Redes Neurais: Detecção de Câncer Cerebral

Gabriel Candelária Wiltgen Barbosa
gabrielcandelaria@alunos.utfpr.edu.br
Universidade Técnológica Federal - Paraná (UTFPR)
Apucarana, Paraná, Brasil

Camila Costa Durante
camiladurante@alunos.utfpr.edu.br
Universidade Técnológica Federal - Paraná (UTFPR)
Apucarana, Paraná, Brasil

***Resumo*—This case study investigates the application of neural network models, Multilayer Perceptrons (MLPs), Deep Neural Networks (DNNs) and Convolutional Neural Networks (CNNs), to detect and classify three types of brain cancer using tomography images. The primary objective is to evaluate and compare the efficiency of these models in cancer detection, ensuring that their hyperparameters are kept as consistent as possible to enable a fair comparison. The study aims to identify the most effective model for this specific medical imaging task and demonstrate the potential of these models.**

***Index Terms*—Brain cancer, Image, Multilayer Perceptrons, Deep Neural Networks, Convolutional Neural Networks**

I. Introdução

Na atualidade, a busca por soluções utilizando algoritmos de ”Inteligência Artificial”tem se intensificado bastante, desde o surgimento de IAs voltadas à acessibilidade e automação, como os *Large Language Models* (LLMs), tais como ChatGPT, Gemini, DeepSeek, entre outros, até o aumento e melhoria nas responsáveis por sistemas de recomendações, diagnósticos médicos e análises de imagens. A aplicação de IA na área da saúde, em particular, tem ganhado destaque, especialmente em tarefas como a detecção de doenças a partir de imagens médicas [1]. Esse avanço levou ao objetivo deste projeto: avaliar diferentes abordagens (algoritmos) de redes neurais para a detecção de câncer cerebral a partir de imagens de tomografia.

O foco deste trabalho é a aplicação de modelos de redes neurais, como *Multilayer Perceptrons* (MLPs), *Deep Neural Networks* (DNNs) e *Convolutional Neural Networks* (CNNs), para a análise de imagens de tomografia, visando a detecção e classificação de três tipos de câncer cerebral. A escolha desses algoritmos se deve à sua arquitetura e implementação, que embora não seja de nível tão elevado, é capaz de proporcionar resultados gratificantes, demonstrando o quão perto a humanidade está de automatizar também o setor médico.

O ambiente de treinamento para os algoritmos foi o *DataSet* ”Brain Tumor (MRI Scans)”[2], encontrado no Kaggle, o qual apresenta imagens de tomografias com cérebros afligidos pelo câncer Glioma (1621 imagens), Meningioma (1645 imagens) e Pituitário (1757 imagens), além de imagens de cérebros saudáveis (2000 imagens).

Tumores do tipo glioma, conforme descrito no artigo ”Glioma: an overview of current classifications, characteristics, molecular biology and target therapies”[3], originam-se nas células gliais e representam os tumores primários mais comuns do sistema nervoso central, desenvolvendo-se, na maioria dos casos, no cérebro. Segundo o artigo ”An overview of meningiomas”[4], os meningiomas são tumores cerebrais que surgem nas meninges — membranas que revestem e protegem o cérebro e a medula espinhal. Geralmente benignos, são frequentemente descobertos de forma incidental em exames de imagem. Já os tumores pituitários, como discutido no artigo ”Pituitary Adenomas: An Overview”[5], desenvolvem-se na glândula pituitária (hipófise). Embora sejam predominantemente benignos, podem interferir na produção hormonal, afetando diversas funções do organismo.

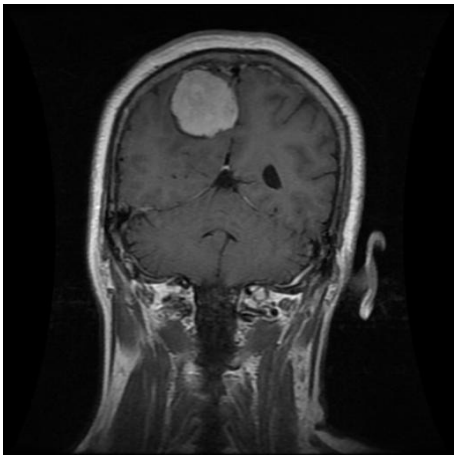

Figura 1: Exemplo de cérebro com Meningioma [2]

Na totalidade, foram implementados quatro agentes e utilizados mais três agentes pré-treinados, sendo que, dos implementados, apenas um dos MLP (*Multilayer Perceptron*) foi feito sem a utilização de bibliotecas especializadas, um agente desenvolvido a fim de manter sua simplicidade. Os demais implementados foram feitos utilizando a biblioteca *Keras* [6], sendo eles, outro MLP, um modelo DNN (*Deep Neural Network*) e um modelo CNN (*Convolutional Neural Network*). Ademais, os agentes pré-treinados utilizados foram o VGG-16 [7], o ResNet-50 [8] e o InceptionV3 [9].

