UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

ALANA CRISTINE NÓBREGA DE ALMEIDA

DETECÇÃO DE TUMORES CEREBRAIS EM IMAGENS DE RESSONÂNCIA MAGNÉTICA UTILIZANDO REDES NEURAIS CONVOLUCIONAIS

CAMPINA GRANDE
2024

ALANA CRISTINE NÓBREGA DE ALMEIDA

DETECÇÃO DE TUMORES CEREBRAIS EM IMAGENS DE RESSONÂNCIA MAGNÉTICA UTILIZANDO REDES NEURAIS CONVOLUCIONAIS

Trabalho de Conclusão de Curso submetido à Coordenação de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande, Campus Campina Grande, como parte dos requisitos necessários para obtenção do título de Graduado em Engenharia Elétrica.

Orientador: Gutemberg Gonçalves dos Santos Júnior, D.Sc.

CAMPINA GRANDE
2024

Alana Cristine Nóbrega de Almeida

Detecção de tumores cerebrais em imagens de ressonância magnética utilizando redes neurais convolucionais

Trabalho de Conclusão de Curso submetido à Coordenação de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande, Campus Campina Grande, como parte dos requisitos necessários para obtenção do título de Graduado em Engenharia Elétrica.

Trabalho aprovado em: / /

Gutemberg Gonçalves dos Santos Júnior, D.Sc.
Orientador

Danilo Freire de Souza Santos, D.Sc.
Avaliador

Campina Grande
2024

Dedico este trabalho à minha família, em especial, aos meus pais, irmã e ao meu noivo.

AGRADECIMENTOS

A minha mãe, minha maior inspiração hoje e sempre, por seus esforços sem medida para me dar uma boa educação, absolutamente nada disso seria possível sem o seu amparo diário, e acima de tudo por ter me feito enxergar paixão em aprender e estudar sempre mais. Ao meu pai pelos domingos assistindo auto esporte que me fez criar curiosidade por esse mundo de robótica. A minha irmã pelas noites viradas na mesa da sala estudando comigo. E ao meu noivo por sempre acreditar em mim e me dar suporte a todo momento.

Expresso minha gratidão a todos os professores do Departamento de Engenharia Elétrica por dedicarem-se à mais nobre das profissões: a de disseminar seu próprio conhecimento.

Em especial ao professor Gutemberg Júnior. Agradeço ao senhor por nos apresentar ao universo da automação e mostrar o quanto o processo pode ser feliz quando se ama o que se está fazendo. Obrigada por tratar todos a sua volta com tanto respeito. E por ter apresentado o primeiro projeto que me incentivou a estudar programação, esse fato me levou ao trabalho que tenho hoje, pelo o qual sou tão grata.

"Depois do medo, vem o mundo" -

Clarice Lispector

ABSTRACT

“ This report details the process of developing and implementing a Convolutional Neural Network using Transfer Learning from the VGG19 model. It includes explanations and references about the evolution of Computer Vision, as well as practical aspects such as image preprocessing, model building, training, inference, and validation. In addition, it discusses the Grad-CAM technique, which provides insight into the algorithm’s decision-making process through heatmap visualization.”

Keywords: Convolutional Neural Network; Transfer Learning; VGG19; Computer vision; Grad-CAM.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

AP	Aprendizado Profundo
GPU	*Graphics Processing Unit*
RAM	*Random Access Memory*
GB	*Gigabyte*
RMC	Ressonância Magnética Cerebral
RM	Ressonância Magnética
RGB	*Red, Green, Blue*
VGG	*Visual Geometry Group*
Grad-CAM	*Gradient-weighted Class Activation Mapping*
AUC	*Area Under The Curve*
CAD	*computer-assisted diagnosis*

SUMÁRIO

1 INTRODUÇÃO

Neste capítulo será apresentada a justificativa, os objetivos gerais e específicos e, por fim, uma breve explicação sobre a organização do trabalho.

1.1 JUSTIFICATIVA

A Ressonância Magnética (RM) é um exame de imagem rico em detalhes, onde através de um campo magnético é possível capturar uma imagem com nível de nitidez alto, permitindo uma diferenciação entre a variedade de tecidos presentes no corpo humano, com mais exatidão do que uma tomografia, por exemplo. Por essa razão é o mais quisto dos profissionais da saúde quando se trata de diagnóstico de doenças potencialmente fatais, como a presença de tumores malignos.

Este trabalho foi desenvolvido utilizando como base imagens de RM, com ou sem diagnostico de tumores cerebrais. Estes tumores possuem algumas características em comum, apesar de normalmente não se disseminarem por outros órgãos, sendo em sua maioria malignos. O diagnóstico precoce pode ser um fator decisivo para sobrevivência do paciente, até mesmo quando se trata de um tumor benigno, já que o seu crescimento tem a capacidade de gerar danos irreversíveis com o passar do tempo, devido à sensibilidade da área em que se encontram.

O desafio do diagnóstico está na identificação do tumor na imagem, que podem possuir formatos e tamanhos imprevisíveis, dificultando a avaliação do profissional e a identificação da doença. Com o objetivo de auxiliar na resolução desse problema, foi iniciada a prática de diagnósticos assistidos por computadores, do inglês *computer-assisted diagnosis* (CAD), promovendo uma análise automática de imagens médicas.

Esses estudos que buscam um padrão entre imagens têm se tornado comum com o advento da prática do aprendizado de máquina, do inglês *Machine Learning*. Essa tecnologia tem a habilidade de aumentar a performance de um algoritmo, o tornando especialista em uma atividade, treinando uma capacidade específica ao fornecer mais dados para que aprenda com eles.

O que parece extremamente complexo para um ser humano identificar a olho nu, mesmo que capacitado, pode ser alcançado com exatidão através de um algoritmo bem treinado. O objetivo deste trabalho é a criação, treinamento e avaliação de um algoritmo de identificação de tumores em RM, utilizando apenas recursos totalmente gratuitos e acessíveis.

Esse estudo além de se mostrar de grande valia para uma aplicação prática, é bastante rico em conhecimento. Ele possibilita a união do *Machine Learning* com o campo de *Data Science*, abrindo as portas para várias ferramentas que possibilitam aumentar a performance do código e catalogar essa evolução, através de gráficos e estudos desses números de evolução.

No desenvolvimento de algoritmos de classificação com Redes Neurais Convolucionais,

o volume do banco de dados que alimenta o treinamento dessa rede, possui um grande impacto no valor da sua acurácia final, porém um grande banco de dados exige uma capacidade de hardware proporcional. Uma forma de contornar essa problemática é a utilização da técnica de Transferência de Aprendizado, onde é possível utilizar um modelo que já foi treinado em outro servidor e integrá-lo em um outro algoritmo.

Nesse trabalho o banco de dados selecionado se encaixa nesse cenário, e a intenção da sua escolha foi a possibilidade de aplicação de métodos e técnicas como a Transferência de Aprendizado, mostrando que é possível atingir uma boa acurácia ao contornar essa dificuldade.

