Nas CNNs o conceito é parecido porém com as devidas limitações de capacidade do cérebro humano. Essa arquitetura é forma por um conjunto de neurônios arranjados em 3 dimensões (largura, altura e profundidade) arranjadas em 4 tipos de camadas: Camada convolucional, não linear, *pooling* e camada totalmente conectada.

2.6.1.1 Camada Convolucional

Como dito anteriormente, na região do córtex visual primário existem células em uma estrutura bidimensional que respondem a estímulos somente de uma parte da imagem recebida. Nas redes neurais convolucionais esse é o papel da camada convolucional.

Para simular a visão a camada convolucional 2.17 é formada por um conjunto de filtros (ou *kernels*) com dimensões menores que as da entrada. Esses filtros deslizam em um número fixo de unidade sobre toda entrada calculando o produto escalar entre seus valores e a entrada. Essa operação é chamada convolução e o seu resultado é chamado *activation map*.

A figura 2.17 mostra o exemplo de uma operação de convolução com filtros 3x3x3 em uma imagem 8x8x3. Cada aplicação de filtro em uma posição da imagem gera um único valor. Note que em relação as dimensões originais, o *activation map* resultante tem dimensões menores.

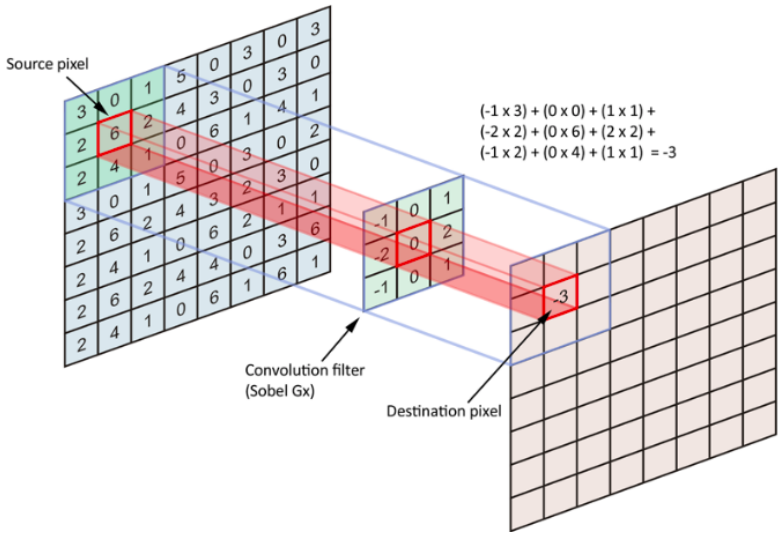


Figura 2.17: Exemplo de aplicação de um filtro 3x3x3 em uma imagem de dimensões 8x8x3. Fonte: Daphne Cornelisse, Freecodecamp, Acessado em Nov 2018, https://medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050.

Algumas vezes, reduzir as entradas muito rapidamente podem dificultar uma boa performance no modelo. Note que as dimensões dos *activation maps* são totalmente determinados pelo valor do *stride*. Para controlar o encolhimento das entradas é utilizada a técnica de *zero-padding*. Nela são adiconados zeros nas bordas dsa imagens, tornando-as maiores e permitindo um melhor controle das dimensões.

2.6.1.2 Não Linear

Como explicado na seção 2.5.1 a grande maioria dos problemas do mundo real tem natureza não-linear. O objetivo dessa camada é adicionar esse comportamento ao modelo treinado 2.18. A função mais utilizada aqui é também a *Rectified Linear Unit (ReLU)*.

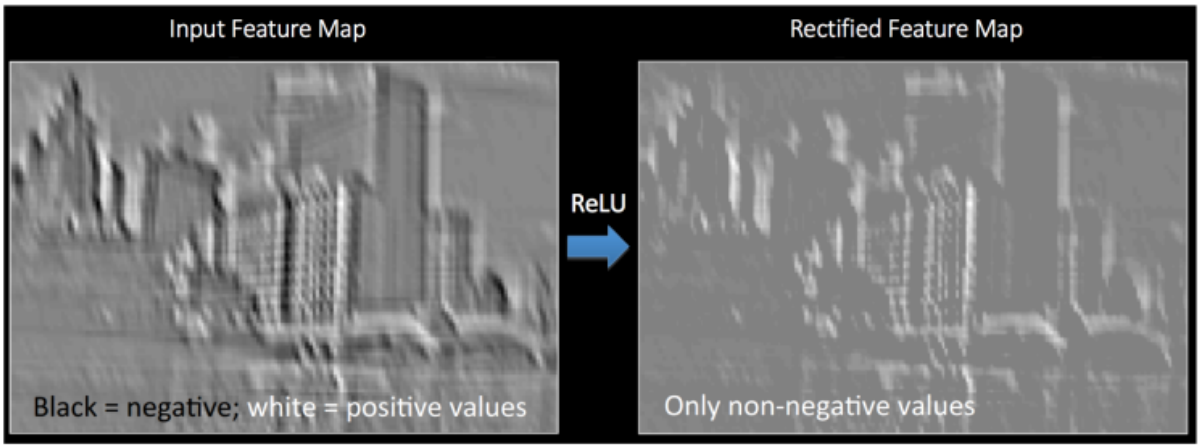


Figura 2.18: Resultado da aplicação da camada ReLU em um mapa de ativação. Note que os valores negativos (pretos) foram todos substituidos por números não negativos (brancos). Fonte: Rob Fergus, New York University, Acessado em Nov 2018, http://mlss.tuebingen.mpg.de/2015/slides/fergus/Fergus_1.pdf.

2.6.1.3 Pooling

Como mencionado anteriormente, na região do córtex visual primário há células mais complexas que recebem estímulos visuais e são invariantes a pequenas transições de posição das imagens. Nas redes neurais convolucionais esse é o objetivo da camada de *pooling*.

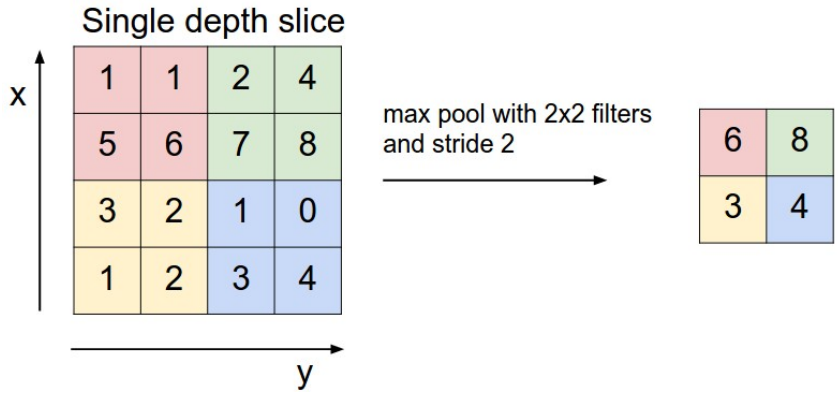


Figura 2.19: Aplicação da operação *max-pooling* em uma matriz. Fonte: Rob Fergus, New York University, Acessado em Nov 2018, CS231n, University of Stanford, Acessado em Nov 2018, http://cs231n.github.io/convolutional-networks/.

Assim como as camadas convolucionais, essa camada também apresenta filtros que deslizam sobre uma entrada de maior dimensão em um número fixo de unidades. Porém ao invés de fazer o produto interno essa camada aplica outra função. O caso mais comum é o *max-pooling* que tem como resultado o maior valor encontrado no filtro.

A consequência dessa operação é tornar as entradas invariantes com qualquer mudança da imagem (e.g. imagems espelhadas e invertidas). Além disso ela reduz as dimensões da entrada e consequentemente a quantidade de parâmetros, o custo computacional e o *overfitting*.

2.6.1.4 Camada Totalmente Conectada

Todas as camadas anteriormente explicadas são responsáveis pela criação de uma representação das entradas em uma série de características criadas de forma automatizada. A camada totalmente conectada é um *multilayer perceptron* responsável por receber a entrada gerada pelas camadas anteriores e as classificá-las de acordo com as classes pré-determinada.

CAPÍTULO 3

Metodologia

Esse capítulo descreve os passos utilizados para a execução do experimento bem como as fundamentações teóricas necessárias para seu entendimento. Na primeira seção será feita apresentação do banco de dados. Em seguida serão explicadas a seleção, treinamento e avaliação dos modelos. E por fim o ambiente em que o experimento foi executado.

A estrutura deste capítulo se dará da seguinte forma: No início de cada seção serão explicadas a metodologia utilizada no experimento de acordo com o tópico abordado. As subseções tem por objetivo explicar os conceitos necessários.

3.1 Banco de Dados

Para a realização do experimento foi utilizado o banco de dados *Reducing False Arrhythmia Alarms in the ICU: the PhysioNet/Computing in Cardiology Challenge 2015* da Physionet [GAG+13]. A Physionet é um website de acesso livre gerenciado por membros do *MIT's Lab for Computational Physiology*. Nele estão contidos grandes repositórios de sinais fisiológicos e *softwares open-source* relacionados. O banco utilizado faz parte de um desafio anual ocorrido em 2015 com o intuito de reduzir os alarmes falsos em arritmias cardíacas dentro de UTIs.

O banco consiste de 750 gravações de até 4 tipos diferentes de monitores hospitalares utilizados para monitoramento de pacientes, são eles: Eletrocardiógrafo, oxímetro de pulso, respirador e transdutor de pressão arterial. Cada gravação pode conter até 4 sinais sendo 3 obrigatórios e 1 opcional. Nos obrigatórios temos 2 eletrodos diferentes de um eletrocardiógrafo que não necessariamente geraram o alarme. Os outros dois sinais podem ser de qualquer um dos outros monitores.

Os alarmes podem ser resultantes de 5 tipos de arritmias cardíacas graves, são elas: Assistolia, bradicardia extrema, taquicardia extrema, taquicardia ventricular e fibrilação ventricular. Mais informações sobre esses problemas podem ser encontrados no capítulo 2. Exemplos das gravações podem ser vistos na figura 3.1.

O desafio divide os dados em dois tipos de problema. No problema do tipo *"real-time"* as gravações tem 5 minutos de duração. Já no *"retrospective"* as gravações contém 30 segundos a mais. É importante ressaltar que para respeitar as normas americanas de equipamentos hospitalares (ANSI/AAMI EC13:2002) a janela de início do alarme é entre 4:50min e 5:00min independente do tipo de problema.

De forma a diminuir o viés dos dados, as gravações são feitas em 4 hospitais escolhidos aleatoriamente dos Estados Unidos e da Europa. Além disso, são utilizados equipamentos dos 3 maiores fabricantes de equipamentos de monitoramento hospitalar. Além disso, a classificação de um sinal em alarme falso ou verdadeiro necessita de aprovação de pelo menos 2 de 3 especialistas. Finalmente, gravações que vem do mesmo paciente não ultrapassam mais do que 3 tipos diferentes de alarme obrigatoriamente separados por pelo menos 5 minutos entre eles.

(A)

(B)

Figura 3.1: Exemplos de gravações contidos no banco de dados na janela entre 4:30min e 5:00min. Na figura (A) é possível observar apenas um sinal extra de pletismograma (PLETH). Já a figura (B) apresenta 2 sinais extras: um de pressão arterial (ABP) e um respiratório (RESP). Fonte: *Physionet's LightWAVE Tool*

.

3.2 Treinamento

Para diminuir o tempo e os recursos necessários para o treinamento de uma arquitetura de *deep learning* o experimento foi baseado no paradigma de *transfer learning*. A idéia é usar a arquitetura somente como um extrator de características das imagens. Para isso foram geradas imagens a partir das gravações que são dadas como entrada em uma rede VGG16 [SZ14a] sem a camada totalmente conectada e carregada com os pesos originais que ganharam o desafio da ImageNet. A saída dessa rede (chamadas *bottleneck features*) são usadas para treinar um *multilayer perceptron* com as configurações abaixo:

- **Otimizador**: *Stochastic Gradient Descendent (SGD)*
- **Função de perda**: *Binary Cross Entropy*
- **Função de ativação**:
  - **Camadas Intermediárias**: *Rectified Linear Unit (ReLU)*
  - **Camada de Saída**: *Sigmoid*

Para a busca dos hiperparâmetros foi feito um *Grid Search* em 3 etapas utilizando os valores descritos na tabela 3.1. Na primeira parte são variados a quantidade de camadas intermediárias e o dropout. Na segunda é a vez da taxa de aprendizado e tamanho do batch. Por fim é variado o momentum e seu tipo (Nesterov ou não).

