DOI: 10.5748/19CONTECSI/PSE/EDU/6933

AVALIAÇÃO DO USO DO TENSORFLOW E DO PYTORCH EM UMA REDE NEURAL ARTIFICIAL UTILIZADA PARA O RECONHECIMENTO FACIAL.

Marcos Rodrigo Mendes Saavedra ; https://orcid.org/0000-0002-4698-6202
Universidade Federal do Pará

Flávio Ramon Almeida de Souza ; https://orcid.org/0000-0002-8022-2468
Universidade Federal do Pará

Josivaldo de Souza Araújo ; https://orcid.org/0000-0001-6890-6923
Universidade Federal do Pará

EVALUATION OF THE USE OF TENSORFLOW AND PYTORCH IN AN ARTIFICIAL NEURAL NETWORK USED FOR FACIAL RECOGNITION.

AVALIAÇÃO DO USO DO *TENSORFLOW* E DO *PYTORCH* EM UMA REDE NEURAL ARTIFICIAL UTILIZADA PARA O RECONHECIMENTO FACIAL.

ABSTRACT: *The use of heterogeneous computing (CPU and GPU) in general purpose application processing has evolved exponentially in recent years. This type of architecture aims to accelerate the achievement of results, which contributes to the reduction in processing time. For this reason, these systems have also been used in applications that need to recognize some type of pattern, such as facial recognition. Facial recognition systems have gained notoriety in recent years because they are more accurate and non-invasive in the process of identifying people previously registered in a database. With this objective, this work uses two parallel libraries, TensorFlow and PyTorch, in order to evaluate the reduction of processing time in a facial recognition application that uses an Artificial Neural Network. The results were promising with the reduction of processing time up to three times when compared to sequential processing time.*

Keywords: *TensorFlow, Pytorch, Video Board, Facial Recognition, Artificial Neural Network.*

RESUMO: O uso da computação heterogênea (CPU e GPU) no processamento de aplicações de propósito geral evoluiu exponencialmente nos últimos anos. Esse tipo de arquitetura tem como finalidade a aceleração na obtenção dos resultados, o que contribui para a redução no tempo de processamento. Por este motivo, esses sistemas passaram a ser utilizados, também, em aplicações que necessitam reconhecer algum tipo de padrão, como por exemplo, os de reconhecimento facial. Sistemas de reconhecimento facial ganharam notoriedade nos últimos anos por serem mais precisos e não invasivos no processo de identificação de pessoas previamente cadastradas em uma base de dados. Com esse objetivo, esse trabalho utiliza duas bibliotecas paralelas, *TensorFlow* e *PyTorch*, com a finalidade de avaliar a redução do tempo de processamento em uma aplicação de reconhecimento facial que utiliza uma Rede Neural Artificial. Os resultados se mostraram promissores com a redução do tempo de processamento em até três vezes quando comparado com o tempo de processamento sequencial.

Palavras-chave: *TensorFlow*, *Pytorch*, Placa Gráfica, Reconhecimento Facial, Rede Neural Artificial.

1. INTRODUÇÃO

Nas últimas décadas, com o avanço da tecnologia, para se acessar informações, equipamentos e até mesmo lugares é necessário utilizar algum sistema de autenticação que comprove a identificação de quem está acessando. Essa identificação pode ser realizada através de sistemas que utilizem cartões de acesso ou senhas digitadas. No entanto, esses sistemas podem ser adulterados, de forma que terceiros se passem pelos verdadeiros usuários (desde que tenham a posse do cartão de acesso ou senha), o que faz com que tenham

prejuízos de violação de dados, fraude de identidade, perdas econômicas e de privacidade. Soma-se a isso, ainda, o desafio dos usuários de memorizar algumas dezenas de diferentes senhas (Souza et. al., 2022).

Por conta disso, sistemas de reconhecimento facial têm se tornado uma importante área de pesquisa para a comunidade em Segurança Computacional, pois proporciona a identificação de cada indivíduo como único, através das suas características pessoais e de uma forma não invasiva. Esses sistemas estão evoluindo a ponto de atingir desempenhos similares ao humano através do uso da Inteligência Artificial (Souza and Araújo, 2021).

A Inteligência Artificial (IA) é uma tecnologia de grande relevância na análise de grandes volumes de dados e devido ao seu potencial, vem impactando significativamente aplicações e serviços, com campos cada vez mais específicos. Um dos campos presentes na IA está o de Aprendizado de Máquina (ou *Machine Learning - ML*), onde um conjunto de métodos, ou algoritmos, utilizam a probabilidade e a estatística para obter resultados mais satisfatórios em uma tarefa. Entre as diversas abordagens de ML, pode-se destacar as Redes Neurais Artificiais (RNA) (Araújo, et. al. 2022).

