Jefferson Xavier Nóbrega

Uso de rede neural convolucional na identificação de hipertensão arterial

Campo Grande (MS) - Brasil
Setembro de 2022

Jefferson Xavier Nóbrega

Uso de rede neural convolucional na identificação de hipertensão

Dissertação apresentada ao Programa de Mestrado Stricto Sensu em Computação Aplicada, mantido pela Universidade Federal do Mato Grosso do Sul, como parte dos requisitos para a obtenção do título de Mestre em Computação Aplicada.

Orientador:	Prof. Dr. Milton Ernesto Romero Romero
Coorientador:	Prof. Dr. Evandro Mazina Martins
Banca:	Profa. Dra. Vera Regina Fernandes da Silva Marães
Banca:	Prof. Dr. Fábio Iaione

Campo Grande (MS) - Brasil
Setembro de 2022

Resumo

A hipertensão arterial ou pressão arterial alta sustentada é uma doença crônica com grande ocorrência no Brasil e no mundo. É um dos principais indicadores para a ocorrência de acidente vascular cerebral (AVC), enfarte, aneurisma arterial e insuficiência renal e cardíaca. Impacta diretamente na frequência cardíaca e na sua variabilidade: pacientes hipertensos costumam apresentar uma menor variação na frequência cardíaca se comparados a normotensos. Porém, ainda existe controversia se é possível determinar a hipertensão arterial somente pela Variabilidade da Frequência Cardíaca (VFC). Esta dissertação tem como objetivo estudar a questão ainda aberta se a Variabilidade da Frequência Cardíaca tem informação suficiente para definir a hipertensão, baseada no treinamento de uma Rede Neural Convolucional capaz de classificar um paciente entre saudável ou hipertenso e todos os desafios inerentes a este processo. Este treinamento é feito com base nos valores do intervalo RR do eletrocardiograma. Os intervalos RR podem apresentar *outliers* - amostras inválidas/incorretas - que devem ser devidamente tratados e não devem ser levados em consideração para o treinamento. Devido a baixa quantidade de arquivos para treinamento da Rede Neural Convolucional, foi utilizada a técnica *data augmentation* para oferecer uma maior fonte de aprendizagem. Através da Transformada Wavelet Contínua, estes arquivos de texto contendo o intervalo RR em milissegundos são convertidos em uma matriz com informações no domínio do tempo e da frequência. Por fim, é definido um modelo de Rede Neural Convolucional que é treinada para realizar a classificação dos pacientes. Os resultados são promissores mas dependem da quantidade de exames de pacientes que neste caso devem ser aumentada. O trabalho faz recomendações de como deve ser continuado este estudo, mas o sistema implementado na linguagem Python mostra utilidade para aprofundar o problema.

Palavras-chave: Intervalo RR, variabilidade da frequência cardíaca, rede neural convolucional, transformada wavelet, hipertensão.

Lista de Figuras

Lista de Abreviaturas

BPM Batimentos por minuto

CNN Convolutional Neural Network, em português, Rede Neural Convolucionall

ECG Eletrocardiograma

VFC Variabilidade da Frequência Cardíaca

Sumário

Capítulo 1

Introdução

A hipertensão arterial ou pressão arterial alta sustentada é uma doença crônica caracterizada pelos níveis elevados da pressão sanguínea nas artérias. Ela acontece quando os valores das pressões máxima(sistólica) e mínima(diastólica) são iguais ou ultrapassam os 140/90 mmHg [1].

As doenças cardiovasculares são a principal causa de morte no mundo. Em 2016, estima-se que 17,9 milhões de pessoas morreram por doenças cardiovasculares. Isto representa 31% (quase 1/3) de todas as mortes no planeta[2].Com números tão alarmantes, é necessário cuidado e atenção à nossa saúde cardíaca. Diversos estudos foram feitos relacionados à análise da VFC e a sua relação com a saúde cardiovascular.Vanderlei et al. [3] define que mudanças nos padrões da VFC fornecem um indicador sensível e antecipado de comprometimentos na saúde e que uma alta variabilidade na frequência cardíaca é sinal de boa adaptação, caracterizando um indivíduo saudável, com mecanismos autonômicos eficientes, enquanto que, a baixa variabilidade é frequentemente um indicador de adaptação anormal e insuficiente do sistema nervoso autônomo, implicando na presença de mau funcionamento fisiológico no indivíduo. Rosa [4] e Maraes[5] observaram uma menor variação da frequência cardíaca de hipertensos quando comparada à de normotensos.

Uma maneira fácil e não invasiva de analisar a VFC é através dos intervalos RR do Eletrocardiograma (ECG) dos pacientes. Entretanto, a análise manual dos intervalos RR do ECG leva tempo. Isto sem falar que o resultado do eletrocardiograma pode conter *outliers*, que são sinais inválidos que não devem ser considerados na análise. A presença de *outliers* exige que seja realizado um filtro para removê-los. Fazer este filtro de forma manual, além de demandar tempo, pode também levar à erros humanos, como por exemplo, a remoção de um sinal válido.

Surgem como opções de apoio para a análise dos intervalos RR, soluções automatizadas para removerem os *outliers* e classificar se o paciente é saudável ou se tem alguma cardiopatia. Para que um sistema consiga classificar corretamente um paciente, ele deve primeiro aprender os padrões apresentados por pessoas saudáveis e pessoas não saudáveis. É neste ponto que a Convolutional Neural Network, em português, Rede Neural Convolucionall (CNN) pode auxiliar e identificar características para essa análise automática.

Segundo [6], a Rede Neural Convolucional é uma classe de rede neural artifical que vem sendo aplicada com sucesso no processamento e análise de imagens digitais. Ainda segundo [6],as CNNs são capazes de aplicar filtros em dados visuais, mantendo a relação de vizinhança entre os pixels da imagem ao longo do processamento da rede. Este tipo de rede vem sendo amplamente utilizada, principalmente nas aplicações de classificação, deteccção e reconhecimento em imagens e vídeos.

Devido à característica da CNN em trabalhar dados visuais, é necessário converter os intervalos RR - que são originalmente representados em números inteiros, normalmente em milissegundos - para uma forma que a CNN consiga entendê-los e manipulá-los. É possível fazer essa conversão através da transformada Wavelet. Conforme estudos de [7, 8, 9], aplicar a transformada de wavelet contínua em intervalos RR resulta em uma boa entrada para que a CNN possa ser treinada.

Capítulo 2

Trabalhos Relacionados

Este trabalho é uma continuação dos trabalhos [7, 8, 9], nesta ordem.

[7] implementou técnicas para tratamento e análise dos sinais de frequência cardíaca, fazendo uso da Transformada de Wavelet (discreta e contínua). Seu trabalho consiste na implementação de técnicas automáticas para filtragem e análise dos sinais coletados pelo cardiofrequencímetro. Foram feitos experimentos com os filtros: Butterworth de 4ª ordem, Chebychev I de 4ª ordem, Chebychev II de 4ª ordem e Bessel de 4ª ordem mas nenhum destes se mostrou satisfatório visto que apresentavam problemas como a suavização da curva ou a redução geral da amplitude do sinal. Como alternativa foram elaborados e experimentados filtros de janela de médias e suporte de medianas, sendo que o filtro com suporte de medianas apresentou melhores resultados. Ainda foram comparadas duas hipóteses no tratamento de outliers: remoção do outlier e substituição do outlier. Este trabalho ainda analisa individualmente algumas amostras no domínio do tempo e da frequência e consegue, a partir das definições de taquicardia e bradicardia, identificar se a amostra em questão se encaixa em algum desses grupos. Também foram aplicadas as transformadas waveletes contínua (mexican hat) e discreta, onde foi possível notar diferenças entre o resultado para uma pessoa saudável e para uma pessoa com arritmia. Considera em sua conclusão que a filtragem com a abordagem de janela de medianas se apresentou bastante eficiente, tendo melhor desempenho que a filtragem com janela de médias e filtragem manual. O trabalho não desenvolveu CNN, apesar de deixar evidências promissoras na classificação do resultado da transformada wavelet.

[8] implementou uma rede neural convolucional para diagnosticar as variações de frequência cardíacas com a maior precisão possível. Também fez uso das Transformadas de Wavelet discreta e contínua, para transformar os arquivos com intervalo RR em entrada para a CNN. O autor fez uso da técnica *data augmentation* para transformar os 65 arquivos originais em 32500 arquivos de entradas para o treinamento da CNN. Adotou como padrão a quantidade de 348 amostras de intervalo RR para cada arquivo/paciente. Conclui que as redes neurais convolucionais são eficientes para treinamento de análise de variabilidade da frequência cardíaca caso tenham um bom número de arquivos para o treinamento. A acurácia em predições foi superior à 90%.

[9] além de fazer ajustes na CNN e na filtragem proposta por [7] e [8], também criou uma interface Web para que arquivos com intervalos RR sejam submetidos e classificados. Também precisou criar dados sintéticos a fim de aumentar a quantidade de dados para treinamento da CNN. Trabalhou especificamente com a Transformada Wavelet contínua. Consta em sua conclusão que tanto a filtragem quanto a classificação podem ser melhoradas.

Estes três trabalhos se basearam na mesma base de dados - sendo que variaram na forma de gerar dados sintéticos. Todos fizeram a conversão do intervalo RR para Batimentos por minuto (BPM).

[10] analisou o impacto da hipertensão - e da diabetes - na VFC e os resultados sugerem que a hipertensão causa ainda maiores alteração na VFC de diabéticos.

[11] analisou se o histórico de hipertensão nos pais influenciava na VFC dos filhos e como resultado obteve que, no domínio do tempo, a VFC foi significativamente mais baixa nos filhos de pais hipertensos do que os filhos de pais saudáveis. Também houveram alterações visíveis no domínio da frequência (através da análise espectral) que indicam uma menor VFC para os filhos de hipertensos.

[12] analisou o ECG de crianças diagnosticadas precocemente com hipertensão arterial e percebeu a diminuição da VFC. O desvio padrão de todos os intervalos RR (SDNN) foi significativamente inferior no grupo com hipertensão, sendo que o SDNN é o principal indicador para doenças cardíacas na análise pelo domínio no tempo [13].

A métrica BPM foi utilizada na grande maioria dos trabalhos revisados. Ela é inversamente proporcional aos valores de intervalo RR. Wallot et al.[14] compara a utilização do BPM e do interalo RR em um cenário de descanso-exercício-descanso e sugere que o intervalo RR permite uma melhor análise do que o BPM padrão, principalmente no domínio não linear . Já Goldberger et al. [15] comparou qual parâmetro descreve melhor as alterações autonômicas que ocorrem após o exercício e que fornece maior significância prognóstica. Encontrou resultados semelhantes entre a análise pelo RR e pelo BPM em dados de pessoas em repouso, porém ao analisar dados em exercício e pós-exercício algumas interações podem ser observadas pelo BPM e não pelo RR. Concluiu que nem sempre é válido fazer a conversão de um para outro, devendo ser observado qual tipo de análise e cenário será trabalhado.

Capítulo 3

Fundamentação Teórica

3.1 Hipertensão Arterial

A hipertensão arterial ou pressão alta é uma doença crônica caracterizada pelos níveis elevados da pressão sanguínea nas artérias. Ela acontece quando os valores das pressões máxima(sistólica) e mínima(diastólica) são iguais ou ultrapassam os 140/90 mmHg [1]. Estima-se que, no Brasil, 388 pessoas morrem por dia por hipertensão, sendo que de 2006 a 2016 o registro de mortalidade por hipertensão é próximo a 500 mil. Em 2016 a hipertensão foi o motivo de mais de 980 mil internações e atendimentos ambulatoriais no Brasil, gerando um custo superior à R$ 60 milhões aos cofres públicos.[1]A hipertensão também é um fator de risco importante e independente para doença cardiovascular, acidente vascular cerebral (AVC) e doença renal [16][17]. Também é comum a sua coexistência com a diabetes[10]. Na grande maioria dos casos a hipertensão é herdada dos pais, mas existem vários fatores relacionados ao estilo de vida do paciente que influenciam na pressão arterial [1]. Entre eles:

- Fumo;
- Consumo de bebidas alcoólicas;
- Obesidade;
- Estresse;
- Elevado consumo de sal;
- Níveis altos de colesterol;
- Falta de atividade física;

3.2 Variabilidade da Frequência Cardíaca

A Variabilidade da Frequência Cardíaca (VFC) descreve as oscilações dos intervalos entre ondas R do Eletrocardiograma (ECG) e, segundo[18] constitui um potente e independente indicador de mortalidade cardiovascular. Estas alterações na frequência cardíaca são normais e esperadas, já que o coração não é uma máquina e tem a habilidade de responder aos múltiplos estímulos fisiológicos e ambientais, como a respiração, o exercício físico, estresse mental, entre outros.[3]

O ECG é o método mais básico e acessível de analisar a VFC de uma pessoa, principalmente por ser não invasivo e fácil de usar[19]. É o procedimento mais utilizado para auxiliar o diagnóstico das doenças cardíacas[20]. Também é definido por [21] como de fácil manuseio, reprodutível e de baixo custo operacional. Através do resultado do eletrocardiograma, o profissional da área de saúde consegue verificar a VFC do paciente, entre outros fatores relacionados a saúde cardiovascular. Um eletrocardiograma normal pode ser visto em 3.1.

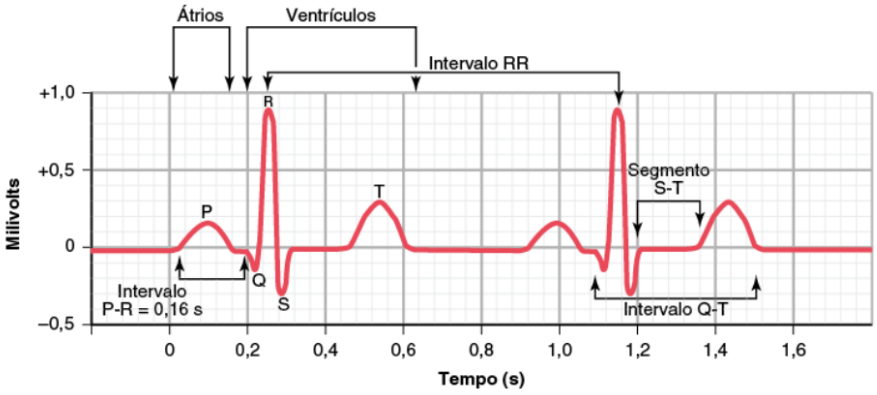


Figura 3.1: Dois batimentos cardíacos em um eletrocardiograma normal. Fonte: Guyton & Hall Tratado de Fisiologia Médica

Outro conceito importante é o intervalo RR. O intervalo RR é o período entre a repetição das ondas R, como observamos na figura 3.1. Sendo assim, um intervalo RR de 700ms representa que se passaram 700 milissegundos entre um batimento cardíaco e o batimento seguinte. Quanto mais lentos forem os batimentos cardíacos do paciente, maior será o intervalo RR. O inverso também se aplica: quanto mais rápidos forem os batimentos cardíacos do paciente, menor será o intervalo RR.

Um dos desafios no treinamento da Rede Neural Convolucional para diagnosticar hipertensos é que após o uso de medicamentos a tendência é que a Variabilidade da Frequência Cardíaca do hipertenso fique mais semelhante ao de um normotenso[4]. Este é, obviamente, justamente o propósito do medicamento, mas para fins de treinamento da CNN é uma dificuldade a mais, já que o efeito do medicamento vai retirar - ou ao menos diminuir - as características da hipertensão na VFC, conforme pode ser observado nas figuras 3.2 e 3.3. Sendo assim, seria de grande valia dados de hipertensos recém diagnosticados, ou

seja, ainda não medicados, para compor o treinamento da CNN tendo em vista que no diagnóstico de novos hipertensos, teoricamente o paciente ainda não faz uso de nenhuma medicação. Obter estes dados é díficil, principalmente pela questão ética. Todo indivíduo tem direito a tratamento médico.