1.2 OBJETIVOS GERAIS

Modelar uma Rede Neural Convolucional para detecção de tumor em imagens de Ressonância Magnética Cerebral, com a aplicação da técnica de Transferência de Aprendizado.

1.3 OBJETIVOS ESPECÍFICOS

- Avaliar e investigar os conceitos de Inteligência Artificial, Aprendizado de Máquina, Aprendizado Profundo, Visão Computacional e Transferência de Aprendizado com aplicação para Redes Neurais Convolucionais;
- Utilização de ferramentas atuais como o TensorFlow, Pandas, Keras entre outras bibliotecas e frameworks do mercado;
- Modelar um algoritmo de classificação binária com Transferência de Aprendizado utilizando Redes Neurais Convolucionais;
- Avaliar a performance do modelo, explicitando os conceitos de acurácia e perda, além da aplicação de métodos como Grad-CAM que auxilia na compreensão de erros de previsão.

1.4 ORGANIZAÇÃO DO TRABALHO

No capítulo 2 são abordados os conceitos utilizados como base para desenvolvimento do trabalho, como Redes Neurais Convolucionais e suas funções, além de temas como Transferência de Aprendizado e a Arquitetura do VGG19. No Capítulo 3 são apresentados os aspectos fundamentais para a construção do modelo, complementado pelo capítulo 4 onde é feito a análise do modelo e seus resultados.

2 REFERENCIAL TEÓRICO

A modelagem de uma Rede Neural Convolucional (RNC) é uma atividade que se tornou bastante acessível com a disponibilização de inúmeras bibliotecas e recursos gratuitos para construção. O intuito deste capítulo é apresentar ao leitor os fundamentos teóricos por trás das funções utilizadas para a construção do modelo. Além de uma breve explicação sobre os exames de imagem que compõem o *dataset* a ser usado.

2.1 RESSONÂNCIA MAGNÉTICA CEREBRAL E SUAS TÉCNICAS

A Ressonância Magnética (RM) foi desenvolvida a partir da descoberta da técnica de Ressonância Magnética Nuclear (RMN), (BLOCH, 1946), (PURCELL; TORREY; POUND, 1946), inovação esta cujo impacto científico gerou como fruto um Prêmio Nobel em física aos autores *Felix Bloch* e *Edward Purcell* em 1952. Em 1970, o médico e pesquisador *Dr. Raymond Damadian* se baseou na ideia dos pesquisadores *Bloch* e *Purcell*, aplicando-a a diagnósticos médicos, o que o levou ao entendimento de que células cancerígenas possuíam um comportamento diferente nas imagens RM, emitindo sinais e se destacando. Essa observação levou *Damadian* a criar e patentear a primeira máquina de RM em 1974.(BELLIS, 2019)

São 30 anos que separam a revelação do fenômeno físico da RMN e a aplicação da RM. A justificativa é a complexidade que existe por trás do sistema, são conceitos de eletromagnetismo, supercondutividade e processamento de sinais abordados em conjunto. De forma sucinta, os sinais recebidos pelo antena receptora do equipamento RM possuem diferenças no tempo de chegada, levando a diferenciação dos tecidos que estão emitindo tais sinais. Partindo desse princípio é possível montar uma imagem que reflita a anatomia do corpo humano, como na figura 1. Lembrando que todos esses resultados se tornam ainda mais atrativos unidos ao fato de não exposição a radiação, o que ocorre em outros métodos de diagnóstico como a tomografia.(MAZZOLA, 2015)

Figura 1 – Imagem turbo spin eco ponderada, T1 e T2 são as contantes de tempo.

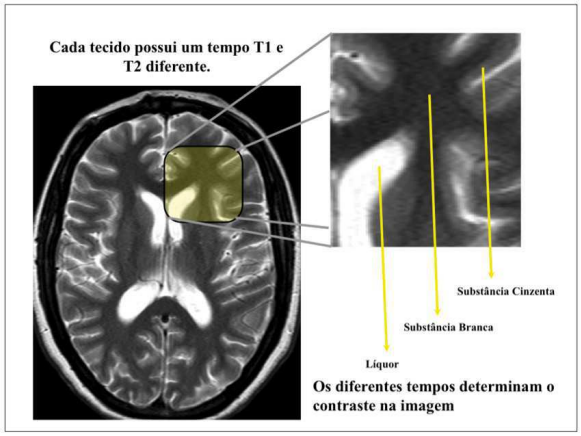
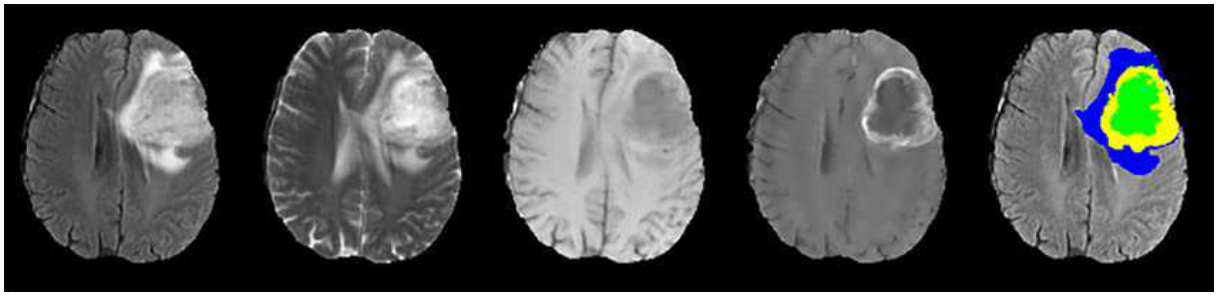


Figura 10. Imagem turbo spin eco ponderada em T2, mostrando na imagem ampliada a resolução de contraste obtida devido as diferenças nos tempos T2 entre os tecidos envolvidos.

Fonte: (MAZZOLA, 2015)

No estudo de imagens de RM algumas técnicas são utilizadas para destacar características na imagem, podendo ser interpretadas como filtros. Nesse trabalho foi utilizado um *dataset* com imagens de RNC onde são vistas três dessas técnicas aplicadas: a T1 ponderada, a T2 ponderada e a Flair. (Nesse momento se faz necessário um ponto de atenção, o T1 e T2, mencionados a partir desse parágrafo não são referentes às constantes de tempo da figura 1 acima, e sim às técnicas de RM). Abaixo é apresentada a figura 2, com quatro representações do mesmo cérebro com presença de tumor, contendo o mesmo ângulo de observação, o corte axial, se diferenciando apenas pelas técnicas aplicadas (Flair, T2, T1, T1ce).