As RNAs utilizam um conjunto de dados finito para realizar o treinamento e os testes, o que faz com que esses sistemas necessitem, às vezes, de grande poder computacional, como sistemas de alto desempenho. Esses sistemas, atualmente, híbridos, contam com um número elevado de núcleos de processamento, tanto em CPU quanto em GPU.

Com o objetivo de avaliar ferramentas e arquitetura paralela, este trabalho apresenta um estudo de caso que tem como base o reconhecimento facial que é realizado através do uso de uma Rede Neural Artificial com placa gráfica e duas bibliotecas paralelas, o *TensorFlow* e o *Pytorch*. A Rede Neural é executada em uma placa gráfica com o objetivo de obter ganhos de desempenho no processo de treinamento e teste. Os resultados mostraram-se satisfatórios com a redução do tempo de processamento quando comparado com a execução sequencial.

2. TRABALHOS RELACIONADOS

Os estudos de bibliotecas paralelas estão revolucionando a computação e sua utilização, em problemas relacionados à identificação de imagens, vem desempenhando um papel de otimização aos algoritmos relacionados à visão computacional.

O trabalho apresentado (Terraza Arciniegas et al, 2022) trouxe uma proposta capaz de analisar a atenção do aluno em aulas online e determinar quando ele está atento a determinada explicação. Para isso, foi desenvolvida uma técnica automatizada baseada em algoritmo RNA de visão computacional com a finalidade de verificar e alterar a dinâmica das aulas quando necessário.

No trabalho de (Cristiano A. Kúnas and Padoin, 2020) é utilizada uma rede neural artificial para detecção de sentimentos, pois é uma área da visão computacional recorrente. O trabalho tem como objetivo treinar uma base de dados pública e aplicar no reconhecimento de sentimentos expressados em textos e postagens utilizando Python junto com os módulos *Keras, Tensorflow* e Scikit-Learn.

O trabalho realizado por (Leandro P. Heck and Padoin, 2020) visa reconhecer seis emoções a partir de expressões faciais. Para isso utiliza uma RNA Convolucional com a linguagem python, as bibliotecas *TensorFlow* 2.0, keras, OpenCV, Scikit-Learn, Numpy, Pandas, Matplotlib e testa em diferentes arquiteturas com CPU e GPU.

3. DESCRIÇÃO DO PROBLEMA

Com objetivo de avaliar as ferramentas propostas, esse trabalho teve como base o sistema desenvolvido em (Souza and Araújo, 2021). No trabalho, foi utilizada uma estrutura

composta de cinco etapas, que vai da captação (aquisição) da imagem, até a transformação desta em um vetor binário, o qual é utilizado como entrada pela Rede Neural, para o reconhecimento facial. As etapas podem ser visualizadas na Figura 1.

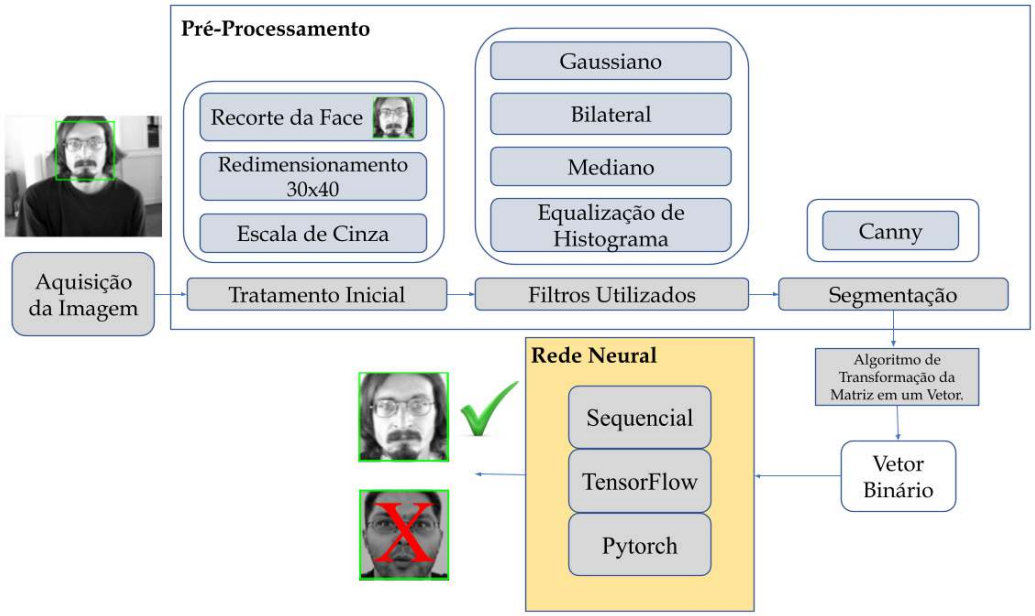

Figura 1. Etapas da execução sequencial.
Fonte: Adaptada de (Souza and Araujo 2021).