Tabela 3.1: Variação dos hiperparâmetros utilizados no *grid search*

Hyperparameters	Values
Hidden Layers	[128], [256], [128, 128], [128, 256], [256, 128], [256, 256][1]
Dropout	0.2, 0.35, 0.5
Batch Size	8, 16, 32
Learning Rate	0.0001, 0.001, 0.025, 0.01
Momentum	0.5, 0.7, 0.9
Nesterov	True, False

[1]Valores entre "["e "]"representam a configuração das camadas intermediárias. (e.g. [128, 256] significa uma arquitetura com uma camada de 128 neurônios seguida de uma com 256)

3.2.1 Transfer Learning

A complexidade no treinamento de uma arquitetura de *deep learning* é muito alta em relação à recursos computacionais e tempo. Para dar um exemplo, o treinamento da arquitetura VGG16 para partipação no desafio da ImageNet em 2014 teve uma duração de 2 - 3 semanas usando 4 GPUs NVidia Titan Black [SZ14a]. Além disso, treinar esse tipo de arquitetura requer

uma massiva quantidade de dados classificados previamente o que muitas vezes não é viável de se coletar. Esses problemas se agravam ainda mais quando a distribuição dos dados de um problema mudam com o tempo. Nesse caso é necessário coletar mais dados e retreinar todo o modelo desde o início [PY+10]. Para solucionar esses e outros problemas surge a idéia de *transfer learning*.

A técnica consiste em aplicar o conhecimento adquirido na resolução de um ou mais problemas para resolver ou melhorar outro problema relacionado [TS10]. Comportamento que pode ser observado também em humanos: Quanto mais próxima uma tarefa passada é de outra, tendemos a resolvê-las de forma parecida.

A forma de aplicação de *transfer learning* depende de acordo com a proximidade dos problemas a serem resolvidos e a quantidade de dados disponíveis. Lembrando que arquiteturas de *deep learning* tem a capacidade de aprender características mais genéricas dos dados nas suas camadas iniciais e nas camadas finais a tendência é de se obter características mais particulares do domínio [YCBL14]. Segundo Chollet [C+15] existem 3 abordagens principais:

- **Treinar um modelo desde o início**: Essa abordagem é interessante quando se tem uma grande quantidade de dados e os problemas são diferentes. Isso porque, mesmo as características mais genéricas seriam voltadas ao problema em questão.
- **Treinar somente algumas camadas**: Nesse caso é assume-se uma certa falta de similaridade com o problema a ser resolvido. A quantidade de camadas a serem alteradas depende da quantidade de dados disponíveis.
- **Usar a arquitetura fonte como extrator de características**: Essa abordagem deve ser preferencialmente utilizada caso os problemas sejam parecidos e não se tenham muitos dados disponíveis.

Nesse projeto a arquitetura foi utilizada como extrator de características, pois apesar dos problemas não serem parecidos a pouca quantidade de dados poderia não permitir um treinamento efetivo de tantas camadas.

3.2.2 VGGNet

A VGG [SZ14a] é uma arquitetura de rede neural convolucional criada por cientistas da Oxford Visual Geometry Group (VGG) e introduzida em Simonyan [SZ14b] após vencer na competição ImageNet Large Scale Visual Recognition Challenge do ano de 2014 [RDS+15]. Essa arquitetura pode ser encontrada de duas formas: VGG16 e VGG19 onde os números se referem a quantidade de camadas com pesos da arquitetura.

Na figura 3.3 o modelo é divido por vários blocos formados por filtros pequenos (3x3 com *padding* 1) empilhados. O *pooling* é feito através de filtros de *max pooling* 2x2 com *padding* de 2. Além disso, é formada por duas camadas totalmente conectadas de 4096 nós seguidas de um classificador *softmax*.

O aparecimento desse modelo foi muito importante para a academia devido á sua arquitetura formada por blocos uniformes que traz simplicidade ao modelo. Além da utilização de uma maior quantidade de filtros pequenos em detrimento de poucos filtros mais largos que mudou o rumo das pesquisas após a sua publicação.

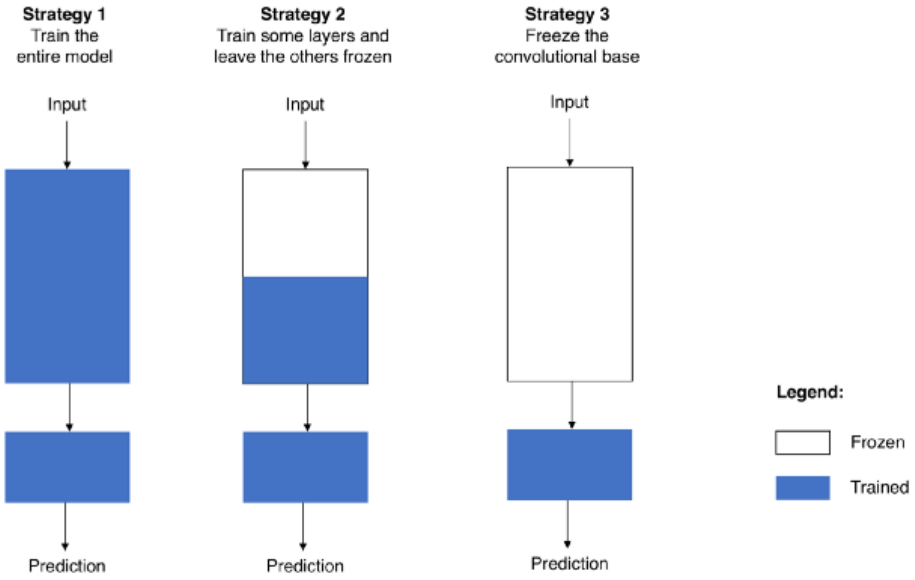


Figura 3.2: Abordagens comuns na aplicação de *transfer learning*. Na imagem os retângulos maiores representam a parte convolucional de um modelo de CNN e os menores representam a camada totalmente conectada do mesmo modelo. A cor azul representa a profundidade do modelo que deve ser treinada e a branca os pesos que estão congelados. Fonte: Pedro Marcelino, Towards Data Science, Acessado em Nov 2018, https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751

3.2.3 Grid Search

Grid Search é uma técnica de otimização de hiperparâmetros baseado em força bruta. O objetivo da técnica é dentro de um conjunto de possíveis valores pré-definidos de hiperparâmetros. Encontrar a combinação que maximize uma determinada métrica de performance.

Como exemplo, suponha uma rede neural onde se quer variar os parâmetros de taxa de aprendizado $LR = \{0.2, 0.4\}$ e tamanho do *batch* $BS = \{8, 16\}$ de acordo com a acurácia. Sendo P o conjunto de possibilidades definida pela tupla (l, b) onde $l \in LR$ e $b \in BS$ temos $P = \{(0.2, 8); (0.4, 8); (0.2, 16); (0.4, 16)\}$. O objetivo da técnica é encontrar qual o elemento em P que retorne o maior valor de acurácia.

3.3 Avaliação e seleção de modelos

Para obter uma estimativa mais robusta da performance dos modelos foi utilizada uma validação cruzada estratificada *5-Fold*. A decisão de reduzir o número de *folds* em relação ao número convencional *10-folds* foi feita devido a reduzida quantidade de dados para execução do experimento.

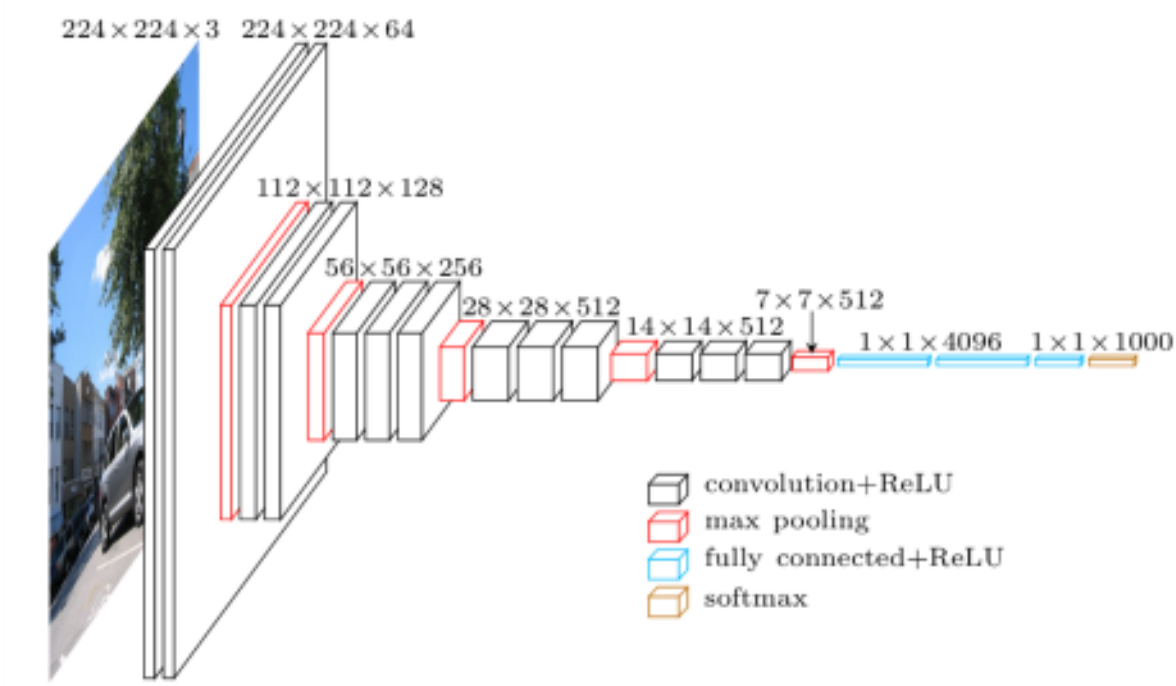


Figura 3.3: Modelo macro da arquitetura VGG19. Na figura as dimensões dos polígonos fazem referência as dimensões encontradas no modelo. Fonte: Leonard Blier, Heuritech Blog, Acessado em Nov 2018, https://blog.heuritech.com/2016/02/29/a-brief-report-of-the-heuritech-deep-learning-meetup-5/

A seleção dos modelos é feita buscando a maximização da AUROC (*Area Under Receive Operating Characteristic*). Porém outras métricas importantes também foram medidas durante o processo, são elas: *Precision*, *Recall*, *F1* e *Accuracy*.

3.3.1 Métricas

Nessa seção serão explicadas as métricas utilizadas na avaliação dos modelos deste trabalho. Para ilustrar as métricas de forma mais didática, dois exemplos serão utilizados. Para ambos suponha um classificador binário para detecção de arritmias cardíacas com uma base de dados de 100 gravações. Os exemplos são:

- **Exemplo 1**: Nesse exemplo temos uma base de dados balanceada com 50 exemplos de cada classe. Nesse caso, 70 exemplos são classificados corretamente (40 verdadeiros e 30 falsos) e 30 exemplos são classificados erroneamente (10 para os verdadeiros e 20 para os falsos)

- **Exemplo 2**: Nesse caso temos uma base de dados desbalanceada com 10 exemplos para a classe falsa e 90 para a classe verdadeira. Desses exemplos, 65 exemplos são classificados corretamente (60 para classe verdadeira e 5 para a classe falsa) e 35 exemplos classificados erroneamente (30 verdadeiros e 5 falsos).

3.3.1.1 Matriz de Confusão

A matriz de confusão é uma configuração de tabela que permite facilitar a visualização de performance de um modelo preditivo. Ela consiste em uma tabela $L \times L$ sendo L o quantidade de classes do problema. Suas células representam a quantidade de exemplos divididos de acordo com seus valores reais e classificados. A figura 3.4 mostra um exemplo de uma matriz de confusão 2×2 que será utilizada como base para explicação dos conceitos abaixo.