Figura 2 – Técnicas de RMC: Flair, T2, T1, T1ce



Fonte: (HU; GU; GU, 2020)

Explicando de uma forma bastante objetiva, apenas para o propósito de identificar as técnicas, na figura é possível distinguir a técnica T1 observando as partes do tecido que possui alta concentração de água retratadas de forma mais escura, por possuir baixo sinal nessa representação. A imagem T1 tem uma formação similar à anatomia cerebral, sendo conhecida como anatômica por essa característica. A parte com mais sinal, visivelmente mais clara, é o tecido conhecido como massa branca, já a parte periférica com um tom um pouco mais escuro,

é a massa cinzenta. Perceba que essa não é a melhor forma para identificar as dimensões do tumor, ela denota muita ênfase a massa branca, fazendo com que o tumor não se destaque nela. As composições moleculares que recebem mais ênfases são: melanina, sangue, gordura, proteína e o contraste. Logo, essa técnica apresenta bons resultados no diagnóstico de casos como hemorragias. A técnica T1ce é a T1 no momento pós contraste.

Analisando a abordagem T2, perceba que é uma imagem predominantemente escura, possui características quase que opostas a T1, dando ênfase e brilho a partes ricas em água.

Já o procedimento Flair é uma combinação da T1 e T2. Possui basicamente todas as características do comportamento de T2, se diferenciando na apresentação do *Líquor*, *líquido cerebrospinal*, composto em sua maior parte por água, expressando um sinal baixo na imagem, nesse ponto se assemelha ao método T1.(HU; GU; GU, 2020).

2.2 INTELIGÊNCIA ARTIFICIAL E SUAS SEGMENTAÇÕES

Os algoritmos com aprendizado de máquina têm realizado conquistas notáveis. Por exemplo, em 1997, o *Deep Blue* desenvolvido pela IBM, realizou o feito de derrotar *Garry Kasparov*, campeão mundial de xadrez(CAMPBELL; HOANE; HSU, 2002). O êxito em uma atividade de estratégia chamou bastante atenção, porém atividades que são tidas para os humanos como simples, o ato de reconhecer um cachorro em uma foto, ou discernir palavras faladas, a pouco tempo não eram tangíveis para Inteligência Artificial (IA). O avanço no estudo das Redes Neurais Convolucionais (RNC) tornaram essas ações possíveis, remontando ao córtex visual do cérebro humano, sendo aplicadas em reconhecimento de imagens e em processamento de linguagem natural (PLN).(GéRON, 2022).

A IA nasceu na década de 1950 a partir do questionamento inicial de se os computadores poderiam ser projetados para “pensar”. O campo da IA abrange os temas Aprendizado de Máquina (AM), Aprendizado Profundo (AP) e RNC. Nessa linha de pensamento o pioneiro em IA, Alan Turing, apresentou em seu artigo (TURING, 1950) o que ficou conhecido como o “Teste de Turing”. O teste consiste, de forma resumida, em um interrogador humano interagindo ao mesmo tempo com uma máquina e um humano, porém o interrogador em questão não tem conhecimento sobre a origem das respostas. Para que a máquina passe no teste, o seu comportamento deve ser igual ou superior ao do humano.

Segundo o livro (CHOLLET, 2017), o pensamento gatilho para o estudo do AP foi que: “seria possível um computador não apenas fazer o que lhe foi prescrito, mas nos surpreender aprendendo de forma automática a realizar uma tarefa? se baseando em dados para entregar as regras e não o contrário”, esse raciocínio abre a discussão de que máquinas não só seriam capazes de atingir a habilidade humana como superá-la em certas execuções.

“[Aprendizado de máquina é o] campo de estudo que possibilita aos computadores a habilidade de aprender sem explicitamente programá-los.” - Arthur Samuel, 1959

Outra perspectiva mais voltada a engenharia:

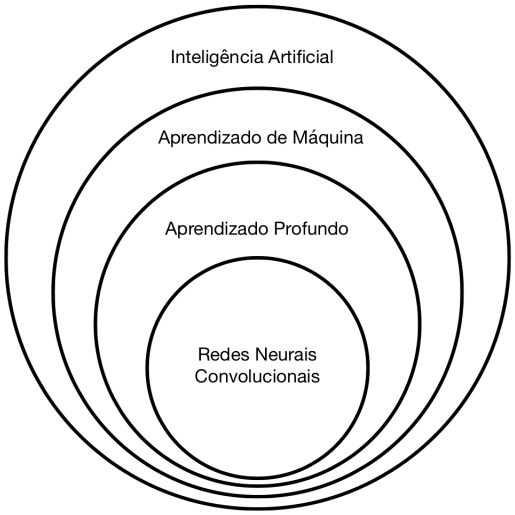
“Alega-se que um programa de computador aprende pela experiência E em relação a

algum tipo de tarefa T e alguma medida de desempenho P se o seu desempenho em T, conforme medido por P, melhora com a experiência E." - Tom Mitchell, 1997

O Aprendizado Profundo é um subconjunto do Aprendizado de Máquina, onde é dado ênfase ao aprendizado por consecutivas camadas. O "profundo" é erroneamente entendido como um aprendizado mais significativo e embasado sobre algo. Na verdade, se refere a ideia de sucessivas camadas de treinamento. Esse aprendizado por repetição, por assim dizer, pode super adaptar o algoritmo ao banco de dados utilizado no treino, problema de *Overfitting* que será abordado mais a frente na seção 2.3.3.

Para encerrar essa explanação sobre a posição dos campos de estudo que aqui serão abordados, temos as Redes Neurais Convolucionais, que por sua vez fazem parte de AP, e se caracterizam pela técnica de criação de mapas de características através da operação de convolução, também será tratado desse tema na próxima seção 2.3.

Figura 3 – Diagrama IA



Fonte: Autoria Própria

2.3 REDES NEURAIS CONVOLUCIONAIS

A RNC foi descoberta e publicada por *Yann LeCun* e colaboradores, em 1998 no artigo (LECUN et al., 1998). O modelo usado para exemplificar a arquitetura de uma RNC no artigo foi o *LeNet-5*, sendo a aplicação descrita o reconhecimento de padrões em documentos escritos à mão com precisão. Esse avanço no reconhecimento de padrões e no campo de visão computacional impactou o meio acadêmico e industrial por ser um grande avanço técnico, além de demonstrar o potencial prático das RNC, sendo uma de suas primeiras aplicações em bancos, com o objetivo de exercer a leitura automatizada de cheques.

