UNIMEP
Universidade Metodista de Piracicaba

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIAS DO MOVIMENTO HUMANO

DESENVOLVIMENTO DE REDE NEURAL CONVOLUCIONAL PARA CLASSIFICAÇÃO DA CAMINHADA, CORRIDA, AGACHAMENTO E SALTO VERTICAL

Convolutional Neural Network development
for gait, running, squat and vertical jump classification

Ricardo Pablo Passos

2022

DOUTORADO

Ricardo Pablo Passos

DESENVOLVIMENTO DE REDE NEURAL CONVOLUCIONAL PARA CLASSIFICAÇÃO DA CAMINHADA, CORRIDA, AGACHAMENTO E SALTO VERTICAL

Tese apresentada ao Programa de Pós-Graduação em Ciências do Movimento Humano, da Universidade Metodista de Piracicaba, para obtenção do Título de Doutor em Ciências do Movimento Humano.

Orientador: Prof. Dr. Guanis de Barros Vilela Junior

PIRACICABA
2022

Ficha Catalográfica elaborada pelo Sistema de Bibliotecas da UNIMEP

Bibliotecário: Joyce Rodrigues de Freitas – CRB: 8/101115

P234d

Passos, Ricardo Pablo
Desenvolvimento de rede neural convolucional para classificação da caminhada, corrida, agachamento e salto vertical / Ricardo Pablo Passos – 2022.
100 fls.; il.; 30 cm.

Orientador (a): Prof. Dr. Guanis de Barros Vilela Junior.
Tese (Doutorado) – Universidade Metodista de Piracicaba, Ciências do Movimento Humano, Piracicaba, 2022.

1. Redes Neurais Convolucionais. 2. Corrida 3. Caminhada I. Passos, Ricardo Pablo . II. Título.

CDD – 612.76

DEDICATÓRIA

Dedico esta, bem como todas as minhas demais conquistas, a ***humanidade***. Em especial para meus queridos alunos e professores

Ao meu parceiro de todas as horas, Sua Alteza Sereníssima Theodoro Tobias Pitoco Passos. *“O amor tem 4 letras e por certo 4 patas. Não diferencia ouro ou um pedaço de lata. Não fala, não sabe ler, mas diz tudo para você com o poder de um olhar. Tão puro e tão leal, tem o dom especial de sempre nos perdoar”* **Bráulio Bessa**.

A minha querida esposa, Jéssica Gonçalves Passos*. “Ce qui est éternel tant qu'il dure et dure pour toujours”. “Aquele que conheceu apenas a sua mulher, e a amou, sabe mais de mulheres do que aquele que conheceu mil”* **Leon Tolstói**.

Ao meu querido amigo Prof. Dr. Guanis de Barros Vilela Junior. *“Liberdade é pouco. O que eu desejo ainda não tem nome”* **Clarice Lispector**.

Esse trabalho é para vocês.

AGRADECIMENTOS

Agradeço a toda sociedade brasileira, que com impostos que não são poucos, financiou toda minha formação no doutorado, tenho uma eterna dividida para com a educação de nosso país.

A todos os integrantes do Núcleo de Pesquisas em Biomecânica Ocupacional e Qualidade de Vida (NPBOQV); *"Não somos mais aquela força dos velhos tempos; quando movíamos céus e terras, hoje somos o que somos; corações heróicos e um único caráter; enfraquecidos pelo tempo e destino, mas fortes na vontade; para lutar, buscar, encontrar, e não se render"* **Alfred Tennyson**.

Ao meu querido amigo e orientador professor Dr. Guanis de Barros Vilela Junior *(...) A mim, ele me ensinou tudo / Ele me ensinou a olhar para as coisas/ Ele me aponta todas as cores que há nas flores/ E me mostra como as pedras são engraçadas/ Quando a gente as tem na mão e olha devagar para elas/ Damo-nos tão bem um com o outro na companhia de tudo/ Que nunca pensamos um no outro/ Vivemos juntos os dois com um acordo íntimo/ Como a mão direita e a esquerda (...)* **Fernando Pessoa**. Muito obrigado por tudo que fez por mim.

Aos membros da banca Dra. Rozangela Verlengia; Dr. Anderson dos Santos Carvalho; Dra. Flávia Baccin Fiorante e Dra. Daniela Faleiros Bertelli Merino meu muito obrigado pelas considerações, ensinamentos, atenção e ajuda para comigo. *"Escolho meus amigos não pela pele ou outro arquétipo qualquer, mas pela pupila. Tem que ter brilho questionador e tonalidade inquietante. A mim não interessam os bons de espírito nem os maus de hábitos. Fico com aqueles que fazem de mim louco e santo. Deles não quero resposta, quero meu avesso. Que me tragam dúvidas e angústias e aguentem o que há de pior em mim." (...)''quero*

amigos sérios, daqueles que fazem da realidade sua fonte de aprendizagem, mas lutam para que a fantasia não desapareça. Não quero amigos adultos nem chatos. Quero-os metade infância e outra metade velhice! Crianças, para que não esqueçam o valor do vento no rosto; e velhos, para que nunca tenham pressa. Tenho amigos para saber quem eu sou. Pois ao vê-los loucos e santos, bobos e sérios, crianças e velhos, nunca me esquecerei de que a "normalidade" é uma ilusão imbecil e estéril." **Oscar Wilde.**

Aos meus amigos e sogros Sr. Nelson A. Gonçalves e Sra. Matilde G. B. Gonçalves*, "Escolho meus amigos pela alma lavada e pela cara exposta. Não quero só o ombro e o colo, quero também sua maior alegria. Amigo que não ri junto, não sabe sofrer junto. Meus amigos são todos assim: metade bobeira, metade seriedade. Não quero risos previsíveis, nem choros piedosos."* **Oscar Wilde.**

Obrigado a todos.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – CAPES – Brasil.

EPÍGRAFE

Motivo

Eu canto porque o instante existe
e a minha vida está completa.
Não sou alegre nem sou triste:
sou poeta.

Irmão das coisas fugidias,
não sinto gozo nem tormento.
Atravesso noites e dias
no vento.

Se desmorono ou se edifico,
se permaneço ou me desfaço,
— não sei, não sei. Não sei se fico
ou passo.

Sei que canto. E a canção é tudo.
Tem sangue eterno a asa ritmada.
E um dia sei que estarei mudo:
— mais nada.

Cecília Meireles

RESUMO

Introdução: Os avanços tecnológicos ocorridos no início do século XXI, especialmente na capacidade de processamento de computadores e a consolidação / compartilhamento de banco de dados, têm possibilitado o crescimento no número de pesquisas na grande área da Inteligência Artificial (IA) e, especialmente nas Ciências do Movimento Humano (CMH) na identificação de movimentos locomotores, no ambiente de trabalho, no esporte e na reabilitação. **Objetivos:** desenvolver e validar uma Rede Neural Convolucional (CNN) capaz de identificar os movimentos: caminhada, corrida, agachamento e salto vertical. **Materiais e Métodos:** a amostra oriunda do banco de dados de imagens CPAQV-100, de livre acesso, onde foram escolhidas aleatoriamente 2400 imagens que após interpolação realizada na CNN atingiu 135 milhões de parâmetros distribuídas para os quatro movimentos locomotores utilizados nesta pesquisa. Estas foram utilizadas, proporcionalmente, nas fases de treinamento, teste e validação da CNN. A linguagem de programação *Python*® e suas bibliotecas *Pandas, Torch, Matplotlib*, dentre outras, foram adotadas para a compilação da mesma. Foi escolhido o ambiente virtual Colab® acessada com um desktop de 32 Megabytes de memória RAM, 2 Terabytes de armazenamento e placa de vídeo Nvidia® RTX 3060.Esta pesquisa faz parte do projeto integrado intitulado *Métodos da Inteligência Artificial Aplicados na Análise do Movimento Humano,* aprovado pelo Comitê de Ética em Pesquisa da Universidade Metodista de Piracicaba sob protocolo número: 4126546, **Resultados:** a CNN apresentou, quando comparada com estudos similares, excelente capacidade de identificação dos movimentos da caminhada, corrida, agachamento e salto vertical com acurácia superior a 97%. A precisão, sensibilidade e especificidade também foram superiores a 0,974. Para corroborar a robustez desses achados foram calculados: o Índice Fowlkes Mallows (FMI=0,971), Coeficiente de Correlação de Matthews (MCC=0,941) *e o* Índice de Youden (IY=0,941). **Conclusão**: neste estudo foram calculadas, além da acurácia, a precisão, sensibilidade, especificidade, FMI, MCC e IY, uma vez que uma CNN que utilize exclusivamente a acurácia pode gerar limitações na sua capacidade preditiva. Tais resultados permitem afirmar que a hipótese da presente pesquisa foi amplamente corroborada, apresentando, portanto, elevado potencial de aplicação prática nas ciências do movimento humano.

Palavras-Chave: Redes Neurais Convolucionais. Corrida. Caminhada. Agachamento. Salto Vertical.

ABSTRACT

Introduction: Technological advances that took place at the beginning of the 21st century, especially in computer processing capacity and database consolidation/sharing, have enabled the growth in the number of research in the large area of Artificial Intelligence (AI) and, especially in Human Movement Sciences (HCM) in the identification of locomotor movements, in the work environment, in sport and in rehabilitation. **Objectives**: to develop and validate a Convolutional Neural Network (CNN) capable of identifying the movements: walking, running, squatting and vertical jumping. **Materials and Methods**: the sample from the free access CPAQV-100 image database, where 2400 images were randomly chosen, which after interpolation performed on CNN reached 135 million parameters distributed for the four locomotor movements used in this research. These were used, proportionally, in the CNN training, testing and validation phases. The Python® programming language and its libraries Pandas, Torch, Matplotlib, among others, were adopted for its compilation. The Colab® virtual environment was chosen, accessed with a desktop with 32 Megabytes of RAM, 2 Terabytes of storage and an Nvidia® RTX 3060 video card. approved by the Research Ethics Committee of the Methodist University of Piracicaba under protocol number: 4126546, **Results**: CNN showed, when compared to similar studies, excellent ability to identify the movements of walking, running, squatting and vertical jumping with an accuracy greater than 97 %. Precision, sensitivity, and specificity were also higher than 0.974. To corroborate the robustness of these findings, the Fowlkes Mallows Index (FMI=0.971), Matthews Correlation Coefficient (MCC=0.941) and the Youden Index (IY=0.941) were calculated. **Conclusion**: in this study, in addition to accuracy, precision, sensitivity, specificity, FMI, MCC and IY were calculated, since a CNN that uses accuracy exclusively can generate limitations in its predictive capacity. Such results allow us to affirm that the hypothesis of the present research was broadly confirmed, presenting, therefore, a high potential for practical application in the sciences of human movement.

Keywords: Convolutional Neural Networks. Running. Walk. Squat. Vertical Jump.

LISTA DE ILUSTRAÇÕES

LISTA DE GRÁFICOS

LISTA DE TABELAS

LISTA DE EQUAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

ANN Recorrentes (RANN)

Back Propagation ANN (BPANN)

Centro de pressão (COP)

Ciências do Movimento Humano (CMH)

Feedback ANN (FBANN)

Feedforward ANN (FANN)

Função Tangente (Tanh)

Inteligência Artificial (IA)

Kungliga Tekniska Högskolan (KTH)

Rede Neural Convolucional (CNN)

Redes Neurais Artificiais (ANN)

Redes Neurais Convolucionais (CNN)

Unidade Retificadora Linear (ReLU)

Visual Geometric Group (VVG)

Wireless Sensor Data Mining (WISDM)

SUMÁRIO

1 INTRODUÇÃO

Os algoritmos da Inteligência Artificial (IA) estão cada vez mais presentes no cotidiano de praticamente todas as áreas de atuação, nas diferentes instâncias da governança, dos negócios e das ciências. Na área da saúde sua presença é cada vez mais sistemática, contribuindo para avaliações e diagnósticos mais precisos.(1)

A IA tem potencial para revolucionar a ciência no sentido de Kuhn, ou seja, da revolução científica e a emergência de um novo paradigma. Algo potencialmente comparável com a revolução copernicana ou a revolução causada pela física quântica há pouco mais de 100 anos. A mesma já está presente em muitas das tarefas cotidianas que realizamos em ambiente virtual, tais como, fotografia digital, compras online, sistema bancário, reservas de hotéis, reconhecimento facial, otimização de fluxos de produção, logística, direção veicular autônoma, diagnóstico de doenças através do reconhecimento de imagens, interação via voz, com assistentes como o Google Assistente, Alexa da Amazon, Watson da IBM, dentre várias outras.

Especificamente, nas Ciências do Movimento Humano (CMH) ainda são raros os estudos que recorram aos algoritmos da IA para classificar as atividades locomotoras, tanto nas atividades cotidianas, atividades laborais e esportivas.

A IA é potencialmente capaz de mudar os paradigmas hegemônicos da biomecânica, especialmente da análise cinética e cinemática do movimento humano que reduzem para compreender este objeto de estudo que é por essência complexo e que opera dentro de padrões não-lineares onde a distribuição normal é rara e não um imperativo metodológico.

Com o advento dos avanços computacionais nas últimas décadas e o aumento da capacidade de processamento dos computadores, ocorreu uma

significativa aproximação entre a IA e neurociência, na década de 40; Mcculloch e Pitts[2]; Hebb[3]; já discutiam aspectos comportamentais e a lógica subjacente da atividade nervosa que pudesse ser simulada através de algoritmos computacionais. Esta aproximação se consolidou exponencialmente apesar da complexidade desta interação, ambas as áreas apresentam objetos de estudo claramente diferenciados e de atuação, enquanto disciplinas, bastante solidificadas, conforme atesta Brooks et al [4], ao afirmar que cabe a neurociência a geração de ideias capazes de acelerar e conduzir as pesquisas relativas à IA.

Segundo Brooks et al[4]; a neurociência tem sido uma rica fonte de inspiração para novos tipos de algoritmos e arquiteturas baseadas em métodos lógico matemáticos que tem dominado tradicionalmente as abordagens na IA.

Especialmente na Biomecânica do movimento humano, o futuro é promissor; as análises do movimento locomotor, em diferentes contextos e situações e para diferentes populações, alimentando bancos de dados em escala mundial possibilitando diagnósticos em situações cotidianas, sejam laborais ou recreacionais, na reabilitação e performance esportiva. Isso possibilitará um redimensionamento nos instrumentos da pesquisa em biomecânica que, cada vez mais, acontecerá in loco e in vivo, por exemplo, pesquisar a eficiência biomecânica de jogadores de futebol, acontecerá durante o jogo, a partir das imagens (dados) capturadas durante a partida.

2 HIPÓTESE

A Rede Neural Convolucional (CNN) desenvolvida para classificar os movimentos da caminhada, corrida, agachamento e salto vertical apresenta métricas satisfatórias, sendo confiável e aplicável em intervenções práticas.

3 JUSTIFICATIVA

A automatização da classificação do movimento humano em diferentes ambientes com elevada acurácia tem assumido importante protagonismo na área das CMH dado sua aplicabilidade em situações cotidianas, sejam laborais ou recreacionais, na performance esportiva e reabilitação.

Diante disso foram escolhidos quatro movimentos essenciais do repertório locomotor humano para o desenvolvimento da presente CNN, são eles: Caminhada, Corrida, Agachamento e Salto Vertical. Trata-se de uma escolha não aleatória, e sim, fruto de uma observação empírica das ações de movimentos predominantes na maioria dos esportes e atividades corporais tais como; futebol, voleibol, basquetebol, atletismo, rugby, beisebol, handebol, badminton, levantamento de peso, crossfit, calistenia, ginástica artística, golfe, futsal, danças, lutas, trilhas, saltos ornamentais, dentre outras. Some-se a esse cenário o fato de que nas chamadas atividades da vida diária, as pessoas, em maior ou menor quantidade e intensidade, caminham e agacham, por exemplo, indo à pé até uma padaria ou agachando para sentar-se em uma cadeira.

4 OBJETIVOS

4.1 Geral

Desenvolver e validar algoritmo com IA para classificação de diferentes movimentos locomotores, são eles: caminhada, corrida, agachamento e salto vertical.

4.2 Específicos

Desenvolver e treinar rede neural convolucional para identificação da caminhada, corrida, agachamento e salto vertical.

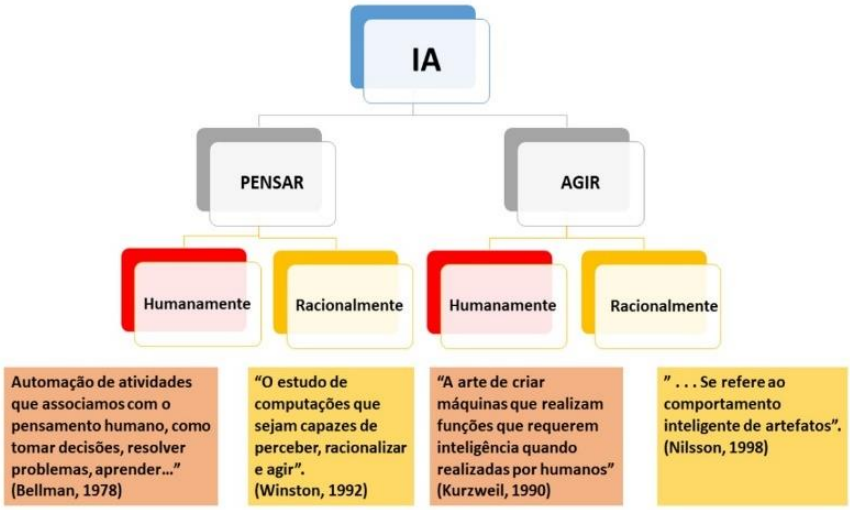
Testar rede neural convolucional para identificação da caminhada, corrida, agachamento e salto vertical.

Validar rede neural convolucional para identificação da caminhada, corrida, agachamento e salto vertical.

5 REVISÃO DA LITERATURA

A IA possui diferentes abordagens metodológicas e epistemológicas que estão sumarizadas na figura 01 [5-8].

Figura 1 – Diferentes abordagens da IA.

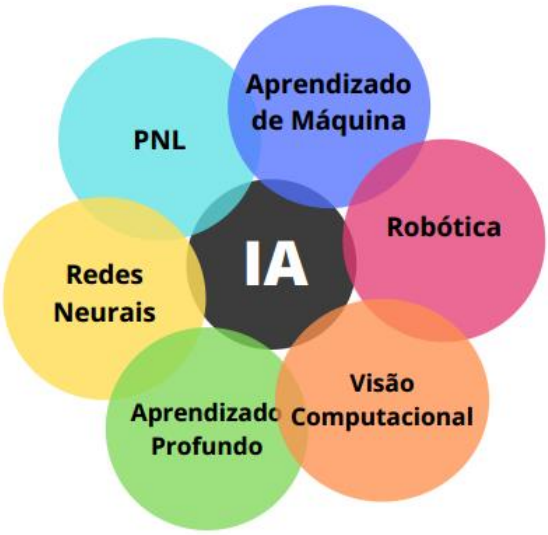


Fonte: o autor.

Trata-se de uma profícua discussão que acaba norteando as estratégias metodológicas dos diferentes grupos de pesquisas. Tais abordagens estão articuladas e existem estudos em todas estas possíveis definições de IA.

A IA possui diferentes subáreas de implementação de seus algoritmos conforme mostra a figura 02, que serão brevemente discutidas a seguir.

Figura 2 – Subáreas da IA.