A etapa inicial do processo diz respeito a aquisição da imagem, ou seja, o momento em que irá ser realizada a captura da face do alvo em questão, e isso pode ser realizado por qualquer dispositivo eletrônico, como por exemplo, câmeras de monitoramento de ruas ou prédios. A imagem adquirida é então armazenada para que possa ser tratada nas etapas seguintes.

O pré-processamento tem como objetivo corrigir possíveis defeitos e imperfeições que possam ter sido ocorridos durante a etapa de aquisição, como por exemplo, a filtragem de ruídos, o aumento do contraste, além de operações de rotação, ou redimensionamento. Na etapa de tratamento inicial, são realizados três procedimentos: o recorte apenas da área do rosto, o redimensionamento da imagem (para o tamanho de 40x30 pixels) e a aplicação de um filtro para a transformação em escala de cinza (esse procedimento é necessário, pois a técnica de detecção de bordas, depende de uma imagem em escala de cinza para determinar os contornos da face humana, e serve, também, separar o primeiro plano (face humana), do fundo. Neste trabalho, foram utilizados os filtros Gaussiano, Equalizador de Histograma, Mediana e o Bilateral (Souza and Araújo, 2021).

Após a aplicação de um dos filtros, a imagem chega à etapa da segmentação, onde o objetivo nesta fase é transformar uma matriz de observação (imagem original) em uma matriz de feições (matriz na escala RGB), empregando alguma função ortogonal ou não-ortogonal, de modo a se obter um espaço de características não correlacionado. Para este trabalho, foi utilizado a técnica de detecção de borda desenvolvida por Canny, embora seja a mais complexa, é também a de melhor desempenho. O detector de bordas Canny possui três objetivos básicos: baixa taxa de erro, os pontos da borda devem ser bem localizados, e resposta única para os pontos de uma borda (Souza and Araújo, 2021).

Após a etapa da segmentação, a imagem passa a conter apenas duas cores básicas: o branco, que na escala RGB decimal, é representado pelo valor (255, 255, 255), e o preto,

que na mesma escala é representado por (0, 0, 0). Dessa forma, é possível transformá-la, por exemplo, em uma matriz bidimensional. Esse procedimento é possível, pois acontece uma leitura na imagem, na qual se percorre pixel a pixel. O trajeto começa no primeiro pixel do canto superior esquerdo e segue em linha reta, da esquerda para a direita, até o final da primeira linha. Finalizada a leitura da primeira linha, a leitura recomeça no início da segunda linha, e o processo segue até chegar ao final da imagem. O objetivo dessa leitura é transformar cada pixel da imagem em um par [x,y] na matriz. Após a definição da matriz bidimensional, é necessário transformá-la em um vetor binário, pois é este vetor que será utilizado como entrada na Rede Neural, para o processo de reconhecimento de faces (Souza and Araújo, 2021).

3.1 Rede Neural Artificial (RNA).

Uma RNA é um algoritmo de Inteligência Artificial utilizada na solução de problemas complexos e com o propósito de identificar padrões para tomadas de decisão de forma mais acertada possível. São consideradas multicamadas, pois são formadas por pelo menos uma camada de entrada, uma camada oculta e uma camada de saída. Neste trabalho, foi utilizado o algoritmo *backpropagation,* onde na camada de entrada são inseridos os dados, a camada oculta é responsável pelo processamento e a camada de saída apresenta o resultado. Quando os dados processados não têm um resultado esperado, o algoritmo refaz o caminho inverso do processamento (Souza et. al, 2020).

4. PARALELIZAÇÃO DA REDE NEURAL ARTIFICIAL

Das etapas apresentadas na Figura 1, para este trabalho, destaca-se o quadro da Rede Neural, pois foi a parte do problema que foi paralelizado, utilizando-se a placa gráfica (GPU). As demais etapas continuaram sendo processadas de forma sequencial pela CPU, como pode ser observado na Figura 2.