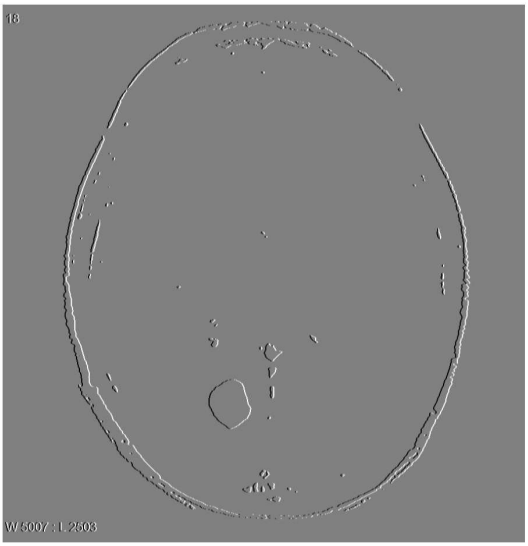
A arquitetura CNN é dividida em camadas, onde cada camada possui uma função. As camadas mais características são as de convolução mencionadas anteriormente. Elas possuem a

função de passar a imagem de entrada por filtros para que seja colocado em evidência os pixels da imagem que realmente vão impactar na identificação do objeto, como destacar bordas ou texturas. Por exemplo, se eu tenho uma imagem de uma ressonância magnética com tumor, figura 4a, os pixeis que são determinantes na minha imagem são as bordas do meu tumor.

Figura 4 – Exemplo

(a) Imagem com tumor sem o filtro

Fonte: (CHAKRABARTY, 2019)

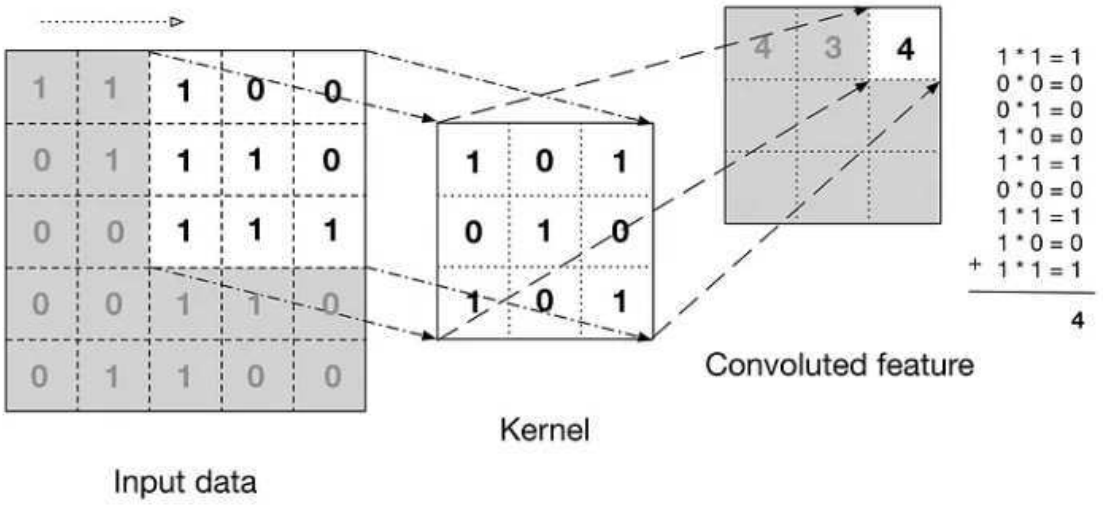


(b) Imagem com tumor com filtro

Fonte: (CHAKRABARTY, 2019)

Para fazer o destaque das características, uma matriz nomeada de *Kernel* percorre toda a matriz da imagem, realizando ao longo do caminho as operações explanadas na figura 5. A matriz *Kernel* vai deslizando coluna por coluna da imagem, e através desse processo é possível destacar só os elementos que importam para a classificação. Como resultado da operação de convolução é obtido uma nova matriz, chamada de "mapa de características", ou *convoluted feature*, e para cada dimensão que a imagem possuir será impresso um mapa de característica desse processo. Se a imagem de entrada do sistema estiver representada em RGB, significa que cada imagem vai gerar 3 mapas de características após a convolução. Os filtros, ou matiz *Kernel*, não precisam ser escolhidos manualmente, a RNC irá aprender quais filtros foram mais eficazes em auxiliar na classificação correta e selecionar eles de forma automática. Na convolução da figura 4b foi utilizada a matriz *Kernel* **Sobel**. Por motivos didáticos, abaixo temos como o processo de convolução funciona de forma visual.

Figura 5 – Mapa de Características



Fonte: (BARBOSA, 2019)

Porém, a RNC não se resume a apenas a operação de convolução, existem outras camadas que a compõem, como as funções de ativação, *polling* e *dropout*. Neste capítulo serão abordadas as funções de ativação que foram aplicadas na solução do trabalho, a função *ReLU* e a *Sigmoid*. No entanto existem outras diversas funções que podem e são utilizadas em RNC como Softmax, Tanh (Hyperbolic Tangent), Leaky ReLU, entre outras.

2.3.1 Função de Ativação Rectified Linear Unit - ReLU

A função *ReLU* é a mais utilizada nas redes de convolução. Ela tem por objetivo transformar os valores de entrada negativos em zero e permanecer inalterados os valores positivos. Essa aplicação é ideal para redes profundas, pois evita problemas de saturação do gradiente. Esse problema se manifesta quando o valor do gradiente se torna tão próximo a zero durante o processo de *Retro Propagação* que isso o impede de atualizar os pesos de forma adequada. Sua fórmula matemática é:

$$ReLU(x) = max(0,x) \tag{1}$$

- Se $x > 0$, a saída é x.
- Se $x \leq 0$, a saída é 0.

2.3.2 Função de Ativação Sigmoid

A função *Sigmoid* transforma todo valor de entrada em uma saída no intervalo de (0, 1), podendo ser aplicada em redes de classificação binária, que é o caso abordado neste trabalho, onde a imagem tem tumor ou não tem tumor. Essa função de ativação deve ser utilizada na última

camada, pois caso seja aplicada em camadas intermediárias ocasiona o problema que citamos anteriormente da saturação do gradiente. Sua fórmula matemática é:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

- e é a base do logaritmo natural.
- x é o valor de entrada.

2.3.3 ***Polling*, *Dropout* e *Overfitting***

Os outros recursos utilizados no treinamento da rede neural convolucional nesse trabalho foram o *Pooling* e o *Dropout*.

A etapa de *Pooling* visa uma subamostragem do mapa de características, deixando apenas os dados mais essenciais. A função de *Pooling* mais utilizada é o *MaxPoling*, faz a extração dos maiores valores da matriz de características. Como pode ser visto na operação convolução, na figura 5, os maiores valores da matriz equivalem às principais características. No entanto, aqui neste trabalho existe uma especificidade: a aplicação de transferência de aprendizado na construção do modelo. Esse tema será aprofundado na seção 2.4.