Fonte: o autor

É comum em projetos de IA a utilização de mais de uma dessas subáreas concomitantemente, dependendo da complexidade do problema a ser solucionado. Por exemplo, na área da saúde são comuns algoritmos inteligentes que recorrem a Redes Neurais, Visão Computacional e Robótica (caso de robôs cirurgiões) ou Aprendizado de Máquinas, Redes Neurais e Aprendizado Profundo (caso do reconhecimento facial e das atividades humanas).

O Aprendizado de Máquina (*machine learning*) enquanto subárea da IA é aquele algoritmo linear capaz de se tornar mais preciso na previsão de resultados, sem nenhuma programação explícita para isso. Já o Aprendizado Profundo (*deep learning*) é um algoritmo não-linear que imita a maneira como os humanos obtêm certos tipos de conhecimento, possuindo arquitetura complexa e abstração praticamente ilimitada, dependendo da capacidade computacional instalada, ou seja, este aprende com os erros e é capaz de fazer associações que talvez sejam difíceis para a maioria dos humanos. Por exemplo, pela voz do sujeito realizar o diagnóstico precoce doença de Alzheimer[9, 10].

A figura 03 mostra as etapas do Aprendizado de Máquina que serão utilizadas nesta pesquisa.

Figura 3 - Aprendizado de Máquina (*machine learning*).

Fonte: o autor

Uma sequência lógica de procedimentos hierárquicos deve ser realizada; as etapas 1, 2 e 3 (Figura 03) usualmente demandam mais tempo no processo da pesquisa, pois com aquisição de dados ineficiente, mal estruturados e com uso equivocado de filtros, são potencialmente capazes de comprometer a eficiência do algoritmo inteligente. As etapas 4, 5 e 6 são implementadas no algoritmo através das bibliotecas da linguagem de programação adotada (Python, no caso desse estudo).

A etapa 7 se refere ao desenvolvimento final, inclusive a forma como o algoritmo será disponibilizado para os usuários.(1)

No presente texto serão utilizadas as expressões e abreviações mais comuns da IA nos idiomas português e inglês, dado que o inglês foi adotado mundialmente, facilitando assim, a comunicação entre pesquisadores de diversos países. A utilização de ambos os idiomas, tem também uma dimensão didática, posto que alguns pesquisadores utilizam as expressões em português.

A presente revisão da literatura foi desenvolvida considerando-se as principais arquiteturas de redes neurais artificiais aplicadas a identificação dos movimentos de Caminhada, corrida, agachamento e salto vertical.

Burton et al [11] desenvolveram quatro algoritmos de *machine learning* para identificação da eficiência da Caminhada a partir da força de reação vertical, dados antropométricos e ângulos articulares com o intuito de diagnóstico de eventuais patologias. A eficiência desses algoritmos oscilou entre 83% e 94%, destacando assim a aplicabilidade de tais CNNs.

Albert et al [12] a partir de dados coletados pelo Kinect® e pela reconstrução 3D da Vicon® (considerado pelos autores padrão ouro) desenvolveram algoritmo convolucional na plataforma Azure Kinect® com elevada acurácia na determinação de parâmetros espaciais da Caminhada.

Lin et al [13] utilizaram redes neurais convolucionais para estimar gasto energético em Caminhadas e corridas a partir da reconstrução tridimensional do movimento atingindo eficiência que oscilou entre 81% e 96%.

Chow et al [14] utilizaram unidades de medida inercial (acelerômetros e giroscópios) para desenvolver CNN capaz de aferir a eficiência da corrida na esteira e no solo. Concluíram que o algoritmo que utilizou dados relativos ao ângulo no joelho apresentou resultados superiores quando comparado com o algoritmo que utilizou ângulos do quadril em ambas as situações.

Jiang et al [15] desenvolveram uma CNN a partir de um modelo de regressão para identificar pequenas alterações no movimento do agachamento utilizando do Centro de pressão (COP) atingindo acurácia de 94% mostrando assim, a potencial aplicabilidade de redes neurais artificiais no movimento.

Athavale et al [16] realizaram pesquisa sobre atividades físicas cotidianas (Caminhada, corrida, agachamento e saltos) utilizando banco de dados vgg16 com mais de 10 mil amostras de acelerometria capturadas a partir de smartphone. Obtendo 79,55% de acurácia.

Estler [17] realizou estudos utilizando redes neurais recorrentes para predição de forças de contato no joelho. A partir do banco de dados de código aberto (opensource) “Grand Challenge” e uma rede neural *Long Short-Term Memory* (LSTM), obtendo R^2= 0,77 e o RMSE=0,27.

Hu e wang [18] utilizaram redes neurais de aprendizado profundo e *big data* (dados antropométricos e cinéticos de membros inferiores) para predição de risco de lesões no tendão patelar em jogadores de basquetebol (caminha, corrida, agachamento e salto). Quando sob máximos ângulos de flexão do joelho foram identificadas diferenças significantes entre os grupos de alto e baixo risco com p<0,01. O algoritmo foi capaz de predizer para os grupos de alto e baixo risco de lesão as acurácias de 66.82% e 66.20% respectivamente. Estudo semelhante realizado por Chalangari [19] para estimar riscos de lesões no ligamento cruzado anterior, utilizou uma CNN baseada em análise 3D do movimento locomotor a partir do Kinect® da Microsoft® , concluindo que existe uma elevada consistência entre os dados coletados no Kinect® (infravermelho) e os preditos no algoritmo.

Kwon et al [20] realizaram pesquisa a partir de dados coletados de unidades de medidas inerciais (acelerômetros e vídeos) durante a realização de exercícios livres na ginastica (Caminhada, corrida, agachamento e salto) destacando a aplicabilidade de redes neurais artificiais para identificação de movimentos locomotores em situações de eventuais oclusões parciais de segmentos corporais. Pesquisas semelhantes de Alcantara et al [21] desenvolveram rede neural

recorrente que usa dados do acelerômetro para prever a força de reação do solo na corrida, inclinações, subida e descida, que facilita a previsão de variáveis cinéticas e cinemáticas fora do ambiente de laboratório. Isso representa um passo substancial no sentido de quantificar e monitorar com precisão as cargas externas experimentadas pelo corpo ao correr ao ar livre.

Zaroug et al [22] desenvolveram uma rede neural artificial (LSTM) utilizando amostragem a partir da reconstrução 3D durante a Caminhada em esteira. Todas as trajetórias preditas pela CNN apresentaram elevada correlação com valores superiores 0,98.

Sultana et al [23] desenvolveram algoritmo de aprendizado profundo para classificar eventos de quedas de idosos em ambiente doméstico. Utilizaram uma rede convolucional (2D CNN) e uma rede recursiva que utiliza dados espaço temporais. Os resultados obtidos apresentaram acurácia na classificação de eventos de queda de 99%. Estudo similar foi realizado por Hu et al [24] para prevenções decorrentes de quedas utilizando sensores de IMU colocados no punho direito, no tronco (L5/S1), coxas e pernas enquanto os sujeitos caminhavam sobre 7 diferentes superfícies (horizontal, declive e aclive, inclinação à direita e à esquerda, e subir e descer escadas. Atingindo uma acurácia entre 90% e 92% em todas as superfícies.

Jaouedi et al [25] desenvolveu a partir do banco de dados de imagens *Kungliga Tekniska Högskolan* (KTH) uma rede neural recorrente para analisar padrões de movimentos (Caminhada, corrida, agachamento dentre outros) obtendo acurácia máxima de 92%.

Kim e Cho [26] desenvolveram por meio de uma CNN um sistema de baixo custo para reconhecimento do movimento humano em tempo real para identificação

de padrões de movimento a partir de vídeos acessados do Youtube®. A CNN desenvolvida nesse estudo apresentou melhor confiabilidade quando comparada com outras cinco arquiteturas de CNNs atingindo acurácia de 93,69%. Pesquisa semelhante realizada por Kumar et al [27] utilizaram rede neural recorrente e banco de dados KTH e UFC sports para identificação através de modelos de aprendizado profundo a atividade locomotora humana atingindo acurácia média de 96.8%.

Martindale et al [28] desenvolveram uma rede neural recorrente capaz de identificar 8 diferentes atividades locomotoras (repouso, Caminhada, trotar, correr, sentar, subir escadas, pedalar, saltar), sendo 6 dessas atividades cíclicas atingindo F1 score de 98,2%

He e Li [29] ao pesquisarem a identificação de padrões no futebol americano desenvolveram rede neural recorrente e convolucional com entrada LSTM para identificar posturas e situações comuns no jogo (posição ortostática, balanço de perna, chutando a bola) atingindo acurácia de 92,3%.

Ghate e Hemalatha [30] a partir de dados obtidos com acelerômetros, giroscópios e velocidade angular utilizando smartphones, desenvolveram algoritmo de aprendizado profundo com CNN e o classificador Random Forest para identificar movimento locomotor humano, obtendo acurácia máxima de 98.2%.

Albaba et al [31], pesquisaram o conjunto de dados Wireless Sensor Data Mining (WISDM) de código aberto, que tem seis (caminhar, correr, ficar em pé, sentar, subir e descer escadas). Cada tipo dessas atividades consiste em valores em termos dos eixos (X, Y e Z) portanto sendo dados 3D. Aplicaram dois tipos de algoritmos de aprendizado profundo que são CNN e Rede Neural Recorrente - LSTM. Descobriram que, ao usar a CNN, a acurácia foi de 81% enquanto a acurácia da Rede Neural Recorrente foi de 91%.

Soares et al [32] buscaram identificar e quantificar os movimentos realizados por um atleta de taekwondo durante os treinos por meio de técnicas de *deep learning* aplicadas aos dados coletados em tempo real. Considerando as especificidades dos movimentos, geralmente rápidos e principalmente com alta incidência nas pernas, concluíram que os melhores resultados foram obtidos com modelos de camadas de convolução, como CNN mais LSTM e modelos de aprendizagem profunda ConvLSTM, com mais de 90% em termos de validação de acurácia.

5.1 Redes Neurais Artificiais (ANNs)

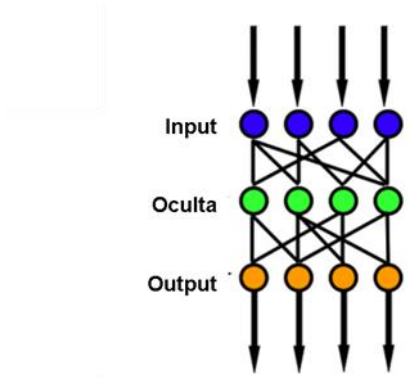
Uma ANN é um algoritmo capaz de trabalhar com um elevado número de rotinas altamente interconectadas a partir dos *inputs* que ele recebe do meio ambiente. Esses *inputs* podem ser qualquer tipo de sinal (elétrico, luminoso, térmico, barométrico, sonoro, magnético, dentre outros) e ao serem processados estes *aprendem* a minimizar seus erros e a otimizar sua acurácia. Outro aspecto de várias ANNs se refere ao fato de que o erro (ou ruído) é convertido em dado e assim ela é capaz de *aprender* com os erros que comete. É importante esclarecer que este *aprender* é um simulacro do *aprender humano* e só é possível graças a utilização massiva de cálculos matemáticos e estatísticos. Nesse sentido, o aprendizado de máquinas (*machine learning*) é muito diferente do aprendizado humano[33, 34].

Existem alguns tipos de Redes Neurais Artificiais (ANN) com diferentes arquiteturas e soluções de problemas, dentre estás as mais comuns são: *Feedforward* ANN (FANN), Feedback ANN (FBANN), *Back Propagation* ANN (BPANN), ANN Recorrentes (RANN) e Redes Neurais Convolucionais (CNN)[35-37].

5.1.1 Redes Neurais Artificiais *Feedforward (FANN)*

Uma FANN é o modelo mais simples de uma ANN onde as conexões entre os nós não realizam um circuito fechado (*loop*), ou seja, o sentido do processamento é único onde suas *entradas, pesos, camadas e saídas* executam hierarquicamente seus papeis até a solução do problema (conforme mostra figura 04). Nelas não existe nenhum tipo de entre os nós da rede[38, 39].

Figura 4 - Arquitetura de uma FANN.



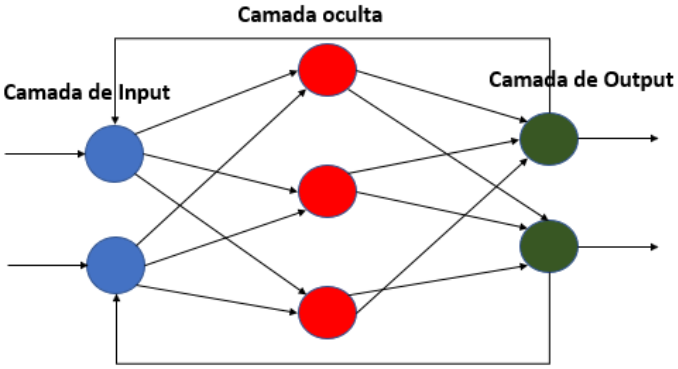
Fonte: o autor

Seus elementos constituintes (*entradas, pesos, camadas e saídas*) são definidos previamente na elaboração da mesma. FANNs são utilizadas para geração, reconhecimento e classificação de padrões.

5.1.2 Redes Neurais *Feedback* (FBANN)

Uma FBANN possui conexões que retroalimentam alguma camada de neurônios anterior (camadas de *inputs* ou ocultas) a partir da camada de *output* ou oculta[40, 41].

Figura 5 - Arquitetura típica de uma FBANN.



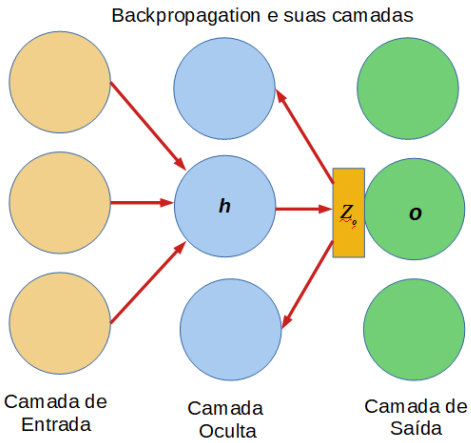
Fonte: o autor

A Figura 05, mostra a arquitetura típica de uma FBANN, observa-se dois neurônios artificiais na camada de *input*, três na camada oculta e dois na camada de *output*, sendo que a camada de *output* possui conexões com a camada de *input*, ou seja, o *feedback*. Este *feedback* poderia acontecer também nos neurônios da camada oculta.

5.1.3 Redes Neurais Artificiais *Back Propagation (BPANN)*

A expressão *Back Propagation* se refere a um algoritmo de treinamento ou de aprendizagem que trabalha com acumulação reversa de informações, aprendendo com a utilização de exemplos prévios utilizados na fase de treinamento da rede. A *back propagation* possui como característica a capacidade de ajustar os pesos e vieses da rede para aprimorar a eficiência de seus *outputs*. Estes ajustes são determinados pelos gradientes da função de otimização destes parâmetros. Essa característica de realizar operações lógicas ciclicamente faz da BPANN uma das mais importantes arquiteturas de ANNs (conforme mostra figura 06) [38-41].

Figura 6 - Estrutura de uma MLP com backpropation.



Fonte: o autor

5.1.4 Redes Neurais Artificiais Recorrentes

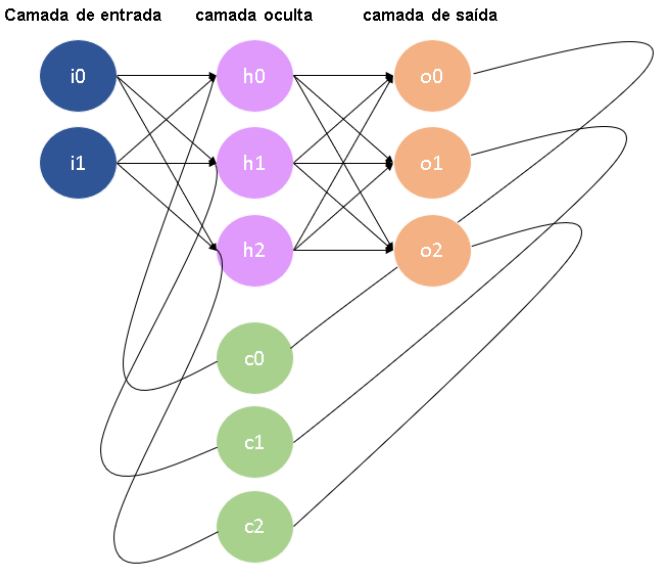
As Redes Neurais Convolucionais (CNN) são bastante eficientes em reconhecer imagens do corpo humano em situações clínicas, como exames por imagens, tais como, RX, ultrassom, tomografia, dentre outras. Entretanto, para análise do movimento humano não existe nenhuma padronização de superclasses e classes que constituam em um banco de dados disponível publicamente que padronize as mais diversas situações locomotoras humanas, nas atividades cotidianas, laborais, recreativas e esportivas[42, 43].

Um dos desafios da automatização do reconhecimento de imagens relativas ao movimento humano e imagens de diagnóstico (fotos ou vídeos) na inteligência artificial, especialmente nas redes neurais convolucionais é a padronização das superclasses e classes.

As ANNs Recorrentes são aquelas que conseguem trabalhar com matrizes quadradas (número de linhas igual ao número de colunas) de dados ou funções, realizando operações em *loop*, sendo bastante eficientes na identificação de padrões que variam com o tempo[44, 45].

Outra característica das ANNR se refere ao fato de que usualmente são adicionadas conexões à camada oculta apresentando assim um comportamento temporal dinâmico[46, 47]. (conforme mostra figura 07)

Figura 7 - Rede Neural Recursiva de Jordan.



Fonte: o autor

Na figura 07, “i” são os neurônios presentes na camada de entrada, “h” são os neurônios presentes na camada oculta e “o” são os neurônios presentes na camada de saída e “c” são os neurônios presentes de controle na rede de Jordan, que é utilizada em algoritmos que precisam lidar com a interpretação e tradução semântica entre diferentes idiomas, especialmente, no formato de áudio, por exemplo, através da fala natural de um sujeito é possível diagnostica-lo precocemente com Alzheimer antes da doença se manifestar de maneira crônica. Outro exemplo, a partir de um banco de dados das acelerações do Centro de Gravidade (CG) de um atleta realizando um salto carpado na ginástica artística, é possível compreender a eficiência do salto durante toda sua execução [29].

5.1.5 Redes Neurais Convolucionais (CNN)

As redes neurais convolucionais se originaram nas primeiras pesquisas em IA e existem desde o final do século passado. As CNN foram estendidas para introduzir novos tipos de camadas nas Redes Neurais Artificiais (ANNs) para melhorar sua capacidade de acomodar diferentes posições, tamanhos. Além disso, as redes agora têm dezenas ou mesmo centenas de camadas mais profundas que podem ser usadas para criar modelos hierárquicos de imagens, sons, tabuleiros de jogos e outras estruturas de dados espaciais[48, 49].

As redes neurais convolucionais foram adotadas por pesquisadores devido ao seu sucesso em tarefas relacionadas à visão. É usado não apenas para detectar movimento em várias situações, mas também para detectar, explicar e rastrear ativamente objetos no espaço físico[50, 51].