Modelos mais simples e que requerem menos poder computacional não se encaixam muito no escopo de pesquisas relacionadas a avanços na Inteligência Artificial, contudo, são excelentes para retratar o potencial delas e mostrar que, para grande parte dos problemas, não é necessário o uso de um

agente de alto nível para atingir resultados gratificantes. Por este fator, os modelos em questão estão sendo estudados, a fim de demonstrar seu potencial não só no âmbito médico, como nesse caso, mas também em um contexto geral.

II. Trabalhos Relacionados

Um exemplo de aplicação de redes neurais na detecção de doenças a partir de imagens médicas é o artigo "A utilização de redes neurais convolucionais para a identificação de doenças respiratórias em radiografias de tórax"[10], no qual os autores implementam um modelo de rede neural convolucional (CNN) para a detecção de pneumonia. O modelo demonstrou alta eficácia, alcançando uma acurácia de 96,96% no conjunto de teste.

Em outro trabalho de referência, "Detecção de câncer de mama por meio de imagens infravermelhas utilizando Redes Neurais Convolucionais"[11], o autor redimensiona imagens termográficas para a detecção de câncer de mama e avalia qual modelo pré-treinado (AlexNet, GoogleNet, ResNet-18, VGG-16 e VGG-19) apresenta o melhor desempenho. Os melhores resultados foram observados com os modelos VGG-16 e VGG-19, ambos treinados por 20 épocas, alcançando uma acurácia de teste de 77,5%.

Os demais trabalhos de referência utilizam redes neurais convolucionais para a detecção de câncer em imagens de ressonância magnética (IRM) do cérebro, empregando três abordagens distintas.

No artigo "Detecção de tumor cerebral a partir de análise de imagens médicas usando inteligência artificial"[12], o autor realiza o pré-processamento das imagens, criando uma máscara que determina a localização do tumor. As máscaras, combinadas com as imagens, são utilizadas para alimentar o modelo implementado, permitindo que ele aprenda a gerar máscaras para detectar o tumor cerebral nas IRMs de teste. O modelo apresentou uma acurácia variando entre 94% e 98%, dependendo da imagem de teste.

No trabalho intitulado "Reconhecimento de Tumores Cerebrais Utilizando Redes Neurais Convolucionais"[13], foi implementada uma CNN alimentada com imagens divididas em 5 conjuntos para a realização de validação cruzada. Desses conjuntos, 4 foram utilizados para treinamento e 1 para teste, alternando o uso a cada execução. O modelo obteve uma cobertura de 87% para as imagens com tumores e 66% para as imagens saudáveis.

Por fim, o artigo "O USO DE APRENDIZADO PROFUNDO NA CLASSIFICAÇÃO DE RESSONÂNCIAS MAGNÉTICAS PARA DETECÇÃO DE TUMOR CEREBRAL"[14] realiza o pré-processamento das imagens, incluindo realce de contraste, normalização, aplicação de cortes e redimensionamento. As imagens pré-processadas foram utilizadas em modelos pré-treinados de arquitetura transformers (MViT) e CNNs (EfficientNet, EdgeNeXT e ConvNeXT) para avaliar a melhor performance. A avaliação dos modelos resultou em uma acurácia Top-1 de 99,11% para o modelo MViT. A acurácia Top-1 refere-se à proporção de vezes em que o modelo classificou corretamente a imagem com a categoria mais provável entre todas as classes possíveis, ou seja, a porcentagem de vezes em que o modelo acertou a primeira escolha.

III. Metodologia de Desenvolvimento

Sessão dedicada à descrição dos procedimentos metodológicos com os quais foi desenvolvido este trabalho. As sub-seções seguintes detalham segmentos específicos que compõem o trabalho na totalidade, contendo informações importantes para seu desenvolvimento.

A. Tratamento do DataSet

As imagens do *DataSet* estavam organizadas em pastas separadas de acordo com suas respectivas classes: glioma, *healthy* (saudável), meningioma e *pituitary* (pituitário). Foram selecionadas 1.400 imagens de cada classe para treino, enquanto o restante foi reservado para teste.

Para evitar viés, as imagens de treino e teste foram embaralhadas de maneira mapeada, armazenando o caminho da imagem juntamente com sua *label* (rótulo), correspondente à classe: 0 para glioma, 1 para saudável, 2 para meningioma e 3 para pituitário.

Após o embaralhamento, as imagens foram renomeadas de forma padronizada como 'train_image_i' e 'test_image_i', onde i representa a ordem das imagens. Elas foram então armazenadas em pastas denominadas 'X_train' e 'X_test', correspondentes aos conjuntos de treino e teste, respectivamente.