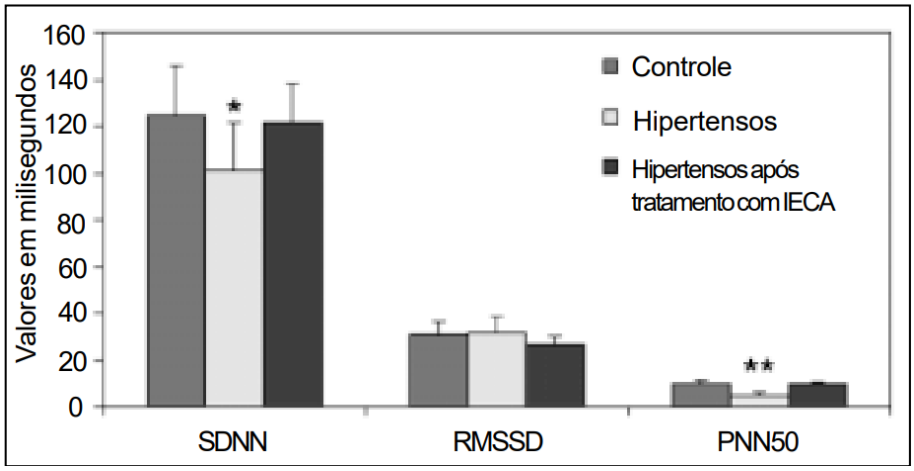


Figura 3.2: Comparação em índices da VFC em hipertensos antes e depois do uso de medicamentos no domínio do tempo. Fonte: [4]

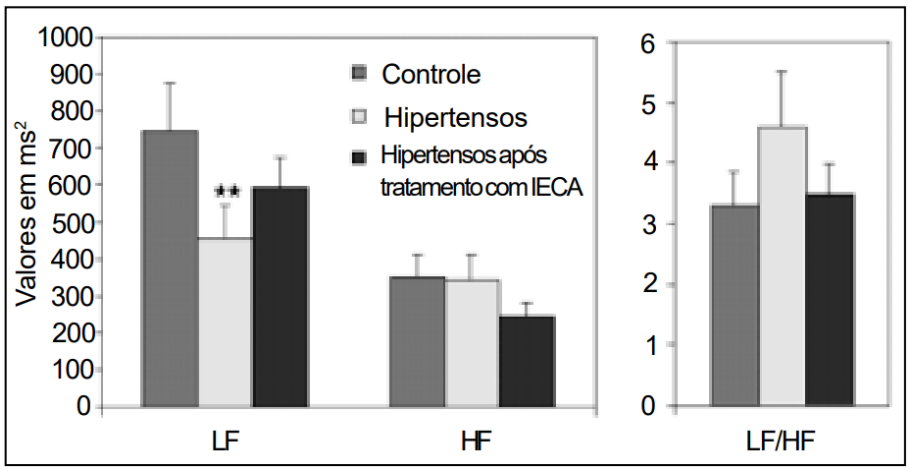


Figura 3.3: Comparação em índices da VFC em hipertensos antes e depois do uso de medicamentos no domínio da frequência. Fonte: [4]

3.3 Transformada Wavelet

Transformadas matemáticas são úteis quando queremos informações que não estão visíveis - ou não estão facilmente visíveis - na forma original do sinal. As vezes, uma informação que não está visível no domínio do tempo está visível no domínio da frequência[22]. A Transformada Wavelet é aplicada em um sinal para extrair informações no domínio da frequência e do tempo. O fato de extrair as informações também no domínio do tempo é

a principal vantagem em comparação à Transformada de Fourier.
A Transformada de Fourier é mais antiga que a Transformada Wavelet. É uma das transformadas mais amplamente utilizadas no processamento de sinais e é comumente contextualizada nos estudos relacionados à Wavelet. A transformada de Fourier fornece apenas informações da frequência, mas não nos diz quando, no domínio do tempo, estas frequências existem. Isto faz a transformada de Fourier ideal e recomendada para sinais estacionários, ou seja, quando a variação no sinal é constante. Como vimos na definição da VFC, isto não se aplica ao sinal do coração representado através do eletrocardiograma.
A Transformada Wavelet tem sido utilizada de maneira satisfatória em vários estudos, desde sua aplicação no sinal da voz humana para identificar doenças relacionadas ao sistema vocal, no processamento de imagens e até na predição de séries econômicas, como visto em [23].
[24] define que a Transformada Wavelet de uma funcão pertencente ao espaco das funções quadraticamente integráveis, $L^2\{\mathbb{R}\}$, é sua decomposição numa base formada por expansões/compressões e translações de uma única função-mãe $\psi(t)$, chamada de wavelet.
De maneira resumida e direta, consiste em passar o sinal original em vários filtros passa alta e passa baixa, que filtra as altas frequências ou baixas frequências de um sinal. Esse procedimento é repetido [22]. Um exemplo prático apresentado por [22] supõe a existência de um sinal que tem frequências de até 1000Hz. Em um primeiro momento este sinal é dividido em duas partes passando o sinal nos filtros de passa altas e passa baixas que resulta em duas versões diferentes do mesmo sinal, sendo uma parte do sinal que corresponde a 0-500 Hz (passa baixas) e a outra parte correspondente a 500-1000 Hz (passa altas). Então é usada uma das duas partes (normalmente a parte do passa baixas) ou ambas, e repete o processo. Essa operação é chamada decomposição. Assumindo que a parte usada foi a do passa baixas, agora teremos 3 conjuntos de dados, cada uma correspondente ao mesmo sinal nas frequências 0-250 Hz, 250-500 Hz e 500-1000 Hz. Esse processo é repetido até o sinal ser decomposto a um nível predefinido. Se todas as partes forem reunidas e plotadas em um gráfico 3-D, teremos o tempo em um eixo, frequência no segundo e amplitude no terceiro.
Uma importante propriedade da Tranformada Wavelet é o escalamento. Escalar uma Wavelet significa simplesmente dilatá-la ou contrai-la[25]. Quanto menor o fator de escala, mais contraída será a Wavelet, e vice-versa. Com isso, temos que no caso do fator de escala ser baixo, a Wavelet será contraída e consequentemente teremos maior precisão em detalhes que mudam rapidamente, que são os sinais de alta frequência. Já para um alto fator de escala, a Wavelet é dilatada e temos maior precisão em detalhes que mudam vagarosamente, que são os sinais de baixa frequência[25]. O fator de escala é normalmente representado pela letra a.
Outra importante propriedade é a translação. Transladar uma Wavelet significa movê-la horizontalmente (eixo das abcissas)[25].
Há 2 tipos de Transformadas Wavelets: a contínua e a discreta. Baseado no resultados dos estudos de [7] e [8], que seguem a mesma linha de pesquisa, este estudo também aplicará a transformada contínua. A wavelet mãe utilizada é a Ricker, comumente chamada de "Mexican Hat" devido ao seu formato.

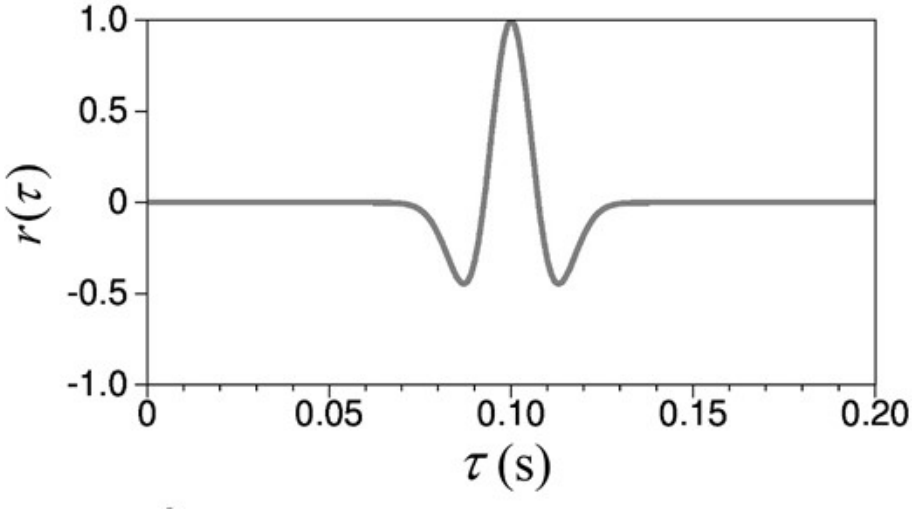


Figura 3.4: Wavelet Ricker (Mexican Hat). Fonte: [26]

3.4 Rede Neural Convolucional

A Rede Neural Convolucional, também chamada de *Convolutional Neural Network* (CNN), é um tipo de Rede Neural Artificial. [27] define a Rede Neural Artificial como um sistema computacional de processamento que é inspirada na maneira que os sistemas nervosos biológicos (como por exemplo, o cérebro) funcionam. Ainda segundo [27], a principal diferença entre a Rede Neural Convolucional e a Rede Neural Artificial é que a CNN é principalmente usada no campo de reconhecimento de padrões dentro de imagens, o que permite configurá-la mais facilmente e inserir *features* específicas para imagens.
A Rede Neural Convolucional é composta por neurônios - também chamados de "nós" que se otimizam durante a aprendizagem. Estes neurônios são organizados em três dimensões: altura, largura e profundidade.
A CNN é composta por três tipos de camadas: *convolutional*, *pooling* e *fully-connected*. As duas primeiras são responsáveis pela extração de *features*, enquanto a terceira mapeia as *features* em um formato de saída, como por exemplo, uma classificação [28].
Features podem ser descritas como características do modelo a ser processado. Por exemplo, se uma CNN fosse aprender dados para classificar seres humanos em homem ou mulher, características físicas poderiam ser *features* que influenciariam no resultado da classificação. No caso do aprendizado em imagens, normalmente as *features* são relacionadas aos pixels das imagens: sua cor e os pixels em sua volta. [29] define que o principal objetivo da camada convolucional é detectar *features* visuais em imagens, como bordas, linhas, alternância de cores, etc., o que é uma propriedade bem interessante, já que uma característica aprendida em um ponto da imagem pode ser reconhecida em um outro ponto distinto. Isto difere em relação às Redes Neurais apenas de camadas *fully-connected*, pois a rede teria que aprender todo o padrão novamente caso a característica aparecesse em um ponto distinto do que já foi aprendido.
Além disso, as camadas convolucionais podem aprender padrões complexos devido a sua capacidade de aprender hierarquias espaciais de padrões preservando os relacionamentos

espaciais. Exemplificando: Uma primeira camada convolucional aprende elementos básicos e uma segunda camada aprende padrões compostos por esses elementos básicos[29].
O resultado de saída da camada convolucional serve como dado de entrada para a camada *pooling*, que aparece sempre após a operação de convolução. Uma definição curta para o objetivo dessa camada é que ela simplifica a informação obtida anteriormente pela camada convolucional[29]. Existe mais de uma maneira de fazer o *pooling* mas a mais tradicional é o *max_pooling*, onde é mantido apenas o maior valor de uma matriz - o tamanho dessa matrix é definida na arquitetura da CNN. Pode-se observar na figura 3.5 como uma matriz 12x12 passa a ser representada em uma matriz 6x6 sem perder sua essência.

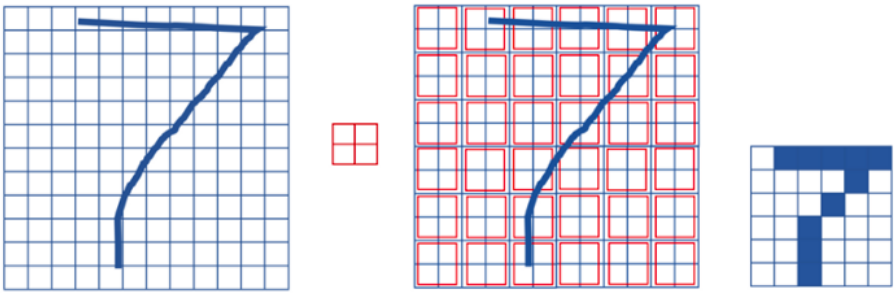


Figura 3.5: Exemplo de operação max-pooling. Fonte: [29]

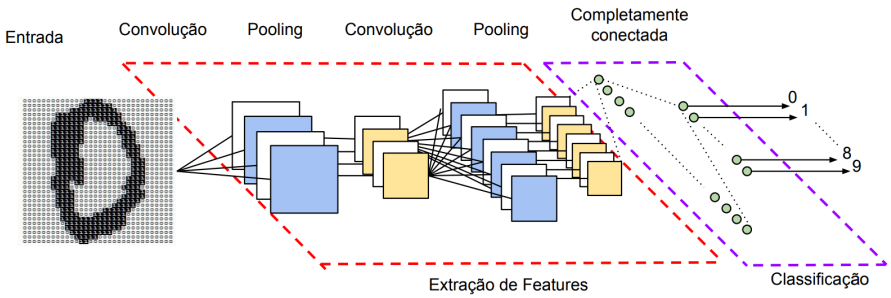


Figura 3.6: Exemplo de estrutura de CNN. Fonte: [30]

3.5 Interpolação por Spline Cúbica

A interpolação, de uma maneira simplista, consiste em determinar o valor de um ponto dentro de um gráfico ou tabela a partir de valores conhecidos de outros pontos. No contexto deste trabalho, a interpolação foi utilizada para gerar novas amostras para substituir valores *outliers*.
Existem várias formas de interpolação. A utilizada neste trabalho foi a Interpolação por Spline Cúbica. Ela leva este nome por usar polinômios de grau 3. Esta técnica é utilizada também no software Kubios, que é um dos softwares de referência na área de análise da VFC. A maior vantagem da Interpolação por Spline Cúbica em comparação com outros métodos mais simples de interpolação é o fato desta gerar curvas mais suaves.

Capítulo 4

Materiais e Métodos

Os arquivos com sinal RR que montam a base de dados utilizada para treinar a Rede Neural Convolucional foram recebidos da Universidade de Brasília. As coletas foram realizadas por meio do cardiofrequencímetro *Polar Wearlink*, que é um sensor acoplado a uma cinta elástica que é posicionada no tórax do paciente. A coleta foi realizada em março de 2018, tendo sido aprovada pelo Comite de Ética da Faculdade de Ciências da Saúde (FS) da Universidade de Brasília (CAAE 38386714.8.0000.0030).
Os critérios de inclusão foram:

- ser do sexo masculino;
- ter entre 18 e 25 anos;
- praticantes ou não de esportes físicos.

Já os critérios de exclusão foram:

- portador de doenças cardiorrespiratórias;
- distúrbios hormonais e/ou metabólicos;
- tabagismo;
- etilismo frequente;
- usuário de drogas ou medicamentos regulares que pudessem influenciar nas respostas cardiovasculares durante a coleta da VFC.

Foram utilizadas amostras de 2 grupos de pessoas: saudáveis e hipertensos. Estes grupos juntos somam 115 arquivos (65 saudáveis e 50 hipertensos). Para ambos os grupos, foram utilizadas apenas as amostras onde os pacientes estavam na posição supino (deitado) e em repouso, resultando na quantidade de 87 arquivos (65 saudáveis e 22 hipertensos). Durante a análise dos índices, que será descrita adiante, foi possível perceber que 7 arquivos de pessoas saudáveis estavam duplicados. Após eliminá-los, ficamos então com 80 (58

saudáveis e 22 hipertensos). Apesar de haver amostras de pessoas saudáveis em exercício, não foram utilizadas pois o exercício gera alteração na VFC. O mesmo critério justifica a não utilização de amostras de pessoas na posição em pé e sentado.

Capítulo 5

Desenvolvimento

5.1 Filtro

O primeiro ponto a ser resolvido é o problema da existência de *outliers*. Foi desenvolvido um algoritmo para realizar a filtragem.
Este algoritmo varre todas as amostras de um sinal RR buscando por *outliers*. Como não existe uma representação matemática capaz de identificar 100% dos *outliers* foi criada uma variável que recebe um valor que representa a variação máxima permitida entre uma amostra RR e as amostras em sua volta. Com os arquivos que temos atualmente para treinar a Rede Neural Convolucional, a variação permitida definida nesta variável foi de 130, que representam 130 milissegundos. Com este valor, foi possível perceber a remoção da grande maioria dos *outliers* de fácil identificação visual e ao mesmo tempo poucas ocorrências de remoção de amostras que parecem ser variações reais na frequência cardíaca de um paciente, conforme pode ser visto na figura 5.1. Também foi testado definir essa variação permitida com 200ms, conforme definição feita em [7], mas desta vez vários *outliers* eram tratados como amostras válidas e não eram substituídas: os *outliers* visualmente destoaram do padrão das amostras ao seu redor.

Também foi possível verificar um comportamento satisfatório do algoritmo na substituição de *outliers* em sequência (figura 5.2). Esta situação exige um pouco mais de cuidado na substituição, já que a maioria dos métodos de substituição de *outliers* baseiam-se nos valores ao redor do *outlier*, ou seja, por mais que um *outlier* seja corretamente identificado, a sua substituição pode gerar um valor inapropriado caso leve em consideração valores de outros *outliers* no cálculo do novo valor.