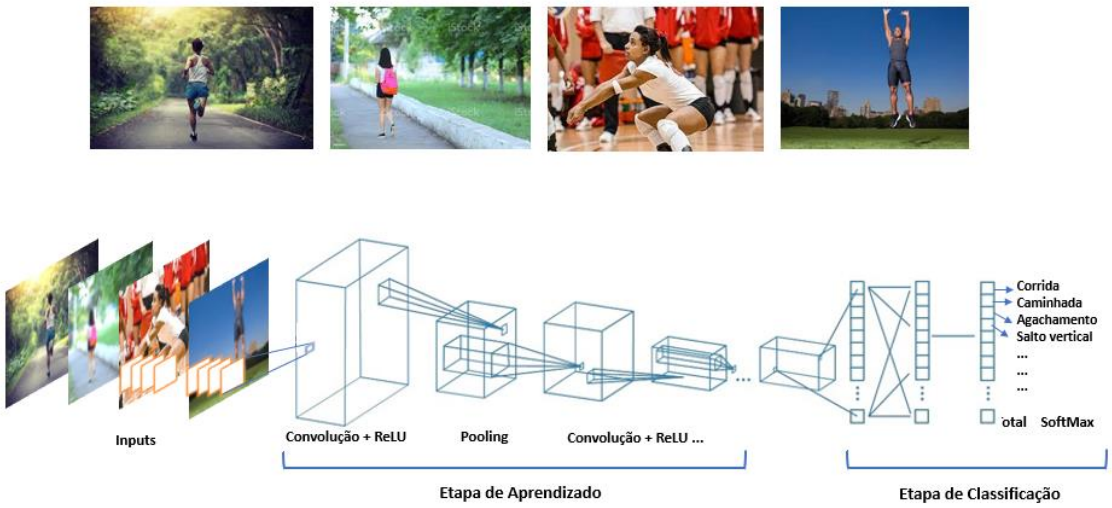
As Redes Neurais Convolucionais (CNN) são uma classe de redes neurais artificiais do tipo *feed-forward;* a mesma é utilizada com sucesso no processamento e análise de imagens digitais. Do reconhecimento facial, identificação de digitais a diagnósticos de patologias a partir do RX, DEXA, tomografia e ultrassom, com um potencial de aplicabilidade nas ciências do movimento humano que provavelmente norteará boa parte dos avanços científicos na área nos próximos anos[52, 53].

A matemática subjacente às CNNs se refere ao fato que toda imagem digital é uma sequência numérica que são tratadas como matrizes com uma quantidade de *pixels*, que podemos transformar em um vetor (por exemplo uma matriz 3x3 em um vetor 9x1). Uma CNN é um algoritmo capaz de transformar cada pixel de uma imagem em uma sequência numérica característica de cada objeto ou movimento que se queira identificar[54-56].

É importante lembrar que convoluções na matemática, são operações entre duas funções para obtenção uma terceira função. Sendo a base na *deep learning* e nos últimos anos, emergiram como ciência que conduz a pesquisa em redes neurais artificiais. A CNN revolucionou a visão computacional, oferecendo os mais altos níveis de resultados para muitas tarefas básicas, além de análises de linguagem natural muito avançadas, reconhecimento de som, aprendizado reforçado, dentre outras. As empresas de tecnologia estão usando redes neurais convolucionais para diversos fins, dentre eles podemos destacar: 1) Detecção e rotulação de objetos, lugares, plantas, animais e pessoas; 2) Converter a voz humana em texto; 3) Sintetizar sons e vozes naturais.

Funcionalmente uma CNN realiza, graças ao poder dos processadores da atualidade, milhões de cálculos para mapear cada pixel de uma imagem na sua entrada, associando a cada um deles um valor numérico, possibilitando a identificação de diferentes objetos que ela foi treinada a identificar. A figura 08 mostra, esquematicamente sua estrutura[57, 58].

Figura 8 - As diferentes etapas de uma CNN.



Fonte: o autor

Na prática a convolução é um filtro que identifica um valor numérico a cada pixel (ou conjunto de pixels) associado à função de classificação (usualmente *Softmax* ou ReLU) possibilita a redução da dimensão da imagem através de operações matriciais (*pooling*).

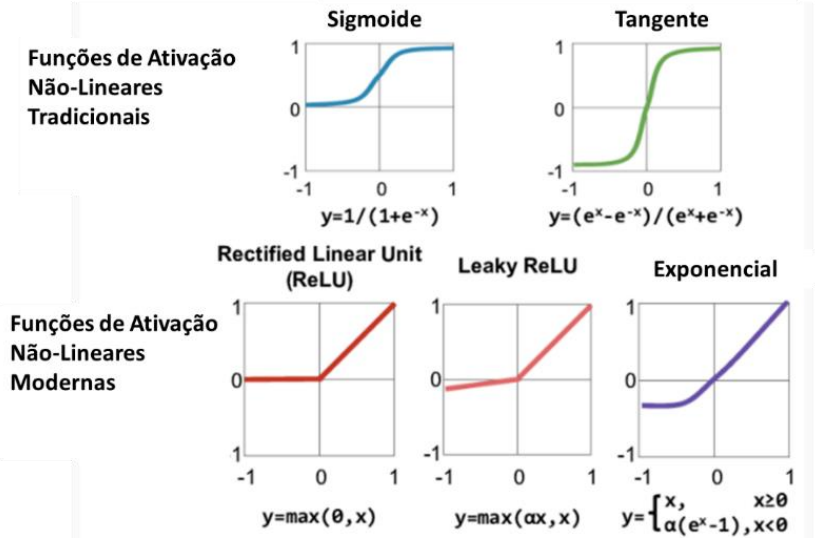
A seguir serão exemplificados os tipos mais comuns das chamadas Funções de Ativação em redes neurais artificiais, estas são importantes em todas as etapas dos algoritmos da IA sendo utilizadas várias vezes nos mesmos.

Funções de Ativação

Uma *Função de Ativação* é especializada em definir um *output* em um nó de uma ANN a partir de seus *inputs*; o termo *ativação* se refere à lógica do LIGA-DESLIGA, onde LIGA equivale a 1 e DESLIGA equivale a 0.

É importante ressaltar que apenas funções não-lineares são capazes de resolver problemas complexos, nesse sentido, especialmente em algoritmos para identificação de movimentos locomotores e em diferentes contextos ambientais[57, 59, 60].

Figura 9 - Diferentes tipos de funções de ativação usadas na IA.



Fonte: o autor

Equação 1 - Função Sigmoide

Trata-se de uma função não-linear. E quando aplicada em gradientes pequenos a rede não aprende pois os valores ficam próximo de 0.

Eq. 1

$$\boldsymbol{f(x) = \frac{1}{1 + e^{-x}}}$$

Função Logit.

A função de ativação Logit é o inverso da função sigmoide, sendo utilizada em modelos que permitam a predição de valores tomados por uma variável categórica, frequentemente binária, a partir de uma série de variáveis explicativas contínuas e/ou binárias.

Equação 2 - Função Tangente (Tanh).

A função tangente é também utilizada nas ANNs, sendo que seus valores de output oscilam entre -1 e 1.

Eq. 2

$$\tanh(x) = 2 / (1 + e \wedge (-2x))^{-1}$$

Equação 3 - Unidade Retificadora Linear (ReLU).

Essa função é bastante utilizada em RNA pois utiliza os próprios erros que podem retroalimentar camadas de neurônios. A ReLU não ativa todos os neurônios ao mesmo tempo, pois não permite a saturação da função de ativação, isto possibilita a melhoria da eficiência no processamento.

Eq. 3

$$f(x) = \max(0, x)$$

Nos algoritmos de *machine learning* as funções sigmoide e tangente usualmente perdem eficiência quando a rede neural artificial possui várias camadas, mas estas são eficientes quando aplicadas na camada de output executando operações de classificação. Nas camadas internas é recorrente a utilização da função ReLU, especialmente em redes convolucionais.

6 MATERIAIS E MÉTODOS

O presente estudo foi desenvolvido na linguagem Python® dado sua flexibilidade e bibliotecas específicas para tratamento dos dados e exibição dos resultados.

> "… [machine learning é] uma área de Inteligência Artificial que tem como objetivo desenvolver técnicas computacionais que permitam a predição e o aprendizado de determinados comportamentos ou padrões automaticamente a partir de experiências acumuladas por exemplos e problemas anteriores." (81).

Quando falamos em Machine Learning, estamos também falando de estatística. Isso porque o Aprendizado de Máquina só pôde ser criado graças à ampla variedade de técnicas estatísticas desenvolvidas nos últimos tempos. Graças ao aumento do poder de processamento dos computadores, as abordagens estatísticas e matemáticas evoluíram ao ponto de permitir o uso em aprendizado de máquina.

E devido aos resultados que o aprendizado de máquina tem gerado até agora, os profissionais e especialistas da área estão buscando aplicá-lo em projetos mais complexos, como em diagnósticos médicos e em vários outros problemas sociais e empresariais relevantes. Uma enorme quantidade de cálculos avançados estão embutidos em toda a presente tese, área da matemática (álgebra linear, cálculo vetorial, cálculo diferencial e integral, etc) e a estatística (clássica e bayesiana) neste trabalho em ciências do movimento humano. (81)

A Figura 10 sintetiza os procedimentos metodológicos utilizados.

Figura 10 - Procedimentos metodológicos.

Fonte: O autor

As principais bibliotecas do Python® utilizadas foram: Torch, Pandas, Numpy, Matplolib, dentre outras apresentadas abaixo durante a importação.

6.1 As bibliotecas do Python utilizadas

O código abaixo mostra a compilação da importação das referidas bibliotecas.

```
# PyTorch
from torchvision
import transforms, datasets, models
import torch
from torch import optim, cuda
from torch.utils.data import DataLoader, sampler
import torch.nn as nn
import zipfile
torch.__version__
torch.manual_seed(123)

import warnings
warnings.filterwarnings('ignore', category=FutureWarning)

# Ferramentas de ciência de dados
import numpy as np
import pandas as pd
import os

# Manipulações de imagem
from PIL import Image

# Útil para examinar a rede
from torchsummary import summary

# Utilitário de cronometragem
from timeit import default_timer as timer

# Visualizações
```

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams['font.size'] = 14

# Imprimindo todas as saídas
InteractiveShell.ast_node_interactivity = 'all'
```

Essa etapa é necessária pois com a importação das bibliotecas é possível começar a programar em Python® utilizando os recursos como suporte.

6.2 Aquisição de imagens para consolidação do banco de dados

No escopo desta pesquisa todos os dados foram imagens relativas ao movimento humano (fotografias e vídeos) que no algoritmo desenvolvido foram transformadas em dados numéricos para ser possível a classificação. As imagens, majoritariamente, pertencem aos bancos de dados CPAQV100 [61] e Fileni [37], e imagens públicas coletadas na internet.

Para todos os problemas com ciência de dados, a formatação adequada do *banco de dados* é determinante na eficiência da ANN.

O banco de dados foi estruturado com imagens coloridas sobre as atividades locomotoras utilizadas no presente estudo, em diferentes resoluções, que foram tratadas com diferentes filtros totalizando mais de 135 milhões de imagens.

Os arquivos foram importados para a nuvem utilizando a ferramenta Colab® que é um ambiente de programação de linguagem Python®.

6.3 Treinamento do modelo

Durante o treinamento, a ANN ajusta seus pesos por meio da *backpropagation* e gradiente descendente; os dados podem ser acessados diversas vezes até que a rede possa classificar as imagens com acurácia satisfatória.

O modelo foi executado por 10 *epochs*, o que significa que examinou todos os dados de treinamento 10 vezes; em cada *epoch*, o programa acompanha o progresso no conjunto de dados *(Loss function e accuracy)*.

6.4 Parâmetros

Os parâmetros nesta célula podem ser alterados conforme necessário. Um dos benefícios de usar PyTorch® é que podemos mover facilmente diferentes elementos do modelo ou dados para CPUs.

```
# Localização dos dados
datadir = '/'
traindir = datadir + 'train/'
validdir = datadir + 'valid/'
testdir = datadir + 'test/'

save_file_name = 'name.pt'
checkpoint_path = 'name.pth'

# Mudar para ajustar o hardware
batch_size = 128

# Treinar em um gpu
train_on_gpu = cuda.is_available()
print(f'Train on gpu: {train_on_gpu}')

# Número de gpus
if train_on_gpu:
    gpu_count = cuda.device_count()
    print(f'{gpu_count} gpus detected.')
    if gpu_count > 1:
        multi_gpu = True
    else:
        multi_gpu = False
```

Ao contrário da CPU, que faz o processamento de uma série de tarefas dos mais diferentes tipos, a GPU se dedica inteiramente à execução de atividades gráficas, como exibição de vídeos, games, edição e modelagem 3D.

6.5 Exploração de dados

Os dados foram divididos em treinamento (50%), validação (25%) e teste (25%) sendo organizados em classes conforme mostra o código a seguir:

```
/ datadir
    / train
        / class1
        / class2
        .
        ...
    / valid
        / class1
        / class2
        .
        ...
    / test
        / class1
        / class2
        .
        ...
```

Esta é uma organização padrão dos dados para CNNs e torna simples associar os rótulos corretos às imagens (superclasses e classes). A seguir os números de imagens em cada categoria e no tamanho das imagens.

```
# Listas vazias
categories = []
img_categories = []
n_train = []
n_valid = []
n_test = []
hs = []
ws = []

# Repita em cada categoria
for d in os.listdir(traindir):
    categories.append(d)

# Número de cada imagem
    train_imgs = os.listdir(traindir + d)
    valid_imgs = os.listdir(validdir + d)
    test_imgs = os.listdir(testdir + d)
    n_train.append(len(train_imgs))
    n_valid.append(len(valid_imgs))
    n_test.append(len(test_imgs))

# Encontre estatísticas para imagens de trainamento
    for i in train_imgs:
        img_categories.append(d)
        img = Image.open(traindir + d + '/' + i)
        img_array = np.array(img)
        # Shape
        hs.append(img_array.shape[0])
        ws.append(img_array.shape[1])

# Quadro de dados das categorias
cat_df = pd.DataFrame({'category': categories,
                       'n_train': n_train,
                       'n_valid': n_valid, 'n_test': n_test}).\
    sort_values('category')

# Quadro de dados de imagens de treinamento
image_df = pd.DataFrame({
    'category': img_categories,
    'height': hs,
    'width': ws
```

```
    })

    cat_df.sort_values('n_train', ascending=False, inplace=True)
    cat_df.head()
    cat_df.tail()
```

Tabela 1 - Número de imagens em cada classe.

Categoria	N_Treinamento	N_Validação	N_Teste
Caminhada	300	150	150
Corrida	300	150	150
Agachamento	300	150	150
Salto vertical	300	150	150

Fonte: o autor

6.6 Distribuição de Imagens

Existem entre 300 e 150 imagens de treinamento em cada categoria. O baixo número de imagens de treinamento pode resultar em pontuações reduzidas em algumas categorias.

Categorias com mais exemplos terão uma taxa maior de aprendizagem. Uma maneira de contornar o pequeno número de imagens é por meio do aumento de dados (*data augmentation*) que está descrito no item 6.10.

6.7 Definição do tamanho das imagens

As próprias imagens têm formas muito diferentes. Pode-se observar isso com as estatísticas de tamanhos de imagens por categoria.

```
    img_dsc = image_df.groupby('category').describe()
    img_dsc.head()
```

Tabela 2 - Altura da imagem (Pixels).

Categoria	n	média	DP	min	25%	50%	75%	máx.
Caminhada	300	1404,57	688,19	168,0	1080,0	1080,0	2146,0	2146,0
Corrida	300	722,00	294,83	480,0	480,0	480,0	1080,0	1080,0
Agachamento	300	523,84	13,52	480,0	528,0	528,0	528,0	528,0
Salto vertical	300	1835,95	644,34	388,0	2146,0	2146,0	2146,0	2146,0

Fonte: o autor

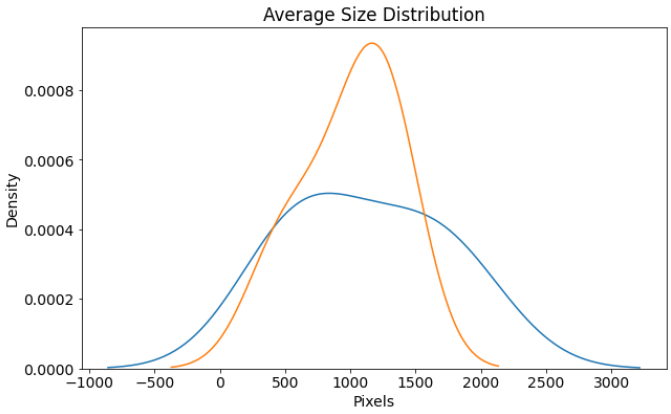
Tabela 3 - Largura da imagem (Pixels)

Categoria	n	média	DP	min	25%	50%	75%	máx.
Caminhada	300	1265,24	503,56	183,0	966,0	966,0	1920,0	1920,0
Corrida	300	1280,37	526,76	848,0	848,0	848,0	1920,0	1920,0
Agachamento	300	480,00	99,25	99,25	450	966	966	1920
Salto vertical	300	948,95	0,00	480	480	480	480	480

Fonte: o autor

```
plt.figure(figsize=(10, 6))
sns.kdeplot(
    img_dsc['height']['mean'], label='Average Height')
sns.kdeplot(
    img_dsc['width']['mean'], label='Average Width')
plt.xlabel('Pixels')
plt.ylabel('Density')
plt.title('Average Size Distribution')
```

Figura 11 - Definição do tamanho médio.



Fonte: o autor

Após as imagens passarem pela rede pré-treinada, elas foram ajustadas para 224 x 224 pixels. Esse é o tamanho das imagens *Imagenet* e, portanto, o que o modelo espera receber. As imagens maiores serão achatadas enquanto as imagens menores serão interpoladas.

```
def imshow(image):
    """Display image"""
    plt.figure(figsize=(6, 6))
    plt.imshow(image)
    plt.axis('off')
    plt.show()

# Imagem de exemplo
x = Image.open(traindir + ' /gait/gait (100).jpg')
np.array(x).shape
imshow(x)

(1080, 1920, 3)
```

Figura 12 – Exemplo de imagem para caminhada.

Fonte: o autor

```
x = Image.open(traindir + '/gait/gait (100).jpg')
np.array(x).shape
imshow(x)
(2146, 966, 3)
```

Figura 13 - Exemplo de imagem para corrida

Fonte: o autor

```
x = Image.open(traindir + '/squat/squat (92).jpg')
np.array(x).shape
imshow(x)
(3126, 976, 3)
```

Figura 14 - Exemplo de imagem para agachamento

Fonte: o autor

```
x = Image.open(traindir + '/jump/jump (58).jpg')
np.array(x).shape
imshow(x)
(0144, 366, 3)
```

Figura 15 - Exemplo de imagem para salto vertical

Fonte: o autor

6.8 Pré-processamento de imagens

Para preparar as imagens da rede, foram redimensionadas e normalizadas para cada canal de cor subtraindo um valor médio e dividindo por um desvio padrão, foi aumentado os dados de treinamento neste estágio. Essas operações são feitas usando transformações de imagem, que preparam os dados para uma rede neural.

6.9 Interpolação de dados

Como há um número limitado de imagens em algumas categorias, foi necessária a interpolação para aumentar artificialmente o número de imagens "vistas" pela rede. Ou seja, para o treinamento, as imagens são recortadas, redimensionadas e rotacionadas aleatoriamente. Uma transformação aleatória diferente é aplicada a cada *época* (durante o treinamento), de forma que a rede veja efetivamente muitas versões diferentes da mesma imagem. Todos os dados também são convertidos em tensores do Pytorch® antes da normalização. Os dados de validação e teste não são aumentados, mas apenas redimensionados e normalizados. Os valores de normalização são padronizados para *Imagenet*.