As *labels* foram salvas em arquivos .txt, associando o nome das imagens ao número referente à classe. Esses arquivos foram armazenados em pastas nomeadas 'y_train' para o conjunto de treino e 'y_test' para o de teste.

Ademais, foi aplicado o processo de *one-hot-encoding* para as *labels*, utilizando o comando *to_categorical()*, o qual organiza as classificações em vetores, binarizando seus valores.

B. Pré-processamento

As imagens do DataSet passaram por alterações antes de serem alimentadas as redes neurais de cada algoritmo, com exceção da MLP implementada sem uso da biblioteca *Keras*. Essas alterações também serviram como forma de estudo para verificar possíveis melhoras ou pioras durante o processo de treinamento de cada agente. O pré-processamento era composto por sete diferentes tipos de filtragem, afim de, avaliar o impacto que cada um traria aos agentes, sendo eles, *Grayscale* (i), *RGB* (ii), *Negative* (iii), *HSV* (iv), *Binary* (v), *Blur* (vi) e *Canny* (vii).

(i) *Grayscale* - Faz com que a imagem fique apenas em escalas da cor cinza, reduzindo o número de canais de cor para apenas um.
(ii) *RGB* - (*Red*, *Green*, *Blue*) Filtro padrão utilizado pela maioria das imagens.
(iii) *Negative* - Consiste na inversão dos valores contidos nos canais de cor da imagem em questão.
(iv) *HSV* - (*Hue*, *Saturation*, *Value*) Filtro baseado também em três canais, porém, vinculado a matriz, saturação e valor da imagem em questão.

(v) *Binary* - Converte a imagem para preto e branco, reduzindo o número de canais de cor para um.
(vi) *Blur* - Aplica um desfoque para suavizar detalhes e reduzir ruído.
(vii) *Canny* - Realiza um processo de separação de bordas ou contornos, reduzindo também o número de canais de cor para um.

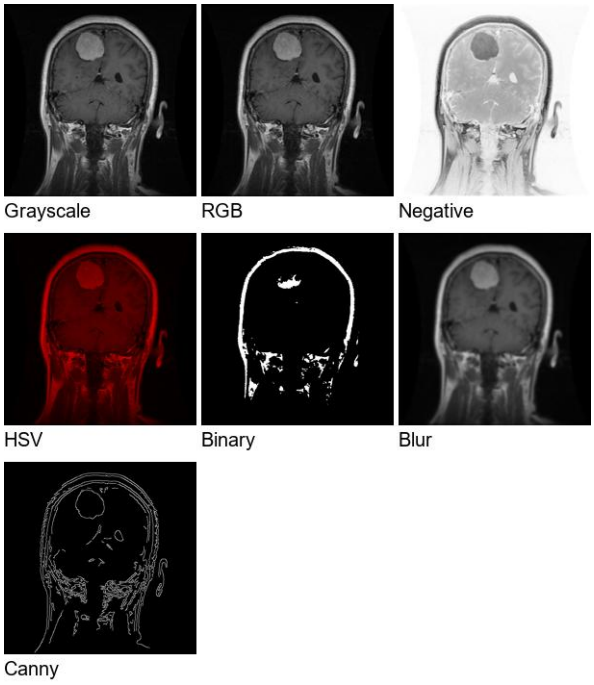

Figura 2: Exemplo de aplicação de cada um dos filtros

Ademais, durante o pré-processamento foi necessário aplicar a reescala de resolução das imagens, as quais foram ajustadas para um tamanho de 60x60 por questões de consumo de memória da GPU que estava sendo utilizada para treinar os agentes. Essa reescala foi realizada para todo modelo, com exceção do MLP implementado sem o uso do *Keras* e do agente do modelo pré-treinado InceptionV3, o qual necessitava de uma entrada de no mínimo 75x75 pixels.

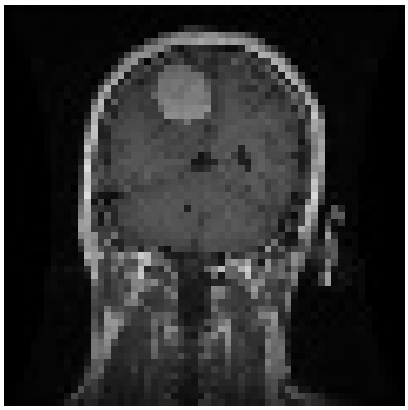

Figura 3: Imagem RGB pré-processada de 60x60 pixels (Ampliada para melhor visualização)

C. Hiperparâmetros

Afim de estudo e comparação, os modelos foram treinados visando manter o maior nível de similaridade possível em relação aos parâmetros e hiperparâmetros estabelecidos. Devido a este fator, o número de épocas máxima para todos os modelos foi de 100 épocas, o *batch size* foi definido para 128, o otimizador utilizado foi o "Adam"[15], a função para calculo de *loss* (perda) utilizada foi a *categorical_crossentropy*, a separação do conjunto de treino para validação foi de 20% e, foi aplicado um *early stop* (parada antecipada) de *patience* (paciência) para até 10 épocas com restauração de pesos ativa.