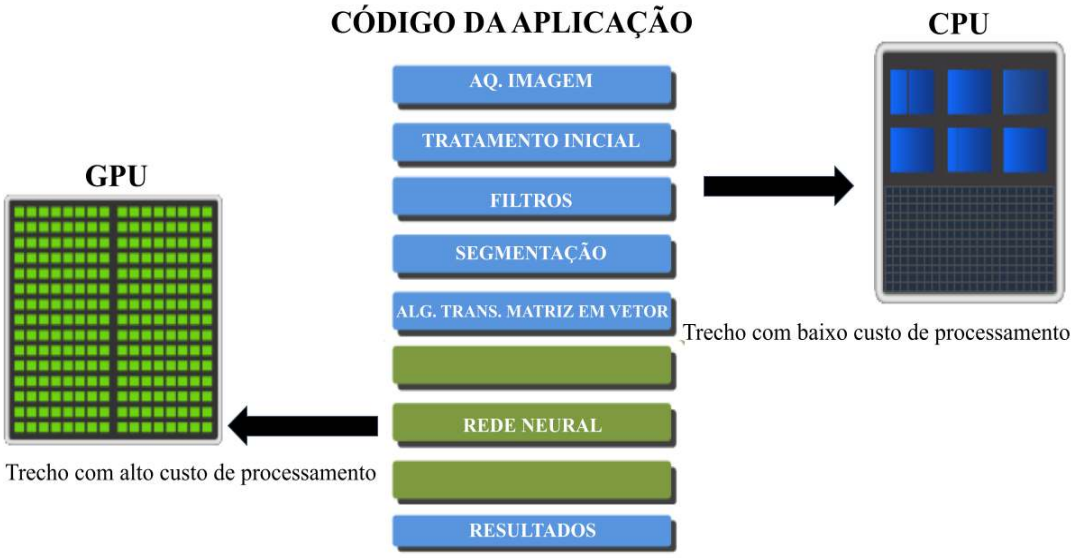

Figura 2. Etapas da execução paralela.
Fonte: Adaptada de (Cheng et al., 2014).

A escolha da forma de processamento deve levar em consideração a necessidade de o algoritmo manter a ordem de precedência e se os dados podem ser executados de maneira sequencial ou se as entradas de dados não possuem dependência de suas saídas (Cheng at. al, 2014).

A paralelização ocorre por meio de uma configuração em grades para uma, duas ou três dimensões, definidas de forma menos complexa, por meio de um conjunto de instruções próprias via API CUDA (Kirk and Hwu 2010). A grade é basicamente representada pelo número total de threads organizadas em blocos e, dependendo da dimensão do problema, essas threads são identificadas em seus blocos específicos em um escopo de execução global (Cheng et al., 2014). Todas as threads são inicializadas, teoricamente, ao mesmo tempo, mas na verdade, isso depende do dispositivo GPU utilizado, por conta de um conceito presente no CUDA chamado de *warp.*

Ele possui um ambiente que permite ao desenvolvedor a utilização do C/C++ como ferramenta, contudo, também suporta outras linguagens, dentre elas o Python, principal linguagem utilizada na criação do software da aplicação. Normalmente, os dados são armazenados unidimensionalmente. Mesmo quando uma visão lógica multidimensional de dados é usada, ele ainda mapeia para armazenamento físico unidimensional. Determinar como distribuir dados entre threads está intimamente relacionado há como esses dados são armazenados fisicamente, bem como a forma como a execução de cada thread é ordenado. A maneira como se organiza as threads tem um efeito significativo no desempenho do programa.

4.1. Metodologia Aplicada no Trabalho

O trabalho de (Souza and Araújo, 2021) apresenta uma proposta de aplicação de uma Rede Neural Artificial (RNA) utilizando um código sequencial para reconhecimento facial. O objetivo inicial foi identificar em qual etapa do processo de reconhecimento se consumia mais tempo de processamento, e se verificou que era na etapa de treinamento e teste da RNA. Dessa forma, optou-se pelo modelo de programação paralela utilizando placas gráficas, que entre as mais utilizadas estão as GPUs da NVIDIA[1], que são muito utilizadas na solução de problemas complexos e de caráter geral.

Para este trabalho foi utilizada uma imagem presente na base (BioID, 2019). Essa imagem passou pelas etapas descritas na Figura 1. Para validar a proposta as imagens foram divididas em dois conjuntos, o de treinamento e o de testes, com proporções de 75% e 25%, respectivamente. Dos 75% do conjunto de treinamento, 25% foram utilizadas como conjunto de validação.

O código sequencial da Rede Neural foi modificado de forma a ser executado em uma placa gráfica (GPU), utilizando as bibliotecas paralelas *TensorFlow* e *Pytorch*, ambas escritas em Python, o que acabou contribuindo para este trabalho, já que a aplicação utilizada foi totalmente desenvolvida nesta linguagem. Os testes foram executados utilizando os filtros destacados na Figura 1.