```
     # Transformações de imagem
     image_transforms = {
         # O treinamento usa aumento de dados
         'train':
         transforms.Compose([
             transforms.RandomResizedCrop(size=256, scale=(0.8, 1.0)),
             transforms.RandomRotation(degrees=15),
             transforms.ColorJitter(),
             transforms.RandomHorizontalFlip(),
             transforms.CenterCrop(size=224),   # Padrões de rede de imagem
transforms.ToTensor(),
             transforms.Normalize([0.485, 0.456, 0.406],
                                  [0.229, 0.224, 0.225])  # Padrões de rede de
imagem    ]),
         # A validação não usa aumento
         'val':
         transforms.Compose([
             transforms.Resize(size=256),
             transforms.CenterCrop(size=224),
             transforms.ToTensor(),
             transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
         ]),
         # O teste não usa aumento
         'test':
         transforms.Compose([
             transforms.Resize(size=256),
             transforms.CenterCrop(size=224),
             transforms.ToTensor(),
             transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
         ]),
     }
```

6.10 Exemplos de Aumento

Para mostrar como o aumento funciona, é utilizada a seguinte função que plotará um tensor como uma imagem.

```
def imshow_tensor(image, ax=None, title=None):
    """Imshow for Tensor."""

    if ax is None:
        fig, ax = plt.subplots()

    # Definindo um canal de cor como a terceira dimensão
    image = image.numpy().transpose((1, 2, 0))

    # Invertendo as etapas de pré-processamento
    mean = np.array([0.485, 0.456, 0.406])
    std = np.array([0.229, 0.224, 0.225])
    image = std * image + mean

    # Recorte os valores de pixel da imagem
    image = np.clip(image, 0, 1)

    ax.imshow(image)
    plt.axis('off')

    return ax, image
```

```
ex_img = Image.open('/content/datadir/train/gait/gait (200).jpg')
imshow(ex_img)
```

Figura 16 - Exemplo de imagem para Caminhada.

Fonte: o autor

```
t = image_transforms['train']
plt.figure(figsize=(24, 24))

for i in range(16):
    ax = plt.subplot(4, 4, i + 1)
    _ = imshow_tensor(t(ex_img), ax=ax)

plt.tight_layout()
```

```
<Figure size 1728x1728 with 0 Axes>
```

Figura 17 - Multiplicando as imagens de Caminhada.

Fonte: o autor

```
ex_img = Image.open('/content/datadir/train/running/running (1).jpg')
plt.figure(figsize=(24, 24))

for i in range(16):
    ax = plt.subplot(4, 4, i + 1)
    _ = imshow_tensor(t(ex_img), ax=ax)

plt.tight_layout()

<Figure size 1728x1728 with 0 Axes>
```

Figura 18 - Multiplicando as imagens de corrida.

Fonte: o autor

6.11 Iteradores de dados

Para evitar o carregamento de todos os dados na memória de uma vez, é recomendado utilizar o *DataLoaders* de treinamento. Primeiro, é criado um objeto de conjunto de dados a partir das pastas de imagens e depois transferido para um *DataLoader*. No momento do treinamento, o DataLoader carregará as imagens do disco, aplicará as transformações e gerará um lote. Para treinar e validar, deve-se iterar por todos os lotes no respectivo DataLoader. Um aspecto crucial é embaralhar os dados antes de transmiti-los à rede. Isso significa que a ordem das categorias de imagem muda em cada passagem pelos dados (uma passagem pelos dados é uma *época* de treinamento).

```
     # Conjuntos de dados de cada pasta
     data = {
         'train':
         datasets.ImageFolder(root=traindir,
transform=image_transforms['train']),
         'val':
         datasets.ImageFolder(root=validdir,
transform=image_transforms['val']),
         'test':
         datasets.ImageFolder(root=testdir, transform=image_transforms['test'])
     }

     # Iteradores de carregador de dados
     dataloaders = {
         'train':        DataLoader(data['train'],        batch_size=batch_size,
shuffle=True),
         'val': DataLoader(data['val'], batch_size=batch_size, shuffle=True),
         'test': DataLoader(data['test'], batch_size=batch_size, shuffle=True)
     }
```

```
     trainiter = iter(dataloaders['train'])
     features, labels = next(trainiter)
     features.shape, labels.shape

     (torch.Size([128, 3, 224, 224]), torch.Size([128]))
```

Um lote é composto pelo tamanho, canais de cor, altura, largura (*batch_size, color_channels, height, width*). Deve haver 2 classes diferentes.

Pode ser confirmado isso da seguinte maneira.

```
n_classes = len(cat_df)
print(f'There are {n_classes} different classes.')

len(data['train'].classes)

Existem 2 classes diferentes
```

6.12 Modelos Pré-Treinados para Classificação de Imagens

O PyTorch® tem vários modelos pré-treinados que podem ser utilizados. Todos esses modelos foram desenvolvidos com a utilização da biblioteca *Imagenet*, que consiste em milhões de imagens em mais de 1000 categorias. Nos modelos pré-treinados foram substituídas suas primeiras camadas pelo algoritmo de classificação desenvolvidos no presente estudo.

6.13 Abordagem

A abordagem para utilização do modelo de reconhecimento de imagem pré-treinado foi estruturada nas seguintes etapas:

1. Carregar pesos pré-treinados de uma rede em um grande conjunto de dados,
2. Desabilitar todos os pesos nas camadas inferiores (convolucionais). As camadas desabilitadas podem ser ajustadas dependendo de similaridade da tarefa com um grande conjunto de dados de treinamento.
3. Substituir a parte do classificador (totalmente conectada) da rede por um classificador personalizado. O número de saídas deve ser definido igual ao número de classes, no presente estudo, quatro classes de movimentos, portanto, com quatro saídas.
4. Treine apenas as camadas do classificador personalizado (totalmente conectadas) para a tarefa para um conjunto de dados menor.

A ideia por trás do pré-treinamento são as primeiras camadas convolucionais dos recursos de extração de uma CNN que são relevantes para muitas tarefas de reconhecimento de imagem.

As redes pré-treinadas são eficientes para uma variedade de tarefas e resultam em uma redução significativa no tempo de treinamento e, geralmente, em aumentos no desempenho.

Os modelos disponíveis no PyTorch® estão listados abaixo com o número correspondente de parâmetros na tabela 04.

```
model_options = pd.read_csv('models.csv')
model_options
```

Tabela 4 - Modelos de CNN disponíveis nas bibliotecas do PyTorch®.

Modelos de CNN	**Nº de Parâmetros**
AlexNet	61100840
DenseNet	7978856
Inception3	27161264
SqueezeNet	1248424
alexnet	61100840
densenet121	7978856
densenet161	28681000
densenet169	14149480
densenet201	20013928
inception_v3	27161264
resnet101	44549160
resnet152	60192808
resnet18	11689512
resnet34	21797672
resnet50	25557032
squeezenet1_0	1248424
squeezenet1_1	1235496
vgg11	132863336
vgg11_bn	132868840
vgg13	133047848
vgg13_bn	133053736
vgg16	**138357544**
vgg16_bn	138365992
vgg19	143667240

Fonte: o autor

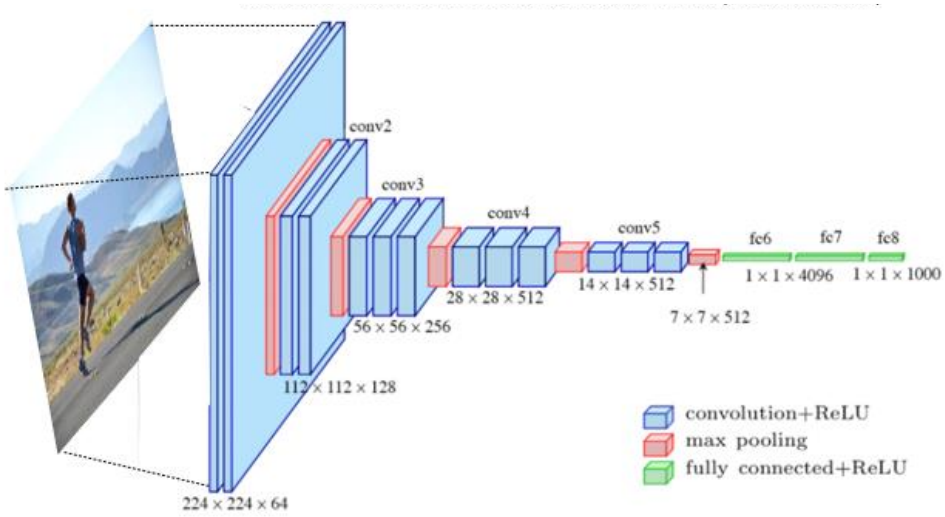
Figura 19 - Os 24 tipos de arquiteturas de CNN

A figura 19 mostra as 24 CNN que disputaram em 2014 uma competição relativa a identificação de imagens (evento que ficou conhecido como ImageNet).

Na presente pesquisa foi utilizada a CNN chamada *Visual Geometric Group* (*vgg16) que possui 16 camadas com* parâmetros treináveis e outras camadas com não possuem nenhum parâmetro treinável (as chamadas *Max pool layers*). Este modelo de rede *vgg16* foi utilizado pois obteve o melhor desempenho sendo executado no melhor tempo e com melhor acurácia.

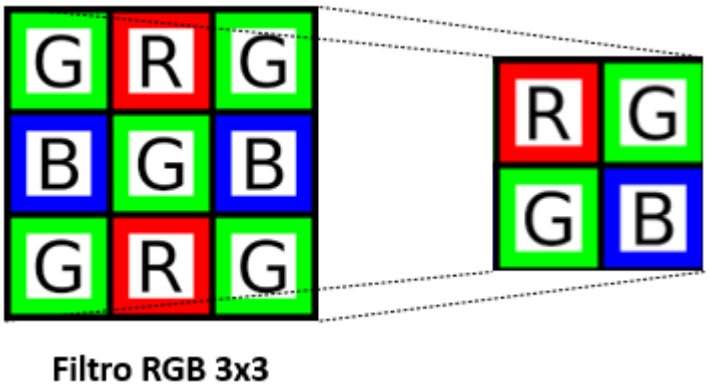
A figura 20 mostra a arquitetura da CNN-VGG com suas 16 camadas de treinamento, destacadas com as cores azul e verde.

Figura 20 – Estrutura da CNN com 16 camadas treináveis (azuis e verdes)



Nesta CNN, como as imagens eram coloridas foi utilizado filtro RGB-Bayer de ordem 3x3 pixels, pois esta é a menor configuração para identificação precisa dos pixels que estão acima ou abaixo, à direita ou à esquerda em relação ao pixel central, conforme ilustra a figura 21.

Figura 21 - Filtro RGB-Bayer utilizado na CNN-VVG



Outra vantagem deste filtro se refere ao fato de que diante da necessidade de utilizar uma ordem de 2x2, necessariamente, a matriz conterá 2 pixels G (verde), um B (azul) e um R (vermelho); este cenário é especialmente útil na identificação de bordas ou contornos em uma imagem.

6.14 Construção do modelo

Primeiro, carregue o modelo com pesos pré-treinados.

```
     model = models.vgg16(pretrained=True)
     model

     VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace)
    (4):    MaxPool2d(kernel_size=2,    stride=2,    padding=0,    dilation=1,
ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace)
    (9):    MaxPool2d(kernel_size=2,    stride=2,    padding=0,    dilation=1,
ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace)
    (16):    MaxPool2d(kernel_size=2,    stride=2,    padding=0,    dilation=1,
ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace)
    (23):    MaxPool2d(kernel_size=2,    stride=2,    padding=0,    dilation=1,
ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace)
    (30):    MaxPool2d(kernel_size=2,    stride=2,    padding=0,    dilation=1,
ceil_mode=False)
       )
       (classifier): Sequential(
         (0): Linear(in_features=25088, out_features=4096, bias=True)
         (1): ReLU(inplace)
         (2): Dropout(p=0.5)
         (3): Linear(in_features=4096, out_features=4096, bias=True)
         (4): ReLU(inplace)
         (5): Dropout(p=0.5)
         (6): Linear(in_features=4096, out_features=1000, bias=True)
       )
     )
```

6.15 Desabilitar camadas iniciais

Desabilitar todas as camadas existentes na rede definindo *require_grad by false*.

```
# Desabilitar as primeiras camadas
for param in model.parameters():
    param.requires_grad = False
```

6.16 Adicionar no classificador personalizado

Treinando um classificador que consiste nas seguintes camadas

• Totalmente conectado com ativação ReLU *(n_inputs, 256);*

• Abandono com 40% de chance de perda;

• Totalmente conectado com saída log *softmax (256, n_classes);*

Para construir o classificador personalizado, foi utilizado o módulo *nn.Sequential ()* que permite especificar cada camada uma após a outra.

O classificador personalizado à camada final na rede vgg16 já treinada. Ao adicionar camadas extras, elas são definidas como *require_grad =True* por padrão.

Essas serão as únicas camadas treinadas.

```
n_inputs = model.classifier[6].in_features

# Adicionar no classificador
model.classifier[6] = nn.Sequential(
    nn.Linear(n_inputs, 256), nn.ReLU(), nn.Dropout(0.4),
    nn.Linear(256, n_classes), nn.LogSoftmax(dim=1))

model.classifier

Sequential(
  (0): Linear(in_features=25088, out_features=4096, bias=True)
  (1): ReLU(inplace)
  (2): Dropout(p=0.5)
  (3): Linear(in_features=4096, out_features=4096, bias=True)
  (4): ReLU(inplace)
  (5): Dropout(p=0.5)
  (6): Sequential(
    (0): Linear(in_features=4096, out_features=256, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.4)
    (3): Linear(in_features=256, out_features=100, bias=True)
    (4): LogSoftmax()
  )
)
```

A saída final será o log de probabilidades que podemos usar na perda de probabilidade de log negativo.

```
total_params = sum(p.numel() for p in model.parameters())
print(f'{total_params:,}
 total parameters.')
total_trainable_params = sum(
    p.numel() for p in model.parameters() if p.requires_grad)
print(f'{total_trainable_params:,} training parameters.')
```

```
135,310,404 total de parâmetros.
1,049,860 treinamento parâmetros.
```

Mesmo com apenas algumas camadas definidas como treináveis, ainda existem mais de um milhão de parâmetros (pesos) que serão atualizados durante o treinamento.

Ao ajustar este modelo, ele funciona bem no *Imagenet* e, como as imagens são relativamente semelhantes, é esperado que o modelo transfira facilmente seu conhecimento do *Imagenet* para o nosso conjunto de dados.

6.17 Função para carregar no modelo pré-treinado

É possível recompilar todo esse código em uma única função que retorna um modelo pré-treinado.

```
def get_pretrained_model(model_name):
    """Retrieve a pre-trained model from torchvision

    Params
    -------
        model_name (str): name of the model (currently only accepts vgg16
and resnet50)

    Return
    --------
        model (PyTorch model): cnn

    """

    if model_name == 'vgg16':
        model = models.vgg16(pretrained=True)

        # Desabilitar as primeiras camadas
        for param in model.parameters():
            param.requires_grad = False
        n_inputs = model.classifier[6].in_features

        # Adicionar no classificador
        model.classifier[6] = nn.Sequential(
            nn.Linear(n_inputs, 256), nn.ReLU(), nn.Dropout(0.2),
            nn.Linear(256, n_classes), nn.LogSoftmax(dim=1))

    elif model_name == 'resnet50':
        model = models.resnet50(pretrained=True)

        for param in model.parameters():
            param.requires_grad = False

        n_inputs = model.fc.in_features
        model.fc = nn.Sequential(
            nn.Linear(n_inputs, 256), nn.ReLU(), nn.Dropout(0.2),
            nn.Linear(256, n_classes), nn.LogSoftmax(dim=1))
```

```
        # Mover para gpu e desabilitar
        if train_on_gpu:
            model = model.to('cuda')

        if multi_gpu:
            model = nn.DataParallel(model)

        return model
```

6.18 Mapeamento de classes para índices

Para acompanhar as previsões feitas pelo modelo, foi realizado um mapeamento de classes para índices. Isso permite saber a classe real para uma determinada previsão.

```
model.class_to_idx = data['train'].class_to_idx
model.idx_to_class = {
    idx: class_
    for class_, idx in model.class_to_idx.items()
}

list(model.idx_to_class.items())[:10]

[(0, 'gait'), (2, 'running')]
```

6.19 Perda de treinamento e otimizador

A perda de treinamento foi determinada pela probabilidade de log negativo e o otimizador de Adam. O otimizador é aplicado aos parâmetros do modelo (apenas alguns dos quais requerem um gradiente).

Perda (critério): acompanha a própria perda e os gradientes da perda em relação aos parâmetros do modelo (pesos)

Otimizador: atualiza os parâmetros (pesos) com os gradientes

```
criterion = nn.NLLLoss()
optimizer = optim.Adam(model.parameters())
```

Abaixo serão apresentados os parâmetros (pesos) que serão atualizados pelo otimizador durante o treinamento.

```
for p in optimizer.param_groups[0]['params']:
    if p.requires_grad:
        print(p.shape)

torch.Size([256, 4096])
torch.Size([256])
torch.Size([4, 256])
torch.Size([4])
```

6.20 Treinamento

Para treinamento, é realizado uma iteração por meio do *DataLoader* de treinamento, cada vez passando um lote pelo modelo. Uma passagem completa pelos dados de treinamento é conhecida como uma época, e foi treinado por um determinado número de épocas ou até que a parada precoce tenha iniciado. Após cada lote, é calculado a perda (com critério (saída, metas)) e, em seguida, calculado os gradientes de perda em relação aos parâmetros do modelo com *loss.backward ()*. Isso usa autodiferenciação e *backpropagation* para calcular os gradientes.

Depois de calcular os gradientes, é aplicado um *optimizer.step ()* para atualizar os parâmetros do modelo com os gradientes. Isso é feito em cada lote de treinamento, portanto, implementando um gradiente descendente estocástico (ou melhor, uma versão dela com momentum conhecida como Adam). Para cada lote, também é calculado a precisão do monitoramento e, após a conclusão do *loop* de treinamento, inicia-se o loop de validação, isso será usado para realizar a parada antecipada.

6.21 Parada antecipada

A parada antecipada interrompe o treinamento quando a perda de validação não diminuiu por várias épocas. Cada vez que a perda de validação diminui, os pesos do modelo são salvos para que possa ser carregado mais tarde o melhor modelo.

A parada antecipada é um método eficaz para evitar ajustes excessivos nos dados de treinamento. Se continuarmos o treinamento, a perda de treinamento continuará diminuindo, mas a perda de validação aumentará porque o modelo está começando a memorizar os dados de treinamento.

Parar antecipadamente evita que isso aconteça e salvando o modelo a cada época em que a perda de validação diminuir, é possível recuperar o modelo que se sair melhor com os dados de validação.

A parada antecipada é implementada iterando os dados de validação no final de cada período de treinamento e calculando a perda. Os dados de validação completos são usados todas as vezes e registrados se a perda diminuiu ou não. Se não o fez durante algumas épocas, o treinamento é interrompido, recuperado os melhores pesos e os devolvidos.

6.22 Função de treinamento

A função abaixo treina a rede enquanto monitora vários parâmetros diferentes, conforme descritos no código abaixo. É importante destacar que na linguagem Python® o símbolo *hashtag* (*#)* presente nos códigos referem-se às explicações de cada etapa do algoritmo.