D. Algoritmos MLP

Foram implementados dois modelos para o algoritmo de *Multilayer Perceptrons* (MLP). O primeiro se trata de um código feito do princípio e, sem uso de bibliotecas especializadas, contendo a implementação completa dos procedimentos *forward*, no qual o algoritmo passa os dados de entrada através das camadas a fim de gerar uma saída, utilizando os valores dos pesos, *bias* e, funções de ativação. Através da saída gerada, é possível obter o erro, o qual é utilizado no processo de *backward* através da sua retropropagação até a camada de entrada, calculando os gradientes entre camadas durante o processo e, utilizando-os para realizar os ajustes de peso através do otimizador utilizado, que no caso foi o *Stochastic Gradient Descent* (SGD). Ademais, esse algoritmo foi gerado com uma camada de entrada, apenas uma camada densa oculta (128 neurônios) e uma camada de saída com ativação sigmoidal.

O algoritmo seguinte, foi implementado utilizando a biblioteca do *Keras* e, apresentando uma arquitetura contendo uma camada de entrada, duas camadas densas (128, 64 neurônios respectivamente) com ativação *relu* e uma camada de saída com ativação sigmoidal.

E. Algoritmo DNN

Esse algoritmo de redes neurais profundas (DNN), utiliza apenas de camadas densas e de *dropout* (*Dropout()*), sendo que, as camadas de *dropout* tem a função de desligar neurônios para forçar a readaptação do modelo ainda durante seu treinamento, com o intuito de atrasar ou impedir o *overfiting* (cenário onde o modelo passa a decorar as respostas, perdendo potencial).

Flatten(input_shape = 60, 60, nChannels)
Dense(128, activation = "relu")
Dropout(0.2)
Dense(64, activation = "relu")
Dropout(0.2)
Dense(32, activation = "relu")
Dense(4, activation = "sigmoid")

Tabela I: Arquitetura da DNN

F. Algoritmo CNN

O algoritmo feito utilizando as redes neurais convolucionais (CNN) possui uma arquitetura um pouco mais complexa que as demais, possuindo camadas convolucionais (*Conv2D()*), de *pooling* (*MaxPooling2D()*) e de *flatten* (*Flatten()*).

As camadas de convolução (*Conv2D()*) são as responsáveis pela extração de características das imagens, algo que é feito através da aplicação de *kernels* sobre a imagem, que são pequenas matrizes que realizam a detecção das características através do seu deslizamento (convolução) sobre a imagem, realizando multiplicações de elementos e o produto escalar entre o filtro e a região da imagem que ele está cobrindo no momento, gerando por fim, um mapa de características.

Desta forma, a camada de *pooling* (*MaxPooling2D()*) entra em ação, reduzindo o tamanho espacial do mapa de características gerado pela camada de convolução, mantendo as características mais relevantes ao modelo.

Por fim, a camada de achatamento ou *flatten* (*Flatten()*), realiza a transformação dimensional do mapa de características em um vetor unidimensional, tornando possível seu processamento pelas camadas densas que seram responsáveis pela classificação.

Conv2D(64, kernel_size=(3, 3), activation='relu', input_shape = 60, 60, nChannels)
Conv2D(64, kernel_size=(3, 3), activation='relu')
MaxPooling2D(pool_size=(2, 2))
Conv2D(256, kernel_size=(3, 3), activation='relu')
MaxPooling2D(pool_size=(2, 2))
Conv2D(128, kernel_size=(3, 3), activation='relu')
MaxPooling2D(pool_size=(2, 2))
Dropout(0.5)
Flatten()
Dense(256, activation='relu')
Dense(256, activation='sigmoid')
Dropout(0.2)
Dense(4, activation='sigmoid')

Tabela II: Arquitetura da CNN

G. Modelos Pré-treinados

Os três modelos pré-treinados utilizados foram travados em suas arquiteturas, para destacar seu desempenho apenas como agentes pré-treinados na solução da detecção em questão. Contudo, a fim de minimizar o gasto com treinamento, mas possibilitar que o mesmo ocorresse, foi necessário aplicar uma camada de *global avarage pooling* (*GlobalAveragePooling2D()*) para reduzir as dimensões espaciais e, então, mais duas camadas densas (256 neurônios), para então, realizar a classificação em uma camada final.

H. Exigências do Ambiente de Desenvolvimento

Os algoritmos foram feitos utilizando o *Python* versão 3.8.10, biblioteca *Keras* versão 2.9.0, *matplotlib* versão 3.7.5, *Numpy* versão 1.24.4, *OpenCV* versão 4.10.0.84, *scikit-learn* versão 1.3.2 e, *tensorflow* versão 2.9.0.