- **Verdadeiros Positivos (TP)**: Um exemplo é considerado verdadeiro positivo quando é previsto como verdadeiro e realmente é (e.g. Um paciente detectado com uma arritmia cardíaca que realmente teve).
- **Verdadeiros Negativo (TN)**: Um exemplo é considerado um verdadeiro negativo quando seu valor é previsto e realmente é falso. (e.g. Um paciente com um ritmo cardíaco normal detectado como tal).
- **Falso Positivo (FP)**: Também chamado de erro tipo I. Um exemplo é considerado um falso positivo quando é previsto como verdadeiro e na realidade é falso (e.g. Detecção de uma arritmia em um paciente sem problemas). O objetivo deste trabalho é a redução desse tipo de erro.
- **Falso Negativo (FN)**: Também chamado de erro tipo II. Um exemplo é considerado um falso negativo quando é previsto como falso e na realidade é verdadeiro (e.g. Detecção de um ritmo cardíaco normal em um paciente sofrendo de arritmia). Esse tipo de erro pode ser considerado muito grave em uma aplicação médica: não detectar um problema a tempo pode ser o diferencial entre a vida e a morte de um paciente.

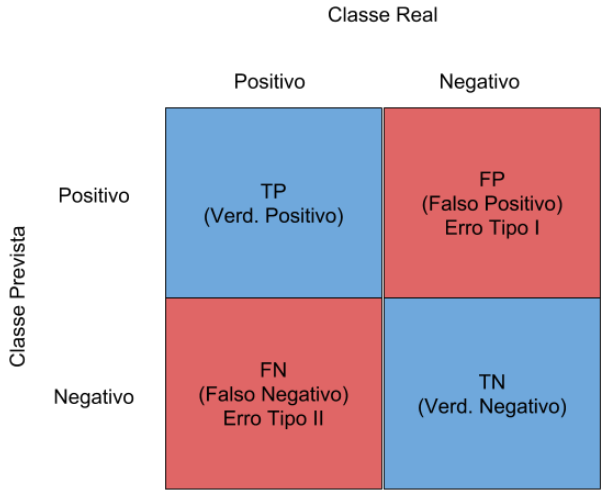
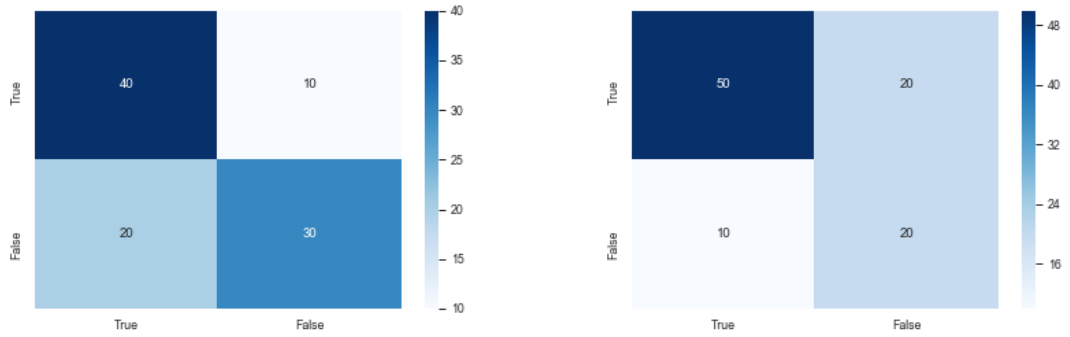


Figura 3.4: Configuração de uma matriz de confusão para classificações binárias (2×2). As células em azul representam as classificações corretas e as em vermelho representam os dois tipos de erros. Fonte: Autor

As matrizes de confusão dos exemplos ficam então da seguinte forma:



(a) Exemplo 01 (b) Exemplo 02

Figura 3.5: Matrizes de confusão referentes aos exemplos 01 (A) e 02 (B). Na imagem quanto maior a intensidade da cor, maior o valor relativo em relação ao total. Fonte: Autor

Apesar de não ser uma métrica, da matriz de confusão podem ser derivadas várias métricas importantes para a avaliação de modelos preditivos (e.g. acurácia, precisão, revocação e etc). Por isso ela será usada como base para a explicação de todas as métricas que seguem esse trabalho.

3.3.1.2 Acurácia (*Accuracy*)

A acurácia é uma das métricas mais simples para o cálculo de performance estatística. Ela consiste na razão da quantidade de previsões certas em relação a quantidade de previsões totais. Mais formalmente, ela pode ser definida como:

$$\text{Acurácia} = \frac{\text{\# previsões corretas}}{\text{\# previsões totais}} = \frac{TP+TN}{TP+TN+FP+FN} \tag{3.1}$$

Seguindo os exemplos, temos então $\textbf{acurácia}_{\textbf{ex01}} = 0.7$ e $\textbf{acurácia}_{\textbf{ex02}} = 0.65$. Um problema na acurácia é que seu valor não reflete muito bem na performance dos modelos em bases desbalanceadas. Suponha que no exemplo 02 um classificador preveja que todos os casos pertençam a classe verdadeira (i.e. uma acurácia de 0.9) teríamos um bom valor de acurácia porém uma baixa performance real do modelo (o modelo sempre erraria a classificação das classes falsas).

3.3.1.3 Precisão (*Precision*), Revocação (*Recall*) e F1

Como forma de obter uma melhor avaliação em bases de dados duas métricas são muito utilizadas: Precisão e Revocação. A precisão (equação 3.2) é uma medida que quantifica a habilidade do modelo de encontrar somente os casos relevantes. Já a revocação (equação 3.3) é

uma medida que quantifica a capacidade de um modelo de encontrar os casos relevantes dentro de um conjunto de dados. A figura 3.6 ilustra visualmente as duas métricas.

$$\text{Precisão} = \frac{TP}{TP+FP} \tag{3.2}$$

$$\text{Revocação} = \frac{TP}{TP+FN} \tag{3.3}$$

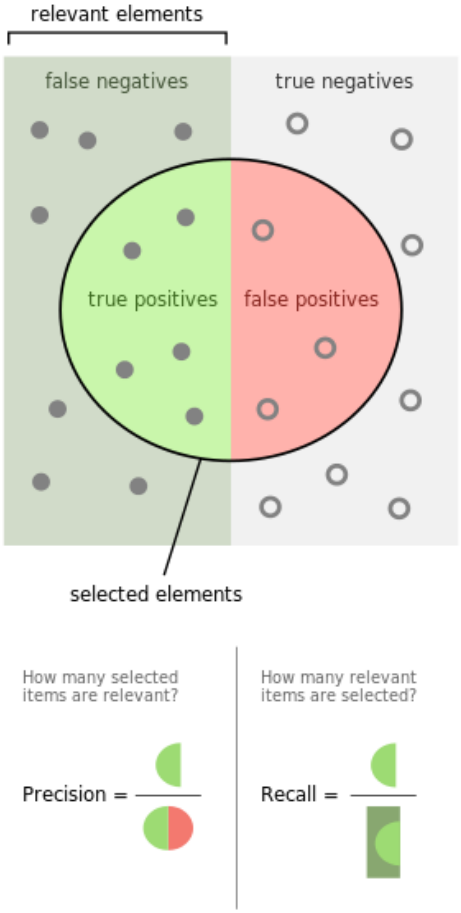


Figura 3.6: Ilustração do cálculo da precisão e revocação. Fonte: Walber, Wikimedia, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=36926283

Nos exemplos então temos que $\textbf{precisão}_{\textbf{ex01}} = 0.67$ e $\textbf{precisão}_{\textbf{ex02}} = 0.92$. Para a revocação temos $\textbf{revocação}_{\textbf{ex01}} = 0.8$ e $\textbf{revocação}_{\textbf{ex02}} = 0.67$. Comparando o modelo do exemplo 02 e um que preveja todos os casos verdadeiros (precisão de 0.9 e revocação de 1) temos na precisão

um indicativo que o modelo do exemplo 2 tem uma melhor performance na identificação de falsos positivos.

A precisão e revocação geralmente são avaliadas juntas devido ao *trade-off* envolvido na maximização de algumas dessas métricas. Em geral, maximizar um dos valores minimiza o outro. A escolha da maximização de alguma delas depende da necessidade do modelo. No modelo proposto, por exemplo, há um sentido maior na maximização da precisão devido ao fato dessa métrica levar em consideração os falsos positivos.

Quando não se tem um objetivo tão concreto para maximizar, uma abordagem melhor pode ser a maximização do F1 (equação 3.4). A F1 combina as duas métricas através do uso de uma média harmônica, o uso desse tipo de média é uma forma de dar um maior peso para valores muito pequenos, ou seja, caso um das métrica tenha uma precisão baixa o mesmo ocorrerá com o valor de F1 mesmo que a revocação seja alta.

$$F1 = 2 \times \frac{Precision \times Recall}{Precison + Recall} \tag{3.4}$$

Voltando novamente aos exemplos, temos $\mathbf{f1_{ex01}} = 0.73$ e $\mathbf{f1_{ex02}} = 0.77$. Para o caso de um classificador que prediz todos os exemplos como da classe verdadeira temos um valor de F1 de 0.94.

3.3.1.4 AUROC (*Area Under the Curve - Receiving Operating Characteristic*)

Antes de explicar o conceito de AUROC é necessário explicar o que é a curva ROC (figura 3.7). A curva ROC é uma forma de visualização da performance de um modelo binário. Nela o eixo X representa a taxa de falsos positivos (FPR - equação 3.6) e o eixo Y representa a taxa de verdadeiros positivos (TPR ou Revocação - equação 3.3). Os pontos nos eixos (*thresholds*) correspondem a probabilidade de definir um exemplo como pertencente a alguma das classes, ou seja, para um valor de *threshold* de 0.2 qualquer entrada que obtiver uma probabilidade maior ou igual a este valor é considerado como da classe verdadeira.

$$FPR = \frac{FP}{FP + TN} \tag{3.5}$$

Portanto, quanto melhor o classificador, mais próxima a sua curva está do ponto superior esquerdo. Esse fato permite calcular a área em baixo da curva (AUROC) resumindo toda a curva em um único número. De maneira formal, seja f(x) a função que representa a curva ROC de um modelo a AUROC é definida pela equação 3.5:

$$AUROC = \int_0^1 f(x)dx \tag{3.6}$$

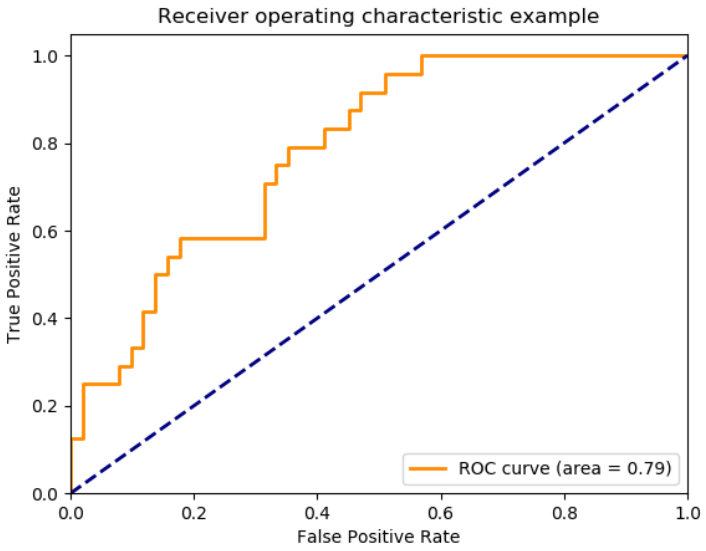


Figura 3.7: Exemplo de curva ROC (linha contínua) para um classificador binário. A linha tracejada representa a curva de um modelo que classifica suas entradas aleatoriamente. Fonte: Sklearn, acessado em Nov 2018, https://scikit-learn.org/stable/modules/model_evaluation.html#roc-metrics

3.3.2 Validação Cruzada K-Fold

Como visto anteriormente, o objetivo de um algoritmo de aprendizagem de máquina é conseguir se aproximar de uma função que modela o fenômeno estudado. Isso é feito usando uma amostra de dados que não necessariamente representam a população inteira. Então, para um modelo ser bom, ele deve ter uma boa performance nos casos não vistos por ele. Isso é chamado generalização.