```
# Definição do modelo e otimização

def train(model,
          criterion,
          optimizer,
          train_loader,
```

```
                valid_loader,
                save_file_name,
                max_epochs_stop=3,
                n_epochs=10,
                print_every=2):
        """Train a PyTorch Model

        Params
        --------
            model (PyTorch model): cnn to train
            criterion (PyTorch loss): objective to minimize
            optimizer (PyTorch optimizier): optimizer to compute gradients of
model parameters
            train_loader (PyTorch dataloader): training dataloader to iterate
through
            valid_loader (PyTorch dataloader): validation dataloader used for
early stopping
            save_file_name (str ending in '.pt'): file path to save the model
state dict
            max_epochs_stop (int): maximum number of epochs with no improvement
in validation loss for early stopping
            n_epochs (int): maximum number of training epochs
            print_every (int): frequency of epochs to print training stats

        Returns
        --------
            model (PyTorch model): trained cnn with best weights
            history (DataFrame): history of train and validation loss and
accuracy
        """

    # Parada antecipada de inicialização
        epochs_no_improve = 0
        valid_loss_min = np.Inf

        valid_max_acc = 0
        history = []

    # Número de épocas já treinadas (se estiver usando pesos carregados no
modelo)
        try:
            print(f'Model has been trained for: {model.epochs} epochs.\n')
        except:
            model.epochs = 0
            print(f'Starting Training from Scratch.\n')

        overall_start = timer()

    # Loop principal
        for epoch in range(n_epochs):

    # manter o controle da perda de treinamento e validação a cada época
            train_loss = 0.0
            valid_loss = 0.0

            train_acc = 0
            valid_acc = 0

    # Definido para treinamento
            model.train()
            start = timer()

    # Treinamento do loop
            for ii, (data, target) in enumerate(train_loader):

    # Tensors to gpu
                if train_on_gpu:
                    data, target = data.cuda(), target.cuda()
```

```
     # Gradientes limpo
                optimizer.zero_grad()

     # As saídas previstas são probabilidades de log
                output = model(data)

     # Perda e backpropagation  de gradientes
                loss = criterion(output, target)
                loss.backward()

     # Atualize os parâmetros
                optimizer.step()

     # Rastreie a perda de treinamento multiplicando a perda média pelo número
de exemplos em lote
                train_loss += loss.item() * data.size(0)

     # Calcule a precisão encontrando a probabilidade máxima de log
                _, pred = torch.max(output, dim=1)
                correct_tensor = pred.eq(target.data.view_as(pred))
     # É necessário converter o tensor correto de int para float para a média
               accuracy = torch.mean(correct_tensor.type(torch.FloatTensor))
               # Multiply average accuracy times the number of examples in batch
                train_acc += accuracy.item() * data.size(0)

                # Acompanhe o progresso do treinamento
                print(
                    f'Epoch: {epoch}\t{100 * (ii + 1) / len(train_loader):.2f}%
complete. {timer() - start:.2f} seconds elapsed in epoch.',
                    end='\r')

                # Após o término dos loops de treinamento, inicie a validação
            else:
                model.epochs += 1

                # Não precisa acompanhar gradientes
                with torch.no_grad():
                    # Definido para o modo de avaliação
                    model.eval()

                    # Loop de validação
                    for data, target in valid_loader:
                        # Tensores para gpu
                        if train_on_gpu:
                            data, target = data.cuda(), target.cuda()

                        # Passar para a frente
                        output = model(data)

                        # Perda de validação
                        loss = criterion(output, target)
                        # Multiplique a perda média * o número de exemplos
                        valid_loss += loss.item() * data.size(0)

                        # Calcular a precisão da validação
                        _, pred = torch.max(output, dim=1)
                        correct_tensor = pred.eq(target.data.view_as(pred))
                        accuracy = torch.mean(
                            correct_tensor.type(torch.FloatTensor))
                        # Multiplique a precisão média pelo número de exemplos
                        valid_acc += accuracy.item() * data.size(0)

                    # Calculate average losses
                    train_loss = train_loss / len(train_loader.dataset)
                    valid_loss = valid_loss / len(valid_loader.dataset)

                    # Calcular a precisão média
```

```
                    train_acc = train_acc / len(train_loader.dataset)
                    valid_acc = valid_acc / len(valid_loader.dataset)

                    history.append([train_loss,     valid_loss,     train_acc,
valid_acc])

                    # Imprimir resultados de treinamento e validação
                    if (epoch + 1) % print_every == 0:
                        print(
                            f'\nEpoch:      {epoch}      \tTraining      Loss:
{train_loss:.4f} \tValidation Loss: {valid_loss:.4f}'
                        )
                        print(
                            f'\t\tTraining Accuracy: {100 * train_acc:.2f}%\t
Validation Accuracy: {100 * valid_acc:.2f}%'
                        )

    # Salve o modelo se a perda de validação diminuir
                    if valid_loss < valid_loss_min:

    # Salvar modelo
                        torch.save(model.state_dict(), save_file_name)
                        # Melhoria de pista
                        epochs_no_improve = 0
                        valid_loss_min = valid_loss
                        valid_best_acc = valid_acc
                        best_epoch = epoch

                    # Caso contrário, aumente a contagem de épocas sem melhora
                    else:
                        epochs_no_improve += 1
                        # Acione a parada antecipada
                        if epochs_no_improve >= max_epochs_stop:
                            print(
                                f'\nEarly Stopping! Total epochs: {epoch}. Best
epoch:  {best_epoch}  with  loss:  {valid_loss_min:.2f}  and  acc:  {100  *
valid_acc:.2f}%'
                            )
                            total_time = timer() - overall_start
                            print(
                                f'{total_time:.2f}  total  seconds  elapsed.
{total_time / (epoch+1):.2f} seconds per epoch.'
                            )

                            # Carregue o melhor dict de estado
                            model.load_state_dict(torch.load(save_file_name))
                            # Anexe o otimizador
                            model.optimizer = optimizer

                            # Histórico de formatação
                            history = pd.DataFrame(
                                history,
                                columns=[
                                    'train_loss', 'valid_loss', 'train_acc',
                                    'valid_acc'
                                ])
                            return model, history

        # Anexe o otimizador
        model.optimizer = optimizer

        # Registre o tempo geral e imprima as estatísticas
        total_time = timer() - overall_start
        print(
            f'\nBest epoch: {best_epoch} with loss: {valid_loss_min:.2f} and
acc: {100 * valid_acc:.2f}%'
        )
        print(
```

```
            f'{total_time:.2f}   total   seconds   elapsed.   {total_time   /
(epoch):.2f} seconds per epoch.'
        )
        # Histórico de formatação
        history = pd.DataFrame(
            history,
            columns=['train_loss', 'valid_loss', 'train_acc', 'valid_acc'])
        return model, history
```

De 10 *epochs* utilizadas a que obteve melhores resultados foi a *epoch* 5. O código abaixo mostra algumas *epochs* com suas perdas e acurácias nas fases de treinamento e validação.

```
    # Identificação da melhor epoch
    model, history = train(
        model,
        criterion,
        optimizer,
        dataloaders['train'],
        dataloaders['val'],
        save_file_name=save_file_name,
        max_epochs_stop=5,
        n_epochs=10,
        print_every=2)

    Começando o treinamento do zero
    /usr/local/lib/python3.7/dist-packages/torch/nn/functional.py:718:
UserWarning: Named tensors and all their associated APIs are an experimental
feature and subject to change. Please do not use them for anything important
until   they   are   released   as   stable.   (Triggered   internally   at
/pytorch/c10/core/TensorImpl.h:1156.)
      return torch.max_pool2d(input, kernel_size, stride, padding, dilation,
ceil_mode)

    Epoch: 1       Training Loss: 0.0598      Validation Loss: 1.6226
           Training Accuracy: 98.42%   Validation Accuracy: 61.50%

    Epoch: 3       Training Loss: 0.0209      Validation Loss: 2.4572
           Training Accuracy: 99.33%   Validation Accuracy: 59.67%

    Epoch: 5       Training Loss: 0.0185      Validation Loss: 2.4520
           Training Accuracy: 99.83%   Validation Accuracy: 60.00%

    Early Stopping! Total epochs: 5. Best epoch: 0 with loss: 1.23 and acc:
60.00%
    5598.72 total seconds elapsed. 933.12 seconds per epoch.
```

6.23 Métricas utilizadas para aferir a confiabilidade da rede neural

A questão que a acurácia busca responder é: qual o percentual de casos verdadeiros em relação a todos os resultados?

A acurácia é amplamente utilizada como parâmetro central na avaliação de classificadores na inteligência artificial, tal fato se deve à sua eficiência e de ser conceitualmente de simples compreensão.

É importante destacar que acurácia não é sinônimo de precisão uma vez que a primeira se refere a erros sistêmicos e aleatórios e a segunda é exclusivamente ligada a erros aleatórios. A acurácia pode ser calculada pela razão entre o número total de observações que o modelo acertou (VP+VN) e o número total de observações que o modelo previu (VP+VN+FP+FN), (ver eq. 4) [68,69,70].

Como essas métricas (acurácia, precisão, sensibilidade, especificidade, dentre outras) são obtidas através da matriz de confusão, elas são dependentes da capacidade do algoritmo em detectar os verdadeiros positivos (VP), verdadeiros negativos (VN), falsos positivos (FP) e falsos negativos (FN).

Equação 4 – Fórmula acurácia

$$Acurácia = \frac{(\mathrm{VP} + \mathrm{VN})}{(\mathrm{VP} + \mathrm{VN} + \mathrm{FP} + \mathrm{FN})}$$

A precisão por sua vez busca responder à questão: Que porcentagem de VPs prevista está correta?

Trata-se da razão de VP em relação a todos os valores positivos classificados (VP+FP). conforme mostra a eq. 5 [68,69,70].

Usualmente é utilizada quando o impacto dos valores FP são potencialmente mais prejudiciais que os valores FN, por exemplo, em um algoritmo para classificação da hipertensão arterial, um FP resulta na afirmação de que o sujeito é hipertenso (mesmo não sendo), tal fato pode resultar em um diagnóstico

equivocado e ao sujeito ser prescrita a utilização de medicamentos hipotensores, colocando a saúde do mesmo em risco.

Equação 5 – Fórmula precisão

$$Precisão = \frac{(\mathrm{VP})}{(\mathrm{VP} + \mathrm{FP})}$$

Busca responder à questão: Qual a porcentagem de todos os VPs que foram previstos corretamente?

Trata-se, portanto, da razão entre os valores VP e (VP+FN) como é mostrado na eq. 6; é amplamente utilizada em situações em que os FN são potencialmente mais preocupantes que os FP. Mede a taxa de verdadeiros positivos que o modelo consegue predizer, isso possibilita analisar a capacidade do classificador em identificar os FN [68,69,70].

Equação 6 – Fórmula sensibilidade

$$Sensibilidade = \frac{(\mathrm{VP})}{(\mathrm{VP} + \mathrm{FN})}$$

A especificidade busca responder à questão: Qual a porcentagem de todos os valores negativos que foram previstos corretamente?

É uma métrica utilizada em um classificador que avalia a taxa de VN, pode ser calculada pela razão entre os valores VN e (VN+FP), conforme mostra a equação 7.

É importante destacar que a especificidade não fornece necessariamente uma indicação precisa sobre um resultado negativo no teste de classificação porque os resultados negativos do teste podem conter muitos resultados falsos negativos [68,69,70].

Equação 7 – Fórmula Especificidade

$$Especificidade = \frac{\mathrm{VN}}{(\mathrm{VN} + \mathrm{FP})}$$

O FMI é definido como a média geométrica entre a precisão e a sensibilidade, conforme mostra a fórmula na figura 2. [68,69,70] Esse índice apresenta valores entre 0 e 1, sendo que valores mais próximos de 1 indicam maiores similaridades entre dois conjuntos de dados.

Trata-se de um indicador bastante eficiente para dados não relacionados, apresentando, também, resultados bastante satisfatórios em um conjunto de dados com ruídos. É importante destacar que o FMI diminui em função do ruído, portanto, apesar de lidar bem com ruídos, deve ser utilizado com cautela diante das escolhas metodológicas do pesquisador.

Equação 8 – Índice Fowlkes Mallows (FMI)

$$FMI = \left(\frac{VP}{VP + FP} * \frac{VP}{VP + FN}\right)^{1/2}$$

O Coeficiente de Correlação de Matthews (MCC) ou coeficiente ϕ (lê-se “phi”) busca sintetizar em um único número uma matriz de confusão binária (2x2); isto facilita a comparação entre outras matrizes. Usualmente é utilizada em situações com classes com métricas bastante diferentes. Pode ser calculada com a equação:

Equação 9 - Coeficiente de Correlação de Matthews (MCC)

$$MCC = \frac{(VP * VN) - (FP * FN)}{\sqrt{(VP + FP)(VP + FN)(VN + FP)(VN + FN)}}$$

Os valores devem oscilar no intervalo [−1, +1], com valores extremos –1 e +1 alcançados em caso de erro de classificação (para -1) e classificação perfeita (para +1), quando ϕ=0 está métrica é considerada neutra, ou seja, o classificador apresenta probabilidade de acertos de 50%, nesse caso, trata-se de um indicador que deve ser substituído pelo F1 score.

O Índice de Youden (IY) é uma métrica importante para a identificação do ponto de corte em uma curva ROC (sensibilidade *versus* especificidade), sendo, portanto, um estimador da probabilidade de uma decisão.

Equação 10 - O Índice de Youden (IY)

$$IY = \frac{VP}{VP + FN} + \frac{VN}{VN + FP} - 1$$

O IY pode variar entre zero e 1,0, quando zero significa que o teste é totalmente inútil pois não é capaz de distinguir VP e VN. Quando IY=1,0, o teste é considerado perfeito pois não existem FP e FN. Hipoteticamente, se VP=VN=0 teremos IY= -1,0, resultado que indica a incapacidade do classificador em identificar casos verdadeiros, apresentando sensibilidade, especificidade e acurácia nulas. Não obstante, é possível que a soma da sensibilidade com a especificidade seja menor que 1,0, e neste caso IY será negativo. Esse problema é fruto da rotulação invertida, bastando trocar positivos e negativos para adequar o IY entre 0,0 e 1,0[68,69,70].

6.24 Ajustes finos: *overfitting* e *underfitting*

Um dos principais desafios metodológicos nos algoritmos da inteligência artificial, especialmente no *machine learning*, refere-se à confiabilidade das

métricas usualmente adotadas nos classificadores, independente da área de aplicação. São comuns pesquisas, onde essas métricas não vão além da acurácia, precisão, especificidade, sensibilidade.

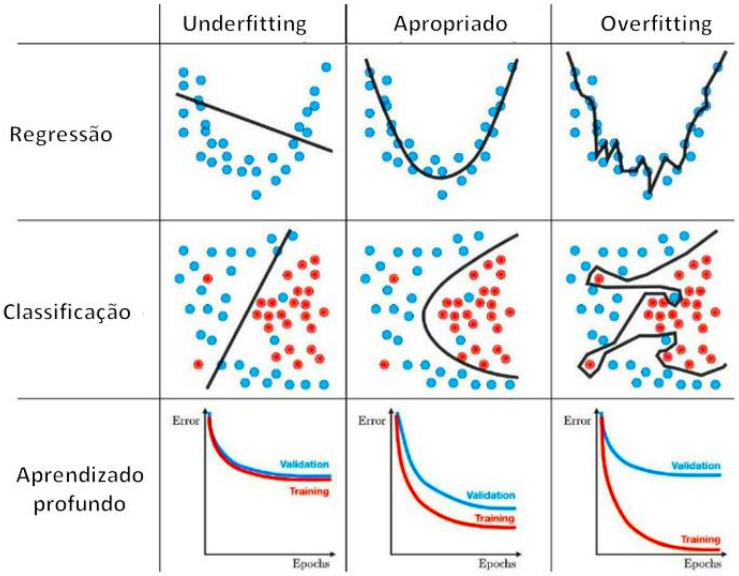
Entretanto, mesmo que essas métricas apresentem resultados satisfatórios, o risco de o algoritmo não ser eficiente o suficiente, a depender do que se espera dele, não deve ser desprezado. Em outras palavras, um pesquisador pode obter valores superiores a 0,95 em todas essas métricas e na prática o classificador apresentar eficiência aquém da desejada [68,69,70].

Tal fato, usualmente, é fruto do chamado *overfitting*, situação de natureza complexa na qual o modelo é bastante eficiente na fase de testes, mas não apresenta resultados satisfatórios diante de um conjunto novos de dados a serem classificados. Como essas métricas (acurácia, precisão, sensibilidade, especificidade, dentre outras) são obtidas através da matriz de confusão, elas são dependentes da capacidade do algoritmo em detectar os verdadeiros positivos (VP), verdadeiros negativos (VN), falsos positivos (FP) e falsos negativos (FN)[68,69,70].

Underfitting é um cenário em ciência de dados em que um modelo de dados é incapaz de capturar a relação entre as variáveis de entrada e saída com precisão, gerando uma alta taxa de erro no conjunto de treinamento e nos dados não vistos. Ocorre quando um modelo é muito simples, o que pode ser resultado de um modelo que precisa de mais tempo de treinamento, mais recursos de entrada ou menos regularização. Assim como o *overfitting*, quando um modelo é subajustado, ele não consegue estabelecer a tendência dominante nos dados, resultando em erros de treinamento e desempenho insatisfatório do modelo (Figura 19). Se um modelo não puder generalizar bem para novos dados, ele não poderá ser aproveitado para

tarefas de classificação ou previsão. A generalização de um modelo para novos dados é, em última análise, o que nos permite usar algoritmos de aprendizado de máquina todos os dias para fazer previsões e classificar dados.

Figura 22 - Ajustes finos: overfitting e underfitting



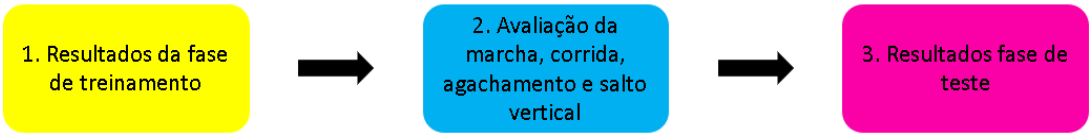
Fonte: O autor

Podemos entender melhor o *overfitting* olhando para o problema oposto, o *underfitting*. O *underfitting* ocorre quando um modelo é muito simples - informado por poucos recursos ou muito regularizado - o que o torna inflexível no aprendizado do conjunto de dados. Aprendizes simples tendem a ter menos variação em suas previsões, mas mais viés para resultados errados. Por outro lado, aprendizes complexos tendem a ter mais variação em suas previsões [68,69,70].

7 RESULTADOS E DISCUSSÕES

Foram reportados os resultados da CNN desenvolvida para as classes de movimentos locomotores caminhada, corrida, agachamento e salto vertical.

Figura 23 - Fluxograma da apresentação dos resultados.



Fonte: o autor

A figura 20 mostra sequencialmente os resultados obtidos em cada etapa do algoritmo para facilitar a compreensão deles.

Optou-se por detalhar o passo-a-passo dos algoritmos presentes em cada etapa da CNN, a saber: fase de treinamento, de validação e de teste.

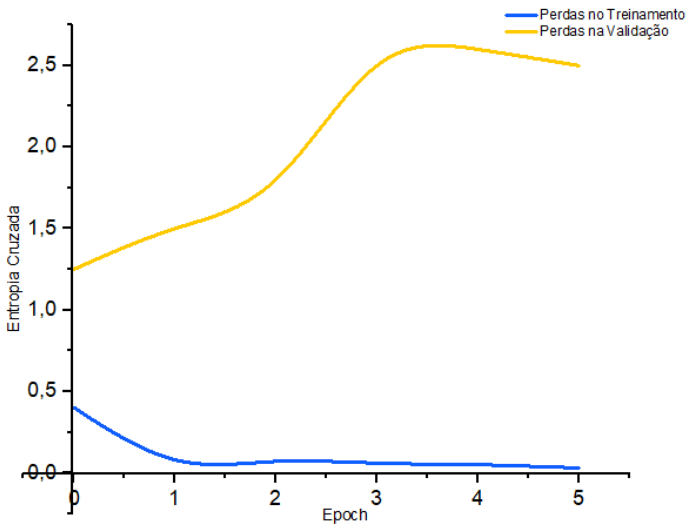
Em seguida serão apresentados os resultados das métricas adicionais que corroboram com as utilizadas em cada uma das fases, são elas: FMI, MCC e IY.

7.1 Resultados da fase de treinamento

Pode-se inspecionar o progresso do treinamento observando o histórico ao longo da epochs.