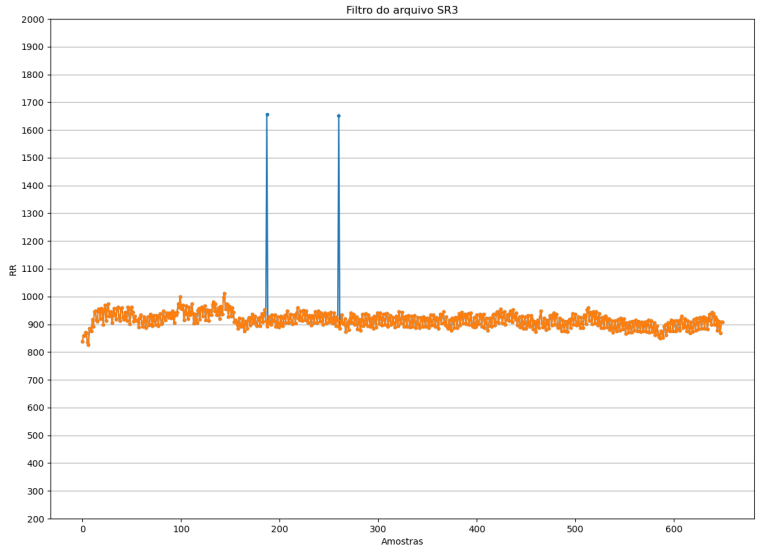


Figura 5.1: Filtragem de arquivo com *outlier* isolado. Na cor azul o sinal original e na cor laranja o sinal filtrado. Fonte: Autor

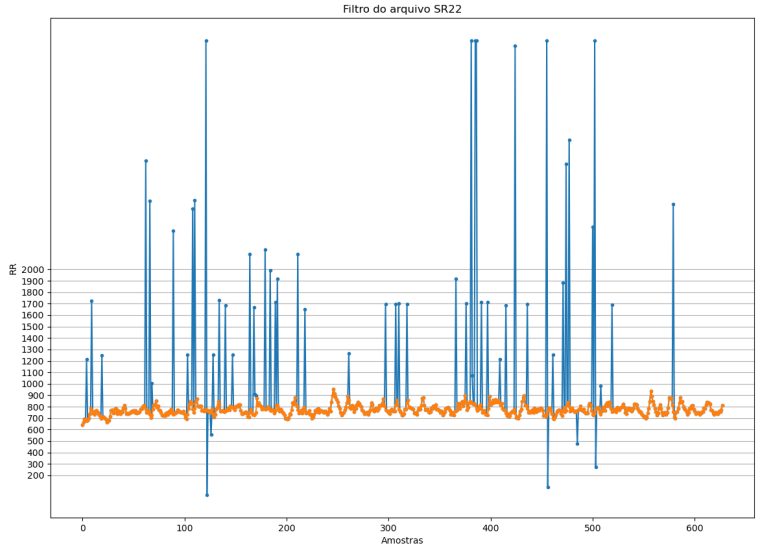


Figura 5.2: Filtragem de arquivo com *outliers* em sequência. Na cor azul o sinal original e na cor laranja o sinal filtrado. Fonte: Autor

Para a substituição dos *outliers*, foi utilizada a interpolação por *spline* cúbica. Este método apresentou bons resultados em simulações removendo amostras válidas e gerando novos valores próximos ao valor substituído. [7] concluiu que a simples remoção do outlier é menos eficiente do que substituí-lo com um sinal mais adequado. Para a interpolação

são usadas 8 amostras válidas existentes no sinal, tendo como regra geral a utilização de 4 amostras anteriores e 4 amostras posteriores ao *outlier*, sendo que essas amostras que são utilizadas para o cálculo são verificadas para que não sejam *outliers*, para evitar o problema citado anteriormente caso estes valores incorretos influenciem no novo valor gerado.
A escolha de utilizar 8 amostras válidas partiu da ideia inicial de replicar o comportamento do filtro do software Kubios e, como apresentou bom resultado, foi mantida.

5.2 Análise dos dados

Existem vários índices utilizados para análise da VFC. Em trabalhos relacionados é possível identificar alguns índices que são usados mais frequentemente. [13] define e lista os índices mais comuns e apropriados, além de recomendar o cenário de aplicação para cada um deles. Esta análise se divide em duas vertentes: linear e não linear. A análise linear ainda é subdividida em mais duas categorias: análise no domínio do tempo e análise no domínio da frequência.
Dentre todos os índices lineares, os que mais comumente são encontradas nos estudos da análise da VFC são:

- rMSSD: raiz quadrada média das diferenças sucessivas entre RR adjacentes
- NN50: quantidade de intervalos RR adjacentes com diferença de duração maior que 50ms
- pNN50: porcentagem dos intervalos RR adjacentes com diferença de duração maior que 50ms
- SDNN: Desvio padrão de todos os intervalos RR
- LF: Componentes de baixa frequência (entre 0,04 e 0,15Hz)
- HF: Componentes de alta frequência (entre 0,15 e 0,5Hz))
- LF/HF: Relação entre a baixa frequência e a alta frequência

Antes de realizar a transformada wavelet nos dados, vamos analisar se os dados são facilmente segregados através de algum desses índices. Para isso, foi utilizada a biblioteca hrv-analysis[31], que é um módulo em Python para análise da VFC em sinais RR e oferece de maneira facilitada o cálculo destes índices. Ao utilizar estes índices para análise no conjunto de dados que possuímos não foi possível identificar nenhum que pudesse satisfatoriamente separar os saudáveis dos hipertensos. Alguns deles até fazem uma maior concentração entre os saudáveis e hipertensos em extremos opostos, mas, ainda assim, sempre misturando-os significativamente . As figuras 5.3a e 5.3b ilustram a análise do sinal RR dos pacientes de acordo com o valor do índice rMSSD de maneira crescente e decrescente, respectivamente. É possível observar que os hipertensos aparecem com mais

Classificação	rmssd	sdnn	mean_nni
saudavel	1,80	11,74	555,97
hipertenso	5,01	22,36	612,82
saudavel	5,26	24,24	644,86
saudavel	7,69	21,70	555,07
saudavel	7,80	19,70	653,34
saudavel	8,18	29,40	667,67
hipertenso	8,52	38,75	613,51
hipertenso	9,82	43,09	712,78
saudavel	10,11	27,47	561,49
hipertenso	10,35	33,50	700,45
hipertenso	10,57	31,10	786,66
hipertenso	10,69	48,61	665,87
hipertenso	10,86	26,01	635,95
saudavel	11,06	29,79	733,38
saudavel	11,53	27,01	607,74
saudavel	12,46	28,34	605,02
hipertenso	13,08	45,75	832,76
saudavel	13,77	32,01	626,30
saudavel	13,77	37,41	777,67
hipertenso	13,97	68,15	775,86

(a) Pacientes ordenados por rMSSD crescente.

Classificação	rmssd	sdnn	mean_nni
hipertenso	61,35	76,97	1168252,00
saudavel	47,64	60,58	821,47
saudavel	46,01	46,43	1136,39
saudavel	45,33	73,24	869,31
hipertenso	45,25	99,70	956,03
saudavel	42,03	60,23	688,76
saudavel	40,45	42,46	917,59
saudavel	35,84	75,12	808,77
hipertenso	35,67	50,83	994,45
saudavel	35,50	52,61	874,57
saudavel	34,97	64,48	725,79
saudavel	34,94	64,54	725,76
saudavel	34,60	64,71	725,89
saudavel	34,35	60,57	847,50
saudavel	33,67	56,64	788,80
saudavel	33,46	52,93	680,25
hipertenso	32,72	49,67	889,23
saudavel	32,58	54,81	838,36
saudavel	30,98	57,14	717,18
saudavel	30,92	38,12	860,58

(b) Pacientes ordenados por rMSSD decrescente.

Figura 5.3: Análise do indíce rMSSD. Fonte: Autor

frequência nos menores rMSSD, entretanto, aparecem também entre os rMSSD mais altos, sendo inclusive uma pessoa hipertensa a pessoa com o maior valor.

O resultado da análise linear no domínio da frequência seguiu a mesma lógica, conforme pode ser visto nas imagens 5.4a e 5.4b, que ilustram a análise pelo índice de HF dos pacientes.

Na análise não linear se destaca o Plot de Poincaré [32]. Também não foi possível realizar uma separação confiável entre saudáveis e hipertensos nesta análise, conforme pode ser visto em 5.5

5.3 Criação de dados sintéticos

Além da filtragem, outro desafio para poder realizar o treinamento da CNN é a relativamente baixa quantidade de arquivos com sinais RR. Devido ao cenário de incertezas de quando seria possível obter mais arquivos com sinais de pacientes reais, principalmente pelo cenário da pandemia do COVID-19, o melhor caminho foi desenvolver um algoritmo para criar arquivos sintéticos a partir dos arquivos reais que já possuíamos.

5.3.1 Divisão dos arquivos em partes com 256 amostras

Primeiramente o algoritmo divide todos os arquivos originais em arquivos de 256 amostras, que foi o número definido como ideal para o treinamento -[18] define 256 amostras como o ideal para análise linear da variabilidade da frequência cardíaca. [32] estima que essas 256 amostras equivalem a cerca de 3,5 min de dados RR em uma frequência cardíaca de 70 batimentos/min. Foi verificado que nenhum arquivo veio com tamanho menor, então

Classificação	lf	hf	lf_hf_ratio
saudavel	3,09	0,86	3,60
hipertenso	102,84	3,83	26,84
saudavel	88,21	8,12	10,86
saudavel	444,53	11,84	37,56
saudavel	85,29	14,53	5,87
hipertenso	300,06	15,53	19,32
saudavel	66,28	20,46	3,24
saudavel	806,13	24,45	32,97
hipertenso	100,24	25,31	3,96
saudavel	86,51	28,23	3,06
hipertenso	200,97	29,60	6,79
hipertenso	105,08	31,05	3,38
saudavel	72,27	34,62	2,09
hipertenso	208,08	35,36	5,89
saudavel	108,11	39,89	2,71
hipertenso	182,54	39,90	4,58
hipertenso	229,03	40,74	5,62
hipertenso	238,39	40,81	5,84
saudavel	157,66	42,00	3,75
saudavel	237,36	45,57	5,21

(a) Pacientes ordenados por HF crescente. Fonte: Autor

Classificação	lf	hf	lf_hf_ratio
saudavel	1187,59	985,91	1,20
hipertenso	2863,30	716,46	4,00
saudavel	903,90	646,85	1,40
saudavel	2573,71	643,89	4,00
saudavel	2199,19	615,14	3,58
saudavel	2013,99	611,74	3,29
saudavel	2136,71	611,01	3,50
saudavel	337,09	499,96	0,67
saudavel	1722,80	496,62	3,47
saudavel	372,44	433,58	0,86
saudavel	750,28	429,28	1,75
saudavel	1022,83	406,38	2,52
saudavel	755,50	378,65	2,00
hipertenso	3126,89	361,66	8,65
saudavel	1233,08	345,88	3,56
saudavel	1495,53	337,17	4,44
saudavel	993,59	328,69	3,02
saudavel	1120,01	299,45	3,74
saudavel	753,66	274,34	2,75
saudavel	1646,60	271,92	6,06

(b) Pacientes ordenados por HF decrescente. Fonte: Autor

Figura 5.4: Análise do índice HF. Fonte: Autor

ratio_sd2_sd1	sd1	sd2	classificacao
1,53	19,62	29,94	saudavel
1,75	32,56	57,02	saudavel
1,84	28,62	52,79	saudavel
2,25	21,88	49,27	saudavel
2,34	33,71	78,76	saudavel
2,60	21,74	56,60	saudavel
2,66	25,26	67,29	hipertenso
2,68	29,74	79,81	saudavel
2,78	15,87	44,14	saudavel
2,79	25,12	70,03	saudavel
2,87	23,15	66,32	hipertenso
2,93	19,41	56,83	saudavel
3,00	23,68	71,01	saudavel
3,07	32,08	98,49	saudavel
3,10	18,18	56,38	saudavel
3,14	18,38	57,75	saudavel
3,21	23,83	76,48	saudavel
3,21	23,05	74,01	saudavel
3,22	15,77	50,79	saudavel

(a) Pacientes ordenados pela taxa de SD2/SD1 crescente. Fonte: Autor

ratio_sd2_sd1	sd1	sd2	classificacao
13,03	1,27	16,55	saudavel
9,70	9,88	95,87	hipertenso
9,15	3,72	34,07	saudavel
9,04	6,03	54,47	hipertenso
9,03	7,56	68,33	hipertenso
8,87	3,54	31,43	hipertenso
8,72	6,94	60,54	hipertenso
7,12	5,79	41,18	saudavel
6,96	18,49	128,69	hipertenso
6,92	9,25	64,04	hipertenso
6,40	7,32	46,81	hipertenso
5,80	7,48	43,35	hipertenso
5,70	15,13	86,24	saudavel
5,55	5,44	30,20	saudavel
5,37	15,38	82,57	hipertenso
5,34	7,15	38,18	saudavel
5,34	9,74	52,00	saudavel
5,29	7,82	41,40	saudavel
5,15	14,32	73,73	saudavel

(b) Pacientes ordenados pela taxa de SD2/SD1 decrescente. Fonte: Autor

Figura 5.5: Divisão do arquivo original SR1. Fonte: Autor

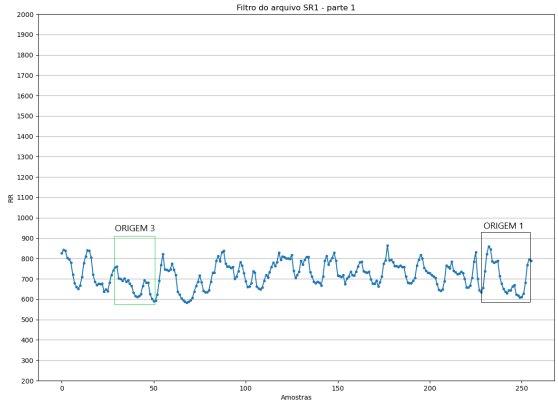
não houve problema neste sentido.

5.3.2 Preenchendo arquivos que ficaram com menos de 256 amostras

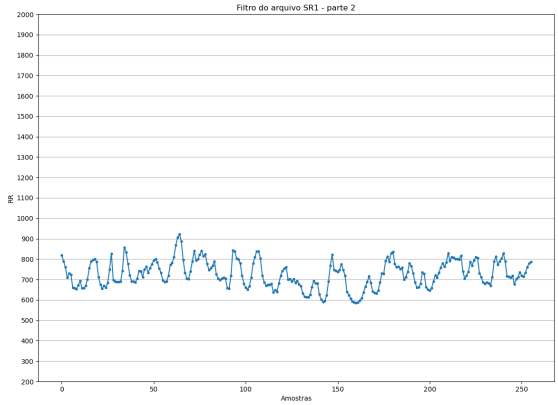
Após a divisão, alguns arquivos ficaram com tamanho menor do que 256. Exemplificando: se o arquivo original tem 696 amostras, então serão gerados 2 arquivos com 256 amostras e 1 arquivo com as 184 amostras restantes. Foi necessário preencher este arquivo que ficou menor até que ficasse exatamente com 256 amostras. Na figura 5.6 é possível ver uma representação visual deste exemplo, gerada pelo algoritmo.
Em um primeiro momento a solução desenvolvida foi um algoritmo que preenchia este arquivo menor apenas com dados de outros pacientes, não permitindo utilizar amostras do mesmo paciente. A intenção era fazer com que as amostras preenchidas não ficassem repetidas se comparadas com outros trechos de arquivos do mesmo paciente. Entretanto, por mais que eram selecionadas para preenchimento apenas amostras que tivessem um valor RR próximo à última amostra real do paciente, o resultado final era um sinal que não fazia sentido pois não se assemelhava ao sinal de uma pessoa real. Então o algoritmo foi alterado e a regra foi invertida: o preenchimento deve ser feito apenas com trechos de amostras do mesmo paciente. Desta forma o resultado ficou mais satisfatório, e por mais que ainda precise ser validado no resultado final do treinamento da CNN, visualmente o sinal já é mais semelhante a um sinal real.

5.3.3 Criação de novas amostras a partir dos arquivos existentes

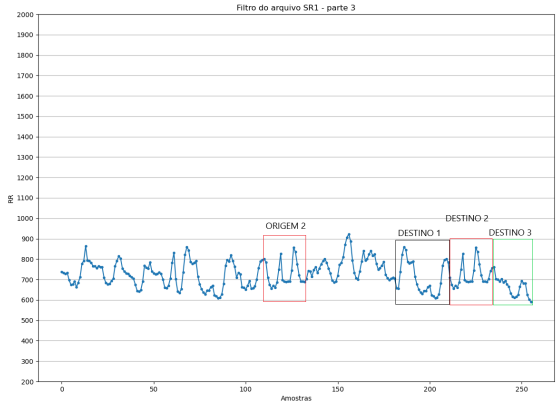
Esta primeira etapa de dividir os arquivos originais em arquivos menores com 256 amostras já elevou o número de arquivos para treinamento. Apesar de já ser um aumento significativo - mais que o triplo -, ainda não é considerado um bom número para ser utilizado no treinamento. Então foi adicionado mais um passo ao algoritmo para que a partir de cada um dos arquivos já prontos fosse retirado um pedaço aleatório com 25 amostras seguidas. A partir destas 25 amostras, o algoritmo segue a mesma ideia anterior, de completar os arquivos com sequências de amostras compatíveis do mesmo paciente. Na figura 5.7 é visualizado um arquivo totalmente criado pelo algoritmo. Este procedimento garantiu que o número de arquivos subisse para 328. Para a primeira avaliação da CNN, 328 arquivos foi considerado satisfatório.