Usar toda a amostra disponível para treinar o modelo pode causar um problema chamado *overfitting*. Esse problema ocorre quando o modelo aprende tão bem o conjunto de treinamento que ao ser testado usando um novo conjunto de dados a sua performance tem uma queda drástica. A figura 3.8 ilustra um modelo com *overfitting* (linha verde) e um modelo ideal (linha preta) separando as duas classes (azul e vermelho). Note que o modelo sofrendo do problema aprende muito bem o conjunto de treinamento, inclusive o ruído (exemplos de uma classe do lado pertencente a outra) prejudicando a generalização.

Uma forma de evitar esse comportamento é sempre separar uma parte do conjunto disponível para treinamento e outro pra teste (geralmente 70/30). Esse método é chamado de *holdout*.

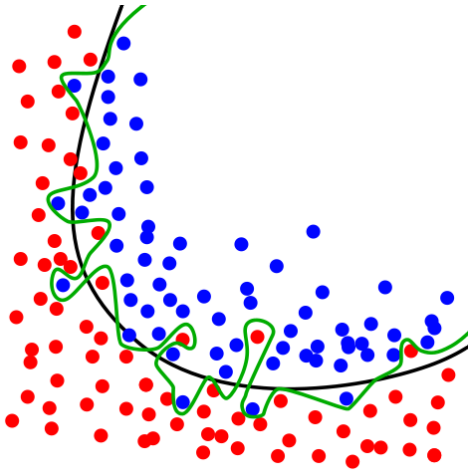


Figura 3.8: Exemplo de *overfitting* (linha verde) contra uma boa classificação (linha preta) para as classes azul e vermelho. Note que o modelo passando por *overfitting* aprende muito bem o conjunto dos dados, inclusive o ruído, o que atrapalha a generalização. Fonte: Chabacano, Wikimedia, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=3610704

Porém há um problema, ao separar esse conjunto em duas partes não se sabe exatamente quais os dados vão para o conjunto de teste e quais para o de treinamento. Em consequência disso, não se sabe se o conjunto de teste é um bom representante da população.

Dentre várias técnicas para resolver esse problema está o **validação cruzada k-fold**. Nessa técnica o conjunto é divido em k subconjuntos mutualmente exclusivos. Desses k conjuntos, 1 é utilizado para validação do modelo (similar ao *holdout*) e k-1 é utilizado para treinamento. O processo é então repetido k vezes e toma-se como resultado a média dos resultados de todos os *folds*. A figura 3.9 ilustra o processo. O objetivo dessa técnica é ter então uma estimativa mais robusta e confiável do modelo, treinando e validando ele com todo o conjunto disponível.

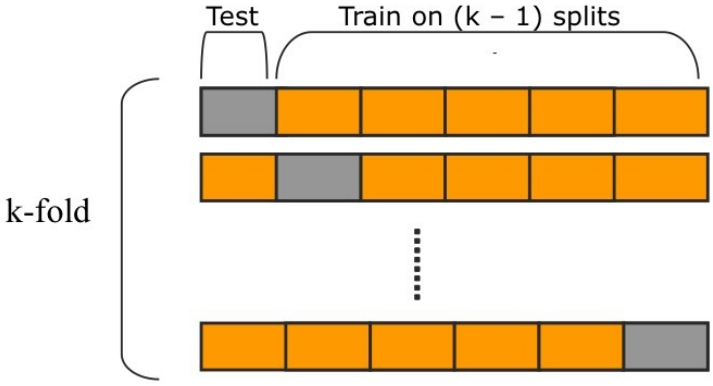


Figura 3.9: Ilustração da divisão de dados em uma validação cruzada *k-fold*. Na imagem os quadrados amarelos ilustram o conjunto de treinamento e o cinza o conjunto de teste. Fonte: Qingkai Kong, Acessado em Nov 2018, http://qingkaikong.blogspot.com/2017/02/machine-learning-9-more-on-artificial.html

3.4 Preparação dos dados

Como as gravações de ambos os problemas tem a mesma forma de captura e classificação, tendo como a única diferença os 30 segundos adicionais no desafio *"retrospective"* foram utilizadas as 750 gravações das 1250 disponíveis (conjunto de testes não disponível). Para evitar inconsistência nos dados os 30 segundos adicionais foram ignorados.

A distribuição dos dados foi feita da seguinte forma: 20% (150 gravações) foram destinadas ao conjunto de teste e 80% (600 gravações) foram utilizadas no processo de validação cruzada explicada na seção anterior, ou seja, em cada *fold* 16% (120 gravações) é utilizado para validação e 64% (480 gravações) são utilizados para treinamento. De todo o conjunto de dados 61% (456 gravações) foram classificados como falsos e 39% (292 gravações) como verdadeiros. Devido a isso, foi utilizado *random oversampling* para reduzir o efeito do desbalanceamento. Um sumário da distribuição pode ser visto na figura 3.10.

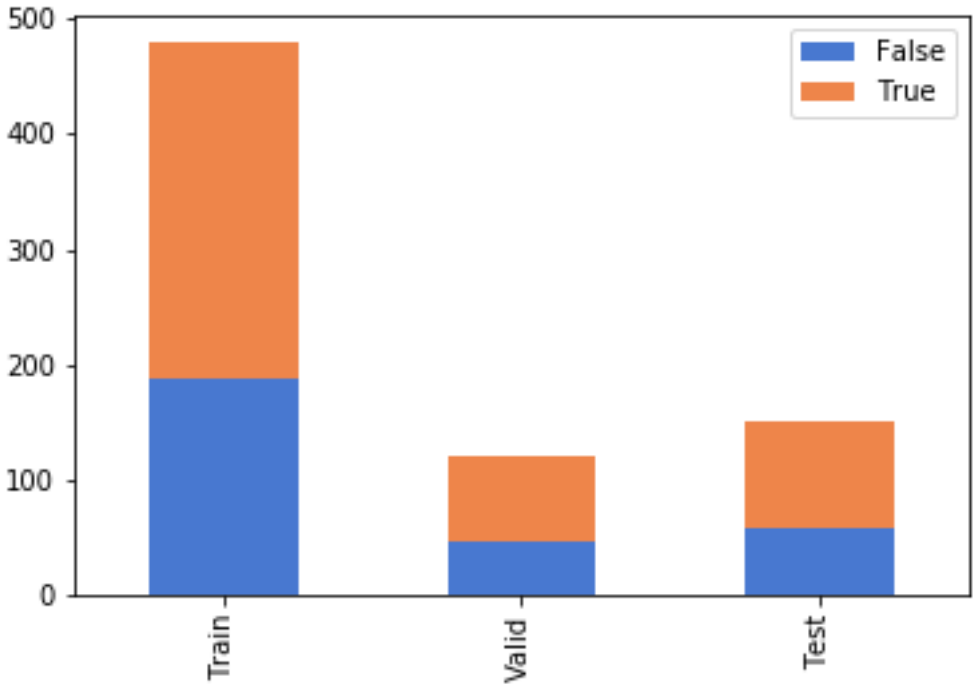
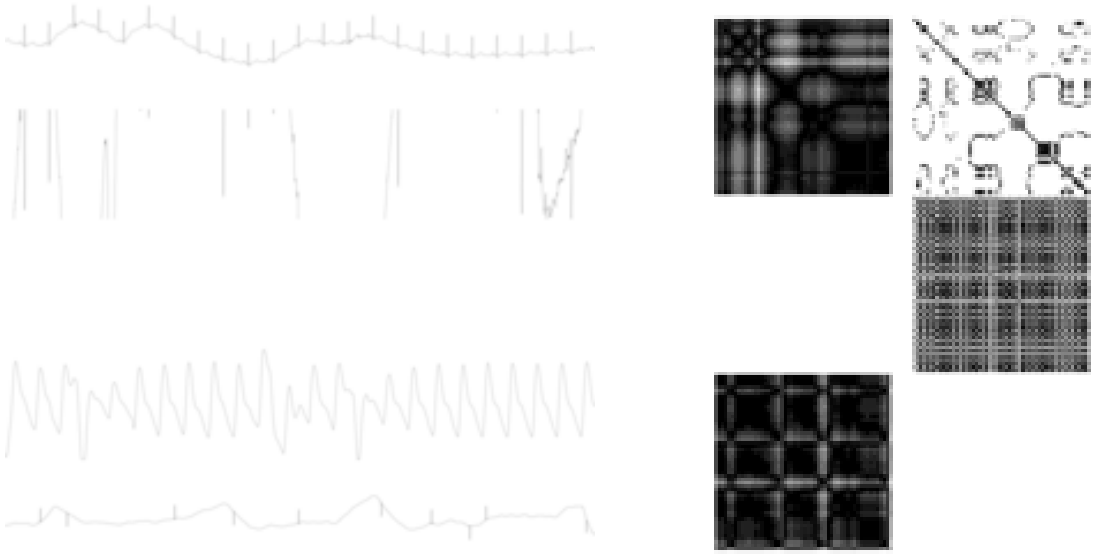


Figura 3.10: Distribuição dos dados de acordo com o conjunto e classe. Fonte: Autor

As figuras 3.11a e 3.11b mostram, respectivamente, uma entrada do tipo sinal e uma entrada do tipo *recurrence plot*. As imagens geradas consistem nos sinais das gravações dispostas na ordem a seguir: 2 sinais de ECG, ABP, PLETH e RESP. Em 3.11a a ordenação é feita de cima para baixo, já em 3.11b da esquerda para direita e de cima para baixo. Espaços em branco referem-se a sinais ausentes (como só há 5 tipos de sinais, o gráfico *recurrence* sempre tem o último espaço vazio). Essa configuração é feita para manter os padrões de uma imagem para outra. O apêndice A mostra alguns exemplos das entradas para alarmes falsos e verdadeiros

para cada uma das possíveis arritmias.

Nem todo o conteúdo da gravação é relevante na identificação dos alarmes falsos. Segundo a descrição do banco de dados, para seguir o padrão ANSI/AAMI EC13:2002 o alarme deve ter sido iniciado entre 4:50 e 5:00 min de gravação. Analisando os possíveis problemas cardíacos temos que no pior caso (i.e. taquicardia extrema) a duração máxima para a identificação da arritmia é de 7.2 segundos. A fim de identificar os problemas de forma otimizada foram utilizados então os 20 segundos finais das gravações (10 seg do padrão + 7s do pior caso + 3 de margem). Além disso, para impedir sobreposição e problemas de escala na geração das imagens, os sinais foram previamente dimensionados em uma escala de -1 a 1, mantendo assim a forma dos sinais.



(a) Sinais (b) *recurrence plots*

Figura 3.11: Exemplos dos sinais utilizados como entrada para a rede neural convolucional. A imagem (A) corresponde a entrada de sinais e a (B) a entrada *recurrence*. As duas imagens são referentes a gravação a165l do banco de dados da physionet. Fonte: Autor

3.4.1 Desbalanceamento

Vários problemas são relacionados a bases de dados desbalanceadas. Segundo Japkowicz [JS02] os problemas podem variar de acordo com 4 principais fatores:

- **Grau de desbalanceamento**
- **Complexidade dos dados**
- **Tamanho do conjunto de treinamento**
- **Classificador**

Em resumo, o aumento do grau de desbalanceamento, o aumento da complexidade dos dados cria pequenos agrupamentos da classe minoritária em meio as da classe majoritária. Esse fator dificulta a capacidade do classificador de separar as classes. O tipo de classificador também pode influenciar na performance, isso acontece principalmente porque muitos modelos assumem uma igual proporção das classes e um custo igual entre erros de classificação. Finalmente foi encontrado que todos esses problemas diminuem quanto maior for o conjunto utilizado para treinamento, isso porque com um conjunto maior temos uma maior representatividade de ambas as classes.

Existem várias técnicas para resolver o problema do desbalanceamento dos dados. Essas técnicas podem ser divididas em duas categorias principais: *undersampling* e *oversampling*. Na primeira se reduz a quantidade de dados da classe majoritária e na segunda se aumenta a quantidade de elementos da classe minoritária.

Para esse experimento foi escolhido a técnica de *random oversampling*. A técnica consistem em escolher aleatoriamente algumas entradas e repeti-las dentro do conjunto de treinamento. Ela foi escolhida devido à sua simplicidade e do reduzido tamanho do conjunto de dados.