Código-fonte do projeto disponibilizado via GitHub:

BrainCancerDetection

IV. Resultados e Discussões

Todos os algoritmos, excluído o MLP implementado sem a biblioteca *Keras*, foram treinados utilizando cada um dos filtros no pré-processamento para avaliar seu desempenho e trazer uma comparação mais precisa em relação aos melhores agentes obtidos, os quais estão presentes na tabela abaixo:

Modelo	**Filtro**	**Acurácia no Teste**
MLP	*Grayscale*	89.53%
MLP *Keras*	*HSV*	90.51%
DNN	*Grayscale*	90.93%
CNN	*HSV*	97.33%
VGG-16	*Negative*	93.68%
ResNet-50	*Grayscale*	87.63%
InceptionV3	*Negative*	90.51%

Tabela III: Melhores Agentes de cada modelo

A. Agentes MLP

A MLP implementada sem o uso do *Keras* teve um desempenho gratificante por ser a mais simples e com menor gasto computacional, e mesmo com estes fatores obteve um agente que atingiu 89.53% de acurácia no teste.

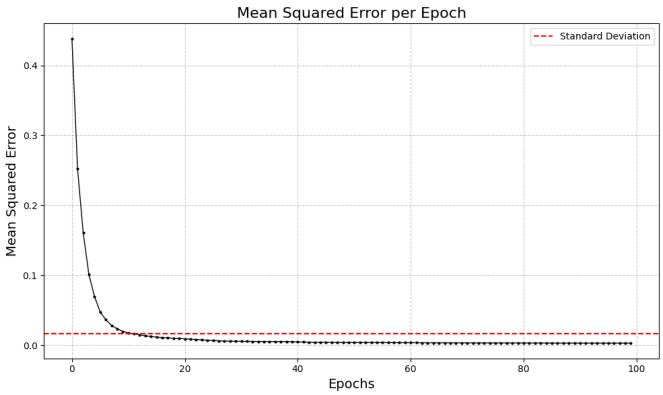

Figura 4: Gráfico do erro médio quadrático por época do MLP

Através do gráfico é possível observar que a convergência do agente ocorreu antes das 100 épocas, porém, devido a como foi implementado não havia um critério preciso para realização de uma parada antecipada, apenas um limite (*threshold*) relacionado ao erro quadrático médio configurado para 0.0001, o qual não foi atingido.

Já os agentes do MLP com *Keras* obtiveram uma acurácia máxima de 90.51%, algo que, é relativamente decepcionante, não por ser um resultado ruim, mas por não ser significativamente maior quando comparado ao MLP sem o *Keras*, que usa apenas uma camada densa. Um aumento de menos de 1% de acurácia no teste, neste caso, não justifica o aumento na complexidade do modelo.

Filtro	Perda - Validação	Acurácia - Validação	Acurácia - Teste
Grayscale	0.3698	88.21%	90.37%
RGB	0.3792	87.32%	89.95%
Negative	0.5971	76.69%	81.31%
HSV	0.3785	87.23%	90.51%
Binary	0.5869	80.35%	81.59%
Blur	0.4357	82.94%	87.35%
Canny	0.6372	78.21%	83.13%

Tabela IV: Agentes MLP *Keras* por Filtro

B. Agentes DNN

Assim como os agentes MLP *Keras* não foi possível notar um aumento considerável de desempenho, obtendo um valor máximo de acurácia no teste de 90.93% pelo melhor agente. Sendo assim, existe uma melhora, mas não é o suficiente para justificar o aumento na complexidade do modelo e o tempo investido para seu aperfeiçoamento e treino.

Filtro	Perda - Validação	Acurácia - Validação	Acurácia - Teste
Grayscale	0.3934	87.32%	90.93%
RGB	0.3784	89.37%	90.72%
Negative	0.5368	80.35%	83.77%
HSV	0.3877	87.67%	90.23%
Binary	0.6513	81.07%	81.59%
Blur	0.3517	87.58%	89.95%
Canny	0.7292	75.89%	80.39%

Tabela V: Agentes DNN por Filtro

C. Agentes CNN

O modelo da CNN foi o capaz de obter os resultados mais gratificantes do projeto, sendo capaz de obter uma acurácia máxima de 97.33% no teste pelo melhor agente treinado, algo que superou as expectativas. O modelo CNN em questão passou por inúmeros testes, assim como os demais agentes, para chegar a uma arquitetura aperfeiçoada que fosse capaz desse resultado, e com ele, o tempo gasto em seu aperfeiçoamento e treinamento são justificáveis.