4.1.1. TensorFlow.

O *TensorFlow*[2] possui um conjunto de ferramentas, bibliotecas e recursos da desenvolvidos pela comunidade, isso permite a utilização de *Machine Learning (ML)* de última geração e aos desenvolvedores criar e implantar aplicativos com os recursos dele oferecidos. Pode ser executado sobre diversas plataformas e arquiteturas, incluindo CPUs e GPUs. Atualmente, é um dos principais frameworks do mercado para criação de *Deep Learning*. Com ele, é possível agilizar e facilitar o processo de obtenção de dados, treinar modelos, realizar predições e refinar resultados futuros. O *TensorFlow* possui uma arquitetura dividida em 3 partes:

- Processamento dos dados;

[1] https://developer.nvidia.com/cuda-zone
[2] https://www.tensorflow.org/federated

- Construção de modelos;
- Treinamento e estimativa dos modelos criados;

Os dados são definidos em forma de tensores, ou seja, de matrizes multidimensionais, onde esses dados são processados de acordo com a definição de um grafo, criado para definir a função da execução. O *TensorFlow* (Geron, 2019) pode dividir o grafo em paralelo, tanto em várias CPUs como em GPUs. Isso proporciona ao desenvolvedor a trabalhar apenas na lógica do problema, pois permite utilizar funções prontas apenas necessitando ajustar os *datasets* para definir as suas saídas.

4.1.2. Pytorch.

O *Pytorch*[3] (Howard and Gugger, 2020) é uma biblioteca expressiva para trabalhos envolvendo aprendizado de máquina *(Machine Learning),* pois proporciona um conjunto de funções destinadas ao tratamento de equações, principalmente da álgebra linear, fornecendo velocidade e simplicidade para execução dos algoritmos. Sua utilização é baseada nos tensores, que são extensão lógica de matrizes multidimensionais capazes de organizar e representar os dados a serem treinados (Pointer, 2019).

A biblioteca é uma estrutura de aprendizado de máquina de código aberto pertencente ao *Pytorch* com recursos estáveis, capaz de manter total desempenho, compatibilidade entre versões a longo prazo e um vasto conjunto de dados, arquiteturas e modelos para o trabalho com visão computacional.

4.2. Configuração do Sistema Utilizado:

Os testes foram executados em um computador com processador Intel Core i7 3770, 8 cores, com 8 GB de RAM, utilizando sistema operacional Linux Fedora 36, linux kernel 5.18.13-200, (distribuição i3 Spin).

A placa gráfica é uma Nvidia GeForce GTX 1660, com memória global de 6 GB, versão CUDA 11.6 e com 1.280 CUDA Cores. Para os testes em GPU foram utilizadas as bibliotecas *Tensorflow 2.6.0* (essa API utiliza o Keras para o treinamento de RNA) (LLC, 2022), e *tensorflow-io-gcs-filesystem 0.24.02.6.1* para o *TensorFlow* e *Torch 1.12.0*, *Torchvision 0.13.0* para os testes com o *Pytorch*, assim como algumas bibliotecas do Python como a *opencv-python 4.5.5.64*, para projetos utilizando visão computacional, *numpy 1.22.3*, utilizado no processamento dos dados em arrays, *keras 2.8.0*, *Keras-Preprocessing 1.1.2* usados no *Tensorflow*, permitindo os testes com a rede neural.

5. RESULTADOS

A primeira análise feita, após a execução dos algoritmos propostos no trabalho, foi a média dos tempos na função treinamento e teste para os filtros definidos na Figura 1 (Gaussiano, Bilateral, Mediano e Equalização de Histograma) e utilizados na etapa de pré-processamento. Os resultados foram obtidos através da média de 10 execuções de cada filtro. O objetivo é avaliar o comportamento da Rede Neural Artificial (RNA) com cada um desses filtros e com cada uma das bibliotecas propostas nas fases de treinamento e teste.

Observa-se na Figura 3, um tempo maior no treinamento sequencial utilizando o filtro Bilateral. O treinamento sequencial mais eficiente foi atingido pelo filtro Mediano, seguido dos filtros Equalizador e Gaussiano. Já quando a comparação é realizada entre as bibliotecas paralelas, observa-se pouca variação nos tempos de execução dos filtros com o *TensorFlow,* porém, a biblioteca apresenta ganhos expressivos quando comparada a execução sequencial. Porém, quando o treinamento é realizado com a biblioteca *Pytorch*, o

[3] https://pytorch.org/

tempo de execução, além de ser menos da metade do tempo sequencial, como pode ser observado para os filtros Gaussiano, Bilateral e Equalizador de Histograma, também são melhores do que os apresentados pelo *TensorFlow*.