(a) Primeiras 256 amostras retiradas do arquivo original SR1



(b) Segundo bloco de 256 amostras retiradas do arquivo original SR1



(c) Terceiro bloco de amostras do arquivo original SR1, contendo as amostras preenchidas pelo algoritmo

Figura 5.6: Divisão do arquivo original SR1. Fonte: Autor

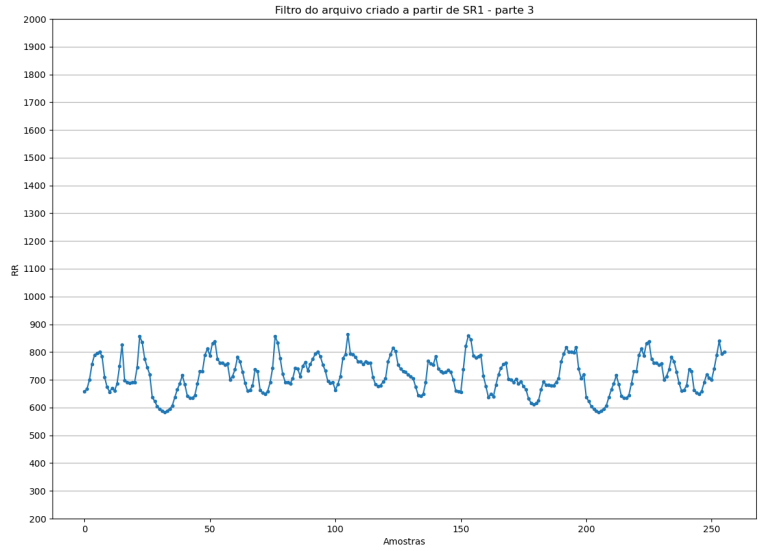


Figura 5.7: Visualização de arquivo totalmente criado a partir de uma sequência de amostras de um arquivo real. Fonte: Autor

5.3.4 Decisões técnicas na criação dos arquivos sintéticos

Seguem alguns detalhes técnicos do algoritmo utilizado para completar arquivos menores do que 256 amostras e criar arquivos sintéticos. Vários destes parâmetros podem ser alterados para obter um melhor desempenho na geração de arquivos:

- Para preencher um arquivo, é utilizado o valor da última amostra RR presente para encontrar outros intervalos que sejam compatíveis. Por exemplo, se a última amostra de um arquivo é 700, que equivale a 700ms, então o algoritmo vai varrer arquivos do mesmo paciente procurando alguma amostra que possua o mesmo valor de 700.
- Em complemento ao item anterior, quando não é encontrado o valor exato da última amostra, então o algoritmo faz uma busca mais flexível que agora irá buscar amostras dentro de um *range* de valores. Há uma variável que controla o *range* de diferença permitido para que um intervalo seja considerado compatível. Este *range* foi utilizado com o valor 20, que equivale a 20ms. Por exemplo, se a última amostra de um arquivo que precisa ser completado é 720, então o algoritmo vai varrer arquivos do mesmo paciente procurando amostras que estejam entre 700ms (20ms a menos que a última amostra) e 740ms (20ms a mais do que a última amostra). Caso não seja encontrada nenhuma amostra compatível na primeira busca flexível, então o algoritmo repete este passo e soma o valor da variável no *range* permitido. Desta forma, caso o processo precise ser repetido uma vez, o range será de 40ms (20 ms + 20ms). Se precisar repetir uma segunda vez o range será de 60ms (40ms +20ms) e assim por diante até que encontre um valor compatível.

- Número de amostras que são retiradas em bloco de um arquivo para preencher outro arquivo: 25. Isto quer dizer que sempre que uma amostra é considerada compatível para preencher um arquivo menor, então serão levadas as 25 amostras em sequência. As únicas exceções são quando um arquivo precisa de menos do que 25 para chegar à 256 amostras ou quando uma amostra compatível é encontrada a menos de 25 amostras do final do arquivo. Isto também está configurado como variável.

5.3.5 Transformada Wavelet Contínua

Através da Transformada Wavelet Contínua iremos transformar os sinais RR - que até aqui são arquivos .txt com uma sequência de números inteiros - em uma matriz que servirá de dados de entrada para a CNN. Foi utilizada a biblioteca Python pywavelet[33] para esta transformada. Uma ilustração de como é realizada esta transformada pode ser observada na figura 5.8.

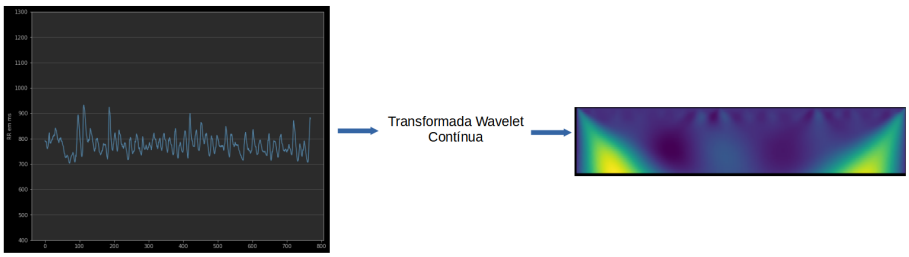
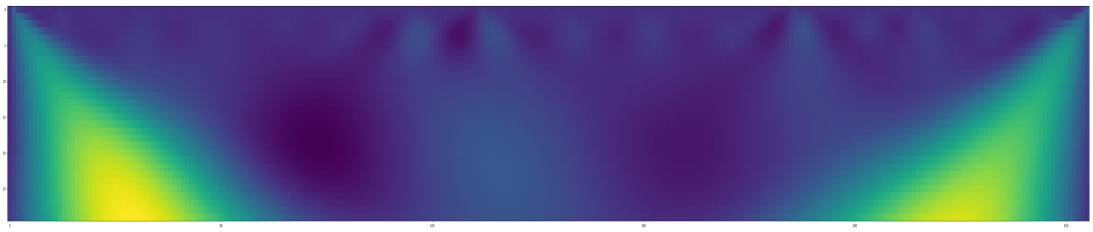


Figura 5.8: Ilustração da entrada e saída da Transformada Wavelet Contínua. Fonte: Autor

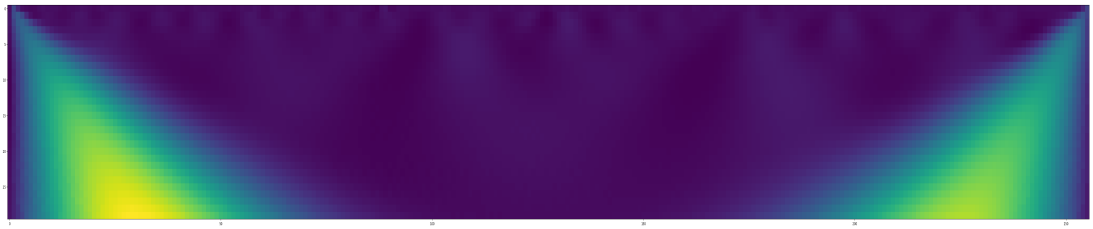
Este é o código que realiza esta transformada e gera a figura do resultado da Tranformada Wavelet.

```
import pywt
import numpy as np
import matplotlib.pyplot as plt

arquivo = open('caminhoDoArquivo')
conteudo = arquivo.readlines()
cwtCoefs, cwtFreqs = pywt.cwt(data=conteudo[:256],
    scales=np.arange(1,31), wavelet='mexh')

plt.figure(figsize=(80,100))
plt.imshow(cwtCoefs)
```

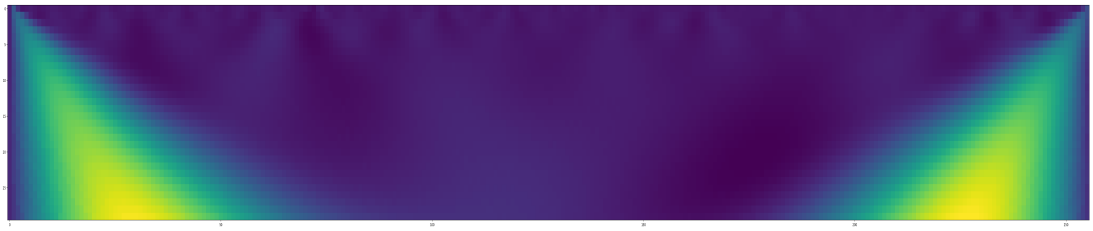
Como a Rede Neural Convolucional precisa de dados de entrada de um único tamanho, estamos passando pela transformada apenas as primeiras 256 amostras de cada arquivo. Nas figuras 5.9 e 5.10 é possível verificar alguns exemplos de saída da transformada para saudáveis e hipertensos, respectivamente.



(a) Transformada para o paciente saudável SR5. Fonte: Autor

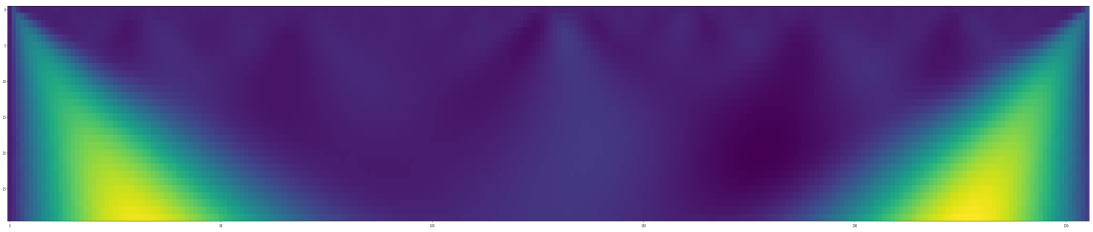


(b) Transformada para o paciente saudável SR6. Fonte: Autor

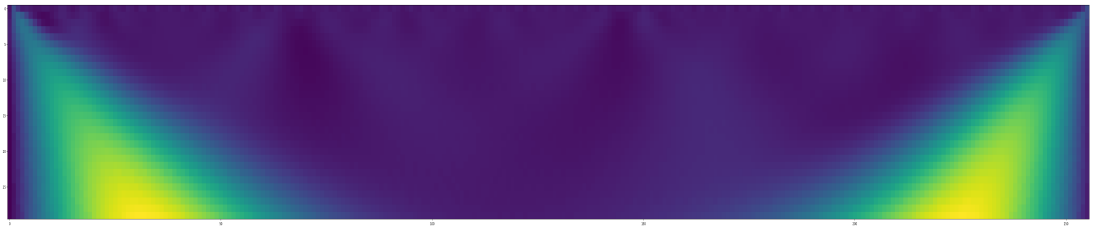


(c) Transformada para o paciente saudável SR7. Fonte: Autor

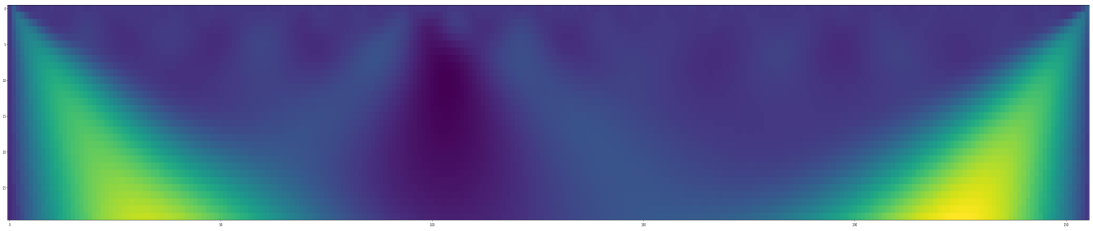
Figura 5.9: Resultado da Transformada Wavelet em pacientes saudáveis



(a) Transformada para o paciente hipertenso ES. Fonte: Autor



(b) Transformada para o paciente hipertenso JL. Fonte: Autor



(c) Transformada para o paciente hipertenso L-AR. Fonte: Autor

Figura 5.10: Resultado da Transformada Wavelet em pacientes hipertensos

5.3.6 Treinamento da Rede Neural Convolucional

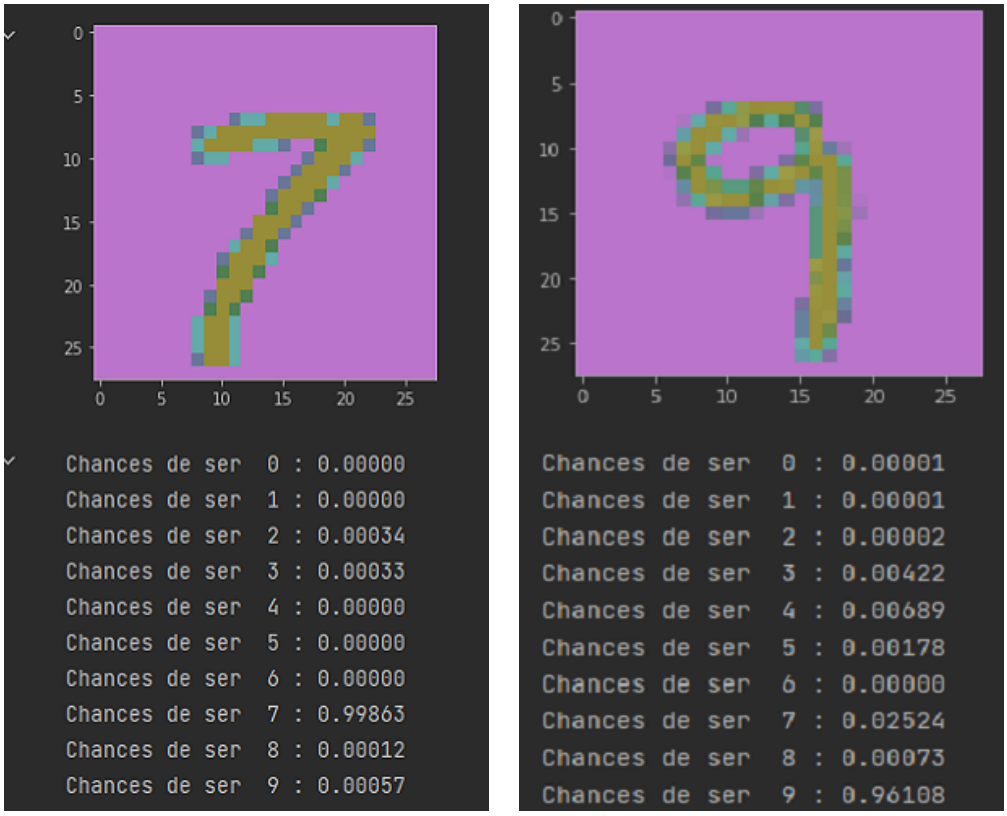
Como ponto de partida, foi replicada a CNN de [8], que apresenta a arquitetura abaixo:

```
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(30, 256, 1),
activation='relu', padding='same',
kernel_constraint=max_norm(3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(512, activation='relu',
kernel_constraint=max_norm(3)))
model.add(Dense(2, activation='softmax'))
model.compile(loss='categorical_crossentropy',
optimizer=SGD(learning_rate=0.01), metrics=['accuracy'])
model.summary()

model.fit(coefsMh, coefsLabel,
    epochs=3,
    batch_size=32,
    validation_data=(coefsMhT, coefsLabelT),
    shuffle=True)
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_5 (Conv2D)           (None, 30, 256, 32)       320

 max_pooling2d_5 (MaxPooling  (None, 15, 128, 32)      0
 2D)

 flatten_5 (Flatten)         (None, 61440)             0

 dense_10 (Dense)            (None, 512)               31457792

 dense_11 (Dense)            (None, 2)                 1026

=================================================================
Total params: 31,459,138
Trainable params: 31,459,138
Non-trainable params: 0
_________________________________________________________________
```

Figura 5.11: Resumo da arquitetura da CNN inicial. Composta por uma camada de convolução, seguida por uma camada de max pooling, uma camada flatten e duas camadas dense. Fonte: Autor

Para validar que esta CNN continua sendo precisa, foi utilizado o dataset MNIST[34], que é um dataset público que contém imagens de números escritos a mão. O resultado



(a) Predição para o número 7 (b) Predição para o número 9

Figura 5.13: Predições no dataset MNIST

foi de mais de 95% de acurácia (Figura 5.12), o que pode ser comprovado ao realizar a validação em alguns arquivos de teste (Figura 5.13).