Quando é utilizada a metodologia de transferência de aprendizado, é necessário fazer o encaixe entre o modelo pré treinado, neste trabalho foi o VGG19, e o novo modelo que está sendo construído. No caso do encaixe com o VGG19, a saída das últimas camadas será uma matriz 3D, no entanto a entrada das primeiras camadas do novo modelo é uma matriz 2D, nesse ponto se faz necessário um redimensionamento da saída do VGG19. Para essa operação é possível utilizar a função *GlobalAveragePooling2D()*. Ela vai reduzir a dimensionalidade, compactar as características aprendidas, e evitar o *Overfitting*. Fazendo uma comparação entre ela e a função *Pooling* mais comum *MaxPooling2D()*, enquanto a *MaxPooling* seleciona o valor máximo em uma região da matriz, a *GlobalAvaragePooling* executa o redimensionamento através de uma média, sendo mais adequado para compactar características de alta dimensão.

A problemática do *Overfitting* pode ser evitada com a função *Dropout*. O *Overfitting* ocorre quando o treinamento leva o modelo a ter um super ajuste a base de dados, e no momento em que ele tem contato com outros dados os resultados não são bons, mesmo que a acurácia durante o treinamento tenha sido satisfatória. Esse tema é trazido com mais riqueza de detalhes no artigo (SRIVASTAVA GEOFFREY HINTON, 2014). Ele menciona que apesar do *Dropout* ter o impacto negativo de um maior tempo de treinamento da rede, ele age prevenindo o *Overfitting* desativando aleatoriamente unidades da rede neural durante o processo de treinamento, colocando como 0 o valor de algumas entradas, e escalonando para cima os valores das entradas restantes, de tal forma que o somatório entre todas as entradas não é alterado (KERAS, 2023). Lembrando que esse processo mesmo sendo aplicado entre as camadas do modelo, no momento de inferência as entradas da rede serão todas ativadas normalmente, aplicando somente durante o treinamento.

2.4 TRANSFERÊNCIA DE APRENDIZADO

Para abordar esse tema foi utilizado como base o livro *"Hands-On Machine Learning"* (GéRON, 2022). O autor apresenta a técnica de transferência de aprendizado, tradução de como é chamado no livro, *trasnfer learning*, onde são reutilizadas camadas pré-treinadas no nosso modelo. Vendo o caso aplicado nesse trabalho, foram utilizadas camadas do modelo VGG19 previamente treinadas com o banco *ImageNet*. Isto permite o aproveitamento dos pesos, sendo necessário fazer apenas o ajuste para o *dataset* do trabalho, reutilizando as últimas camadas do VGG19 e integrando elas ao novo modelo. Dessa forma, é possível desfrutar das vantagens que o livro aponta, como acelerar de forma considerável o processamento, diminuir a capacidade de hardware necessária para um bom treinamento e o principal para o caso aqui abordado, atingir uma boa acurácia sem *Overfitting*, mesmo possuindo um banco de dados consideravelmente pequeno, 253 imagens, sento em media 200 para treinamento e 53 para validação.

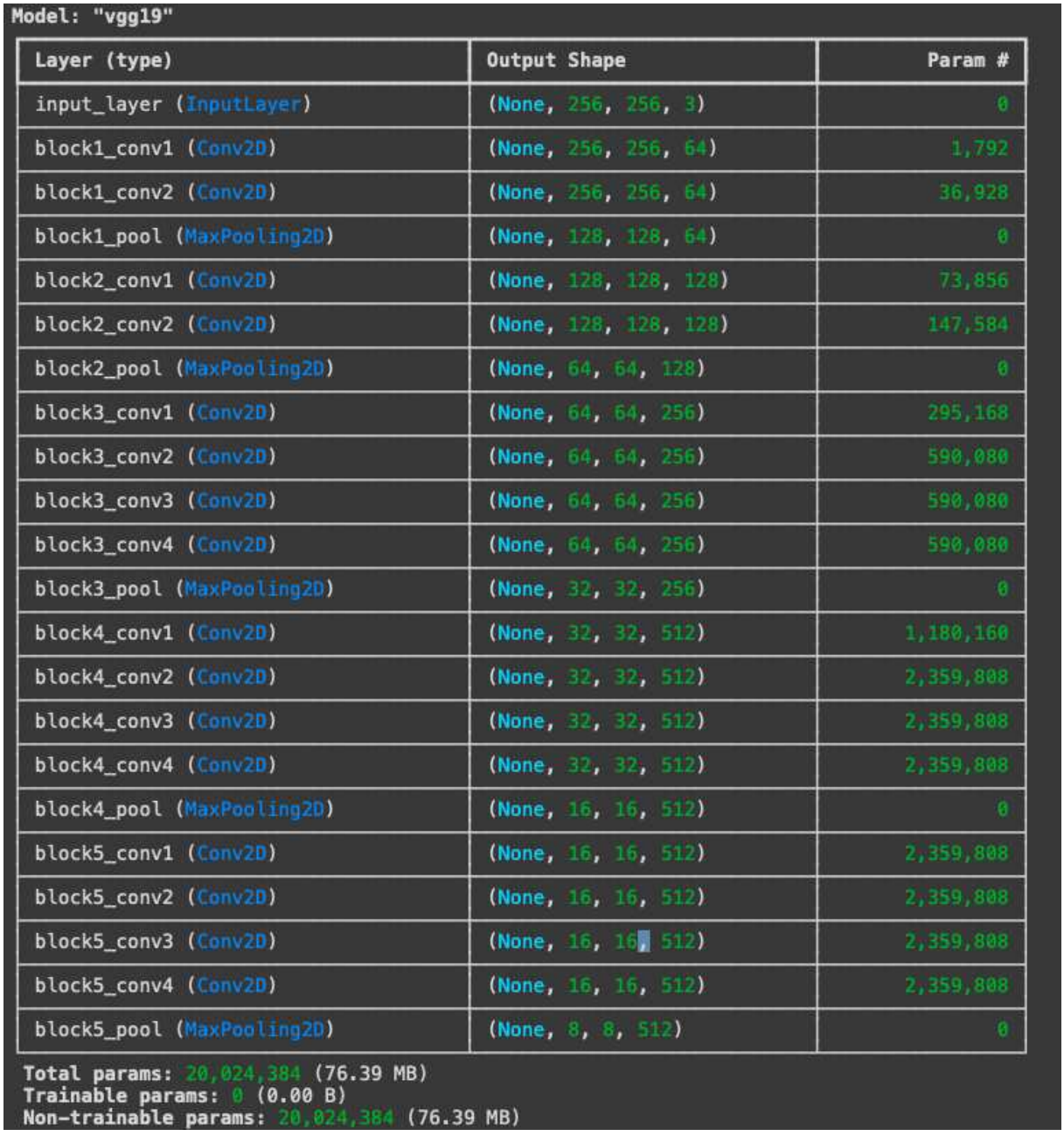
2.4.1 **VGG19**

O modelo VGG19 escolhido para a aplicação, é um aprimoramento do modelo *VGGNet*, também desenvolvido por *Karen Simonyan* e *Andrew Zisserman* no laboratório de pesquisa *Visual Geometry Group (VGG) da Universidade de Oxford*, (GéRON, 2022). Ambos os modelos foram divulgados no artigo *"Very Deep Convolutional Networks for Large-Scale Image Recognition"* no ano de 2014, (SIMONYAN; ZISSERMAN, 2014). O modelo *Visual Geometry Group* (VGG) vem da família de RNCs, utilizando camadas convolucionais em estruturas profundas para identificar características em imagens. Os aspectos individuais da composição do VGG19 é que este possui 19 camadas de profundidade, sendo elas 16 camadas de convolução e 3 para conexão.