Filtro	Perda - Validação	Acurácia - Validação	Acurácia - Teste
Grayscale	0.2068	94.37%	97.26%
RGB	0.1901	95.08%	97.19%
Negative	0.2618	92.58%	95.57%
HSV	0.1461	95.98%	97.33%
Binary	0.4125	89.19%	90.93%
Blur	0.2307	94.28%	95.08%
Canny	0.4807	87.23%	90.09%

Tabela VI: Agentes CNN por Filtro

D. Agentes Pré-Treinados

Os agentes pré-treinados não obtiveram resultados muito elevados, sendo o melhor deles o VGG-16 com um agente que obteve 93.68% de acurácia no teste, contudo, isso pode estar diretamente ligado ao fato desses agentes não serem treinados desde o zero no *DataSet* utilizado, algo que, caso ocorresse, proporcionaria resultados possivelmente melhores que os demais modelos implementados, inclusa a CNN de maior desempenho. Por outro lado, a complexidade do treino desses agentes foi bem reduzida, sendo apenas o mesmo que realizar o treino de uma MLP com duas camadas densas, sendo assim, o desempenho do VGG-16 pode ser considerado um ótimo desempenho para a complexidade que exige, vencendo em questão de acurácia da MLP *Keras* com uma arquitetura que levou mais tempo para ser aperfeiçoada e possui a mesma complexidade de treino (duas camadas densas).

Filtro	Perda - Validação	Acurácia - Validação	Acurácia - Teste
Grayscale	0.3477	87.76%	92.13%
RGB	0.3752	89.01%	93.25%
Negative	0.3529	88.39%	93.68%
HSV	0.3263	89.28%	91.99%
Binary	0.5467	81.42%	86.58%
Blur	0.3798	86.87%	88.97%
Canny	0.5272	82.41%	87.56%

Tabela VII: Agentes VGG-16 por Filtro

Filtro	Perda - Validação	Acurácia - Validação	Acurácia - Teste
Grayscale	0.3929	82.49%	87.63%
RGB	0.3875	82.67%	87.21%
Negative	0.4947	79.55%	84.19%
HSV	0.6626	72.41%	76.67%
Binary	0.5177	81.69%	85.80%
Blur	0.8421	64.01%	67.04%
Canny	0.6063	77.41%	82.02%

Tabela VIII: Agentes ResNet-50 por Filtro

Filtro	Perda - Validação	Acurácia - Validação	Acurácia - Teste
Grayscale	0.4797	88.03%	90.23%
RGB	0.4450	89.01%	90.02%
Negative	0.5080	87.05%	90.51%
HSV	0.4830	87.41%	88.97%
Binary	0.6759	82.58%	83.13%
Blur	0.4154	89.55%	90.09%
Canny	0.8395	79.73%	85.10%

Tabela IX: Agentes InceptionV3 por Filtro

E. Comparações Finais

Realizando uma análise mais detalhada em um contexto mais geral, a maioria dos agentes obteve uma porcentagem de acurácia menor ao se tratar do câncer glioma e meningioma, tendo grande parte de seu erro concentrado nessas duas categorias. Algo possível de se observar através da seguinte tabela comparativa com os melhores agentes de cada modelo:

Modelo	Acurácia			
	Saudável	Glioma	Meningioma	Pituitário
MLP	94%	88.23%	72.65%	94.39%
MLP *Keras*	93.16%	88.23%	79.18%	95.23%
DNN	94.5%	89.59%	78.36%	94.39%
CNN	98.33%	96.83%	93.87%	98.31%
VGG-16	98.83%	88.23%	88.57%	91.87%
ResNet-50	94.5%	78.73%	71.42%	92.71%
InceptionV3	96.33%	87.78%	81.63%	88.51%

Tabela X: Acurácia por categoria dos melhores agentes de cada modelo

Ademais, foram gerados gráficos comparativos para demonstrar a margem de épocas em que os melhores agentes chegaram ao *early stop*, ou seja, a sua convergência, sendo possível também, observar o desempenho deles através das épocas.