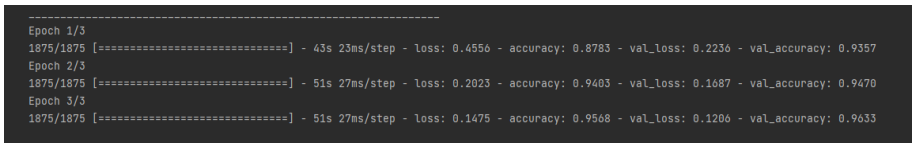


Figura 5.12: Resultado da CNN inicial para o dataset MNIST. Acurácia superior a 95%. Fonte: Autor

Para ter um conjunto de dados pelo menos semelhante ao utilizado no código original, foi adaptado o algoritmo para criar dados sintéticos através da *Shift Data Augmentation*[35]. Foram feitos 2 pequenos ajustes para adequar o código para o nosso cenário:

- Alteração no formato de entrada (input_shape) já que estamos trabalhando com 256 amostras e o código original trabalhava com 348;
- Alteração de unidade na última camada Dense, pois nós temos apenas duas possíveis saídas (saudável ou hipertenso) enquanto o código original tinha três (arritmia, hipertenso ou saudável);

A *Shift Data Augmentation* foi aplicada acrescentando um valor randômico de 1 a 3 para cada amostra dos arquivos originais. Esta etapa foi repetida 100 vezes para cada arquivo. O código e uma representação visual podem ser visualizados nas figuras 5.14 e 5.15.

```
for arquivoSaudavel in arquivosSaudaveis:
     arquivo = open(caminhoArquivosSaudaveis+'/'+arquivoSaudavel, 'r')
     conteudo = arquivo.readlines()

    for u in range(0 ,100):
        frequencies = []  #frequencies recebe todos os dados do arquivo e converte em int
        for n in conteudo:
         frequencies.append(int(n)+ random.randint(1,3))
        result = open('caminhoDestino'+arquivoSaudavel+' - '+str(u)+".txt", 'w')
        for i in range(len(frequencies)):
            result.write(str(frequencies[i])+"\n")
        result.close()
```

Figura 5.14: Código onde a data augmentation é realizada. Fonte: Autor

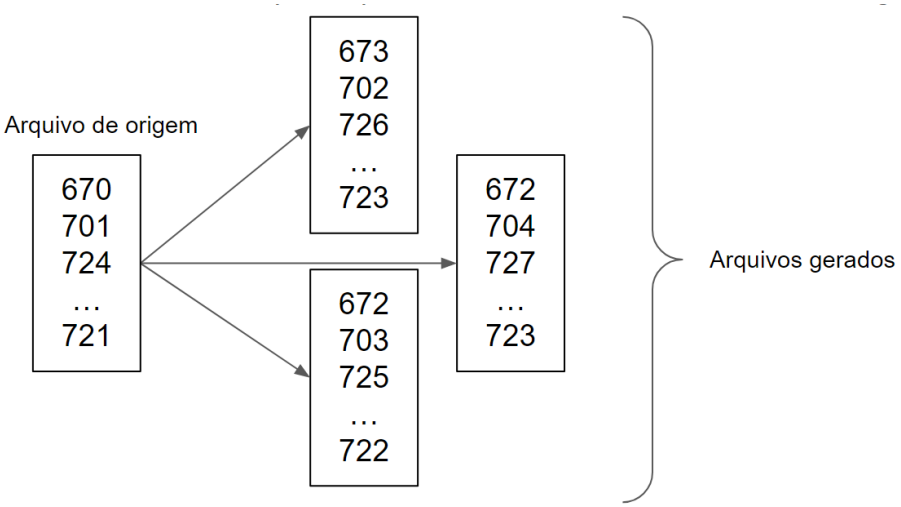


Figura 5.15: Ilustração de como a data augmentation gera novos arquivos. Fonte: Autor

Inicialmente a ideia era utilizar esta técnica de *data augmentation* somente para validar a CNN e para o treinamento seriam utilizados os dados sintéticos criados em em 5.3.3. Porém, durante a validação da Rede Neural Convolucional foi possível observar um resultado muito melhor utilizando da mesma técnica de *data augmentation* empregada por [8]. Ao utilizar os dados criados em 5.3.3, a acurácia ficou em torno de 60%. Mesmo tentando otimizar a Rede Neural (utilização da camada dropout, aumento significativo no número de épocas, aumento no número de camadas convolucionais com maior número de filtros) ainda não foi possível obter um resultado satisfatório. Já com este conjunto de dados foi possível obter por volta de 80% de acurácia na CNN. O resultado não é exatamente igual ao obtido em [8] mas isto é esperado, até pelo fato dos dados serem diferentes.

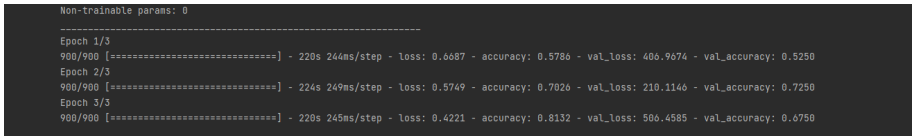


Figura 5.16: Resultado do treinamento com a CNN inicial. Fonte: Autor

Para testar na prática a acurácia desta CNN, foi realizada a classificação dos 80 arquivos originais. Como resultado, obtivemos um total de 67 acertos, cerca de 83,75% de acurácia. O número está de acordo com a acurácia alcançada pelo modelo ao fim do treinamento, que foi de 81,32%.

Classificação	Num. de entradas	Num. de acertos	% de acerto
Saudáveis	58	45	77,58
Hipertensos	22	22	100
Total	80	67	83,75

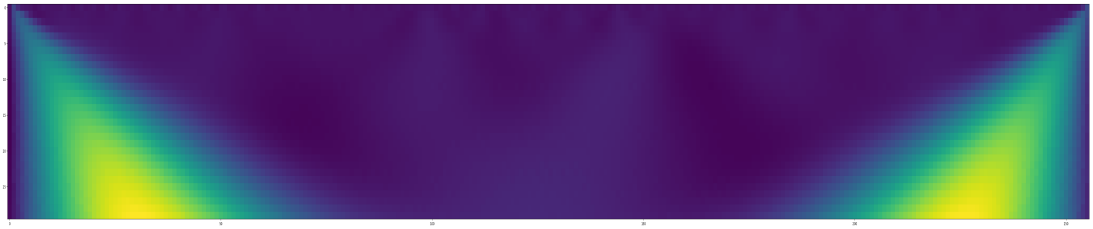
Tabela 5.1: Acurácia da Rede Neural Convolucional ao realizar a predição dos 80 arquivos originais

Sensibilidade	Especificidade
77,58%	100%

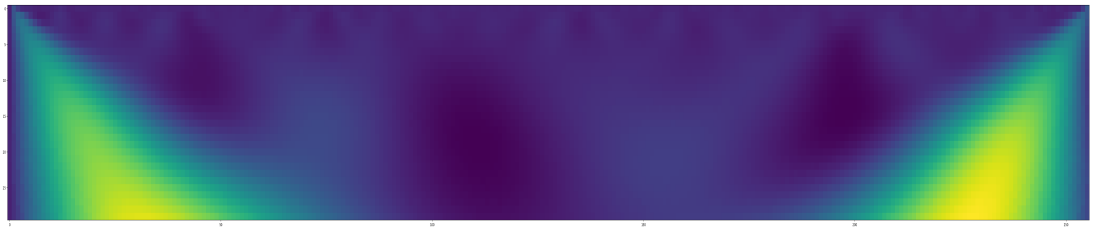
Tabela 5.2: Sensibilidade e especificidade ao realizar a predição dos 80 arquivos originais

Podemos supor que a CNN encontrou alguns padrões apresentados nos hipertensos para alguns pacientes saudáveis, e por isso classificou-os erroneamente. A figura 5.17 retrata o resultado da transformada para 2 desses saudáveis que foram classificados incorretamente.

Estes 80 arquivos originais não foram utilizados no treinamento, porém deram origem a todos os arquivos que foram utilizados. A fim de validar a acurácia com entradas totalmente estranhas à CNN, vamos utilizar alguns arquivos de pacientes saudáveis que foi possível obter na base de dados aberta disponível em [36]. Foram usados os dados de 3 adultos - entre 17 e 39 anos - saudáveis e a CNN fez a classificação correta para todos os casos. Foram usados dados apenas destes 3 adultos pois são os únicos que estão em uma faixa de idade semelhante aos pacientes que tiveram seus dados coletados neste estudo. Não foi possível encontrar uma base de dados pública com dados de hipertensos para realizar uma validação semelhante.



(a) Transformada para o paciente saudável SR3. Fonte: Autor



(b) Transformada para o paciente saudável SR4. Fonte: Autor

Figura 5.17: Resultado da Transformada Wavelet em pacientes saudáveis que foram classificados como hipertensos

Capítulo 6

Considerações

A versão final da Rede Neural Convolucional apresentou uma boa acurácia, superior a 80%. Porém, a falta de dados para validar este resultado é um ponto negativo que deve ser levado em consideração. Com os dados disponíveis ela foi bem sucedida em classificar todos os hipertensos.
A análise da Variabilidade da Frequência Cardíaca através de índices lineares e não lineares não apresentou resultado determinante para indicar se um paciente é saudável ou hipertenso. Isto é conflitante com outros trabalhos onde foi possível observar claramente uma VFC menor para pacientes hipertensos em relação a normotensos. Uma hipótese sobre este fato é que talvez os dados utilizados neste estudo tenha origem em um grupo de pacientes muito heterogêneo. Por exemplo, se no grupo de pessoas saudáveis temos atletas e sedentários, já teremos dentros deste mesmo grupo comportamento diferente na VFC. Um outro parâmetro que pode ter influenciado na análise dos índices é o uso de medicamentos por hipertensos. O hipertenso já diagnosticado normalmente faz uso de medicamentos para controlar sua pressão arterial e isto acaba tornando os dados de sua VFC mais semelhantes com os de uma pessoa normal. Para estes casos, seria de grande valia ter uma base de dados de pessoas recém diagnosticadas com hipertensão e que, portanto, ainda não façam uso de medicamentos. Isto se faz importante pois espera-se que este padrão de VFC de hipertenso não medicado seja o mesmo padrão que será encontrado ao fazer a análise dos dados de hipertensos ainda não diagnosticados.
Algo que não foi abordado neste trabalho mas que vale a pena investigar mais a fundo é a seleção da faixa do sinal RR que será utilizada para treinamento/predição na CNN. Em geral a Transformada Wavelet Contínua funciona bem para sinais não estacionários e não deveria haver preocupação em aproveitar apenas "janelas"do sinal, mas para treinar a Rede Neural Convolucional é necessário que todos os dados de entrada tenham o mesmo tamanho. Neste trabalho foram utilizadas as primeiras 256 amostras para cada arquivo. O ideal seria encontrar o intervalo de 256 amostras que melhor represente o sinal como um todo e não uma parte fixa para todos os arquivos.
Inicialmente a ideia era utilizar como entrada para o treinamento da CNN os dados sintéticos criados em 5.3.3. Porém, durante a validação da Rede Neural Convolucional foi possível observar um resultado muito melhor utilizando da mesma técnica de *data augmentation* empregada por [8]. Com essa técnica, além de gerar um número muito maior

de arquivos (foram gerados cerca de 32 mil arquivos, considerando dados de treinamento e de validação), também apresentou uma acurácia maior na CNN. Ao utilizar os dados criados em 5.3.3, a acurácia ficou em torno de 60%. Mesmo tentando otimizar a Rede Neural (utilização da camada dropout, aumento significativo no número de épocas, aumento no número de camadas convolucionais com maior número de filtros) ainda não foi possível obter um resultado suficientemente bom. Também não deve ser ignorada a grande diferença na quantidade de arquivos. Não deve ser descartada a hipótese de que é possível otimizar a estrutura da Rede Neural Convolucional para que consiga obter boa acurácia mesmo com estes dados. Uma outra possibilidade seria refatorar o algoritmo de criação de arquivos para que gerasse uma quantidade próxima aos 30 mil arquivos e então avaliar novamente como a CNN performa no treinamento.

Bibliografia

[1] Ministério da Saúde. *Hipertensão (pressão alta)*. https://www.gov.br/saude/pt-br/assuntos/saude-de-a-a-z/h/hipertensao-pressao-alta-1#:~:text=O%20que%20%C3%A9%20hipertens%C3%A3o,(ou%2014%20por%209).. [Disponível em 17/05/2022]. 2020.

[2] OPAS. *Doenças cardiovasculares*. https://www.paho.org/pt/topicos/doencas-cardiovasculares, (acessado em 29/11/2020). 2020.

[3] Luiz Carlos Marques Vanderlei et al. "Noções básicas de variabilidade da frequência cardíaca e sua aplicabilidade clínica". pt. Em: *Brazilian Journal of Cardiovascular Surgery* 24 (jun. de 2009), pp. 205–217. ISSN: 0102-7638. URL: http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0102-76382009000200018&nrm=iso.

[4] Antônio da Silva Menezes Júnior, Humberto Graner Moreira e Murilo Tavares Daher. "Análise da variabilidade da freqüência cardıaca em pacientes hipertensos, antes e depois do tratamento com inibidores da enzima conversora da angiotensina II". Em: *Arquivos brasileiros de cardiologia* 83 (2004), pp. 165–168.

[5] Jagmeet P Singh et al. "Reduced heart rate variability and new-onset hypertension: insights into pathogenesis of hypertension: the Framingham Heart Study". Em: *Hypertension* 32.2 (1998), pp. 293–297.

[6] Renan de Paula Rosa. "Método de classificação de pragas por meio de rede neural convolucional profunda". Em: *Dissertação (Mestrado Computação Aplicada) - Universidade Estadual de Ponta Grossa, Ponta Grossa* (2018). URL: http://www.sciencedirect.com/science/article/pii/S0010482518302713.

[7] Larissa Fernandes Maraes. "Tratamento e análise de sinais de variabilidade da frequência cardíaca". Em: *Trabalho de Conclusão de Curso (Engenharia de Computação) - Universidade Federal de Mato Grosso do Sul* (2018).

[8] Ulysses de Castro Goncalves. "Aplicacação de rede neural convolucional para o estudo da variabilidade da frequência cardíaca". Em: *Trabalho de Conclusão de Curso (Engenharia de Computação) - Universidade Federal de Mato Grosso do Sul* (2019).

[9] Filipe Sampaio de Mello. "Rede neural convolucional em ambiente web para o estudo da variabilidade da frequência cardíaca". Em: *Trabalho de Conclusão de Curso (Engenharia de Computação) - Universidade Federal de Mato Grosso do Sul* (2020).

[10] Paula F Martinez e Marina P Okoshi. *Variabilidade da Frequência Cardıaca em Pacientes com Diabetes e Hipertensão*. 2018.

[11] Walter Vargas e Katya Rigatto. "História Familiar de Hipertensão Prejudica o Balanço Autonômico, mas não a Função Endotelial em Jovens Jogadores de Futebol". Em: *Arquivos Brasileiros de Cardiologia* 115 (2020), pp. 52–58.

[12] Erman Cilsal. "In newly diagnosed hypertensive children, increased arterial stiffness and reduced heart rate variability were associated with a non-dipping blood pressure pattern". Em: *Revista Portuguesa de Cardiologia* 39.6 (2020), pp. 331–338.

[13] Task Force of the European Society of Cardiology the North American Society of Pacing Electrophysiology. "Heart Rate Variability". Em: *Circulation* 93.5 (1996), pp. 1043–1065. DOI: 10.1161/01.CIR.93.5.1043. eprint: https://www.ahajournals.org/doi/pdf/10.1161/01.CIR.93.5.1043. URL: https://www.ahajournals.org/doi/abs/10.1161/01.CIR.93.5.1043.

[14] Sebastian Wallot et al. "Using complexity metrics with RR intervals and BPM heart rate measures". Em: *Frontiers in physiology* 4 (2013), p. 211.

[15] Jeffrey J Goldberger et al. "Comparison of the physiologic and prognostic implications of the heart rate versus the RR interval". Em: *Heart Rhythm* 11.11 (2014), pp. 1925–1933.

[16] Cláudia Maria Salgado e João Thomaz de Abreu Carvalhaes. "Hipertensão arterial na infância". Em: *Jornal de Pediatria* 79 (2003), S115–S124.

[17] Luiz Aparecido Bortolotto. "Hipertensão arterial e insuficiência renal crônica". Em: *Rev Bras Hipertens* 15.3 (2008), pp. 152–5.

[18] VRFS Marães. "Frequência cardıaca e sua variabilidade: análises e aplicações". Em: *Revista andaluza de Medicina del Deporte* 3.1 (2010), pp. 33–42.

[19] Özal Yıldırım et al. "Arrhythmia detection using deep convolutional neural network with long duration ECG signals". Em: *Computers in Biology and Medicine* 102 (2018), pp. 411–420. ISSN: 0010-4825. DOI: https://doi.org/10.1016/j.compbiomed.2018.09.009. URL: http://www.sciencedirect.com/science/article/pii/S0010482518302713.

[20] José Carlos Nicolau et al. "Diretriz de interpretação de eletrocardiograma de repouso". Em: *Arquivos Brasileiros de Cardiologia* 80 (2003), pp. 1–18.

[21] José Feldman e Gerson P Goldwasser. "Eletrocardiograma: recomendações para a sua interpretação". Em: *Revista da SOCERJ* 17.4 (2004), pp. 251–256.

[22] Robi Polikar et al. *The wavelet tutorial*. 1996.

[23] Anderson da Silva SOARES. "Predição de séries temporais econômicas por meio de redes neurais artificiais e transformada Wavelet: combinando modelo técnico e fundamentalista". Diss. de mestr. Escola de Engenharia de São Carlos, Universidade de São Paulo, 2008. Dissertação de Mestrado em Processamento de Sinais de Instrumentação, 2008.