```
plt.figure(figsize=(8, 6))
for c in ['train_loss', 'valid_loss']:
    plt.plot(
        history[c], label=c)
plt.legend()
plt.xlabel('Epoch')
plt.ylabel('Average Negative Log Likelihood')
plt.title('Training and Validation Losses')
```

Gráfico 1 – Entropia Cruzada em função das *epochs* na fase de treinamento



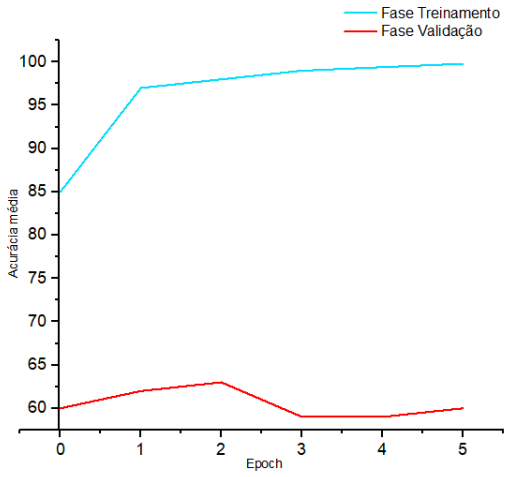
Fonte: o autor

No gráfico 1, pode-se observar como varia a entropia cruzada em função das *epochs*; esta é utilizada em algoritmos que recorrem a *backpropagation* para quantificar a diferença entre duas distribuições de probabilidades na função *SoftMax*.

Como esperado, a perda de treinamento diminui continuamente com as *epochs*. Em um determinado ponto, no entanto, a perda de validação para de diminuir. O uso do *Dropout*, provavelmente, garantiu um baixo *overfitting.* É importante destacar que o *overfitting* refere-se a um modelo onde os ruídos ou flutuações aleatórias nos dados de treinamento são captados e aprendidos como conceitos pelo modelo, ou seja, o algoritmo aprende com seus erros, sendo esta, a característica mais importante das CNNs. Não obstante, na fase de classificação, um eventual *overfitting* ou *underfitting* nas fases de treinamento e validação, podem impactar negativamente na eficiência da rede ao classificar novos dados.

```
plt.figure(figsize=(8, 6))
for c in ['train_acc', 'valid_acc']:
    plt.plot(
        100 * history[c], label=c)
plt.legend()
plt.xlabel('Epoch')
plt.ylabel('Average Accuracy')
plt.title('Training and Validation Accuracy')
```

Gráfico 2 - Acurácia em função das *epochs* na etapa de treinamento



Fonte: o autor

O gráfico 2 mostra como variou a acurácia na etapa de treinamento em função das epochs; observa-se que na primeira epoch na fase de treinamento foi atingida acurácia de 95%, valor esperado, pois nesta fase a rede é treinada com imagens específicas de cada movimento.

Tal como acontece com as perdas, a acurácia do treinamento aumenta (quase até a perfeição), enquanto a acurácia da validação atinge um patamar. O modelo é capaz de atingir uma precisão acima de 80% imediatamente, uma indicação de que os pesos de convolução aprendidos no *Imagenet* foram capazes de ser facilmente transferidos para nosso conjunto de dados.

7.2 Resultados da fase de validação

A próxima função itera através do conjunto de teste para fazer previsões para cada imagem. Ele calcula o desempenho para cada categoria.

```
    def evaluate(model, test_loader, criterion, topk=(1, 4)):
        """Measure the performance of a trained PyTorch model

        Params
        --------
            model (PyTorch model): trained cnn for inference
            test_loader (PyTorch DataLoader): test dataloader
            topk (tuple of ints): accuracy to measure

        Returns
        --------
            results (DataFrame): results for each category

        """

        classes = []
        losses = []
        # Resultados de precisão de retenção
        acc_results = np.zeros((len(test_loader.dataset), len(topk)))
        i = 0

        model.eval()
        with torch.no_grad():

            # Loop de teste
            for data, targets in test_loader:

                # Tensores para gpu
                if train_on_gpu:
                    data, targets = data.to('cuda'), targets.to('cuda')

                # Saída do modelo bruto
                out = model(data)
                # Repita cada exemplo
                for pred, true in zip(out, targets):
                    # Encontre a precisão do topk
                    acc_results[i, :] = accuracy(
                        pred.unsqueeze(0), true.unsqueeze(0), topk)
                    classes.append(model.idx_to_class[true.item()])
                    # Calcule a perda
                    loss = criterion(pred.view(1, n_classes), true.view(1))
                    losses.append(loss.item())
                    i += 1

        # Envie os resultados para um dataframe e calcule a média entre as
classes
        results = pd.DataFrame(acc_results, columns=[f'top{i}' for i in topk])
        results['class'] = classes
        results['loss'] = losses
        results = results.groupby(classes).mean()

        return results.reset_index().rename(columns={'index': 'class'})
```

```
    criterion = nn.NLLLoss()
    # Avalie o modelo em todos os dados de treinamento
    results = evaluate(model, dataloaders['test'], criterion)
    results.head()
```

Tabela 5 - Resultados das acurácias da CNN para identificação da marcha, corrida, agachamento e salto vertical.

Movimento	Acurácia		
	Testagem Inicial	**Testagem Final**	**Perda**
Caminhada	43,33	100,00	1,87
Corrida	40,66	100,00	1,60
Agachamento	100,00	100,00	8.249183e-07
Salto vertical	100,00	100,00	1.049687e-01

Fonte: o autor

7.3 Resultados da fase de teste

Na fase de teste foi utilizado o mesmo número de imagens para as quatro categorias de movimento reportadas até o momento.

Há uma relação entre o número de imagens de treinamento e a acurácia. No entanto, existem algumas classes com imagens cuja qualidade não é satisfatória, mesmo assim, o algoritmo é capaz de interpretá-las eficientemente.

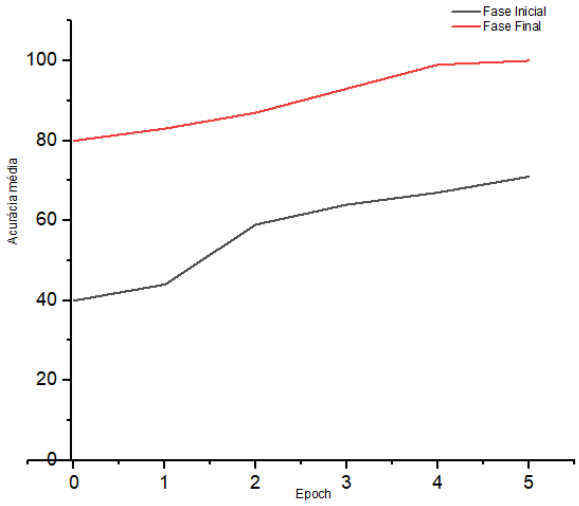
A seguir é mostrado o código para categorização da Caminhada, corrida, agachamento e salto vertical na presente CNN, para imagens escolhidas aleatoriamente e com diferentes níveis de resolução.

```
results = results.merge(cat_df, left_on='class', right_on='category').\
    drop(columns=['category'])

# Plot using seaborn
sns.lmplot(
    y='top1', x='n_train', data=results, height=6)
plt.xlabel('images')
plt.ylabel('Accuracy (%)')
plt.title('Top 1 Accuracy vs Number of Training Images')
plt.ylim(-5, 105)

print('Category with minimum accuracy.')
results.loc[results['top1'].idxmin]

print('Category with minimum images.')
results.loc[results['n_train'].idxmin]
```

```
sns.lmplot(
    y='top4', x='n_train', data=results, height=6)
plt.xlabel('images')
plt.ylabel('Accuracy (%)')
plt.title('Top 4 Accuracy vs Number of Training Images')
plt.ylim(-5, 105)
```

```
# Weighted column of test images
results['weighted'] = results['n_test'] / results['n_test'].sum()

# Create weighted accuracies
for i in (1, 4):
    results[f'weighted_top{i}'] = results['weighted'] * results[f'top{i}']

# Find final accuracy accounting for frequencies
top1_weighted = results['weighted_top1'].sum()
top5_weighted = results['weighted_top4'].sum()
loss_weighted = (results['weighted'] * results['loss']).sum()

print(f'Final test cross entropy per image = {loss_weighted:.4f}.')
print(f'Final test top 1 weighted accuracy = {top1_weighted:.2f}%')
print(f'Final test top 4 weighted accuracy = {top5_weighted:.2f}%')

Final test cross entropy per image = 0.8977.
Final test top 1 weighted accuracy = 71.00%
Final test top 4 weighted accuracy = 100.00%
```

Gráfico 3 - Acurácias da etapa final em função das *epochs*



Fonte: o autor

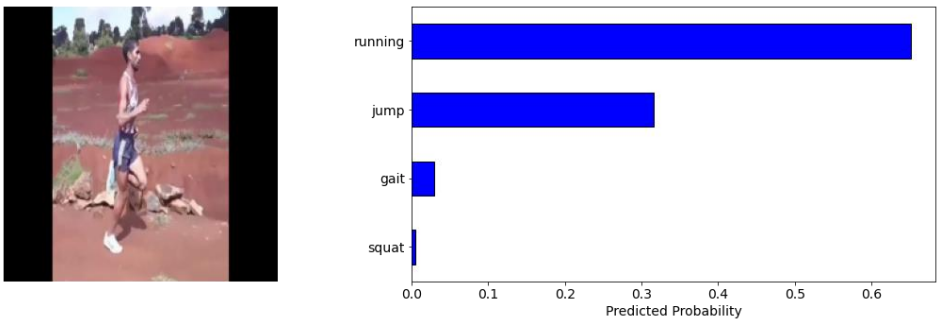
No gráfico 3 observa-se que a acurácia apresentou valores de 40% e 80% nas fases inicial e final, respectivamente. Na *epoch* 5 a acurácia apresentou valores de 70% e 100%, na identificação dos movimentos da Caminhada, corrida, agachamento e salto vertical.

CATEGORIZAÇÃO DOS MOVIMENTOS

A categorização da Caminhada e caminhada a partir de imagens com diferentes resoluções e ambientes é um desafio para o algoritmo de uma CNN; diante disso são apresentados a seguir exemplos nas figuras 18 a 25, destacando a eficiência da rede para classificação das mesmas.

```
running1 = '/content/datadir/test/running/testrunning (71).jpg'
gait2 = '/content/datadir/test/gait/testgait (110).jpg'
squat3 = '/content/datadir/test/squat/testsquat  (150).jpg'
jump4 =  '/content/datadir/test/jump/testjump   (8).jpg'


display_prediction(running1, model, 4)
display_prediction(gait2, model, 4)
display_prediction(squat3, model, 4)
display_prediction(jump4, model, 4)
```

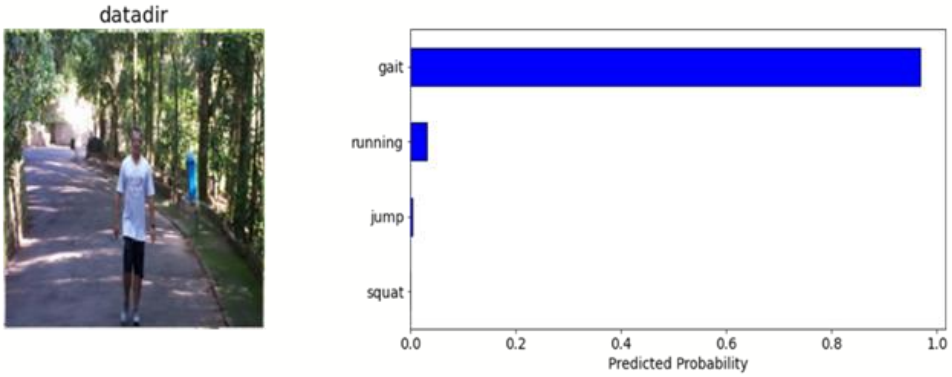
Figura 24 – Exemplo de classificação da corrida com acurácia.



Fonte: o autor

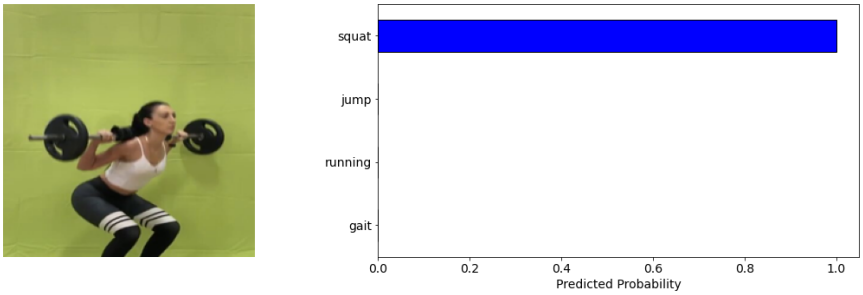
Na figura 20 em função da baixa resolução da imagem a probabilidade da predição da CNN é de mais de 65% para corrida.

Figura 25 - Exemplo de classificação da Caminhada com acurácia.



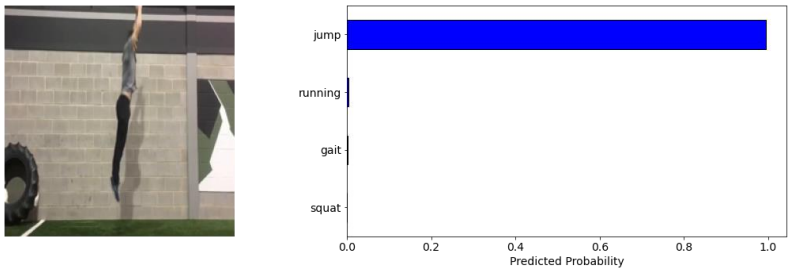
Fonte: o autor

Figura 26 - Exemplo de classificação do agachamento com acurácia.



Fonte: o autor

Figura 27 - Exemplo de classificação do salto vertical com acurácia.

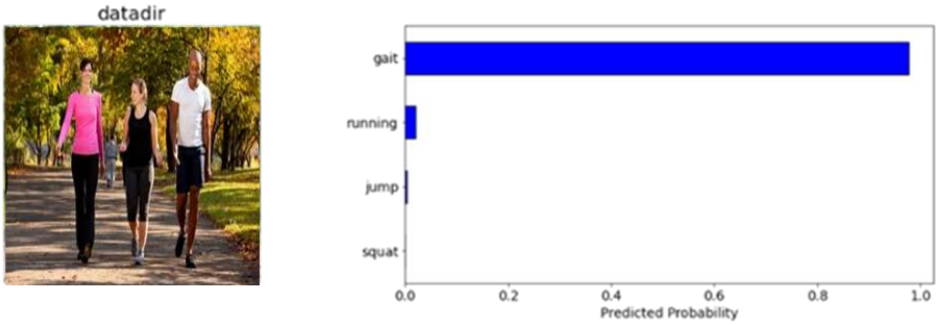


Fonte: o autor

TESTANDO A CAMINHADA

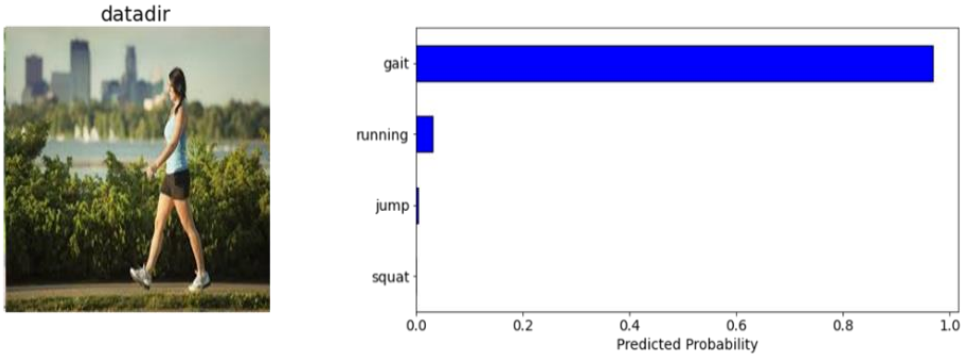
```
display_category(model, 'gait')

class       top1    top4      loss  n_train_x  n_valid_x
0  gait  43.333333  100.0  1.877736        300        150 /n
```

Figura 25 - Exemplo de classificação da Caminhada com acurácia.



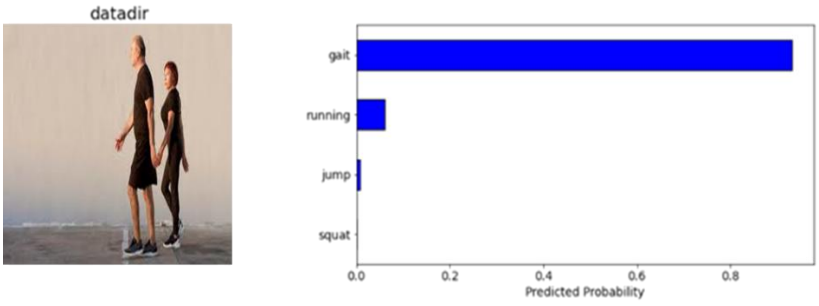
Fonte: o autor

Figura 26 - Exemplo de classificação da Caminhada com acurácia.



Fonte: o autor

Figura 30 - Exemplo de classificação da Caminhada com acurácia.

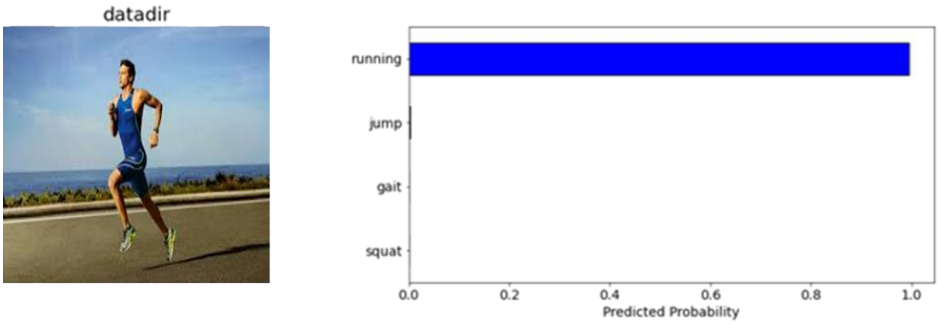


Fonte: o autor

TESTANDO A CORRIDA

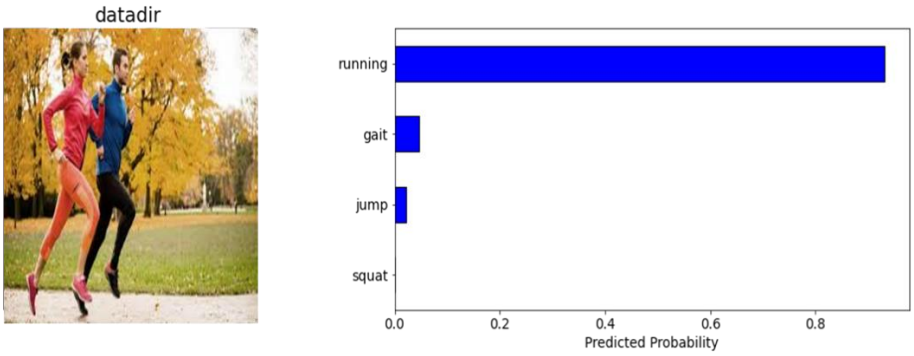
```
display_category(model, 'running')