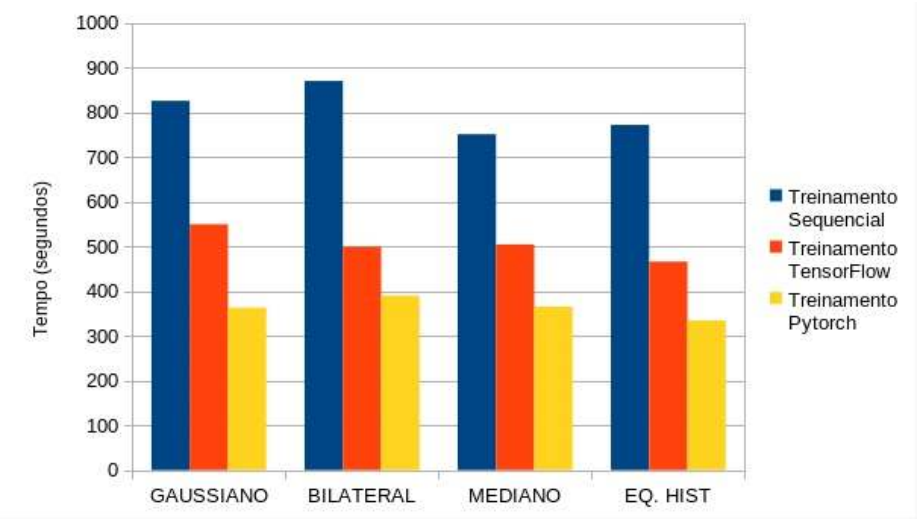

Figura 3. Média do Tempo de Execução na Fase de Treinamento.

Na fase de teste, apresentada na Figura 4, percebe-se que o algoritmo sequencial resultou na melhor eficiência quando comparado as bibliotecas paralelas, e obteve o seu melhor desempenho com o filtro Bilateral, seguido do Mediano, Equalizador e Gaussiano. Já quando a comparação é entre as bibliotecas paralelas, o *TensorFlow* apresentou os piores resultados quando comparado com o *Pytorch*. Esse fato ocorre devido o *Pytorch* organizar os dados em tensores otimizados multidimensionais. Soma-se a isso, o paralelismo executado pelo *Pytorch* é distribuído automaticamente, tornando o processo mais rápido. O *TensorFlow* possui o paralelismo atribuído manualmente, o que faz com que o processo de distribuição dos dados seja um pouco mais demorado.

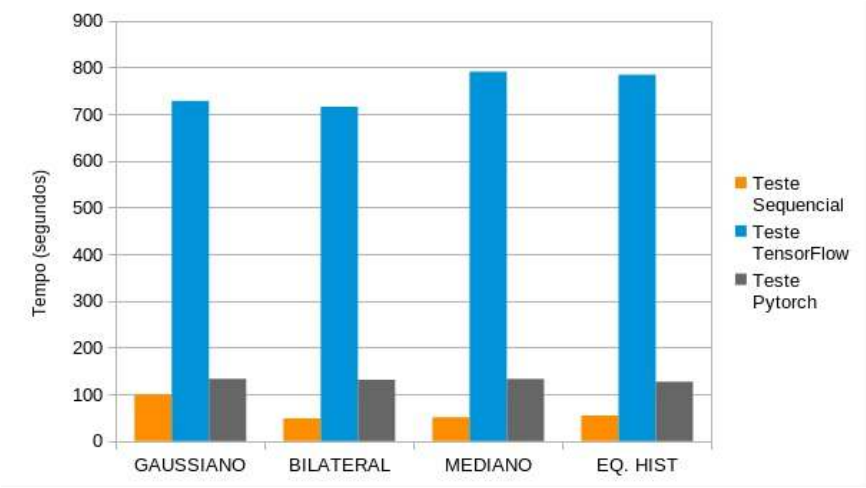

Figura 4. Média do Tempo de Execução na Fase de Teste.

Pode-se observar nas Figuras 3 e 4, que os códigos sequenciais obtiveram os maiores tempos médios de treinamento, em comparação aos outros dois códigos que utilizam as funções paralelas (*TensorFlow* e *Pytorch*) contudo, alcançaram os menores tempos de teste.

Isso por ser explicado por dois motivos: O primeiro motivo é devido ao custo de processamento da herança da classe de dados com outras classes que manipulam os cálculos nas bibliotecas do *TensorFlow* e *Pytorch*. O segundo motivo seria por conta do *overlay* da realocação de memória na placa gráfica, nesse caso, a GPU precisa verificar os espaços livres para fazer a alocação, visto que toda essa informação é lida a partir da memória e não no disco do computador como é feito no treinamento.

A média do tempo de execução total dos algoritmos por filtro, é apresentada na Figura 5. Percebe-se que os tempos obtidos pelo *Pytorch* foram os mais eficientes, seguidos dos *TensorFlow* e sequencial. Os tempos de execução dos algoritmos sequenciais acabaram sendo superiores aos tempos obtidos com as bibliotecas que utilizam GPU.