[24] Sergio L. Netto. Paulo S. R. Diniz Eduardo A. B. da Silva. *Processamento digital de sinais [recurso eletrônico] : projeto e análise de sistemas*. Bookman, 2014.

[25] Elcio Franklin de Arruda. "Análise de distúrbios relacionados com a qualidade da energia elétrica utilizando a transformada Wavelet". Tese de dout. Universidade de São Paulo, 2003.

[26] Yanghua Wang. "The Ricker wavelet and the Lambert W function". Em: *Geophysical Journal International* 200.1 (2015), pp. 111–115.

[27] Keiron O'Shea e Ryan Nash. *An Introduction to Convolutional Neural Networks*. 2015. arXiv: 1511.08458 [cs.NE].

[28] Rikiya Yamashita et al. "Convolutional neural networks: an overview and application in radiology". Em: *Insights into imaging* 9.4 (2018), pp. 611–629.

[29] Jordi Torres. *Convolutional Neural Networks for Beginners using Keras TensorFlow 2*. https://towardsdatascience.com/convolutional-neural-networks-for-beginners-using-keras-and-tensorflow-2-c578f7b3bf25. [Disponível em 17/05/2022]. 2022.

[30] Ana Caroline Gomes Vargas, Aline Paes e Cristina Nader Vasconcelos. "Um estudo sobre redes neurais convolucionais e sua aplicação em detecção de pedestres". Em: *Proceedings of the xxix conference on graphics, patterns and images*. Vol. 1. 4. sn. 2016.

[31] Robin Champseix. *hrvanalysis*. https://aura-healthcare.github.io/hrv-analysis/. [Disponível em 17/05/2022]. 2022.

[32] Bhaskar Roy e Sobhendu Ghatak. "Métodos não-lineares para avaliar mudanças na variabilidade da frequência cardıaca em pacientes com diabetes tipo 2". Em: *Arquivos Brasileiros de Cardiologia* 101 (2013), pp. 317–327.

[33] Gregory R. Lee et al. "PyWavelets: A Python package for wavelet analysis". Em: *Journal of Open Source Software* 4.36 (2019), p. 1237. DOI: 10.21105/joss.01237. URL: https://doi.org/10.21105/joss.01237.

[34] Christopher J.C. Burges Yann LeCun Corinna Cortes. *THE MNIST DATABASE of handwritten digits*. http://yann.lecun.com/exdb/mnist/. [Disponível em 17/05/2022]. 2022.

[35] Rajalingappaa Shanmugamani. *Deep Learning for Computer Vision: Expert techniques to train advanced neural networks using TensorFlow and Keras*. Packt Publishing Ltd, 2018.

[36] I. M. Irurzun et al. *RR interval time series from healthy subjects*. https://www.physionet.org/content/rr-interval-healthy-subjects/1.0.0/.. [Disponível em 17/05/2022]. 2021.

Capítulo 7

Codigo Python

7.0.1 Filtragem dos arquivos originais

```
from scipy import interpolate
from os import listdir
from os.path import isfile, join, splitext
import numpy as np
import matplotlib.pyplot as plt


caminhoArquivosOriginais = 'saudaveis/originais/' #caminho dos arquivos que passarão pelo filtro
caminhoArquivosFiltrados = 'saudaveis/filtrados/' #caminho destino dos arquivos filtrados
caminhoImgFiltragem = caminhoArquivosFiltrados + 'filtragem/' #caminho para guardar imagens da filtragem
arquivos = [f for f in listdir(caminhoArquivosOriginais) if isfile(join(caminhoArquivosOriginais, f)) and not('logPreenchimento' in f)]
variacaoMaximaPermitida = 130
multiplicadorUltimaAmostraValida = 1




def filtraArquivos():
    for arquivo in arquivos:
        arquivoFile = open(caminhoArquivosOriginais+arquivo)
        arquivoConteudo = arquivoFile.readlines()
        arquivoFile.close()
        arquivoConteudo = verificaFimDoArquivo(arquivoConteudo)
        copiaConteudo = arquivoConteudo.copy()
        for indice in range(len(arquivoConteudo)):
            if(ehOutlier(copiaConteudo, indice)):
                print('O valor '+str(int(copiaConteudo[indice]))+' na posição '+str(int(indice))+' é um outlier!')
                copiaConteudo[indice] = interpolacaoSplineCubica(copiaConteudo, indice)
                print('Foi gerado o valor ' + copiaConteudo[indice] + ' para substituir o outlier')
        valores = []
        valoresCorrigidos = []
        for valor in arquivoConteudo:
            valores.append(int(valor.split('\\')[0]))
        for valor in copiaConteudo:
            valoresCorrigidos.append(int(valor.split('\\')[0]))
        plt.figure(figsize=(14, 10))
        plt.plot(valores, marker='.')
        plt.plot(valoresCorrigidos, marker='.')
        plt.yticks(range(200, 2100, 100))
        plt.grid(axis='y')
```

```
41          arquivoNome = arquivo.split('.')[0]
42          plt.title("Filtro do arquivo " + arquivoNome)
43          plt.xlabel("Amostras")
44          plt.ylabel("RR")
45          plt.savefig(caminhoImgFiltragem + arquivoNome)
46          plt.close()
47          plt.clf()
48
49          novoArquivoFile = open(caminhoArquivosFiltrados+arquivo, 'w')
50          novoArquivoFile.writelines(copiaConteudo)
51          novoArquivoFile.close()
52
53  def verificaFimDoArquivo(arquivoConteudo):
54      if (arquivoConteudo[len(arquivoConteudo) - 1] == '\n'):
55          print('Arquivo será corrigido pois tem várias quebras de linha no final: ')
56          while (arquivoConteudo[len(arquivoConteudo) - 1] == '\n'):
57              arquivoConteudo.pop()
58      if (not ('\n' in arquivoConteudo[len(arquivoConteudo) - 1])):
59          arquivoConteudo[len(arquivoConteudo) - 1] += '\n'
60      return arquivoConteudo
61
62  def ehOutlier(arquivoConteudo, indice, ultimaAmostraValida = None,
      indiceUltimaAmostraValida = None):
63
64      if(indice == 0): #trata quando é o primeiro numero do conteudo
65          diferenca = abs(int(arquivoConteudo[indice]) - int(arquivoConteudo[indice + 1]))
      #pega o módulo do numero. distancia do numero ao zero
66          if(diferenca > variacaoMaximaPermitida):
67              return True
68      elif(indice == len(arquivoConteudo)-1): #trata quando é o último numero do conteudo
69          diferenca = abs(int(arquivoConteudo[indice]) - int(arquivoConteudo[indice - 1]))
      #pega o módulo do numero. distancia do numero ao zero
70          if(diferenca > variacaoMaximaPermitida):
71              return True
72      else:
73          diferencaAmostraAnterior = abs(int(arquivoConteudo[indice]) - int(arquivoConteudo
      [indice - 1])) #pega o módulo do numero. distancia do numero ao zero
74
75          if(ultimaAmostraValida != None):
76              #diferencaUltimaAmostraValida = abs(int(arquivoConteudo[indice]) - int(
      ultimaAmostraValida))
77              mediaUltimaAmostraValida = 0
78              for indiceMedia in range(indiceUltimaAmostraValida, indiceUltimaAmostraValida
      -10, -1):
79                  mediaUltimaAmostraValida += int(arquivoConteudo[indiceMedia])
80              mediaUltimaAmostraValida = mediaUltimaAmostraValida/10
81              diferencaUltimaAmostraValida = abs(int(arquivoConteudo[indice]) - int(
      mediaUltimaAmostraValida))
82
83          if(
84              diferencaAmostraAnterior > variacaoMaximaPermitida and not(
      ultimaAmostraValida)or
85              (ultimaAmostraValida and
86                  (diferencaUltimaAmostraValida > variacaoMaximaPermitida *
      multiplicadorUltimaAmostraValida))
87          ):
88              return True
89
90      return False
91
92
93  def interpolacaoSplineCubica(arquivoConteudo, indice):
94      qtdAmostrasParaInterpolacao = 8  # qtd de amostras a analisar ao redor do outlier
95      qtdAmostrasAoRedor = int(
96          qtdAmostrasParaInterpolacao / 2)  # qtd de amostras antes e depois do outlier que
       são usadas para cálculo
97      tamanhoConteudo = len(arquivoConteudo)
98      x_points = y_points = []
99
```

```
100     if (indice == 0):  # caso seja o primeiro elemento, nao pega nenhuma amostra antes
    dele
101         possuiIntervaloAnterior = False
102     else:
103         possuiIntervaloAnterior = True
104 
105     if (indice == tamanhoConteudo - 1):  # caso seja o último elemento, nao pega nenhuma
    amostra depois dele
106         possuiIntervaloPosterior = False
107     else:
108         possuiIntervaloPosterior = True
109 
110     if (possuiIntervaloPosterior == False):
111         # se nao possui intervalo posterior, nao precisa verificar outliers anteriores,
112         # pois todos já foram tratados conforme foram identificados
113         x_points = np.append(x_points, np.arange(indice - qtdAmostrasParaInterpolacao,
    indice))
114         y_points = np.append(y_points, arquivoConteudo[indice -
    qtdAmostrasParaInterpolacao:indice])
115 
116     elif (possuiIntervaloAnterior == False):
117         qtdAmostrasValidas = 0
118         indiceParaInterpolacao = 1
119 
120         while (qtdAmostrasValidas < qtdAmostrasParaInterpolacao):
121             if (not (ehOutlier(arquivoConteudo, indiceParaInterpolacao))):
122                 x_points = np.append(x_points, indiceParaInterpolacao)
123                 y_points = np.append(y_points, arquivoConteudo[indiceParaInterpolacao])
124                 qtdAmostrasValidas += 1
125             indiceParaInterpolacao += 1
126 
127     elif (possuiIntervaloAnterior and possuiIntervaloPosterior):
128         diferencaEntreIndiceEInicio = indice
129         diferencaEntreIndiceEFim = tamanhoConteudo - indice
130 
131         if ((diferencaEntreIndiceEInicio >= qtdAmostrasAoRedor) and (
    diferencaEntreIndiceEFim >= qtdAmostrasAoRedor)):
132             # caso tenha a qtdAmostrasAoRedor necessária antes e depois da amostra para
    interpolação
133             for indiceParaInterpolacao in range(indice - qtdAmostrasAoRedor, indice):
134                 # amostras anteriores ao indice nao precisam ser verificadas como
    outliers, pois já foram tratadas
135                 x_points = np.append(x_points, indiceParaInterpolacao)
136                 y_points = np.append(y_points, arquivoConteudo[indiceParaInterpolacao])
137 
138             qtdAmostrasValidas = qtdAmostrasAoRedor
139             indiceParaInterpolacao = indice + 1
140             ultimaAmostraValida = arquivoConteudo[indice - 1]  # ultima amostra antes do
    outlier
141             indiceUltimaAmostraValida = indice - 1
142 
143             while (qtdAmostrasValidas < qtdAmostrasParaInterpolacao and
    indiceParaInterpolacao < tamanhoConteudo):
144                 if (not (
145                 ehOutlier(arquivoConteudo, indiceParaInterpolacao, ultimaAmostraValida,
    indiceUltimaAmostraValida))):
146                     x_points = np.append(x_points, indiceParaInterpolacao)
147                     y_points = np.append(y_points, arquivoConteudo[indiceParaInterpolacao
    ])
148                     qtdAmostrasValidas += 1
149                     ultimaAmostraValida = arquivoConteudo[indiceParaInterpolacao]
150                 indiceParaInterpolacao += 1
151             else:
152                 if (indiceParaInterpolacao >= tamanhoConteudo):
153                     # chegou ao final do arquivo e não completou a qtd necessaria de
    amostras para interpolação
154                     # vai pegar mais amostras anteriores para completar
155                     # pode ocorrer devido a valores outliers apos a amostra que está no
    indice
156                     for indiceParaInterpolacao in range(
```

```
157                         (indice - qtdAmostrasAoRedor) - 1,
158                         (indice - qtdAmostrasAoRedor) - (qtdAmostrasParaInterpolacao
    - qtdAmostrasValidas) - 1, -1):
159                     x_points = np.append(indiceParaInterpolacao, x_points)
160                     y_points = np.append(arquivoConteudo[indiceParaInterpolacao],
    y_points)
161                     qtdAmostrasValidas += 1
162         else:
163             # nao tem qtdAmostrasAoRedor necessárias antes ou depois para utilizar na
    interpolacao
164             # a amostra que faltar em uma extremidade será compensada na outra
    extremidade
165             faltaAmostraNoInicio = True if diferencaEntreIndiceEInicio <
    qtdAmostrasAoRedor else False
166             qtdAmostrasValidas = 0
167 
168             if (faltaAmostraNoInicio):
169                 x_points = np.append(x_points, np.arange(0, indice))
170                 y_points = np.append(y_points, arquivoConteudo[0:indice])
171                 qtdAmostrasValidas = indice
172                 indiceParaInterpolacao = indice + 1
173 
174                 while (qtdAmostrasValidas < qtdAmostrasParaInterpolacao):
175                     if (not (ehOutlier(arquivoConteudo, indiceParaInterpolacao))):
176                         x_points = np.append(x_points, indiceParaInterpolacao)
177                         y_points = np.append(y_points, arquivoConteudo[
    indiceParaInterpolacao])
178                         qtdAmostrasValidas += 1
179                     indiceParaInterpolacao += 1
180 
181             else:  # falta amostra no final para completar qtdAmostras necessarias para
    interpolação
182 
183                 # amostras anteriores ao indice nao precisam ser verificadas como
    outliers, pois já foram tratadas
184                 x_points = np.append(x_points, range(indice - qtdAmostrasAoRedor, indice)
    )
185                 y_points = np.append(y_points, arquivoConteudo[indice -
    qtdAmostrasAoRedor: indice])
186                 qtdAmostrasValidas = qtdAmostrasAoRedor
187 
188                 possuiAmostraAposIndice = False
189                 indiceParaInterpolacao = indice + 1
190                 while (indiceParaInterpolacao < tamanhoConteudo):
191                     if (not (ehOutlier(arquivoConteudo, indiceParaInterpolacao))):
192                         x_points = np.append(x_points, indiceParaInterpolacao)
193                         y_points = np.append(y_points, arquivoConteudo[
    indiceParaInterpolacao])
194                         qtdAmostrasValidas += 1
195                         possuiAmostraAposIndice = True
196                     indiceParaInterpolacao += 1
197                 else:
198                     # chegou ao final do arquivo e não completou a qtd necessaria de
    amostras para interpolação
199                     # vai pegar mais amostras anteriores para completar
200                     if not (possuiAmostraAposIndice):
201                         # precisa ter pelo menos uma amostra apos o indice para funcionar
     a interpolaçao
202                         # caso nao tenha, replica a ultima amostra valida após o indice
203                         x_points = np.append(x_points, indice + 1)
204                         ultimaAmostraValida = arquivoConteudo[indice - 1]
205                         y_points = np.append(y_points, ultimaAmostraValida)
206                         qtdAmostrasValidas += 1
207 
208                     for indiceParaInterpolacao in range(
209                             (indice - qtdAmostrasAoRedor) - 1,
210                             (indice - qtdAmostrasAoRedor) - (qtdAmostrasParaInterpolacao
    - qtdAmostrasValidas) - 1, -1):
211                         x_points = np.append(indiceParaInterpolacao, x_points)
```

```
                        y_points = np.append(arquivoConteudo[indiceParaInterpolacao],
    y_points)
                        qtdAmostrasValidas += 1

    tck = interpolate.splrep(x_points, y_points)
    return str(int(interpolate.splev(indice, tck))) + '\n'


filtraArquivos()
```

7.0.2 Divisão e criação de arquivos sintéticos

```
from os import listdir
from os.path import isfile, join, splitext
import numpy as np
import matplotlib.pyplot as plt

caminhoArquivos= 'saudaveis/filtrados/'
caminhoArquivosCriados = 'saudaveis/criados/'
caminhoImgArquivosCriados = caminhoArquivosCriados+'img/'
arquivosMenores = np.array([])
intervaloPreenchimento = 25
rangePreenchimento = 20 #utilizada para buscar valores parecidos à ultima mostra. Buscará
     valores que estejam entre (valor - range) e (valor + range)
arquivosCriados = np.array([])
tamanhoIdealArquivos = 256 #num ideal de amostras

class ArquivoCriado:
    nome = ''
    preenchimento = []
    menor = False
    naoEncontrouPreenchimento = False
    origem = None

    def __init__(self, nome):
        self.nome = nome
        self.preenchimento = []
        self.menor = False
        self.naoEncontrouPreenchimento = False
        self.origem = None

    def __str__(self):
        return self.nome

    def __eq__(self, nome):
        if(self.nome == nome):
            return True
        else:
            return False

    def setMenor(self, menor):
        self.menor = menor

    def setNaoEncontrouPreenchimento(self, naoEncontrouPreenchimento):
        self.naoEncontrouPreenchimento = naoEncontrouPreenchimento

    def addPreenchimento(self, preenchimento):
        self.preenchimento.append(preenchimento)

    def setOrigem(self, nomeArquivoOrigem):
        self.origem = nomeArquivoOrigem

    def possuiPreenchimento(self, nomeArquivo): #confirmar se está em uso
        for preenchimento in self.preenchimento:
            if nomeArquivo in preenchimento:
                return True
        return False