class       top1    top4      loss  n_train_x  n_valid_x
2  running  40.666667  100.0  1.607944        300        150 /n
```

Figura 31 - Exemplo de classificação da corrida com acurácia.



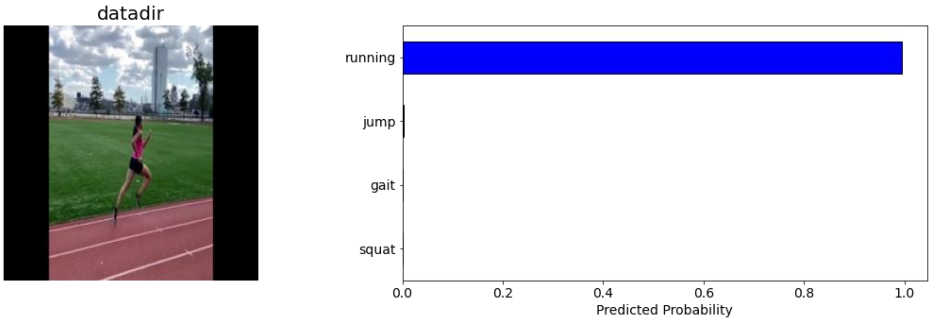
Fonte: o autor

Figura 32 - Exemplo de classificação da corrida com acurácia.



Fonte: o autor

Figura 27 - Exemplo de classificação da corrida com acurácia.

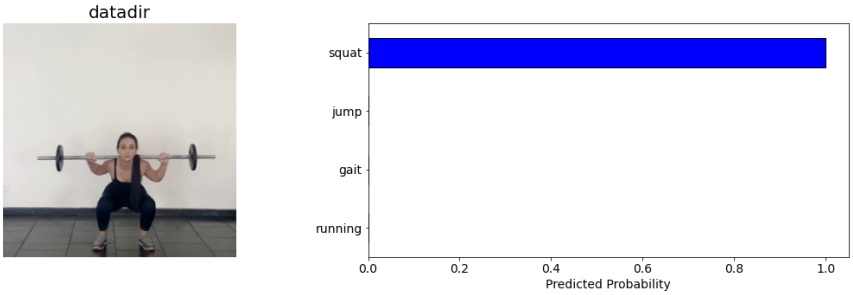


Fonte: o autor

TESTANDO O AGACHAMENTO

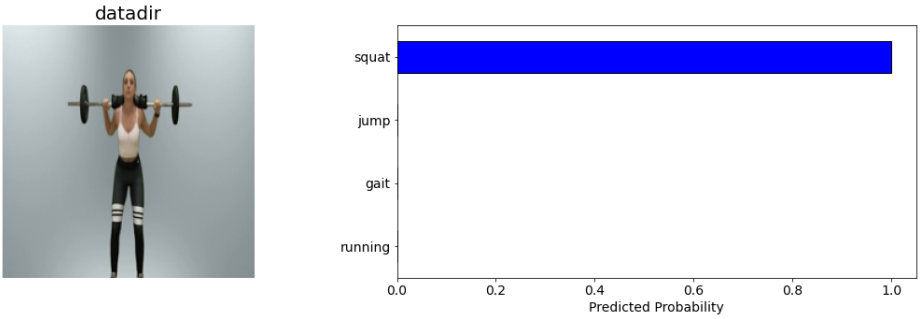
```
display_category(model, 'squat')

   class   top1   top4          loss  n_train_x  n_valid_x
3  squat  100.0  100.0  8.249183e-07        300        150 /n
```

Figura 28 - Exemplo de classificação do agachamento com acurácia.



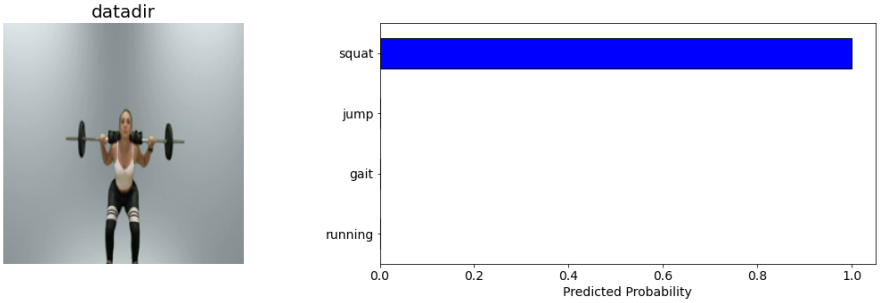
Fonte: o autor

Figura 29 - Exemplo de classificação do agachamento com acurácia.



Fonte: o autor

Figura 30 - Exemplo de classificação do agachamento com acurácia.

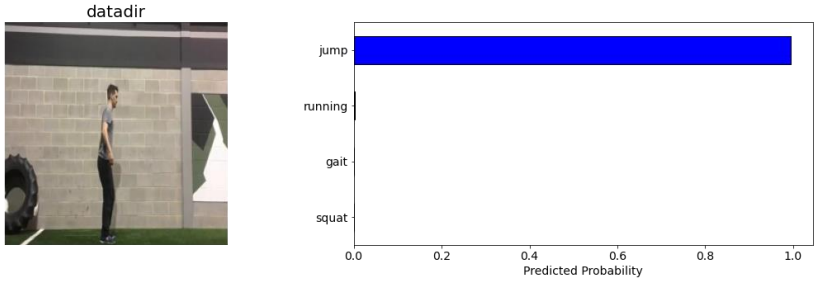


Fonte: o autor

TESTANDO O SALTO VERTICAL

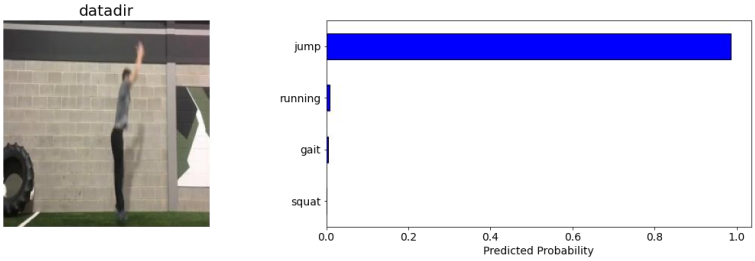
```
display_category(model, 'jump')