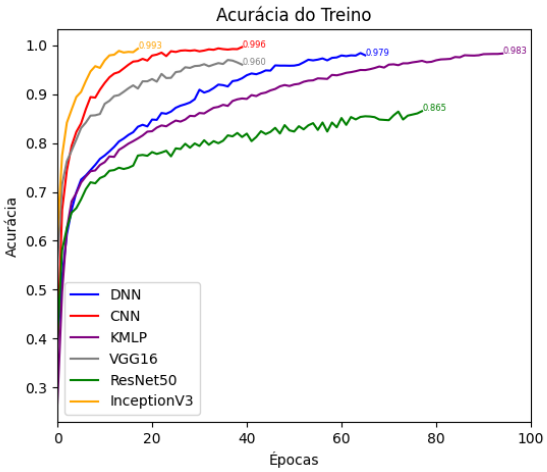

Figura 5: Acurácia do treino por época

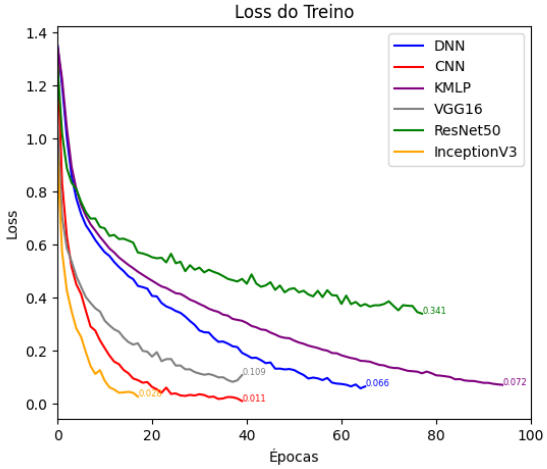

Figura 6: Perda do treino por época

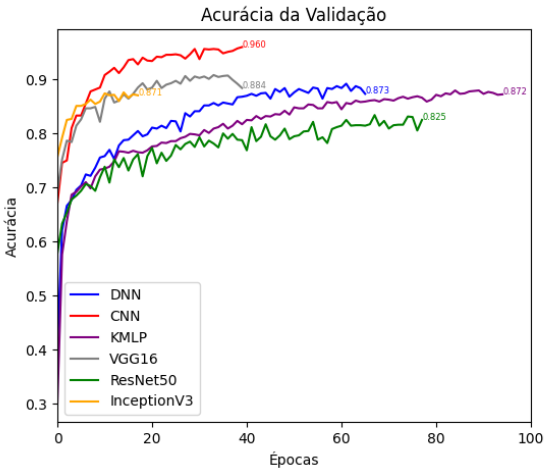

Figura 7: Acurácia da validação por época

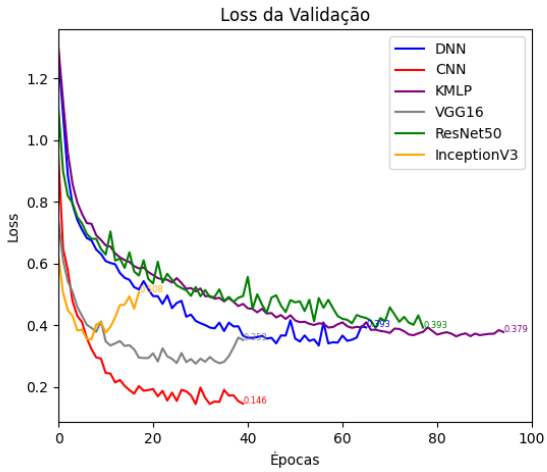

Figura 8: Perda da validação por época

Analisando os gráficos gerados, é possível observar a rápida convergência do agente CNN em relação aos demais, equiparando-se apenas ao pré-treinado VGG-16 e sendo um pouco mais lento que o pré-treinado InceptionV3, contudo, ambos os pré-treinados não obtiveram um resultado consideravelmente próximo ao do agente CNN, demonstrando sua eficácia mesmo comparado a arquiteturas mais robustas de agentes com pesos já treinados.

Demonstra-se notável também, a demora na convergência dos modelos sendo reciproca ao seu nível de complexidade, uma vez que, o MLP *Keras* é o modelo que leva mais tempo para chegar em sua convergência, sendo ele o menos complexo em arquitetura, seguido do agente DNN e posteriormente do agente CNN, marcando a melhora das arquiteturas em relação a convergência e, consequentemente, tempo de treino.

Por fim, ao que diz respeito aos agentes pré-treinados não é possível abstrair muita informação dos resultados obtidos, uma vez que seu treinamento foi superficial e apenas comparável ao nível de complexidade de um agente MLP de dupla camada densa, sendo assim, é possível dizer que os modelos pré-treinados obtiveram um resultado bom, excluído o modelo ResNet-50, que obteve o pior dos resultados dentre os agentes e, dando destaque ao VGG-16, o qual obteve uma rápida convergência e desempenho elevado quando comparado a MLP e DNN, que possuem complexidades semelhantes.

V. Conclusão

O trabalho tinha como objetivo comparar, avaliar e identificar o melhor modelo capaz de realizar a tarefa proposta, além de, demonstrar o potencial dos algoritmos utilizados, dentro e fora do ambiente de treinamento aplicado, algo que, tendo em vista os resultados obtidos, foi realizado com sucesso.

O destaque marcante do modelo CNN implementado com uma arquitetura trabalhada, utilizando a biblioteca do *Keras*, pode estar vinculado ao fato das redes CNN serem especificamente voltadas a tarefa de classificação de imagens, juntamente com os estudos empíricos e testes de arquitetura

que levaram a obtenção dos resultados demonstrados, contudo, vale ressaltar que as arquiteturas dos modelos pré-treinados, caso treinados do princípio no *DataSet*, muito provavelmente trariam resultados melhores, mas a um custo computacional significativamente mais elevado.