Segundo o artigo (SIMONYAN; ZISSERMAN, 2014) as principais características que compõe a arquitetura do VGG19, além das 19 camadas de profundidade, são os filtros de convolução muito pequenos (3x3). A definição desse filtro, matriz *Kernel* foi abordada na seção 2.1, figura 5. A matriz *Kernel* reduzida promove uma maior eficácia na detecção de características importantes para classificação, mesmo estando ela em várias escalas.

A figura abaixo 6 foi criada com a função *Summary*. Ela exibe de uma forma visual a arquitetura VGG19, informando todas as camadas utilizadas.

Figura 6 – Arquitetura VGG19



Model: "vgg19"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 256, 256, 3)	0
block1_conv1 (Conv2D)	(None, 256, 256, 64)	1,792
block1_conv2 (Conv2D)	(None, 256, 256, 64)	36,928
block1_pool (MaxPooling2D)	(None, 128, 128, 64)	0
block2_conv1 (Conv2D)	(None, 128, 128, 128)	73,856
block2_conv2 (Conv2D)	(None, 128, 128, 128)	147,584
block2_pool (MaxPooling2D)	(None, 64, 64, 128)	0
block3_conv1 (Conv2D)	(None, 64, 64, 256)	295,168
block3_conv2 (Conv2D)	(None, 64, 64, 256)	590,080
block3_conv3 (Conv2D)	(None, 64, 64, 256)	590,080
block3_conv4 (Conv2D)	(None, 64, 64, 256)	590,080
block3_pool (MaxPooling2D)	(None, 32, 32, 256)	0
block4_conv1 (Conv2D)	(None, 32, 32, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 32, 32, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 32, 32, 512)	2,359,808
block4_conv4 (Conv2D)	(None, 32, 32, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 16, 16, 512)	0
block5_conv1 (Conv2D)	(None, 16, 16, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 16, 16, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 16, 16, 512)	2,359,808
block5_conv4 (Conv2D)	(None, 16, 16, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 8, 8, 512)	0

Total params: 20,024,384 (76.39 MB)
Trainable params: 0 (0.00 B)
Non-trainable params: 20,024,384 (76.39 MB)

Fonte: Autoria própria com auxilio da função Summary()

2.5 GRAD-CAM

No artigo *"Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization"*, (SELVARAJU et al., 2020) é apresentada a técnica *Gradient-weighted Class Activation Mapping* (*Grad-CAM*). É uma abordagem eficaz para visualizar através de um mapa de calor quais áreas na imagem foram relevantes para a classificação do modelo de que nesta imagem é a presença de um gato ou um cachorro, ou qual que seja a label do modelo usado. A imagem original é sobreposta com o mapa construído e as cores mais quentes indicam os pixels

que possuem maior influência na classificação.

Figura 7 – Imagens c/ convolução e técnica *Grad-CAM*

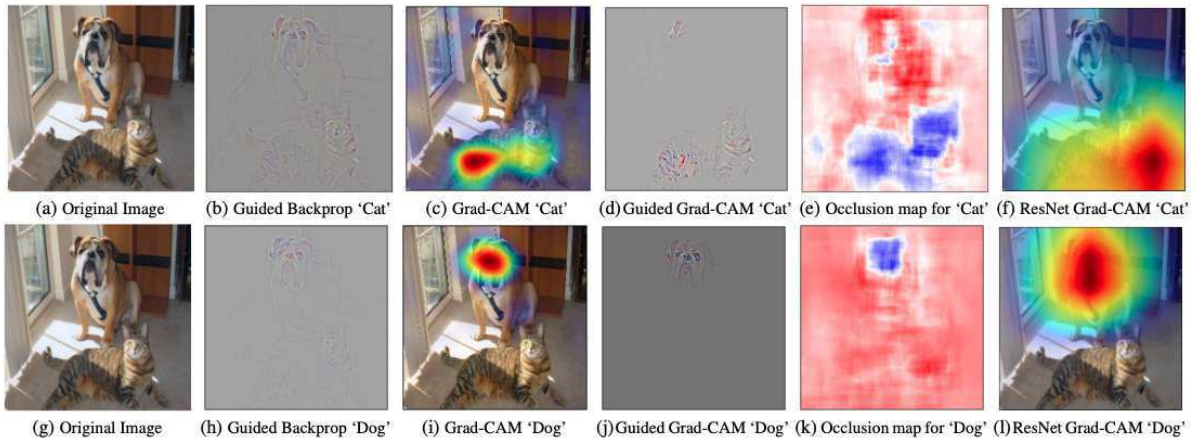


Figure 1: (a) Original image with a cat and a dog. (b-f) Support for the cat category according to various visualizations for VGG-16 and ResNet. (b) Guided Backpropagation [42]: highlights all contributing features. (c, f) Grad-CAM (Ours): localizes class-discriminative regions, (d) Combining (b) and (c) gives Guided Grad-CAM, which gives high-resolution class-discriminative visualizations. Interestingly, the localizations achieved by our Grad-CAM technique, (c) are very similar to results from occlusion sensitivity (e), while being orders of magnitude cheaper to compute. (f, l) are Grad-CAM visualizations for ResNet-18 layer. Note that in (c, f, i, l), red regions corresponds to high score for class, while in (e, k), blue corresponds to evidence for the class. Figure best viewed in color.

Fonte: Artigo (SELVARAJU et al., 2020)

Segundo o artigo (SELVARAJU et al., 2020) a técnica pode ser aplicada em RNCs no geral, trazendo interpretabilidade e confiança no modelo. No exemplo da arquitetura VGG19 citada acima 6, é possível visualizar que as camadas de convolução são todas encapsuladas no modelo. Como saída temos a acurácia e a perda, mostrada na seção 4, mas não é possível ver quais os fatores que levaram a esses valores de métrica. Um exemplo que torna isso muito claro é o que foi mostrado nesse trabalho, na figura 19b na seção 4. Onde o modelo identificou a presença de um tumor na imagem, mas a label da imagem é de que não existe tumor nela, e só com esse resultado de predição não é possível concluir onde na imagem o modelo identificou o tumor. Nesse ponto a aplicação do *Grad-CAM* conseguiu ser bastante feliz, por exibir exatamente onde com mapa de calor, os pesos identificaram um tumor.

Dito isso, também é possível chegar a análise de que a aplicação do *Grad-CAM* com o modelo de VGG19 é uma união eficiente, pela característica do VGG19 de aplicar matrizes *Kernel* de pequenas dimensões em suas convolução, é possível identificar características estejam elas em pequena ou grande escala.