Vale ressaltar que devido a utilização de um validação cruzada *5-fold* para evitar que os exemplos repetidos fossem encontrados em mais de um conjunto (treinamento, validação e teste) o *oversampling* é feito na iteração dos folds. Essa abordagem é executada para evitar uma expectativa otimista nos resultados. A figura 3.12 ilustra a abordagem utilizada.

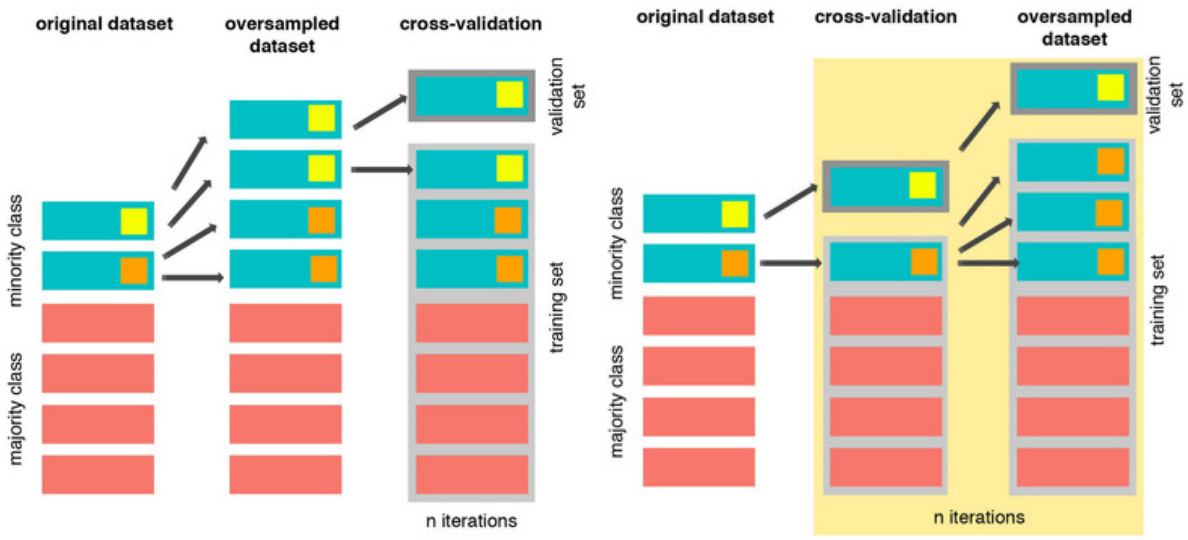


Figura 3.12: *Oversampling* + Validação cruzada feito de forma errada (A) e correta (B). Note que na primeira imagem (A) o fato do *oversampling* ser feito antes da definição dos *folds* repete a entrada (em amarelo) no conjunto de treinamento e teste. A abordagem da segunda imagem (B) evita essa situação aplicando a técnica durante as iterações do *k-fold* evitando assim a contaminação dos dados. Fonte: Marco Altini - https://www.marcoaltini.com/

3.5 Ambiente

Para a realização dos experimentos foi utilizado um ambiente Windows 10 (64-bits), um CPU Intel Core I7 (2.60Ghz), 16GB de Memória RAM e uma GPU Geforce GTX 970M. Na parte de programação foi utilizada a linguagem Python em sua versão v3.5.5 além de bibliotecas, dentre as principais:

- **scikit-learn (v0.19.1)**: Para avaliação de métricas e Validação Cruzada
- **keras (v2.2.2)**: Geração e aplicação de modelos de redes neurais e deep learning
- **tensorflow-gpu (v1.10.0)**: Como backend keras para utilização de GPU
- **pandas (v0.23.4)**: Para manipulação de dados tabulares e exportação de resultados
- **seaborn (v0.9.0)**: Para geração de gráficos

Além disso foi necessária a instalação de drivers adicionais providos pela NVidia, são eles: **Compute Unified Device Architecture (CUDA®)** na sua versão v9.0.176 e **CUDA® Deep Neural Network library (cuDNN)** em sua versão v7.2.1.38 para CUDA v9.0. Esses softwares permitem a utilização da GPU para a realização dos experimentos relacionados a deep learning. Um guia mais detalhado além do código do programa pode ser encontrada em https://github.com/gabriel-gbl.

CAPÍTULO 4

Resultados e Discussão

Neste capítulo serão discutidos e mostrados os principais resultados obtidos durante o experimento. Inicialmente serão discutidos individualmente cada etapa do treinamento como descrito no capítulo 3. Por fim será feita a comparação dos melhores modelos resultantes em relação ao tipo de entrada.

4.1 Experimento 1: Camadas Intermediárias e Dropout

A tabela 4.1 representa os 5 (entre 18) melhores modelos resultantes da variação da configuração das camadas intermediárias (*hidden layers*) e da taxa de *dropout* (i.e. experimento 01). Os resultados são dispostos de acordo com o tipo de entrada e classe dos exemplos (quando possível). A tabela 4.2 representa os hiperparâmetros dos experimentos contidos na tabela 4.1 referenciados pelo número na segunda coluna de ambas as tabelas.

Como o objetivo é a diminuição dos alarmes falsos sem comprometer os alarmes verdadeiros, o critério de escolha do melhor é baseado na maximização da AUROC seguida da precisão e revocação em relação a classe de alarmes falsos. Os melhores resultados estão destacados em negrito.

Analisando os resultados é possível observar uma diferença de performance de 0.18 na AUROC mesmo com as arquiteturas resultantes iguais. Além disso, com a entrada de sinais é possível observar um alto desvio padrão nas métricas se comparado com os *recurrence plots* indicando uma dificuldade maior em aprender os padrões desse tipo de entrada, essa conclusão pode ser reforçada com a presença de uma maior quantidade de camadas intermediárias com este tipo de entrada, ou seja, uma tentativa de se encontrar uma função mais robusta.

Tabela 4.1: Média e desvio padrão da acurácia, f1, precisão, revocação e AUROC das 5 melhores arquiteturas de acordo com o tipo de entrada e seus respectivos hiperparâmetros para o experimento 1.

		Accuracy	F1 (False)[3]	F1 (True)	Precision (False)	Precision (True)	Recall (False)	Recall (True)	AUROC
Recurrence	**0**[1]	0.7±0.06	0.78±0.03	0.51±0.28	0.72±0.07	0.56±0.32	0.86±0.1	0.47±0.26	0.73±0.13
	1	**0.71±0.02**[2]	**0.76±0.05**	**0.63±0.04**	**0.78±0.07**	**0.65±0.06**	**0.76±0.13**	**0.64±0.16**	**0.78±0.02**
	6	0.63±0.14	0.58±0.33	0.64±0.04	0.64±0.36	0.56±0.11	0.53±0.32	0.79±0.14	0.69±0.12
	7	0.7±0.03	0.74±0.02	0.64±0.05	0.77±0.04	0.61±0.03	0.71±0.05	0.68±0.09	0.77±0.03
	12	0.66±0.05	0.72±0.11	0.45±0.26	0.72±0.11	0.51±0.3	0.79±0.24	0.45±0.34	0.74±0.04
Signal	**1**	**0.57±0.17**	**0.46±0.42**	**0.48±0.27**	**0.42±0.39**	**0.43±0.28**	**0.52±0.48**	**0.64±0.41**	**0.6±0.15**
	4	0.61±0.0	0.75±0.0	0.01±0.01	0.6±0.0	0.2±0.45	1.0±0.0	0.0±0.01	0.51±0.01
	7	0.55±0.13	0.5±0.35	0.37±0.31	0.55±0.32	0.48±0.36	0.56±0.47	0.54±0.48	0.56±0.12
	11	0.52±0.11	0.45±0.41	0.23±0.31	0.36±0.33	0.16±0.22	0.6±0.55	0.4±0.55	0.5±0.02
	13	0.48±0.11	0.3±0.41	0.34±0.31	0.44±0.43	0.24±0.22	0.4±0.54	0.6±0.54	0.52±0.01

[1]Números na segunda coluna referenciam os hiperparâmetros contidos na tabela 4.2

[2]Valores em negrito são os melhores resultados de acordo com a AUROC

[3]Valores entre parênteses representam o tipo de alarme

Tabela 4.2: Seleção dos 5 melhores hiperparâmetros de acordo com o tipo de entrada no experimento 1.

		Batch Size	Dropout	Hidden Layers	Learning Rate	Momentum	Nesterov
Recurrence	**0** [1]	16	0.20	[128][2]	0.001	0.7	False
	1	**16**	**0.20**	**[256]**	**0.001**	**0.7**	**False**
	6	16	0.35	[128]	0.001	0.7	False
	7	16	0.35	[256]	0.001	0.7	False
	12	16	0.50	[128]	0.001	0.7	False
Signal	**1**	**16**	**0.20**	**[256]**	**0.001**	**0.7**	**False**
	4	16	0.20	[256, 128]	0.001	0.7	False
	7	16	0.35	[256]	0.001	0.7	False
	11	16	0.35	[256, 256]	0.001	0.7	False
	13	16	0.50	[256]	0.001	0.7	False

[1]Números na segunda coluna referenciam os hiperparâmetros contidos na tabela

[2]Valores entre "["e "]"representam a configuração das camadas intermediárias. (e.g. [128, 256] significa uma arquitetura com uma camada de 128 neurônios seguida de uma com 256)

4.2 Experimento 2: Taxa de Aprendizado e Tamanho do Batch

Similar ao feito na seção 4.1 as tabelas 4.3 e 4.4 representam, respectivamente, os 5 melhores resultados (entre 12) e seus hiperparâmetros correspondentes. Vale lembrar que nesse experimento foram utilizados os melhores hiperparâmetros do experimento anterior e foram variados somente a taxa de aprendizado (*learning rate*) e tamanho do batch (*batch size*).

Observe que com a entrada de sinais há uma alteração somente na taxa de aprendizagem (de 0.001 para 0.0001) em relação ao melhor modelo do experimento anterior, porém essa diferença aumenta em 0.17 a área da curva ROC (AUROC) e torna a estimativa mais exata (diminui o desvio padrão). Já com os *recurrence plots* mesmo com uma alteração nos dois parâmetros não há uma diferença tão significativa na performance.

É importante ressaltar que assim como esperado, nos dois casos houve uma diminuição na taxa de aprendizado. Isso acontece porque com o uso de *transfer learning* assume-se que o modelo original já apresenta uma alta capacidade de generalização e portanto não há necessidade de mudanças bruscas nos pesos originais.

Tabela 4.3: Média e desvio padrão da acurácia, f1, precisão, revocação e AUROC das 5 melhores arquiteturas de acordo com o tipo de entrada e seus respectivos hiperparâmetros para o experimento 2.

		Accuracy	F1 (False)[3]	F1 (True)	Precision (False)	Precision (True)	Recall (False)	Recall (True)	AUROC
Recurrence	**0**[1]	**0.72±0.03**[2]	**0.76±0.05**	**0.63±0.06**	**0.77±0.04**	**0.68±0.12**	**0.77±0.13**	**0.63±0.15**	**0.8±0.02**
	1	0.71±0.05	0.76±0.06	0.63±0.02	0.76±0.02	0.67±0.11	0.77±0.13	0.62±0.08	0.78±0.04
	4	0.72±0.02	0.76±0.02	0.66±0.02	0.79±0.02	0.63±0.03	0.73±0.06	0.69±0.06	0.8±0.02
	8	0.71±0.04	0.76±0.04	0.63±0.06	0.76±0.04	0.65±0.08	0.77±0.1	0.63±0.12	0.79±0.02
	9	0.72±0.03	0.77±0.04	0.63±0.02	0.76±0.02	0.67±0.09	0.78±0.1	0.61±0.08	0.78±0.02
Signal	**0**	0.72±0.03	0.76±0.02	0.65±0.07	0.78±0.05	0.63±0.02	0.75±0.05	0.67±0.13	0.77±0.02
	4	**0.71±0.03**	**0.78±0.01**	**0.59±0.09**	**0.74±0.05**	**0.67±0.02**	**0.83±0.04**	**0.54±0.14**	**0.77±0.02**
	5	0.55±0.14	0.46±0.4	0.36±0.32	0.59±0.37	0.39±0.24	0.56±0.51	0.53±0.5	0.57±0.1
	8	0.73±0.02	0.77±0.01	0.67±0.03	0.79±0.02	0.65±0.02	0.76±0.01	0.69±0.04	0.77±0.02
	9	0.5±0.15	0.3±0.42	0.47±0.27	0.28±0.38	0.37±0.23	0.35±0.49	0.73±0.43	0.56±0.11

[1]Números na segunda coluna referenciam os hiperparâmetros contidos na tabela 4.4
[2]Valores em negrito são os melhores resultados de acordo com a AUROC
[3]Valores entre parênteses representam o tipo de alarme

Tabela 4.4: Seleção dos 5 melhores hiperparâmetros de acordo com o tipo de entrada no experimento 2.