Resultados na casa dos 87.63% (menor desempenho obtido dos melhores agentes) são considerados suficientes para uma grande parte de problemas enfrentados, validando até o uso da MLP implementada de uma maneira simples em um contexto de aprendizado, demonstrando que até o uso de algoritmos não robustos podem trazer resultados favoráveis. Contudo, dependendo do contexto, um acerto de até 97.33% (maior desempenho obtido dos melhores agentes) ainda se demonstra baixo, como no caso em questão, abordando temas relacionados a diagnósticos médicos, os quais, requerem o mínimo de margem de erro o possível, uma vez que, 3% de erro pode custar uma vida.

Ainda assim, o desempenho dos modelos foi satisfatório, alcançando altos níveis de acurácia mesmo com recursos computacionais limitados, que foram o principal fator restritivo do estudo. Para aprofundar a pesquisa, seria necessário utilizar GPUs com maior capacidade de memória e processamento. Isso permitiria, por exemplo, a aplicação de arquiteturas de modelos pré-treinados nesse *DataSet*, além de possibilitar o aumento da resolução das imagens durante o pré-processamento e um ajuste mais equilibrado dos hiperparâmetros, mantendo a eficiência e a agilidade dos agentes durante o treinamento.

REFERÊNCIAS

[1] Jiang, Fei & Jiang, Yong & Zhi, Hui & Dong, Yi & Li, Hao & Ma, Sufeng & Wang, Yilong & Dong, Qiang & Shen, Haipeng & Wang, Yongjun. "Artificial intelligence in healthcare: past, present and future"(2017). BMJ. 2. svn-2017. 10.1136/svn-2017-000101.

[2] Rishabh Malhotra. "Brain Tumor MRI Scans". Kaggle, 2023. [Online]. Available: https://www.kaggle.com/datasets/rm1000/brain-tumor-mri-scans.

[3] Tao Zeng, Daming Cui, Liang Gao. "Glioma: an overview of current classifications, characteristics, molecular biology and target therapies". Front. Biosci. (Landmark Ed) 2015, 20(7), 1104–1115. https://doi.org/10.2741/4362.

[4] Buerki, R. A., Horbinski, C. M., Kruser, T., Horowitz, P. M., James, C. D., & Lukas, R. V. (2018). "An overview of meningiomas". Future oncology (London, England), 14(21), 2161–2177. https://doi.org/10.2217/fon-2018-0006.

[5] Lake, M. G., Krook, L. S., & Cruz, S. V. (2013, September 1). "Pituitary adenomas: An overview". American Family Physician, 88(5), 319–327. https://www.aafp.org/pubs/afp/issues/2013/0901/p319.html.

[6] François Chollet. *Keras*. GitHub, 2015. [Online]. Available: https://github.com/keras-team/keras.

[7] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition", *arXiv preprint arXiv:1409.1556*, 2014. [Online]. Available: https://arxiv.org/abs/1409.1556.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition", *arXiv preprint arXiv:1512.03385*, 2015. [Online]. Available: https://arxiv.org/abs/1512.03385.

[9] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision", *arXiv preprint arXiv:1512.00567*, 2015. [Online]. Available: https://arxiv.org/abs/1512.00567.

[10] S. Guilhermino, P. da Silva e S. Duarte, "A utilização de redes neurais convolucionais para a identificação de doenças respiratórias em radiografias de tórax", *Repositório UFERSA*, 2023. [Online]. Available: https://repositorio.ufersa.edu.br/items/2a1fe317-f5bf-4764-9970-10e5b778a974

[11] E. Chaves, "Detecção de câncer de mama por meio de imagens infravermelhas utilizando Redes Neurais Convolucionais", *Repositório UFU*, 2019. [Online]. Available: https://repositorio.ufu.br/handle/123456789/25938

[12] L. Azevedo, "Detecção de tumor cerebral a partir de análise de imagens médicas usando inteligência artificial", *Repositório UFU*, 2023. [Online]. Available: https://repositorio.ufu.br/handle/123456789/37641

[13] S. Neto, "Reconhecimento de Tumores Cerebrais Utilizando Redes Neurais Convolucionais", *Repositório Unipampa*, 2017. [Online]. Available: https://repositorio.unipampa.edu.br/handle/riu/1911

[14] F. Junior, "O USO DE APRENDIZADO PROFUNDO NA CLASSIFICAÇÃO DE RESSONÂNCIAS MAGNÉTICAS PARA DETECÇÃO DE TUMOR CEREBRAL", *Repositório UFC*, 2023. [Online]. Available: https://repositorio.ufc.br/handle/riufc/76650

[15] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization", *arXiv preprint arXiv:1412.6980*, 2014. [Online]. Available: https://arxiv.org/abs/1412.6980.