2.6 TRABALHOS RELACIONADOS

O trabalho "*Brain tumor classification using deep CNN features via transfer learning* ", (DEEPAK; AMEER, 2019), visa buscar e se aprofundar em técnicas para reconhecimento de tipos de tumores cerebrais em RM. Os tipos considerados para o desenvolvimento da rede são: glioma, meningioma e tumores da hipófise. É um modelo de classificação multi-classe, o que leva a escolha da camada classificadora *softmax*, uma função de ativação que se aplica a essas

situações, assim como neste trabalho foi selecionada a *Sigmoid*, indicada para classificações binárias.

Ele utiliza de RNC como modelo base do trabalho, e faz uso de Transferência de Aprendizado para modelagem, aplicando o modelo *GoogLeNet*. O algoritmo do *GoogLeNet* escolhido, foi pré-treinado com o *dataset* Imagenet, que faz uso de 1,2 milhão de imagens naturais. Durante a descrição da escolha do modelo, é feita uma comparação entre *GoogLeNet* e *AlexNet*, e informado que o motivo da decisão pelo *GoogLeNet* se deu pela sua performance nesta atividade.

O modelo *GoogLeNet* possui duas camadas convolucionais, duas camadas de *pooling* e nove módulos de *inception*, e uma camada totalmente conectada. E para possibilitar o "encaixe" foram adaptadas suas três últimas camadas, a camada totalmente conectada do *GoogLeNet* original foi removida. No seu lugar, uma nova camada de conexão, com um tamanho de saída de três foi inserida. A camada *softmax*, seguindo a camada de conexão, e a camada de classificação baseada em entropia cruzada na saída foram substituídas por novas.

O artigo utiliza o *dataset* do conjunto de dados de RM do *figshare*, e enfatiza o quanto o conceito de aprendizado por transferência é útil quando se trabalha com um banco de dados limitado, esta observação é confirmada no artigo com uma acurácia de 98 por cento.

Como método de validação, o experimento utiliza o processo de validação cruzada, análise da área sob a curva (AUC), precisão, recall, pontuação F e especificidade.

Outro trabalho desenvolvido nessa linha foi o artigo "Convolutional Neural Networks for Medical Image Analysis: Full Training or Fine Tuning¿', (TAJBAKHSH et al., 2016). Ele é discorrido em cima do questionamento "O uso de RNCs profundas pré-treinadas com ajuste fino é suficiente para eliminar a necessidade de treinar uma RNC profunda do zero¿'.

Para responder a pergunta, o experimento trabalha com RNC modeladas com e sem o auxílio de redes pré-treinadas. Como resultado final, o artigo afirma que as RNC desenvolvidas com transferência de aprendizado possuem um comportamento melhor ou similar ao de RNC treinadas do "zero". Como base para a afirmação, são exibidos os desempenhos das redes, a RNC com transferência em seu pior cenário de performance, alcançou um valor de acurácia compatível com a acurácia das RNC sem transferência, e nos outros cenários superou a acurácia.

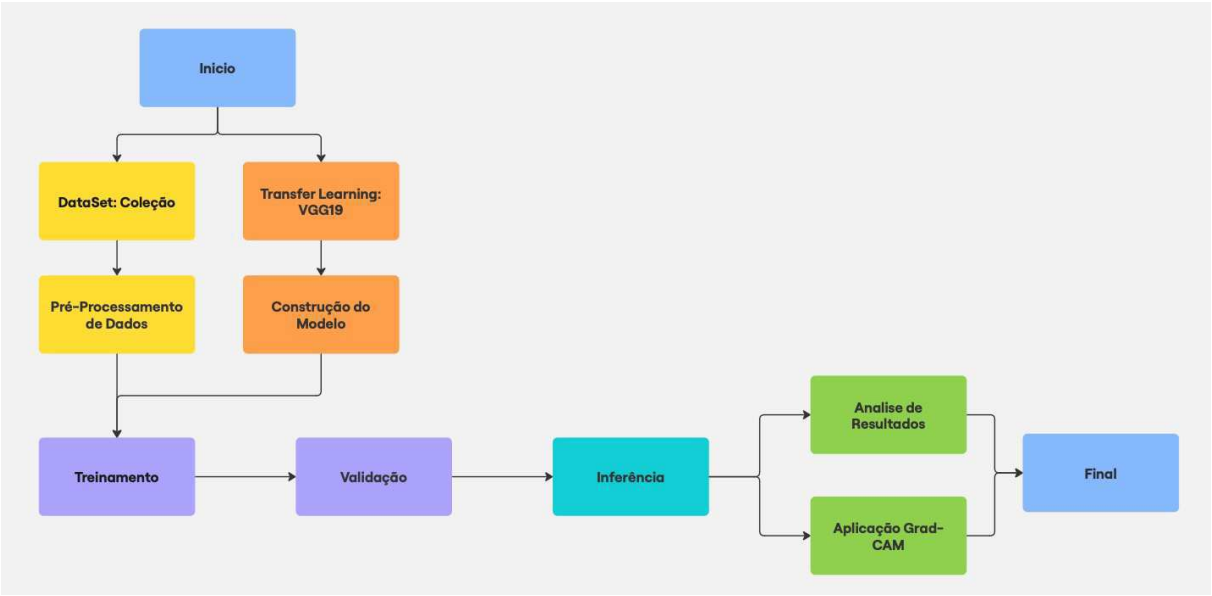
3 METODOLOGIA

Neste capítulo serão descritos os métodos aplicados para classificação das imagens, com e sem a presença de tumor. Também será aprofundado quais componentes foram utilizados ao longo desse processo.

3.1 VISÃO GERAL DO SISTEMA

Em uma visão de alto nível é possível dividir o sistema de classificação de imagens, desenvolvido nesse trabalho, em 5 etapas principais: Coleção e Pré-processamento do *Dataset*, Construção do modelo com *Transfer Learning*, Treinamento e Validação, Inferência, Análise de resultados e Aplicação de ferramentas para análise, como pode ser visto na figura 8.