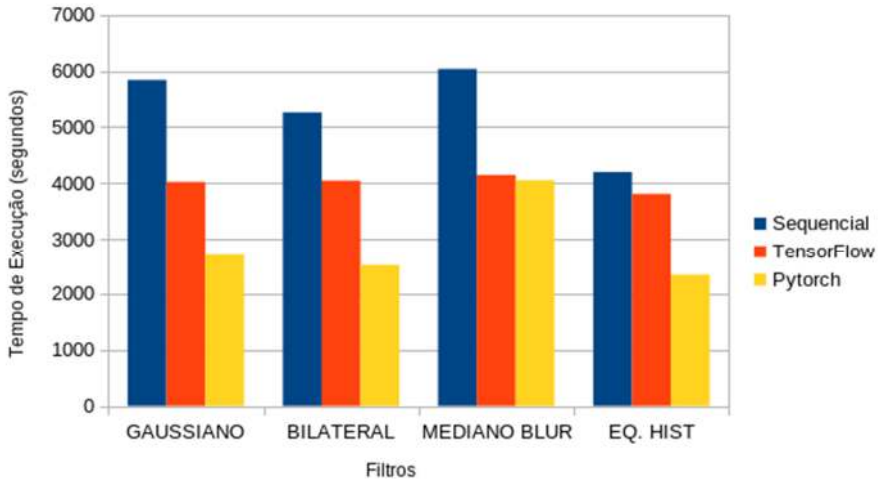

Figura 5. Média do Tempo de Execução Total por Filtro.

A segunda comparação foi a partir da média geral dos tempos obtidos a partir da execução dos algoritmos somando o tempo de execução dos quatros filtros utilizados (Gaussiano, Bilateral, Mediano e Equalizador), ou seja, foram feitas 10 execuções de cada algoritmos (sequencial, *TensorFlow* e *Pytorch*) utilizando todos os filtros e, em seguida, esses tempos foram somados e foi gerada a média dessas execuções Também, foram utilizadas seis diferentes testes (ou seja, imagens de diferentes pessoas) e a separação dos dados por filtros utilizados.

Como pode ser observado na Figura 6, o *Pytorch* teve a melhor média de tempo, depois o *TensorFlow* e, por último, o sequencial. Os resultados dos algoritmos mostram que o tempo de execução sequencial é de mais da metade do tempo de execução do *TensorFlow* e três vezes mais que o tempo de execução do *Pytorch.*

O *Pytorch* obteve o melhor resultado, pois como citado anteriormente, essa biblioteca possui o seu paralelismo distribuído automaticamente através do uso de tensores multidimensionais otimizados, e isso faz com que os dados sejam organizados em tensores antes de enviar para o processamento, isso facilita a manipulação para execução dos *datasets*, em comparação a organização dos dados do *TensorFlow* que foi realizada através da biblioteca *numpy*.

6. CONCLUSÃO

O objetivo desta pesquisa foi apresentar uma avaliação do uso das bibliotecas paralelas *TensorFlow* e *Pytorch*, quando aplicadas à execução de uma Rede Neural Artificial, em

problemas de reconhecimento de padrões, mais especificamente no reconhecimento facial, pois se percebe que, dependendo do tamanho da base utilizada, as fases de treinamento e teste, podem levar um longo tempo de processamento. Com o uso da programação paralela, pode-se reduzir o tempo de execução dessas aplicações, fazendo com que se tenha resultados em intervalos de tempo cada vez menores, principalmente, quando se utiliza uma grande quantidade de técnicas ou filtros para se melhorar o tratamento das imagens realizadas.

Neste trabalho, foram testados quatro diferentes filtros, que produzem diferentes resultados no tratamento das imagens (Gaussiano, Bilateral, Mediano e Equalizador), sendo que o Equalizador de Histograma obteve o melhor resultado geral no tempo de execução, somando-se treinamento e teste, utilizando a biblioteca *Pytorch*.