```

```
56      def strDetalhado(self):
57          detalhes = 'Arquivo '+self.nome+'\n'
58          if(len(self.preenchimento) > 0):
59              detalhes += 'Preenchimentos :'
60              for preenchimento in self.preenchimento:
61                  detalhes += str(preenchimento)
62              else:
63                  detalhes += '\n'
64          else:
65              detalhes += 'Nao houve preenchimento para este arquivo\n'
66          detalhes += '################\n' #separar conteudo de arquivos diferentes
67          return detalhes
68
69
70  #Divide os arquivos já existentes em arquivos de tamanhoIdealArquivos amostras
71  def divideArquivos():
72      arquivos = [f for f in listdir(caminhoArquivos) if isfile(join(caminhoArquivos, f))]
73      arquivosCriadosList = []
74      arquivosMenoresList = []
75      for arquivo in arquivos:
76          arquivo = open(caminhoArquivos+arquivo)
77          conteudoArquivo = verificaFimDoArquivo(arquivo.readlines())
78          arquivo.close()
79          tamanhoArquivo = len(conteudoArquivo)
80          qtdArquivosPodeGerar = tamanhoArquivo//tamanhoIdealArquivos
81          for arquivoGerado in range(1, qtdArquivosPodeGerar+2): #além dos arquivos
      completos, gera um último incompleto (menor que tamanhoIdealArquivos)
82              nomeNovoArquivo = splitext(arquivo.name.split('/')[2])[0] #copia apenas o
      nome do arquivo, sem extensao e pasta. 2 é relativo ao caminho do arquivo
83              nomeNovoArquivo = nomeNovoArquivo+' - parte '+str(arquivoGerado)+'.txt'
84              novoArquivo = open(caminhoArquivosCriados+nomeNovoArquivo, 'w')
85              arquivoCriado = ArquivoCriado(nomeNovoArquivo)
86              if arquivoGerado != qtdArquivosPodeGerar+1:
87                  novoArquivo.writelines(conteudoArquivo[((tamanhoIdealArquivos*
      arquivoGerado)-tamanhoIdealArquivos):(tamanhoIdealArquivos*arquivoGerado)])
88              else: # no último arquivo pega todos os dados restantes (normalmente será
      menos que tamanhoIdealArquivos)
89                  novoArquivo.writelines(conteudoArquivo[((tamanhoIdealArquivos*
      qtdArquivosPodeGerar)):])
90                  if (len(conteudoArquivo[((tamanhoIdealArquivos*qtdArquivosPodeGerar)):])
      < tamanhoIdealArquivos):
91                      arquivosMenoresList.append(nomeNovoArquivo)
92                      arquivoCriado.setMenor(True)
93              novoArquivo.close()
94              arquivosCriadosList.append(arquivoCriado)
95      global arquivosCriados, arquivosMenores
96      arquivosCriados = np.array(arquivosCriadosList)
97      arquivosMenores = np.array(arquivosMenoresList)
98
99  #verifica os arquivos que não ficaram com tamanhoIdealArquivos amostras após a divisao
      realizada em divideArquivos
100 def verificaArquivosMenores():
101     arquivos = [f for f in listdir(caminhoArquivosCriados) if isfile(join(
      caminhoArquivosCriados, f))]
102     arquivosMenores2 = []
103     for arquivo in arquivos:
104         arquivo = open(caminhoArquivosCriados+arquivo)
105         if (len(arquivo.readlines()) < tamanhoIdealArquivos):
106             arquivosMenores2.append(arquivo.name.split('/')[2])
107         arquivo.close()
108     print("Qtd arquivos com menos de ", tamanhoIdealArquivos, " amostras: ", len(
      arquivosMenores2))
109
110
111 #apos pegar intervalos de outros arquivos, verificar se ainda existem arquivos menores
      que tamanhoIdealArquivos
112 def verificaArquivosMenoresAposPreencher():
113     arquivosCriadosMenores = [arquivosCriadosMenores for arquivosCriadosMenores in
      arquivosCriados if arquivosCriadosMenores.menor == True]
114     if(arquivosCriadosMenores):
```

```
115         manipulaArquivosMenores()
116
117 #verifica arquivos que contenham mais de uma quebra de linha no final
118 #também acrescenta quebra de linha caso o último valor não possua
119 def verificaFimDoArquivo(arquivoConteudo):
120     if(arquivoConteudo[len(arquivoConteudo)-1] == '\n'):
121         print('Arquivo será corrigido pois tem várias quebras de linha no final: ')
122         while(arquivoConteudo[len(arquivoConteudo)-1] == '\n'):
123             arquivoConteudo.pop()
124     if(not('\n' in arquivoConteudo[len(arquivoConteudo)-1])):
125       arquivoConteudo[len(arquivoConteudo)-1] += '\n'
126     return arquivoConteudo
127
128 #Manipula arquivos menores que tamanhoIdealArquivos
129 def manipulaArquivosMenores():
130     for arquivoCriadoMenor in (arquivosCriadosMenores for arquivosCriadosMenores in
    arquivosCriados if arquivosCriadosMenores.menor == True):
131         arquivoMenorNome = arquivoCriadoMenor.nome
132         arquivoMenor = open(caminhoArquivosCriados+arquivoMenorNome)
133         arquivoMenorNomeFormatado = arquivoMenorNome.split('-')[0]
134         arquivoMenorConteudo = arquivoMenor.readlines()
135         arquivoMenor.close()
136         ultimoValor = arquivoMenorConteudo[len(arquivoMenorConteudo)-1]
137
138         while(arquivoCriadoMenor.menor == True):
139             houvePreenchimento = False
140             #procura apenas em arquivos com o mesmo prefixo
141             for arquivoAComparar in (
142                     arquivosAComparar for arquivosAComparar in arquivosCriados if (
    arquivosAComparar.nome.split('-')[0] == arquivoMenorNomeFormatado)or
143                     (arquivosAComparar.nome.split('-')[0] == arquivoCriadoMenor.origem)
144                 ):
145                 arquivo = open(caminhoArquivosCriados+arquivoAComparar.nome)
146                 arquivoACompararConteudo = arquivo.readlines()
147                 arquivo.close()
148                 if (ultimoValor in arquivoACompararConteudo):
149                     indiceEncontradoUltimoValor = arquivoACompararConteudo.index(
    ultimoValor)
150                     #garantir que nao haja out of index (tenta pegar valor apos o fim da
    lista)
151                     if(len(arquivoACompararConteudo) > (indiceEncontradoUltimoValor +
    intervaloPreenchimento)):
152                         for valor in arquivoACompararConteudo[(
    indiceEncontradoUltimoValor + 1):(intervaloPreenchimento +
    indiceEncontradoUltimoValor)+1]:
153                             if(len(arquivoMenorConteudo)<tamanhoIdealArquivos): # para o
    arquivo nao ficar maior que tamanhoIdealArquivos
154                                 arquivoMenorConteudo.append(valor)
155                             else:
156                                 break
157                         arquivo = open(caminhoArquivosCriados+arquivoMenorNome, 'w')
158                         arquivo.writelines(arquivoMenorConteudo)
159                         arquivo.close()
160                         ultimoValor = arquivoMenorConteudo[len(arquivoMenorConteudo)-1]
161                         houvePreenchimento = True
162                         #salvando o arquivo que cedeu dados para preenchimento dentro do
    objeto do arquivo receptor
163                         arquivoCriadoMenor.addPreenchimento([arquivoAComparar.nome,
    indiceEncontradoUltimoValor, intervaloPreenchimento])
164                         arquivoCriadoMenor.setNaoEncontrouPreenchimento(False)
165                         if(len(arquivoMenorConteudo) == tamanhoIdealArquivos):
166                             arquivoCriadoMenor.setMenor(False)
167                             break
168                     else:
169                         print('Não foi possível copiar pois o arquivo encontrado não
    possui '+ str(intervaloPreenchimento) +' amostras suficientes para preenchimento')
170             if(not(houvePreenchimento)):
171                 manipulaArquivosMenoresFlexivel(arquivoCriadoMenor, arquivoMenorNome,
    arquivoMenorNomeFormatado, arquivoMenorConteudo, ultimoValor)
172     verificaArquivosMenoresAposPreencher()
```

```
173
174 #manipula os arquivos menores de maneira flexivel, aceitando valores que estejam no range
        informado
175 def manipulaArquivosMenoresFlexivel(arquivoCriadoMenor, arquivoMenorNome,
        arquivoMenorNomeFormatado, arquivoMenorConteudo, ultimoValor,
        rangePreenchimentoAplicado = rangePreenchimento):
176     houvePreenchimento = False
177     for arquivoAComparar in (arquivosAComparar for arquivosAComparar in arquivosCriados
        if (arquivosAComparar.nome.split('-')[0] == arquivoMenorNomeFormatado)or
178             (arquivosAComparar.nome.split('-')[0] == arquivoCriadoMenor.origem)
179             ):
180         arquivo = open(caminhoArquivosCriados+arquivoAComparar.nome)
181         arquivoACompararConteudo = arquivo.readlines()
182         arquivo.close()
183         for valor in arquivoACompararConteudo:
184             if (int(valor) >= (int(ultimoValor) - rangePreenchimentoAplicado)) and (int(
        valor) <= (int(ultimoValor) + rangePreenchimentoAplicado)):
185                 indiceEncontradoUltimoValor = arquivoACompararConteudo.index(valor)
186                 #garantir que nao haja out of index (tenta pegar valor apos o fim da
        lista)
187                 if(len(arquivoACompararConteudo) > (indiceEncontradoUltimoValor +
        intervaloPreenchimento)):
188                     for valorInArquivo in arquivoACompararConteudo[(
        indiceEncontradoUltimoValor + 1):(intervaloPreenchimento +
        indiceEncontradoUltimoValor)+1]:
189                         if(len(arquivoMenorConteudo)<tamanhoIdealArquivos): # para o
        arquivo nao ficar maior que tamanhoIdealArquivos
190                             arquivoMenorConteudo.append(valorInArquivo)
191                         else:
192                             break
193                     arquivoMenor = open(caminhoArquivosCriados+arquivoMenorNome, 'w')
194                     arquivoMenor.writelines(arquivoMenorConteudo)
195                     arquivoMenor.close()
196                     ultimoValor = arquivoMenorConteudo[len(arquivoMenorConteudo)-1]
197                     #salvando o arquivo que cedeu dados para preenchimento dentro do
        objeto do arquivo receptor
198                     arquivoCriadoMenor.addPreenchimento([arquivoAComparar.nome,
        indiceEncontradoUltimoValor, intervaloPreenchimento, 'Preenchimento flexível'])
199                     houvePreenchimento = True
200                     if(len(arquivoMenorConteudo) == tamanhoIdealArquivos):
201                         arquivoCriadoMenor.setMenor(False)
202                         break
203     else:
204         if(houvePreenchimento == False):
205             print(arquivoMenorNomeFormatado,
206             ': Não encontrou nem com flexibilidade a amostra ', ultimoValor,
207             '. Novo range flexível: ', rangePreenchimentoAplicado + rangePreenchimento)
208             manipulaArquivosMenoresFlexivel(
209                 arquivoCriadoMenor, arquivoMenorNome, arquivoMenorNomeFormatado,
210                 arquivoMenorConteudo, ultimoValor, rangePreenchimentoAplicado +
        rangePreenchimento)
211
212 def verificaQuantidadeArquivos():
213     qtdArquivosCriados = len(arquivosCriados)
214     print('Total de Arquivos criados: ', qtdArquivosCriados)
215     if(qtdArquivosCriados < 140): #300
216         criaArquivos()
217
218 #cria um arquivo com parte aleatório de todos arquivos já existentes
219 def criaArquivos():
220     listNovosArquivosCriados = []
221     global arquivosCriados
222     for arquivoCriado in arquivosCriados:
223         arquivoCriadoTxt = open(caminhoArquivosCriados+arquivoCriado.nome)
224         arquivoCriadoConteudo = arquivoCriadoTxt.readlines()
225         arquivoCriadoTxt.close()
226         nomeNovoArquivoCriado = 'criado a partir de '+ arquivoCriado.nome.split('.')[0] +
        '.txt'
227         novoArquivoCriado = open(caminhoArquivosCriados + nomeNovoArquivoCriado, 'w')
228         novoArquivoCriadoConteudo = []
```

```
229         indiceAleatorio = np.random.randint(255-intervaloPreenchimento) # pra garantir
    que na primeira vez tenha um intervalo completo para preenchimento
230         novoArquivoCriadoConteudo = arquivoCriadoConteudo[indiceAleatorio:(
    indiceAleatorio+intervaloPreenchimento)]
231         novoArquivoCriado.writelines(novoArquivoCriadoConteudo)
232         novoArquivoCriado.close()
233         novoArquivoCriado = ArquivoCriado(nomeNovoArquivoCriado)
234         novoArquivoCriado.setMenor(True) # acabou de ser criado com apenas algumas
    amostras, sempre vai ser criado menor
235         novoArquivoCriado.addPreenchimento([arquivoCriado.nome, indiceAleatorio,
    intervaloPreenchimento])
236         novoArquivoCriado.setOrigem(arquivoCriado.nome.split('-')[0])
237         listNovosArquivosCriados.append(novoArquivoCriado)
238
239     arquivosCriados = np.append(arquivosCriados, listNovosArquivosCriados)
240
241     manipulaArquivosMenores()
242     verificaQuantidadeArquivos()
243
244 #cria log que detalha preenchimento dos arquivos
245 def criaLogPreenchimento():
246     conteudoLog = []
247     for arquivoCriado in arquivosCriados:
248         conteudoLog.append(arquivoCriado.strDetalhado())
249     log = open(caminhoArquivosCriados+'logPreenchimento.txt', 'w')
250     log.writelines(conteudoLog)
251     log.close
252
253 #plota arquivos gerados
254 def geraFig():
255         arquivosCriados = [f for f in listdir(caminhoArquivosCriados) if isfile(join(
    caminhoArquivosCriados, f)) and f != 'logPreenchimento.txt']
256         for arquivoCriado in arquivosCriados:
257             valores = []
258             arquivo = open(caminhoArquivosCriados+arquivoCriado)
259             arquivoConteudo = (arquivo.readlines())
260             arquivo.close()
261             for valor in arquivoConteudo:
262                 valores.append(int(valor.split('\\')[0]))
263             plt.figure(figsize=(14, 10))
264             plt.plot(valores, marker='.')
265             plt.yticks(range(200, 2100, 100))
266             plt.grid(axis='y')
267             arquivoNome = arquivoCriado.split('.')[0]
268             plt.title("Filtro do arquivo "+ arquivoNome)
269             plt.xlabel("Amostras")
270             plt.ylabel("RR")
271             plt.savefig(caminhoImgArquivosCriados + arquivoNome)
272             plt.close()
273             plt.clf()
274
275
276 divideArquivos() #descomentar para gerar os arquivos
277 manipulaArquivosMenores()
278 verificaQuantidadeArquivos()
279 print('Fim da execução')
280 criaLogPreenchimento()
281 geraFig()
```