		Batch Size	Dropout	Hidden Layers	Learning Rate	Momentum	Nesterov
Recurrence	**0**[1]	**8**	**0.2**	**[256]**[1]	**0.0001**	**0.7**	**False**
	1	8	0.2	[256]	0.001	0.7	False
	4	16	0.2	[256]	0.0001	0.7	False
	8	32	0.2	[256]	0.0001	0.7	False
	9	32	0.2	[256]	0.001	0.7	False
Signal	**0**	8	0.2	[256]	0.0001	0.7	False
	4	**16**	**0.2**	**[256]**	**0.0001**	**0.7**	**False**
	5	16	0.2	[256]	0.001	0.7	False
	8	32	0.2	[256]	0.0001	0.7	False
	9	32	0.2	[256]	0.001	0.7	False

[1]Números na segunda coluna referenciam os hiperparâmetros contidos na tabela
[2]Valores entre "["e "]"representam a configuração das camadas intermediárias. (e.g. [128, 256] significa uma arquitetura com uma camada de 128 neurônios seguida de uma com 256)

4.3 Experimento 3: Momentum and Nesterov

Nas tabelas 4.5 e 4.6 é possível notar que a aplicação do *momentum* de Nesterov com uma mudança de 0.7 para 0.5 trouxe uma pequena melhora na precisão dos modelos para a classe de alarmes falsos e mantém inalterada a AUROC. A diminuição do *momentum* é outro fator que indica que não se busca uma alteração brusca dos pesos com uma abordagem de *transfer learning*.

Além disso, houveram significativas melhoras nos outras métricas com as alterações da arquitetura. Cabe destacar o aumento da revocação para as classes verdadeiras em ambos os tipos de entrada.

Tabela 4.5: Média e desvio padrão da acurácia, f1, precisão, revocação e AUROC das 5 melhores arquiteturas de acordo com o tipo de entrada e seus respectivos hiperparâmetros para o experimento 3.

		Accuracy	F1 (False)[3]	F1 (True)	Precision (False)	Precision (True)	Recall (False)	Recall (True)	AUROC
Recurrence	**0**[1]	**0.71±0.02**[2]	**0.74±0.04**	**0.66±0.04**	**0.8±0.04**	**0.62±0.04**	**0.7±0.1**	**0.72±0.11**	**0.8±0.01**
	1	0.72±0.02	0.78±0.02	0.6±0.07	0.74±0.03	0.69±0.07	0.83±0.07	0.55±0.12	0.79±0.02
	2	0.71±0.03	0.76±0.03	0.64±0.03	0.77±0.03	0.64±0.05	0.75±0.07	0.66±0.07	0.79±0.02
	3	0.71±0.03	0.75±0.04	0.67±0.02	0.8±0.02	0.62±0.05	0.7±0.08	0.74±0.06	0.79±0.02
	4	0.7±0.03	0.73±0.05	0.64±0.04	0.78±0.05	0.61±0.04	0.7±0.1	0.69±0.11	0.78±0.03
Signal	**0**	**0.72±0.03**	**0.77±0.04**	**0.64±0.07**	**0.78±0.06**	**0.67±0.07**	**0.78±0.11**	**0.64±0.16**	**0.77±0.02**
	1	0.72±0.03	0.78±0.03	0.63±0.05	0.76±0.04	0.68±0.06	0.8±0.07	0.6±0.1	0.77±0.02
	2	0.71±0.01	0.75±0.03	0.65±0.03	0.78±0.04	0.62±0.03	0.72±0.07	0.69±0.09	0.77±0.02
	3	0.71±0.03	0.77±0.02	0.63±0.06	0.76±0.04	0.64±0.03	0.78±0.02	0.62±0.1	0.77±0.02
	4	0.51±0.15	0.33±0.4	0.48±0.23	0.87±0.19	0.48±0.14	0.38±0.5	0.71±0.43	0.56±0.11

[1]Números na segunda coluna referenciam os hiperparâmetros contidos na tabela 4.6
[2]Valores em negrito são os melhores resultados de acordo com a AUROC
[3]Valores entre parenteses representam o tipo de alarme

Tabela 4.6: Seleção dos 5 melhores hiperparâmetros de acordo com o tipo de entrada no experimento 3.

		Batch Size	Dropout	Hidden Layers	Learning Rate	Momentum	Nesterov
Recurrence	**0**[1]	**8**	**0.2**	**[256]**[2]	**0.0001**	**0.5**	**True**
	1	8	0.2	[256]	0.0001	0.5	False
	2	8	0.2	[256]	0.0001	0.7	True
	3	8	0.2	[256]	0.0001	0.7	False
	4	8	0.2	[256]	0.0001	0.9	True
Signal	**0**	**16**	**0.2**	**[256]**	**0.0001**	**0.5**	**True**
	1	16	0.2	[256]	0.0001	0.5	False
	2	16	0.2	[256]	0.0001	0.7	True
	3	16	0.2	[256]	0.0001	0.7	False
	4	16	0.2	[256]	0.0001	0.9	True

[1]Números na segunda coluna referenciam os hiperparâmetros contidos na tabela
[2]Valores entre "["e "]"representam a configuração das camadas intermediárias. (e.g. [128, 256] significa uma arquitetura com uma camada de 128 neurônios seguida de uma com 256)

4.4 Resultados Finais

Como pode ser visto nos valores em negrito da tabela 4.6 as melhores configuração de hiperparâmetros de todo o trabalho resultaram nos mesmos valores exceto no tamanho do *batch*. Já analisando a figuras 4.1 e 4.2 pode-se concluir que não há uma diferença significativa no uso de *recurrence plots* ou sinais. Nos *recurrence plots* é possível notar uma melhor performance nas métricas priorizadas como AUROC e revocação, porém é na entrada de sinais que se vê uma diminuição nos falsos positivos em detrimento de uma maior taxa de falsos negativos.

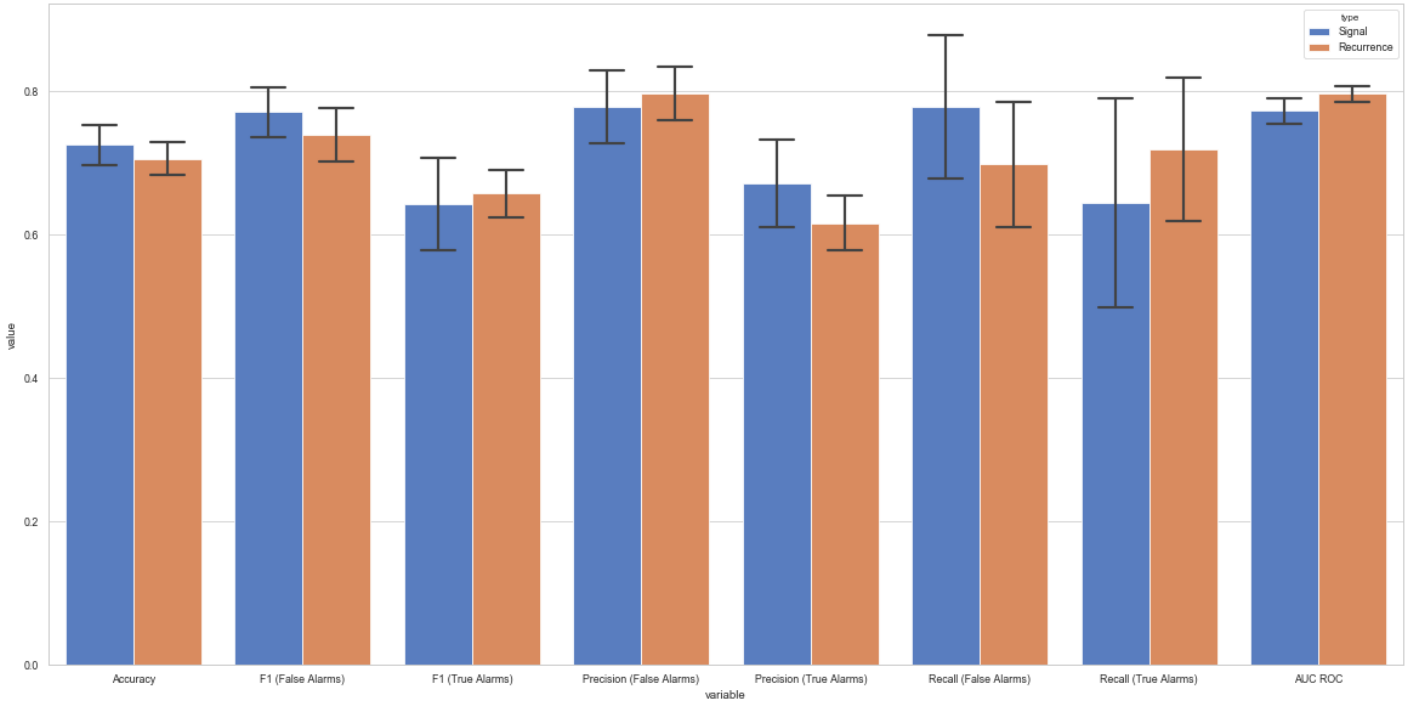
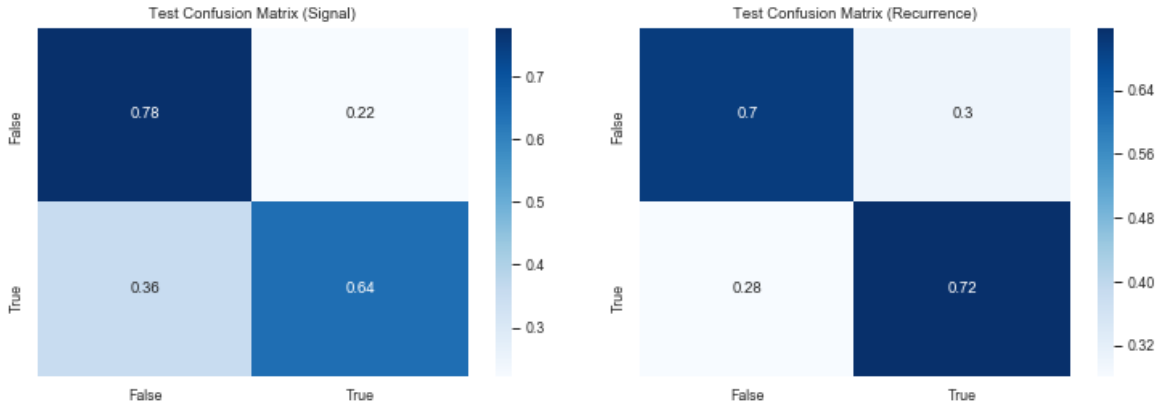


Figura 4.1: Comparação dos melhores resultados de cada tipo de entrada do modelo em relação as suas métricas (e seu desvio padrão). É possível notar que não há uma diferença significativa entre os diferentes tipos de entrada. Fonte: Autor



(a) sinais (b) *recurrence*

Figura 4.2: Matrizes de confusão normalizadas resultantes do conjunto de teste em relação a entrada de sinais (a) e *recurrence plots* (b). Note que é em (a) que há a menor taxa de falsos positivos porém com um aumento dos falsos negativos.

CAPÍTULO 5

Conclusão

Este trabalho teve como objetivo a redução de alarmes falsos de arritmias cardíacas graves em UTIs a partir de representação visual dos sinais utilizando redes neurais convolucionais. Para isso foram utilizadas abordagens modernas como o uso de *recurrence plots* e *transfer learning*.