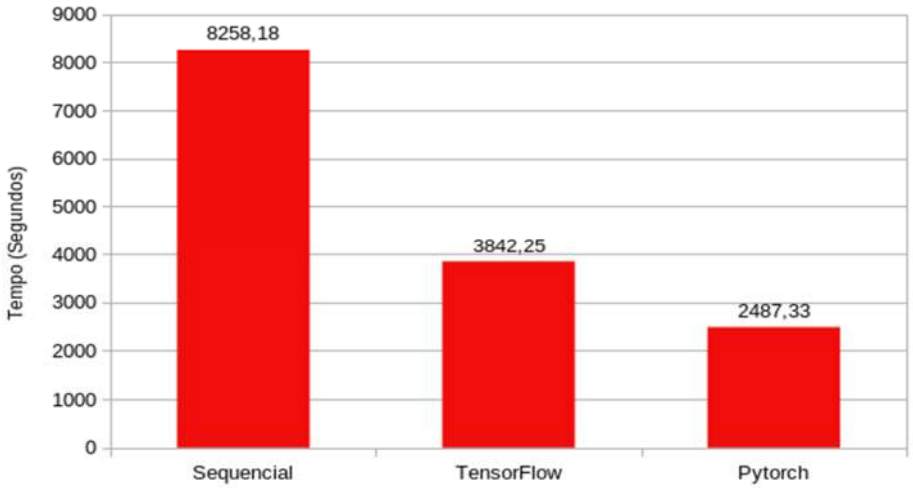

Figura 6. Média do Tempo Geral de Execução Somados os Quatro Filtros.

REFERÊNCIAS BIBLIOGRÁFICAS

ARAÚJO, S. S.; da Ponte, F. R. P.; Oliveira, V.; da Silva, W. S.; Vieira, D.; de Castro, M. F. and Rodrigues, E. B. (2022). Inteligência Artificial e Função como Serviço: Provisionando Aplicações com o AWS Lambda. Minicurso do XXIII Simpósio em Sistemas Computacionais de Alto Desempenho. p. 36-64.

ARCINIEGAS T., D. F., Amaya, M., Carvajal, A. P., Rodriguez-Marin, P. A., Duque-Muñoz, L., and Martinez-Vargas, J. D. (2022). Students' Attention Monitoring System in Learning Environments Based on Artificial Intelligence. *IEEE Latin America Transactions,* Vol. 20, Issue 1, p. 126–132.

BioID FACE DATABASE (2019). Disponível em: https://www.bioid.com/facedb/. Acessado em junho de 2019.

CHENG, J., Grossman, M., and McKercher, T. (2014). *Professional CUDA C Programming.* John Wiley & Sons, Inc. Corporation, N. (2022). *Cuda C Programming Guide*.

DE SOUZA, F., Formigoni Filho, J. R.; Marino, F. C. H.; Sampaio, A. S. (2022) Autenticação segura de pessoas com carteira digital: um estudo no CPQD. Simpósio Brasileiro de Fatores Humanos em Sistemas Computacionais (IHC). Diamantina. p. 48-55. DOI: https://doi.org/10.5753/ihc_estendido.2022.225470.

GÉRON, A. (2019). *Mãos à Obra: Aprendizado de Máquina com Scikit-Learn TensorFlow*, volume 1ª Edição. Editora Alta Books.

HECK, L. P., Künas, C.; and Padoin, E. L. (2020). Análise de desempenho e efetividade de redes neurais convolucionais em plataformas de GPU e CPU aplicadas ao reconhecimento de emoções através de expressões faciais em seres humanos. *XXI Simpósio em Sistemas Computacionais de Alto Desempenho* (WSCAD), p. 8-13. DOI: https://doi.org/10.5753/wscad_estendido.2020.14083.

HOWARD, J. and Gugger, S. (2020). *Deep Learning for Coders with fastai and Pytorch*, volume 1ª Edição. O'Reilly Media, Inc.

KIRK, D. B. and Hwu, W. (2010). *Progamming Massively Parallel Processors.* Second Edition. Elsevier.

KOLB, A., Steiner, H., Sporrer, S., and Jung, N. (2016). *Design of an active multispectral swir camera system for skin detection and face verification.* Journal of Sensors, pages 1–1.

LLC, G. (2022). Tensoflow. Disponível em: www.tensorflow.org.

PLICHOSKI, G. F., Metzger, G., and Chidambaram, C. (2018). A supervised face recognition in still images using interest points. *IX Computer on the Beach*, p. 721–730.

POINTER, I. (2019). *Programming Pytorch for Deep Learning,* volume 1ª Edição. O'Reilly Media, Inc.

SOUZA, F. R. A. and Araujo, J. S. (2021). Análise e Avaliação de Técnicas de Realce de Imagens Digitais Aplicadas ao Reconhecimento Facial com Redes Neurais. *18th International Conference On Information Systems And Technology Management* (CONTECSI). DOI: 10.5748/18CONTECSI/PSE/SEC/6715.

SOUZA, F. R. A., Araújo, F. P. O., Araujo, J. S. (2020). Uma Proposta para o Reconhecimento de Imagens Faciais: Um Estudo de Caso Aplicado à Redes Neurais Artificiais. 17th *International Conference On Information Systems And Technology Management* (CONTECSI), *pages* 2422–2440. DOI: 10.5748/17CONTECSI/PSE-6473.