7.0.3 Transformada Wavelet e Treinamento CNN

```
1 from os import listdir
2 from os.path import isfile, join
3 import numpy as np
4 import pywt
5 import matplotlib.pyplot as plt
6
7
```

```
8  global caminhoArquivosSaudaveis, caminhoArquivosHipertensos, caminhoArquivosSaudaveisTest
       , caminhoArquivosHipertensosTest
9  global arquivosSaudaveis, arquivosHipertensos, arquivosHipertensosTest,
       arquivosSaudaveisTest
10 global caminhoSaudaveisOriginais, caminhoHipertensosOriginais, arquivosSaudaveisOriginais
       , arquivosHipertensosOriginais
11 caminhoArquivosSaudaveis = 'saudaveis/train/'
12 caminhoArquivosHipertensos = 'hipertensos/train/'
13 caminhoArquivosSaudaveisTest = 'saudaveis/test/'
14 caminhoArquivosHipertensosTest = 'hipertensos/test/'
15 caminhoArquivosSaudaveisAug = 'saudaveis/train_aug/'           ##Para uso com data
       augmentation
16 caminhoArquivosHipertensosAug = 'hipertensos/train_aug/'       ##
17 caminhoArquivosSaudaveisTestAug = 'saudaveis/test_aug/'        ##
18 caminhoArquivosHipertensosTestAug = 'hipertensos/test_aug/'    ##
19 arquivosSaudaveis = [f for f in listdir(caminhoArquivosSaudaveis) if isfile(join(
       caminhoArquivosSaudaveis, f)) and f != 'logPreenchimento.txt']
20 arquivosHipertensos = [f for f in listdir(caminhoArquivosHipertensos) if isfile(join(
       caminhoArquivosHipertensos, f)) and f != 'logPreenchimento.txt']
21 arquivosHipertensosTest = [f for f in listdir(caminhoArquivosHipertensosTest) if isfile(
       join(caminhoArquivosHipertensosTest, f)) and f != 'logPreenchimento.txt']
22 arquivosSaudaveisTest = [f for f in listdir(caminhoArquivosSaudaveisTest) if isfile(join(
       caminhoArquivosSaudaveisTest, f)) and f != 'logPreenchimento.txt']
23 caminhoSaudaveisOriginais = 'saudaveis/originais'
24 caminhoHipertensosOriginais = 'hipertensos/originais/'
25 arquivosSaudaveisOriginais = [f for f in listdir(caminhoSaudaveisOriginais) if isfile(
       join(caminhoSaudaveisOriginais, f)) and f != 'logPreenchimento.txt']
26 arquivosHipertensosOriginais = [f for f in listdir(caminhoHipertensosOriginais) if isfile
       (join(caminhoHipertensosOriginais, f)) and f != 'logPreenchimento.txt']
27
28 #carrega conteudo dos arquivos na memoria
29 def carregaArquivos():
30     #Lendo arquivos saudaveis
31     if usarDataAugmentation == False :  #verifica se usa os dados de data augmentation
32         caminho = caminhoArquivosSaudaveis
33         arquivos = arquivosSaudaveis
34     else:
35         caminho = caminhoArquivosSaudaveisAug
36         arquivos = [f for f in listdir(caminhoArquivosSaudaveisAug) if
37                     isfile(join(caminhoArquivosSaudaveisAug, f)) and f != '
       logPreenchimento.txt']
38
39     global rrSaudaveis
40     rrSaudaveis = [] #vai guardar todos
41     for paciente in arquivos:
42         arquivo = open(caminho+paciente)
43         arquivoConteudo = arquivo.readlines()
44         rr = np.array([]) #guarda o individuo atual do for
45         for sinal in arquivoConteudo:
46             rr = np.append(rr,int(sinal))
47         rrSaudaveis.append(rr)
48
49
50     #Lendo arquivos hipertensos
51     if usarDataAugmentation == False:
52         caminho = caminhoArquivosHipertensos
53         arquivos = arquivosHipertensos
54     else:
55         caminho = caminhoArquivosHipertensosAug
56         arquivos = [f for f in listdir(caminhoArquivosHipertensosAug) if
57                     isfile(join(caminhoArquivosHipertensosAug, f)) and f != '
       logPreenchimento.txt']
58
59     global rrHipertensos
60     rrHipertensos = []
61     for paciente in arquivos:
62         arquivo = open(caminho+paciente)
63         arquivoConteudo = arquivo.readlines()
64         rr = np.array([])
65         for sinal in arquivoConteudo:
```

```
66              rr = np.append(rr,int(sinal))
67          rrHipertensos.append(rr)
68
69
70      #Lendo arquivos saudaveis test
71      if usarDataAugmentation == False:
72          caminho = caminhoArquivosSaudaveisTest
73          arquivos = arquivosSaudaveisTest
74      else:
75          caminho = caminhoArquivosSaudaveisTestAug
76          arquivos = [f for f in listdir(caminhoArquivosSaudaveisTestAug) if
77                      isfile(join(caminhoArquivosSaudaveisTestAug, f)) and f != '
    logPreenchimento.txt']
78
79      global rrSaudaveisTest
80      rrSaudaveisTest = []
81      for paciente in arquivos:
82          arquivo = open(caminho+paciente)
83          arquivoConteudo = arquivo.readlines()
84          rr = np.array([]) #guarda o individuo atual do for
85          for sinal in arquivoConteudo:
86              rr = np.append(rr,int(sinal))
87          rrSaudaveisTest.append(rr)
88
89      #Lendo arquivos hipertensos test
90      if usarDataAugmentation == False:
91          caminho = caminhoArquivosHipertensosTest
92          arquivos = arquivosHipertensosTest
93      else:
94          caminho = caminhoArquivosHipertensosTestAug
95          arquivos = [f for f in listdir(caminhoArquivosHipertensosTestAug) if
96                      isfile(join(caminhoArquivosHipertensosTestAug, f)) and f != '
    logPreenchimento.txt']
97
98      global rrHipertensosTest
99      rrHipertensosTest = []
100     for paciente in arquivos:
101         arquivo = open(caminho+paciente)
102         arquivoConteudo = arquivo.readlines()
103         rr = np.array([]) #guarda o individuo atual do for
104         for sinal in arquivoConteudo:
105             rr = np.append(rr,int(sinal))
106         rrHipertensosTest.append(rr)
107
108
109 #cria arquivos augmentation. Nao precisa rodar toda vez
110 def gerarDataAugmentation():
111     import random
112
113     qtdIteracoes = 100  #quantas variações para cada arquivo
114
115     for arquivoSaudavel in arquivosSaudaveis:
116          #open file
117          arquivo = open(caminhoArquivosSaudaveis+'/'+arquivoSaudavel, 'r')
118          conteudo = arquivo.readlines()
119
120          for u in range(0 ,qtdIteracoes):  # cada ciclo cria uma variação do arquivo
    original
121              frequencies = []   #frequencies recebe todos os dados do arquivo e  converte
    em int
122              for n in conteudo:
123                  frequencies.append(int(n)+ random.randint(1,3))  #valor original recebe um
    valor aletarório de 1 a 3
124              result = open(caminhoArquivosSaudaveisAug+arquivoSaudavel+' - '+str(u)+".txt"
    , 'w')
125              for i in range(len(frequencies)):
126                  result.write(str(frequencies[i])+"\n")
127              result.close()
128
129     for arquivoHipertenso in arquivosHipertensos:
```

```
130         arquivo = open(caminhoArquivosHipertensos+'/'+arquivoHipertenso, 'r')
131         conteudo = arquivo.readlines()
132 
133         for u in range(0 ,qtdIteracoes):
134            frequencies = []
135            for n in conteudo:
136             frequencies.append(int(n)+ random.randint(1,3))
137            result = open(caminhoArquivosHipertensosAug+arquivoHipertenso+' - '+str(u)+".
    txt", 'w')
138            for i in range(len(frequencies)):
139                result.write(str(frequencies[i])+"\n")
140            result.close()
141 
142 
143     for arquivoSaudavelTest in arquivosSaudaveisTest:
144         arquivo = open(caminhoArquivosSaudaveisTest+'/'+arquivoSaudavelTest, 'r')
145         conteudo = arquivo.readlines()
146 
147         for u in range(0 ,qtdIteracoes):
148            frequencies = []
149            for n in conteudo:
150             frequencies.append(int(n)+ random.randint(1,3))
151            result = open(caminhoArquivosSaudaveisTestAug+arquivoSaudavelTest+' - '+str(u
    )+".txt", 'w')
152            for i in range(len(frequencies)):
153                result.write(str(frequencies[i])+"\n")
154            result.close()
155 
156 
157     for arquivoHipertensoTest in arquivosHipertensosTest:
158         arquivo = open(caminhoArquivosHipertensosTest+'/'+arquivoHipertensoTest, 'r')
159         conteudo = arquivo.readlines()
160 
161         for u in range(0 ,qtdIteracoes):
162            frequencies = []
163            for n in conteudo:
164             frequencies.append(int(n)+ random.randint(1,3))
165            result = open(caminhoArquivosHipertensosTestAug+arquivoHipertensoTest+' - '+
    str(u)+".txt", 'w')
166            for i in range(len(frequencies)):
167                result.write(str(frequencies[i])+"\n")
168            result.close()
169 
170 
171 def normalizacao():
172     global rrSaudaveis, rrHipertensos
173 
174     maximo = 0
175     minimo = 10000
176     #verifica saudaveis
177     for rrSaudavel in rrSaudaveis:
178         if(max(rrSaudavel) > maximo):
179             maximo = max(rrSaudavel)
180         if(min(rrSaudavel) < minimo):
181             minimo = min(rrSaudavel)
182     #verifica hipertensos
183     for rrHipertenso in rrHipertensos:
184         if(max(rrHipertenso) > maximo):
185             maximo = max(rrHipertenso)
186         if(min(rrHipertenso) < minimo):
187             minimo = min(rrHipertenso)
188 
189     rrSaudaveis = rrSaudaveis/maximo
190     rrHipertensos = rrHipertensos/maximo
191 
192 #salva resultado da transformada para cada arquivo
193 def transformadaWavelet():
194     from keras.utils.np_utils import to_categorical
195 
196     global wavTrain, wavTrainLabel, wavTest, wavTestLabel
```

```
197
198     #wavelet para arquivos train
199     wavTrain = [] #coefs mexican hat
200     wavTrainLabel = [] #label para cnn
201
202     for i, rrSaudavel in enumerate(rrSaudaveis):
203         cwtCoefs, cwtFreqs = pywt.cwt(data=rrSaudavel[:256], scales=np.arange(1,31),
    wavelet='mexh')
204         wavTrain.append(cwtCoefs)
205         wavTrainLabel.append(0)# 0 para saudável, 1 para hipertenso
206
207     for i, rrHipertenso in enumerate(rrHipertensos):
208         cwtCoefs, cwtFreqs = pywt.cwt(data=rrHipertenso[:256], scales=np.arange(1,31),
    wavelet='mexh')
209         wavTrain.append(cwtCoefs)
210         wavTrainLabel.append(1)# 0 para saudável, 1 para hipertenso
211
212     #wavelet para arquivos test
213     wavTest = [] #coefs mexican hat
214     wavTestLabel = [] #label para cnn
215
216     for i, rrSaudavelTest in enumerate(rrSaudaveisTest):
217         cwtCoefs, cwtFreqs = pywt.cwt(data=rrSaudavelTest[:256], scales=np.arange(1,31),
    wavelet='mexh')
218         wavTest.append(cwtCoefs)
219         wavTestLabel.append(0)# 0 para saudável, 1 para hipertenso
220
221     for i, rrHipertensoTest in enumerate(rrHipertensosTest):
222         cwtCoefs, cwtFreqs = pywt.cwt(data=rrHipertensoTest[:256], scales=np.arange(1,31)
    , wavelet='mexh')
223         wavTest.append(cwtCoefs)
224         wavTestLabel.append(1)# 0 para saudável, 1 para hipertenso
225
226     wavTrain = np.array(wavTrain)
227     wavTrain = wavTrain.reshape(-1, 30, 256, 1)
228     wavTrainLabel = np.array(wavTrainLabel)
229     wavTest = np.array(wavTest)
230     wavTestLabel = np.array(wavTestLabel)
231     wavTrain = wavTrain.astype('float32')
232     wavTest = wavTest.astype('float32')
233     wavTrainLabel = to_categorical(wavTrainLabel, 2)
234     wavTestLabel = to_categorical(wavTestLabel, 2)
235
236 #treina a rede neural convolucional
237 def treinaCnn():
238     from tensorflow.keras import Sequential
239     from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
240     from tensorflow.keras.optimizers import SGD
241     from tensorflow.keras.constraints import max_norm
242
243     global model
244
245     model = Sequential()
246     model.add(Conv2D(32, (3, 3), input_shape=(30, 256, 1),
247     activation='relu', padding='same',
248     kernel_constraint=max_norm(3)))
249     model.add(MaxPooling2D(pool_size=(2, 2)))
250     model.add(Flatten())
251     model.add(Dense(512, activation='relu',
252     kernel_constraint=max_norm(3)))
253     model.add(Dense(2, activation='softmax'))
254     model.compile(loss='categorical_crossentropy',
255     optimizer=SGD(learning_rate=0.01), metrics=['accuracy'])
256     model.summary()
257
258     model.fit(wavTrain, wavTrainLabel,
259               epochs=3,
260               batch_size=32,
261               validation_data=(wavTest, wavTestLabel),
262               shuffle=True)
```

```
263
264
265 #treina a cnn com dados do dataet mnist
266 #plota um arquivo aleatório com sua respectiva predição
267 def validaCnnMnist():
268     from tensorflow.keras import Sequential
269     from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
270     from tensorflow.keras.optimizers import SGD
271     from tensorflow.keras.constraints import max_norm
272     import tensorflow as tf
273     import numpy as np
274
275     (x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
276     x_train, x_test = x_train[..., np.newaxis]/255.0, x_test[..., np.newaxis]/255.0
277     from keras.utils.np_utils import to_categorical
278     y_train = to_categorical(y_train, 10)
279     y_test = to_categorical(y_test, 10)
280
281     model = Sequential()
282     model.add(Conv2D(32, (3, 3), input_shape=(28, 28, 1),
283     activation='relu', padding='same',
284     kernel_constraint=max_norm(3)))
285     model.add(MaxPooling2D(pool_size=(2, 2)))
286     model.add(Flatten())
287     model.add(Dense(512, activation='relu',
288     kernel_constraint=max_norm(3)))
289     model.add(Dense(10, activation='softmax'))
290     model.compile(loss='categorical_crossentropy',
291     optimizer=SGD(learning_rate=0.01), metrics=['accuracy'])
292     model.summary()
293
294     model.fit(x_train, y_train,
295         epochs=3,
296         batch_size=32,
297         validation_data=(x_test, y_test),
298         shuffle=True)
299
300     import random
301     amostra = random(0, len(x_test))  #pega uma amostra qualquer
302     a = x_test[amostra]
303     a = a.reshape(-1, 28,28,1)
304     plt.imshow(x_test[amostra,:,:,0], vmin=0, vmax=1)
305     plt.show()
306     result = model.predict(a)
307     for i, r in enumerate(result[0], start=0):
308         print("Chances de ser ", i ,": %.5f" %r)
309
310
311 def salvaModelo():
312     model.save('cnn')
313
314 def carregaModelo():
315     import tensorflow
316     global model
317     model = tensorflow.keras.models.load_model('cnn')
318
319
320 def validaArquivosOriginais():
321     global arquivosHipertensosOriginais, caminhoHipertensosOriginais,
    arquivosSaudaveisOriginais, caminhoSaudaveisOriginais
322     certosTotal = 0
323     certo = 0
324     for i, rrSaudOrig in enumerate(arquivosSaudaveisOriginais):
325         arquivo = open(caminhoSaudaveisOriginais+'/'+rrSaudOrig)
326         conteudo = arquivo.readlines()
327         cwtCoefs, cwtFreqs = pywt.cwt(data=conteudo[:256], scales=np.arange(1,31),
    wavelet='mexh')
328         cwtCoefs = cwtCoefs.reshape(1, 30,256,1)
329         #print(np.argmax(model.predict(cwtCoefs)))
330         if np.argmax(model.predict(cwtCoefs)) == 0 : certo = certo + 1
```

```
331     certosTotal = certo
332     print("Saudaveis Certos:", certo)
333 
334 
335     certo = 0
336     for i, rrHiperOrig in enumerate(arquivosHipertensosOriginais):
337         arquivo = open(caminhoHipertensosOriginais+'/'+rrHiperOrig)
338         conteudo = arquivo.readlines()
339         cwtCoefs, cwtFreqs = pywt.cwt(data=conteudo[:256], scales=np.arange(1,31),
    wavelet='mexh')
340         cwtCoefs = cwtCoefs.reshape(1, 30,256,1)
341         #print(np.argmax(model.predict(cwtCoefs)))
342         if np.argmax(model.predict(cwtCoefs)) == 1 : certo = certo + 1
343     certosTotal += certo
344     print("Hipertensos Certos:", certo)
345 
346     qtdArquivos = len(arquivosHipertensosOriginais)+len(arquivosSaudaveisOriginais)
347 
348     print("Total de arquivos: ", qtdArquivos, ". Total de acertos: ", certosTotal)
349     print("Porcentagem de acertos: ", ((certosTotal*100)/qtdArquivos))
350 
351 
352 #plota a transformada de um arquivo informado
353 def plotaTransformada():
354     arquivo = open('saudaveis/originais/SR5.txt') #arquivo a plotar
355     conteudo = arquivo.readlines()
356     cwtCoefs, cwtFreqs = pywt.cwt(data=conteudo[:256], scales=np.arange(1, 31), wavelet='
    mexh')
357     plt.figure(figsize=(80, 100))
358     plt.imshow(cwtCoefs)
359 
360 
361 
362 #validaCnnMnist() # descomentar caso queira validar a cnn com dataset mnist
363 usarDataAugmentation = True  #alterar caso nao queira gerar/usar a data augmentation.
    Usará os dados indicados como train e test
364 #if usarDataAugmentation == True:
365 #    gerarDataAugmentation() #comentar caso nao queira gerar os arquivos
366 carregaArquivos()
367 normalizacao()
368 transformadaWavelet()
369 treinaCnn()
370 #salvaModelo()
371 #carregaModelo() # se carregar o modelo nao é necessário executar todas as etapas
    relacionadas ao treinamento
372 validaArquivosOriginais()
373 plotaTransformada()
374 
375 
376 print("fim")
```