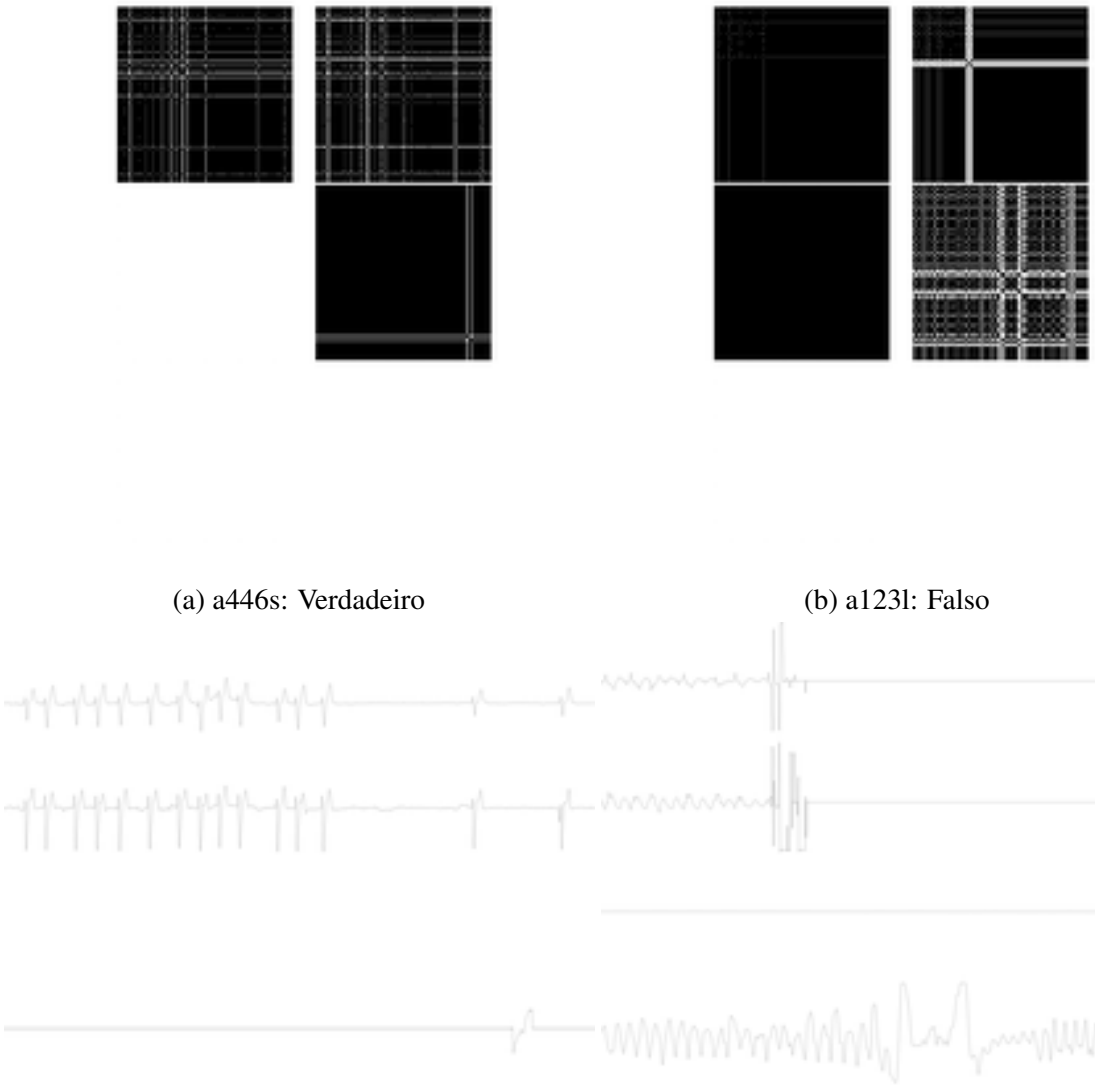
Os resultados dos experimentos mostram que apesar da pouca quantidade de dados foi possível obter bons resultados na identificação de alarmes falsos a partir dos dados utilizados. Além disso, foi mostrado que não há uma diferença significativa na utilização de *recurrence plots* e sinais como descritos na metodologia desse projeto.

5.0.1 Trabalhos Futuros

- Utilização de *recurrence plots* coloridos para aproveitamento máximo de todos os canais identificados na rede VGG.
- Aplicação de outras arquiteturas de redes neurais convolucionais e utilização de *random search* no lugar de *grid search*.
- Treinamento não só das camadas totalmente conectadas, mas das camadas convolucionais mais profundas.
- Diminuição da complexidade do banco de dados (e.g. utilização de somente um equipamento de monitoramento, como o eletrocardiógrafo) para uma aplicação em tempo real de um modelo em UTIs.

APÊNDICE A

Exemplos de Entradas



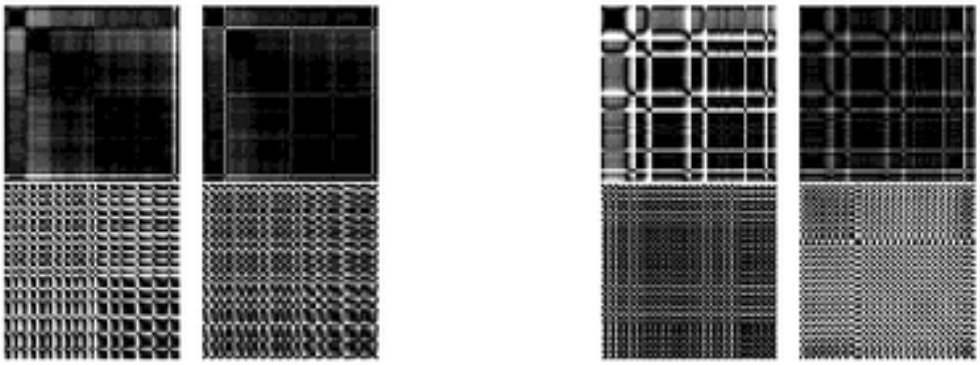
(a) a446s: Verdadeiro

(b) a123l: Falso

(c) a446s: Verdadeiro

(d) a123l: Falso

Figura A.1: Exemplos dos dois tipos de entradas de assistolia utilizadas no experimento para gravações de alarme verdadeiro (a446s) e falso (a123l). Note que na figura (d) os 3 primeiros sinais caracterizam uma assistolia porém o quarto não, esse pode ser um motivo pro alarme ter sido classificado como falso. Fonte: Autor



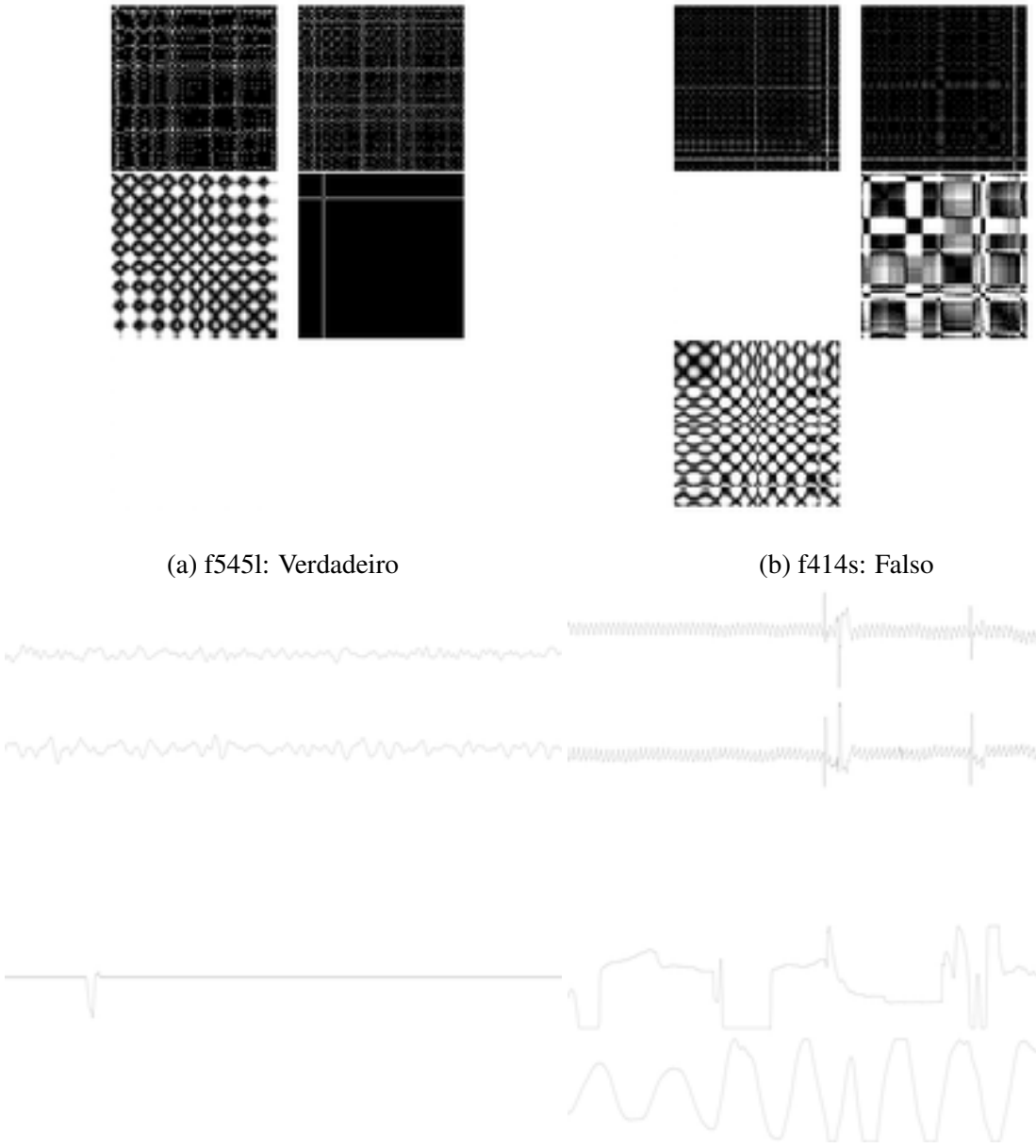
(a) b455l: Verdadeiro

(b) b330s: Falso

(c) b455l: Verdadeiro

(d) b330s: Falso

Figura A.2: Exemplos dos dois tipos de entradas de bradicardia extrema utilizadas no experimento para gravações de alarme verdadeiro (b455l) e falso (b330s). Fonte: Autor