  class   top1   top4      loss  n_train_x  n_valid_x
1  jump  100.0  100.0  0.104969        300        150 /n
```

Figura 31 - Exemplo de classificação do salto com acurácia.



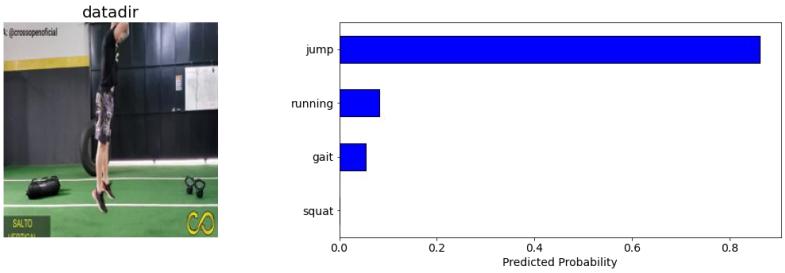
Fonte: o autor

Figura 32 - Exemplo de classificação do salto com acurácia.



Fonte: o autor

Figura 33 - Exemplo de classificação do salto com acurácia.



Fonte: o autor

Os resultados parciais aqui reportados evidenciam que os objetivos do presente estudo foram satisfatórios para os movimentos da Caminhada, corrida, agachamento e salto vertical.

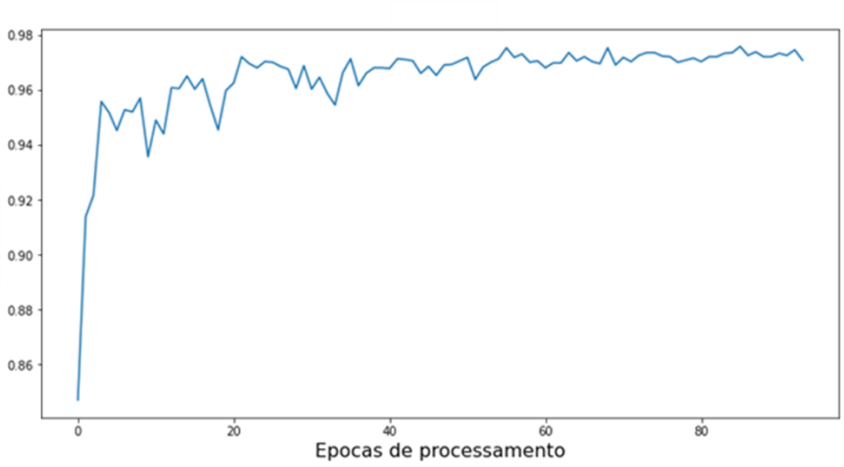
A matriz de confusão característica do algoritmo para classificação destes quatro padrões de movimentos foi:

1938	59
58	1937

A partir dessa matriz foram calculadas todas as métricas utilizadas no presente estudo, são elas: acurácia, precisão, sensibilidade, especificidade.

Os gráficos 4, 5, 6 e 7 mostram respectivamente a acurácia, precisão, sensibilidade e especificidade em função das *epochs* para os quatro movimentos analisados. Nos mesmos observa-se a complexidade destas métricas à medida em que a CNN processa suas diferentes *epochs*.

Gráfico 4 – Acurácia em função das *epochs* na fase final



A seguir são apresentados estudos similares que embora reportem exclusivamente suas acurácias, corroboram os resultados dessa métrica no presente trabalho.

Banjarey et al [71] desenvolveram CNN para classificação das atividades estar sentado, em pé, andando, lendo, dormindo e reclinado na qual obtiveram acurácia de 90,73% esse resultado provavelmente foi influenciado pelas sutis diferenças que podem existir entre as classes estar em *pé* ou *andando* e as classes estar *dormindo*, *lendo* ou *reclinado.*

Analogamente Yin et al [72] construiram uma CNN multiclasse para identificação dos movimentos de caminhada, corrida lenta, subir escadas, descer escadas, sentando-se, ficando de pé atingindo uma acurácia de 96,71%. Provavelmente essa acurácia foi influenciada também pela singularidade dos movimentos escolhidos.

Shi et al [73] compilou uma CNN para classificar os movimentos: caminhada, corrida, sentar, levantar, subir escadas, descer escadas, ficar em pé por 1 minuto apresentando uma acurácia 99,87%. Diante deste resultado é importante a reflexão relativa aos riscos metodológicos em reportar exclusivamente acurácia, pois estudos [67,68,69,70] mostram a importância de recorrer a outras métricas como precisão, sensibilidade e especificidade.

Os pesquisadores Fitjens et al [74] desenvolveram uma CNN para identificação de 19 articulações ou pontos anatômicos de interesse, importantes no movimento humano apresentando acurácia de 93,60%; resultado este bastante satisfatório dado o elevado número de classes.

Os autores Naik e Matlani [75] compilaram uma CNN para categorizar os seguintes movimentos: acenando com os braços, batendo palmas, correndo, caminhando, lutando boxe atingindo acurácia máxima de 98,60%, corroborando também o achado na presente pesquisa.

Xu-Hong et al [76] e Rui Lu[80] desenvolveram algoritmos inteligentes através de CNN para identificação de movimentos no basquetebol e hip-hop, alcançaram acurácias 96,9% e 83,21% respectivamente. Ambas as modalidades são dinâmicas, complexas, apresentando elevada probabilidade de eventuais vieses, especialmente o hip-hop onde existe um amplo espectro de movimentos criados nas diferentes coreografias.

Gráfico 5 – Precisão em função das *epochs* na fase final.

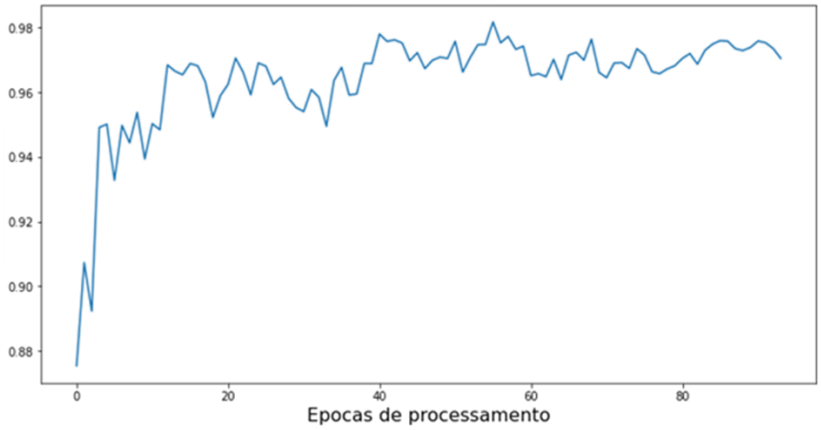


Gráfico 6 – Sensibilidade em função das *epochs* na fase final

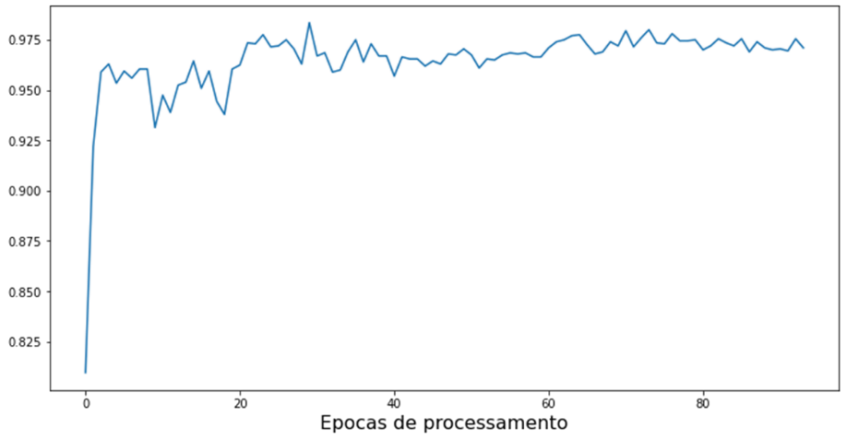


Gráfico 7 – Especificidade em função das *epochs* na fase final

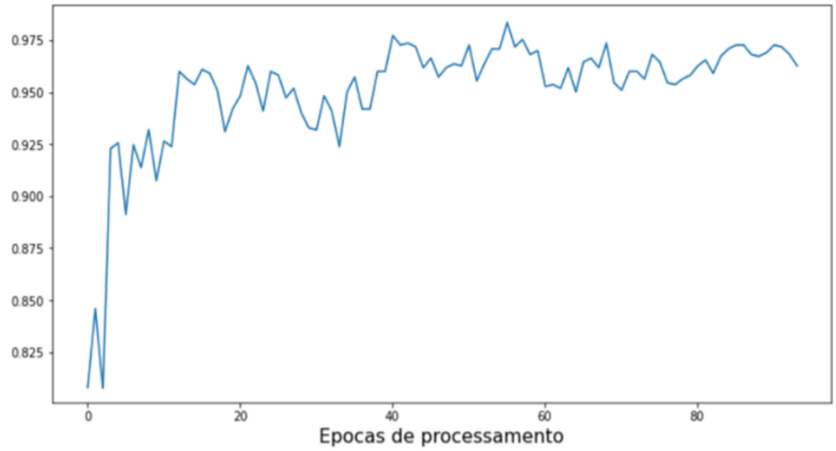


Tabela 6 - Resultados das métricas usuais para a CNN

Acurácia	0,97069
Precisão	0,97045
Sensibilidade	0,97094
Especificidade	0,97044

Na tabela 6 os resultados das métricas foram intencionalmente apresentados com cinco casas decimais para destacar como são sutis as variações das mesmas, decorrentes da diferença entre os números de FP (58) e FN (59).

Os autores Wang Z[77] e Wang Juan [78] desenvolveram CNNs para classificar o movimento humano em modalidades do atletismo sendo elas, salto em distância e corrida de fundo, com acurácias de 82,30% e 98,50% respectivamente. Resultados estes, provavelmente, decorrentes de suas características únicas no reportório locomotor; o que mais uma vez reforça a importância da utilização de outras métricas para garantir uma eventual maior robustez do algoritmo.

Nithyakani P. et al [79] ao pesquisarem a marcha patológica desenvolveram CNN obtendo acurácia de 97,40%; um exemplo de aplicação de algoritmos

inteligentes na reabilitação, mas que provavelmente carece de um arcabouço metodológico que possibilite corroborar este achado. Nesse sentido, ressalta-se a importância da utilização no mínimo das quatro métricas mais utilizadas: acurácia, precisão, sensibilidade e especificidade.

7.4 Resultados das métricas FMI, MCC e IY

De acordo com vários autores [62-67] as métricas FMI, MCC e IY em redes neurais convolucionais são importantes uma vez que auxiliam na depuração do algoritmo desenvolvido, minimizando assim, eventuais *overfitting* e *underfitting* na fase de teste.

Tabela 7 - Resultados das métricas FMI, MCC e IY

FMI	MCC	IY
0,971	0,941	0,941

Tais resultados corroboram [62-,70] com robustez a acurácia, a precisão, a especificidade e a sensibilidade da CNN, apresentando métricas bastante satisfatórias. Apesar de todas as métricas usuais (acurácia, precisão, especificidade e sensibilidade) neste cenário apresentarem resultados maiores que 0,950, foi a sensibilidade do algoritmo, ou seja, sua capacidade em identificar, por exemplo, diferenciar os quatro padrões de movimentos; tarefa nada simples, especialmente para corridas em baixas velocidades. Nesse sentido o MCC e IY obtidos sinalizam que cuidados metodológicos como aumentar o *dataset* de treinamento e/ou diminuir os ruídos, na tentativa de que ambos sejam maiores que 0,950, se consolidam como uma estratégia investigativa importantes.

8 CONSIDERAÇÕES FINAIS

Com os avanços tecnologicos nos ultimos anos, especialmente aqueles relativos a capacidade de processamento computacional e conexões velozes criou-se um cenário que possibilitou a presente pesquisa.

Com a complexidade e transfenomenalidade inerentes à area da saúde o uso de algoritmos inteligentes tem se tornado uma ferramenta importante para identificação de padrões diagnósticos oriundos das mais variadas fontes de dados (imagens, audios, sinais vitais, sequenciamento genético, dentre outros) com a otimização de prognosticos e intervenções que estejam comprometidos com um atendimento humanizado e com menos riscos às pessoas.

A presente Rede Neural Convolucional apresentou resultados satisfatorios para as métricas que são utilizadas pela maioria dos pesquisadores que recorrem a essa ferramenta. Não obstante, a opção metodologica de calcular as métricas FMI, MCC e IY foi decorrente do fato de que eventuais erros podem acontecer quando são utilizadas majoritariamente a acurácia, precisão, sensiblidade e especificidade. Nesse sentido os resultados obtidos e confirmados pelas métricas adicionais aumentam a confiabilidade e robustez da CNN.

Outro desfecho relevante da presente pesquisa refere-se ao seu carater inovador nas Ciências do Movimento Humano especialmente diante da potencial aplicabilidade no esporte, nas atividades laborais e na reabilitação.

REFERÊNCIAS

1. Passos R.P, Vilela Junior G.B. (2018) Inteligência artificial nas ciências da saúde. Centro de Pesquisas Avançadas em Qualidade de Vida. v. 10, n. 1
2. McCulloch WS, Pitts W. A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics. 1943;5(4):115-33.
3. Morris RG. D.O. Hebb: The Organization of Behavior, Wiley: New York; 1949. Brain research bulletin. 1999;50(5-6):437.
4. Brooks R, Hassabis D, Bray D, Shashua A. Turing centenary: Is the brain a good model for machine intelligence? Nature. 2012;482(7386):462-3.
5. Bellman N, Roth RS. The Bellman Continuum: World Scientific Publishing Company; 1987.
6. Kurzweil R, Richter R, Kurzweil R, Schneider ML. The age of intelligent machines: MIT press Cambridge; 1990.
7. Nilsson NJ, Nilsson NJ. Artificial intelligence: a new synthesis: Morgan Kaufmann; 1998.
8. Winston PH. Artificial Intelligence: Addison-Wesley Publishing Company; 1992.
9. Momeni M, Rahmani M. Speech signal analysis of alzheimer's diseases in farsi using auditory model system. Cognitive Neurodynamics. 2021;15(3):453-61.
10. Rohanian M, Hough J, Purver M. Multi-modal fusion with gating using audio, lexical and disfluency features for Alzheimer's Dementia recognition from spontaneous speech. 2021.
11. Burton WS, Myers CA, Rullkoetter PJ. Machine learning for rapid estimation of lower extremity muscle and joint loading during activities of daily living. Journal of biomechanics. 2021;123:110439.
12. Albert JA, Owolabi V, Gebel A, Brahms CM, Granacher U, Arnrich B. Evaluation of the Pose Tracking Performance of the Azure Kinect and Kinect v2 for Gait Analysis in Comparison with a Gold Standard: A Pilot Study. 2020;20(18):5104.
13. Lin B-S, Lee I-J, Fahn C-S, Lee Y-F, Chou W-J, Wu M-L. Depth-Camera Based Energy Expenditure Estimation System for Physical Activity Using Posture Classification Algorithm. 2021;21(12):4216.
14. Chow DHK, Tremblay L, Lam CY, Yeung AWY, Cheng WHW, Tse PTW. Comparison between Accelerometer and Gyroscope in Predicting Level-Ground Running Kinematics by Treadmill Running Kinematics Using a Single Wearable Sensor. 2021;21(14):4633.
15. Jiang Y, Hernandez V, Venture G, Kulić D, K. Chen B. A Data-Driven Approach to Predict Fatigue in Exercise Based on Motion Data from Wearable Sensors or Force Plate. 2021;21(4):1499.
16. Athavale VA, Gupta SC, Kumar D, Savita. Human Action Recognition Using CNN-SVM Model. Advances in Science and Technology. 2021;105:282-90.
17. Estler KE. Predictions of Knee Joint Contact Forces Using Only Kinematic Inputs with a Recurrent Neural Network. 2021.
18. Wu H, Wang L. Analysis of lower limb high-risk injury factors of patellar tendon enthesis of basketball players based on deep learning and big data. The Journal of Supercomputing. 2021.
19. Chalangari P. Deep Learning and Trigonometric Adjustment in Estimation of Lower Extremity Angles: Concordia University; 2020.

20. Kwon H, Wang B, Abowd GD, Plötz T. Approaching the Real-World: Supporting Activity Recognition Training with Virtual IMU Data. 2021;5(3 %J Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.):Article 111.
21. Alcantara RS, Edwards WB, Millet GY, Grabowski AM. Predicting continuous ground reaction forces from accelerometers during uphill and downhill running: A recurrent neural network solution. 2021.
22. Zaroug A, Lai DTH, Mudie K, Begg R. Lower Limb Kinematics Trajectory Prediction Using Long Short-Term Memory Neural Networks. Front Bioeng Biotechnol. 2020;8:362.
23. Sultana A, Deb K, Dhar PK, Koshiba T. Classification of Indoor Human Fall Events Using Deep Learning. Entropy (Basel). 2021;23(3).
24. Hu B, Li S, Chen Y, Kavi R, Coppola S. Applying deep neural networks and inertial measurement unit in recognizing irregular walking differences in the real world. Appl Ergon. 2021;96:103414.
25. Jaouedi N, Boujnah N, Bouhlel M. A Novel Recurrent Neural Networks Architecture for Behavior Analysis. The International Arab Journal of Information Technology. 2021;18(2).
26. Kim J, Cho J. Low-Cost Embedded System Using Convolutional Neural Networks-Based Spatiotemporal Feature Map for Real-Time Human Action Recognition. Applied Sciences. 2021;11(11).
27. Kumar BS, Raju SV, Reddy HV. Human Action Recognition Using A Novel Deep Learning Approach. IOP Conference Series: Materials Science and Engineering. 2021;1042(1).
28. Martindale CF, Christlein V, Klumpp P, Eskofier BM. Wearables-based multi-task gait and activity segmentation using recurrent neural networks. Neurocomputing. 2021;432:250-61.
29. He D, Li L. A Novel Deep Learning Method Based on Modified Recurrent Neural Network for Sports Posture Recognition2020.
30. Ghate V, C SH. Hybrid deep learning approaches for smartphone sensor-based human activity recognition. Multimedia Tools and Applications. 2021.
31. Albaba M, Qassab A, Yilmaz A. Human activity recognition and classification using of convolutional neural networks and recurrent neural networks. International Journal of Applied Mathematics Electronics and Computers. 2020:185-9.
32. Soares F, Carvalho V, Cunha P, Barbosa P. Classification of Taekwondo Techniques using Deep Learning Methods: First Insights. Proceedings of the 14th International Joint Conference on Biomedical Engineering Systems and Technologies2021. p. 201-8.
33. Fausett L. Fundamentals of Neural Networks Architecture Algorithms and Applications 1994.
34. Norvig P. Paradigms of Artificial Intelligence Programming: cases studies in common lisp1991.
35. Capecchi V, Buscema M, Contucci P, D'Amore B. Applications of mathematics in models artificial neural networks 2010.
36. Khemani D. A First Course in Artificial Intelligence2013.
37. Fileni CHP. Desenvolvimento de um aplicativo web responsivo para construção do conhecimento em biomecânica: UNIMEP; 2021.
38. Pyrkov TV, Slipensky K, Barg M, Kondrashin A, Zhurov B, Zenin A, et al. Extracting biological age from biomedical data via deep learning: too much of a good thing? Scientific reports. 2018;8(1):5210.

39. Zia A, Guo L, Zhou L, Essa I, Jarc A. Novel evaluation of surgical activity recognition models using task-based efficiency metrics. International journal of computer assisted radiology and surgery. 2019;14(12):2155-63.
40. Arvin R, Khattak AJ, Qi H. Safety critical event prediction through unified analysis of driver and vehicle volatilities: Application of deep learning methods. Accident; analysis and prevention. 2021;151:105949.
41. Shahtalebi S, Asif A, Mohammadi A. Siamese Neural Networks for EEG-based Brain-computer Interfaces. Annual International Conference of the IEEE Engineering in Medicine and Biology Society IEEE Engineering in Medicine and Biology Society Annual International Conference. 2020;2020:442-6.
42. Ankita, Rani S, Babbar H, Coleman S, Singh A, Aljahdali HM. An Efficient and Lightweight Deep Learning Model for Human Activity Recognition Using Smartphones. Sensors (Basel). 2021;21(11).
43. Feigl T, Kram S, Woller P, Siddiqui RH, Philippsen M, Mutschler C. RNN-Aided Human Velocity Estimation from a Single IMU. Sensors (Basel). 2020;20(13).
44. León J, Escobar JJ, Ortiz A, Ortega J, González J, Martín-Smith P, et al. Deep learning for EEG-based Motor Imagery classification: Accuracy-cost trade-off. PloS one. 2020;15(6):e0234178.
45. Ma X, Qiu S, Du C, Xing J, He H. Improving EEG-Based Motor Imagery Classification via Spatial and Temporal Recurrent Neural Networks. Annual International Conference of the IEEE Engineering in Medicine and Biology Society IEEE Engineering in Medicine and Biology Society Annual International Conference. 2018;2018:1903-6.
46. Sun B, Cao S, He J, Yu L. Affect recognition from facial movements and body gestures by hierarchical deep spatio-temporal features and fusion strategy. Neural networks : the official journal of the International Neural Network Society. 2018;105:36-51.
47. Xie Z, Schwartz O, Prasad A. Decoding of finger trajectory from ECoG using deep learning. Journal of neural engineering. 2018;15(3):036009.
48. Dorschky E, Nitschke M, Martindale CF, van den Bogert AJ, Koelewijn AD, Eskofier BM. CNN-Based Estimation of Sagittal Plane Walking and Running Biomechanics From Measured and Simulated Inertial Sensor Data. Front Bioeng Biotechnol. 2020;8:604.
49. Li Q, Li Q, Liu C, Shashikumar SP, Nemati S, Clifford GD. Deep learning in the cross-time frequency domain for sleep staging from a single-lead electrocardiogram. Physiological measurement. 2018;39(12):124005.
50. Opałka S, Stasiak B, Szajerman D, Wojciechowski A. Multi-Channel Convolutional Neural Networks Architecture Feeding for Effective EEG Mental Tasks Classification. Sensors (Basel). 2018;18(10).
51. Sardari F, Paiement A, Hannuna S, Mirmehdi M. VI-Net-View-Invariant Quality of Human Movement Assessment. Sensors (Basel). 2020;20(18).
52. Jiang Y, Chen C, Zhang X, Chen C, Zhou Y, Ni G, et al. Shoulder muscle activation pattern recognition based on sEMG and machine learning algorithms. Computer methods and programs in biomedicine. 2020;197:105721.
53. Khalili N, Lessmann N, Turk E, Claessens N, Heus R, Kolk T, et al. Automatic brain tissue segmentation in fetal MRI using convolutional neural networks. Magnetic resonance imaging. 2019;64:77-89.
54. Langerhuizen DWG, Bulstra AEJ, Janssen SJ, Ring D, Kerkhoffs G, Jaarsma RL, et al. Is Deep Learning On Par with Human Observers for Detection of

Radiographically Visible and Occult Fractures of the Scaphoid? Clinical orthopaedics and related research. 2020;478(11):2653-9.
55. Lin J, Clancy NT, Qi J, Hu Y, Tatla T, Stoyanov D, et al. Dual-modality endoscopic probe for tissue surface shape reconstruction and hyperspectral imaging enabled by deep neural networks. Medical image analysis. 2018;48:162-76.
56. Mammone N, Ieracitano C, Morabito FC. A deep CNN approach to decode motor preparation of upper limbs from time-frequency maps of EEG signals at source level. Neural networks : the official journal of the International Neural Network Society. 2020;124:357-72.
57. Panwar M, Dyuthi SR, Chandra Prakash K, Biswas D, Acharyya A, Maharatna K, et al. CNN based approach for activity recognition using a wrist-worn accelerometer. Annual International Conference of the IEEE Engineering in Medicine and Biology Society IEEE Engineering in Medicine and Biology Society Annual International Conference. 2017;2017:2438-41.
58. Talo M, Yildirim O, Baloglu UB, Aydin G, Acharya UR. Convolutional neural networks for multi-class brain disease detection using MRI images. Computerized medical imaging and graphics : the official journal of the Computerized Medical Imaging Society. 2019;78:101673.
59. Ogata K, Matsumoto Y. Estimating Movements of Human Body for the Shirt-Type Wearable Device Mounted on the Strain Sensors Based on Convolutional Neural Networks. Annual International Conference of the IEEE Engineering in Medicine and Biology Society IEEE Engineering in Medicine and Biology Society Annual International Conference. 2019;2019:5871-6.
60. Panwar M, Biswas D, Bajaj H, Jobges M, Turk R, Maharatna K, et al. Rehab-Net: Deep Learning Framework for Arm Movement Classification Using Wearable Sensors for Stroke Rehabilitation. IEEE transactions on bio-medical engineering. 2019;66(11):3026-37.
61. Vilela Junior GdB, Pablo Passos R. REDES NEURAIS CONVOLUCIONAIS E PADRÃO CPAQV 100 PARA ANÁLISE DO MOVIMENTO HUMANO. Centro de Pesquisas Avançadas em Qualidade de Vida. 2020.
62. THARWAT, A. (2021) Classification assessment methods. Applied Computing and Informatics Vol. 17 No. 1, pp. 168-192.
63. S. BOUGHORBEL, F. JARRAY, M. ELANBARI, (2017) Optimal classifier for imbalanced data using matthews correlation coefficient metric, PLoS One 12 (6)
64. READ, J. et al. (2011), Classifier chains for multi-label classification. Mach Learn v.85, p.333–359.
65. D M. SOKOLOVA, N. JAPKOWICZ, S. SZPAKOWICZ, (2006) Beyond accuracy, f-score and roc: a family of discriminant measures for performance evaluation, in: Australasian Joint Conference on Artificial Intelligence, Springer, pp. 1015–1021.
66. VILELA JUNIOR, G.B.et al. (2022) Metrics used to evaluate the effciency of classifers in smart algorithm. Revista CPAQV. V.14, n.2.
67. VILELA JUNIOR, G.B. (2022) Artifcial Intelligence classifers in estimation the power of lower members in soccer. 2021. Revista CPAQV. V.13, n.3. DOI: 10.36692/ v13n3-22.
68. VILELA JUNIOR, G.B. (2022) Determinação das métricas usuais a partir da matriz de confusão de classificadores multiclasses em algoritmos inteligentes nas ciências do movimento humano. Revista CPAQV. V.14, n.3. DOI: 10.36692/ v14n2-01.

69. VILELA JUNIOR, G.B. Métricas utilizadas para avaliar a eficiência de classificadores em algoritmos inteligentes. Revista CPAQV. V.14, n.2. DOI: 10.36692/ v14n2-01R.
70. VILELA JUNIOR, G.B. Importância do índice fowlkes-mallows (FMI), do coeficientemde correlação de matthews (MCC) e do índice youden (IY) nos classificadores de inteligência artificial na área da saúde. Revista CPAQV. V.14, n.3. DOI: 10.36692/ v14n3-01.
71. BANJAREY, K., SAHU, S.P., DEWANGAN, D.K. (2022). Human Activity Recognition Using 1D Convolutional Neural Network. In: Shakya, S., Balas, V.E., Kamolphiwong, S., Du, KL. (eds) Sentimental Analysis and Deep Learning. Advances in Intelligent Systems and Computing, vol 1408. Springer, Singapore. https://doi.org/10.1007/978-981-16-5157-1_54
72. YIN, X., LIU, Z., LIU, D. ET AL. A Novel CNN-based Bi-LSTM parallel model with attention mechanism for human activity recognition with noisy data. Sci Rep 12, 7878 (2022). https://doi.org/10.1038/s41598-022-11880-8
73. SHI, W. et al.(2022) HAR Based on Multichannel Convolutional Neural Network With Data Augmentation IEE, DOI: https://doi.org/10.1109/ACCESS.2022.3192452
74. FILTJENS, B. et al. (2022). Skeleton-Based Action Segmentation with Multi-Stage Spatial-Temporal Graph Convolutional Neural Networks https://doi.org/10.48550/arXiv.2202.01727
75. NAIK, A., MATLANI, R., Dynamic Video-Based Activity Recognition using 3D Convolutional Neural Network (2022). Available at: http://dx.doi.org/10.2139/ssrn.4108889
76. XU-HONG et al (2022) Analysis of Basketball Technical Movements Based on Human-Computer Interaction with Deep Learning. Computational Intelligence and Neuroscience Article ID 4247082. https://doi.org/10.1155/2022/4247082
77. WANG, Z. (2022) Long Jump Action Recognition Based on Deep Convolutional Neural Network. Computational Intelligence and Neuroscience Volume 2022, Article ID 3832118, 14 pages https://doi.org/10.1155/2022/3832118
78. WANG JUAN (2022) Convolutional Neural Network and Computer Vision-Based Action Correction Method for Long-Distance Running. Technology Security and Communication Networks Volume 2022, Article ID 1467451, 10 pages https://doi.org/10.1155/2022/1467451
79. NITHYAKANI P. et al. (2022) Classification Of Gait Pathology Using Enhanced Convolutional Neural Network. International Conference on Computer Communication and Informatics (ICCCI) doi: https://doi.org/10.1109/ICCCI54379.2022.9740784
80. RUI LU (2022) Analysis of Main Movement Characteristics of Hip Hop Dance Based on Deep Learning of Dance Movements Computational Intelligence and Neuroscience. Article ID 6794018, 8 pages https://doi.org/10.1155/2022/679401
81.Deisenroth, Marc Peter, A. Aldo Faisal, and Cheng Soon Ong. Mathematics for machine learning. Cambridge University Press, 2020.

ANEXO

UNIVERSIDADE METODISTA DE PIRACICABA - UNIMEP

PARECER CONSUBSTANCIADO DO CEP

DADOS DO PROJETO DE PESQUISA

Título da Pesquisa: MÉTODOS DA INTELIGÊNCIA ARTIFICIAL APLICADOS NA ANÁLISE DO MOVIMENTO HUMANO

Pesquisador: GUANIS DE BARROS VILELA JUNIOR

Área Temática:

Versão: 1

CAAE: 33912120.9.0000.5507

Instituição Proponente: INSTITUTO EDUCACIONAL PIRACICABANO DA IGREJA METODISTA

Patrocinador Principal: Financiamento Próprio

DADOS DO PARECER

Número do Parecer: 4.126.546

Apresentação do Projeto:

Projeto adequadamente apresentado, contendo todos os dados necessários para sua análise.

Objetivo da Pesquisa:

Objetivos claros, coerentes com o desenho do projeto e exequíveis dentro do cronograma exposto.

Avaliação dos Riscos e Benefícios:

Os riscos aos sujeitos são mínimos e o projeto assegura o cuidado para reduzi-los. Os benefícios (diretos e indiretos) aos sujeitos estão presentes e superam os riscos.

Comentários e Considerações sobre a Pesquisa:

Destacam-se a relevância e as contribuições da pesquisa apresentada. As bases teóricas estão adequadas, a metodologia é coerente e a coleta de dados é adequada à proposta.

Considerações sobre os Termos de apresentação obrigatória:

O TCLE apresenta as informações necessárias. Acrescentar no início do texto a frase: Você está sendo convidado(a) a participar de um estudo na área da Ciências do Movimento Humano.

Conclusões ou Pendências e Lista de Inadequações:

O projeto está aprovado.

Considerações Finais a critério do CEP:

Este colegiado acolhe o parecer acima descrito e aprova o projeto.

Endereço: Rodovia do Açúcar, Km 156
Bairro: Taquaral **CEP:** 13.400-911
UF: SP **Município:** PIRACICABA
Telefone: (19)3124-1513 **Fax:** (19)3124-1515 **E-mail:** comitedeetica@unimep.br