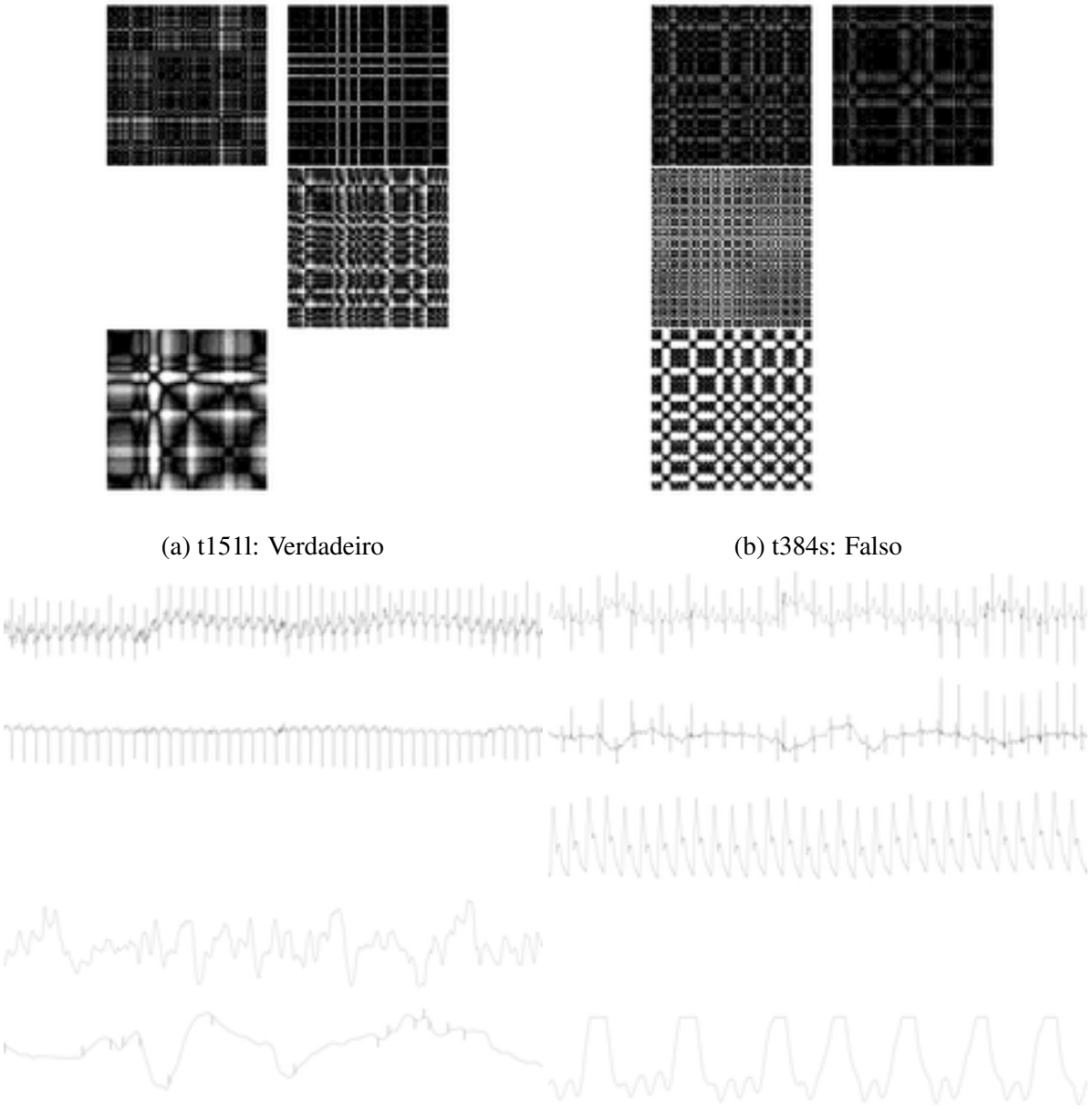
(a) f545l: Verdadeiro

(b) f414s: Falso

(c) f545l: Verdadeiro

(d) f414s: Falso

Figura A.3: Exemplos dos dois tipos de entradas de fibrilação ventricular utilizadas no experimento para gravações de alarme verdadeiro (f545l) e falso (f414s). Note que na figura (c) há um erro na geração da entrada. Fonte: Autor



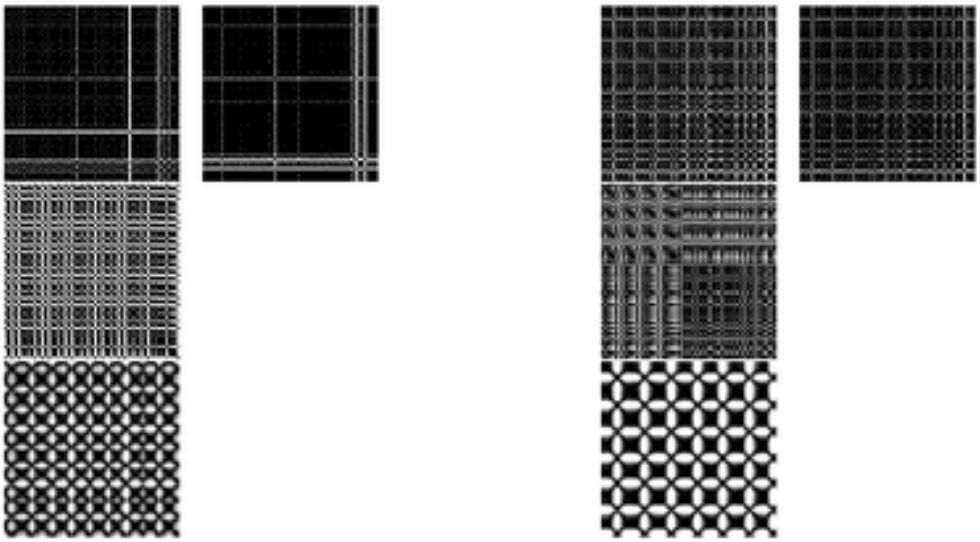
(a) t151l: Verdadeiro

(b) t384s: Falso

(c) t151l: Verdadeiro

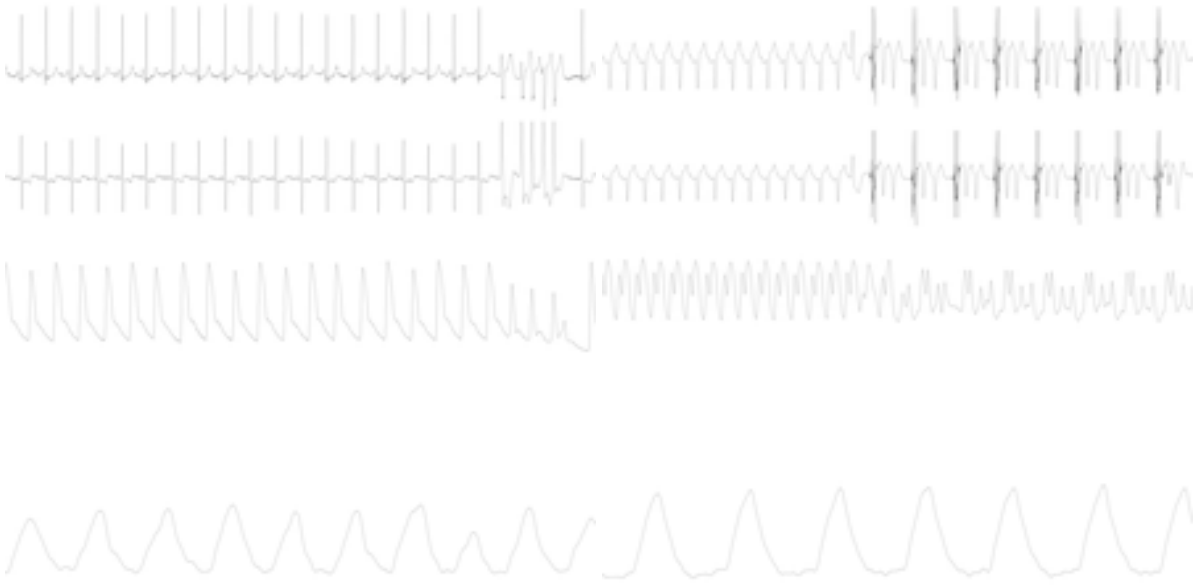
(d) t384s: Falso

Figura A.4: Exemplos dos dois tipos de entradas de taquicardia extrema utilizadas no experimento para gravações de alarme verdadeiro (t151l) e falso (t384s). Fonte: Autor



(a) v793l: Verdadeiro

(b) v830s: Falso



(c) v793l: Verdadeiro

(d) v830s: Falso

Figura A.5: Exemplos dos dois tipos de entradas de taquicardia ventricular utilizadas no experimento para gravações de alarme verdadeiro (v7931) e falso (v830s). Fonte: Autor

Referências Bibliográficas

[AFO+17] U Rajendra Acharya, Hamido Fujita, Shu Lih Oh, Yuki Hagiwara, Jen Hong Tan, and Muhammad Adam. Application of deep convolutional neural network for automated detection of myocardial infarction using ecg signals. *Information Sciences*, 415:190–198, 2017.

[ANS+08] Anton Aboukhalil, Larry Nielsen, Mohammed Saeed, Roger G Mark, and Gari D Clifford. Reducing false alarm rates for critical arrhythmias using the arterial blood pressure waveform. *Journal of biomedical informatics*, 41(3):442–451, 2008.

[Bec06] Daniel E Becker. Fundamentals of electrocardiography interpretation. *Anesthesia progress*, 53(2):53–64, 2006.

[Bra95] Brasil. Resolução cremesp nº 71 - define e regulamenta as atividades das unidades de terapia intensiva. *Diário Oficial do Estado de São Paulo*, nov 1995. Conselho Regional de Medicina do Estado de São Paulo, Poder Executivo, São Paulo, SP, n. 217, 14 nov. 1995. Seção 1.

[BSWI11] Matthias Borowski, Sylvia Siebig, Christian Wrede, and Michael Imhoff. Reducing false alarms of intensive care online-monitoring systems: an evaluation of two signal extraction algorithms. *Computational and mathematical methods in medicine*, 2011, 2011.

[C+15] François Chollet et al. Keras, 2015.

[Cha01] Marie-Christine Chambrin. Alarms in the intensive care unit: how can the number of false alarms be reduced? *Critical Care*, 5(4):184, 2001.

[EKR87] J-P Eckmann, S Oliffson Kamphorst, and David Ruelle. Recurrence plots of dynamical systems. *EPL (Europhysics Letters)*, 4(9):973, 1987.

[GAG+13] A. L. Goldberger, L. A. N. Amaral, L. Glass, J. M. Hausdorff, P. Ch. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley. PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals. *Circulation*, 101(23):e215–e220, 2000 (June 13). Circulation Electronic Pages: http://circ.ahajournals.org/content/101/23/e215.full PMID:1085218; doi: 10.1161/01.CIR.101.23.e215.

[GBCB16] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[hea] heart.org. *Arrhythmia*. https://www.heart.org/en/health-topics/arrhythmia.

[HGD18] Nima Hatami, Yann Gavet, and Johan Debayle. Classification of time-series images using deep convolutional neural networks. In *Tenth International Conference on Machine Vision (ICMV 2017)*, volume 10696, page 106960Y. International Society for Optics and Photonics, 2018.

[HHHH09] Simon S Haykin, Simon S Haykin, Simon S Haykin, and Simon S Haykin. *Neural networks and learning machines*, volume 3. Pearson Upper Saddle River, 2009.

[Ins17] Queensland Brain Institute. *How Neurons Work*. The University of Queensland, November 2017. https://qbi.uq.edu.au/brain-basics/brain/brain-physiology/how-do-neurons-work.

[JS02] Nathalie Japkowicz and Shaju Stephen. The class imbalance problem: A systematic study. *Intell. Data Anal.*, 6(5):429–449, October 2002.

[LC12] Qiao Li and Gari D Clifford. Dynamic time warping and machine learning for signal quality assessment of pulsatile signals. *Physiological measurement*, 33(9):1491, 2012.

[LKL14] Martin Längkvist, Lars Karlsson, and Amy Loutfi. A review of unsupervised feature learning and deep learning for time-series modeling. *Pattern Recognition Letters*, 42:11–24, 2014.

[LRC14] Qiao Li, Cadathur Rajagopalan, and Gari D Clifford. Ventricular fibrillation and tachycardia classification using a machine learning approach. *IEEE Transactions on Biomedical Engineering*, 61(6):1607–1613, 2014.

[Mit97] Tom M Mitchell. Machine learning, ser. *Computer Science Series. Singapure: McGraw-Hill Companies, Inc*, 1997.

[NHN12] Lung National Heart and Blood Institute (NHLBI). *Arrhythmia*. U.S. Department of Health |& Human Services, March 2012. https://www.nhlbi.nih.gov/health-topics/arrhythmia.

[PKM17] B Pyakillya, N Kazachenko, and N Mikhailovsky. Deep learning for ecg classification. In *Journal of Physics: Conference Series*, volume 913, page 012004. IOP Publishing, 2017.

[PT09] Sairam Parthasarathy and Martin J Tobin. Sleep in the intensive care unit. In *Applied Physiology in Intensive Care Medicine*, pages 191–200. Springer, 2009.

[PY+10] Sinno Jialin Pan, Qiang Yang, et al. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[RDS+15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[RHH+17] Pranav Rajpurkar, Awni Y Hannun, Masoumeh Haghpanahi, Codie Bourn, and Andrew Y Ng. Cardiologist-level arrhythmia detection with convolutional neural networks. *arXiv preprint arXiv:1707.01836*, 2017.

[RN16] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

[SS01] Kirk Shelley and S Shelley. Pulse oximeter waveform: photoelectric plethysmography. *Clinical Monitoring, Carol Lake, R. Hines, and C. Blitt, Eds.: WB Saunders Company*, pages 420–428, 2001.

[SZ14a] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *ArXiv e-prints*, September 2014.

[SZ14b] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[TS10] Lisa Torrey and Jude Shavlik. Transfer learning. In *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques*, pages 242–264. IGI Global, 2010.

[YCBL14] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.