Figura 8 – Diagrama do Sistema de Treinamento e Validação da RNC

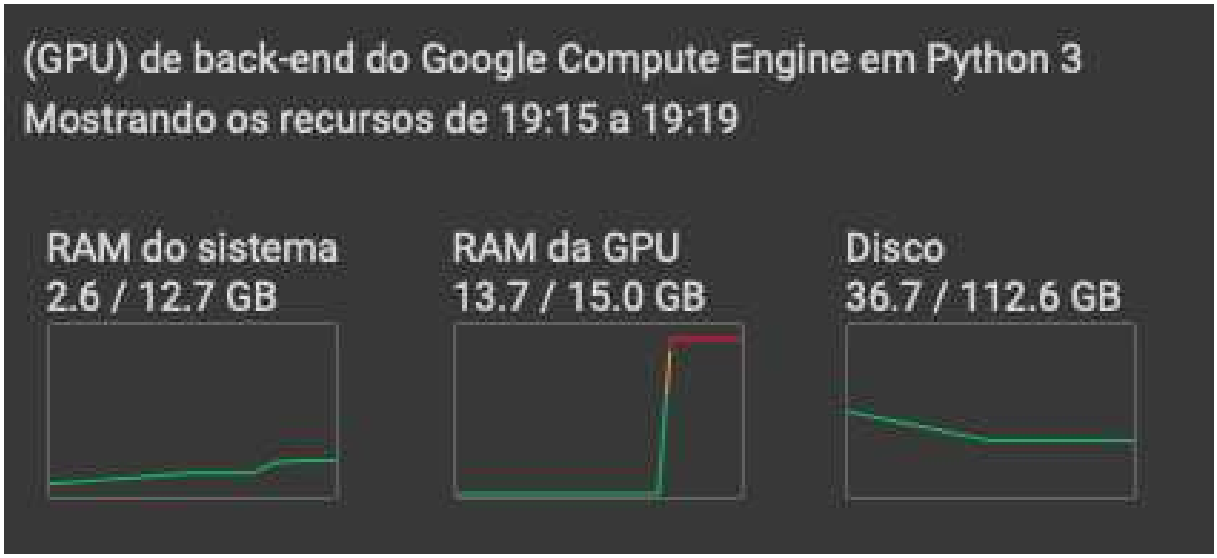


Fonte: Autoria Própria

O algoritmo foi desenvolvido na plataforma do *Google Colab*, utilizando o acelerador de hardware na configuração GPU, com capacidade de RAM do sistema 12.7 GB, RAM da GPU 15 GB e disco 112.6 GB. A plataforma com as configurações citadas, pode ser acessada de forma gratuita.

A figura 9 abaixo foi retirada do *Google Colab* durante o treinamento do modelo, permitindo que seja vista a abstração dos recursos utilizados.

Figura 9 – Recursos *Google Colab* Utilizados no Treinamento



Fonte: Autoria Própria

Para desenvolvimento do código foram utilizadas as bibliotecas Numpy, Pandas, TensorFlow, Keras e Matiplotlib, com a linguagem python na versão 3. As configurações utilizadas acima são comumente empregadas para criação e treinamento de modelos em AP.

3.2 DATASET

Alinhado com o objetivo deste trabalho, que é a detecção de tumores em imagens cerebrais, foi selecionado como banco de dados para treinamento, teste e validação do algoritmo, o *dataset* (CHAKRABARTY, 2019). Composto por imagens de RMC, que como explicado na seção 2.1, às RMC são idealmente aplicadas para diagnósticos precisos de condições neurológicas, como o caso aqui trabalhado.

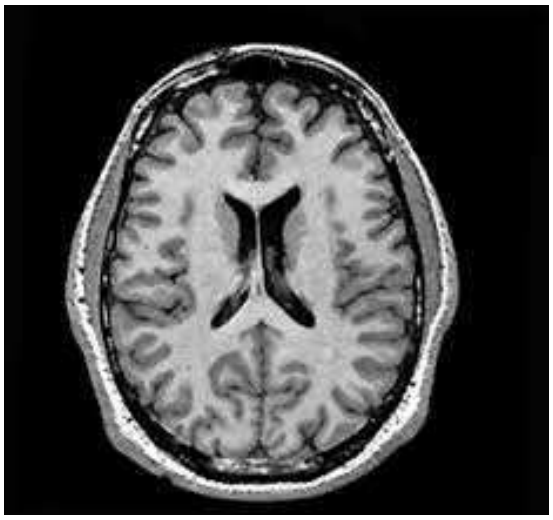
O banco de dados disponibilizado no kaggle (CHAKRABARTY, 2019), possui algumas informações como, o nome do autor e a data de criação do *dataset*, mas não possui especificações quanto a origem exata das imagens. Frequentemente imagens desse contexto podem ser encontradas em fontes públicas de conteúdos médicos e através de colaboração de instituições de saúde que disponibilizam dados para fins de pesquisa.

Pode ser agregado mais informações ao banco de dados, como identificar as técnicas de RMC utilizadas nas imagens, ou o seu tipo de corte para visualização. Empregando como base os conceitos informados na seção 2.1, é possível identificar através da amostra de imagens 10a, que o único corte presente nesse banco de dados é o corte de visão axial, conhecido também como corte horizontal ou transversal, que permite ver as camadas da RM como se o cérebro estivesse sendo analisado de cima.

Outro ponto que pode ser analisado são as técnicas utilizadas. Aqui nesse banco de dados são apresentadas três técnicas: *T1 ponderada*, *T2 ponderada* e *FLAIR*. Essas formas

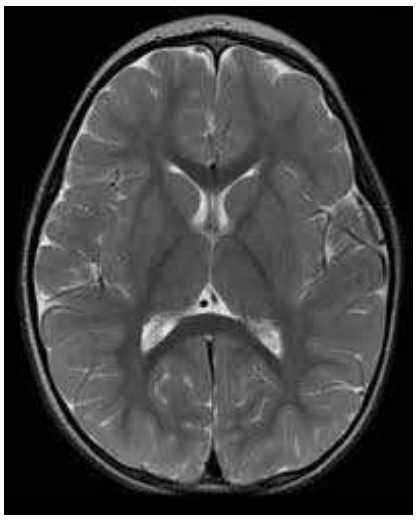
de analisar a anatomia cerebral são úteis na etapa de diagnóstico inicial e planejamento de tratamento.

Figura 10 – Amostra do *Dataset* representando técnicas de RM utilizadas



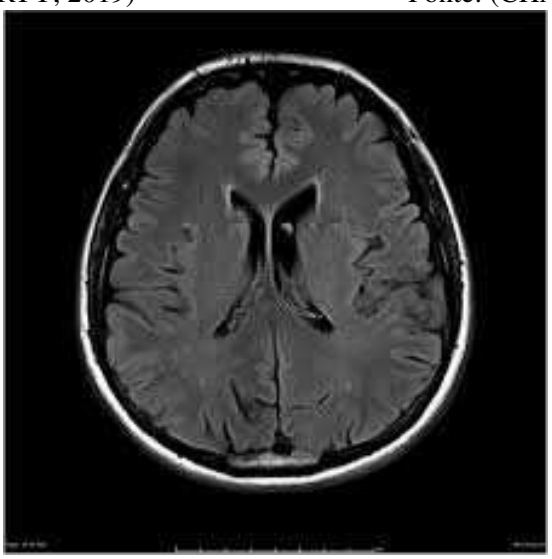
(a) Imagem com técnica T1

Fonte: (CHAKRABARTY, 2019)



(b) Imagem com técnica T2

Fonte: (CHAKRABARTY, 2019)

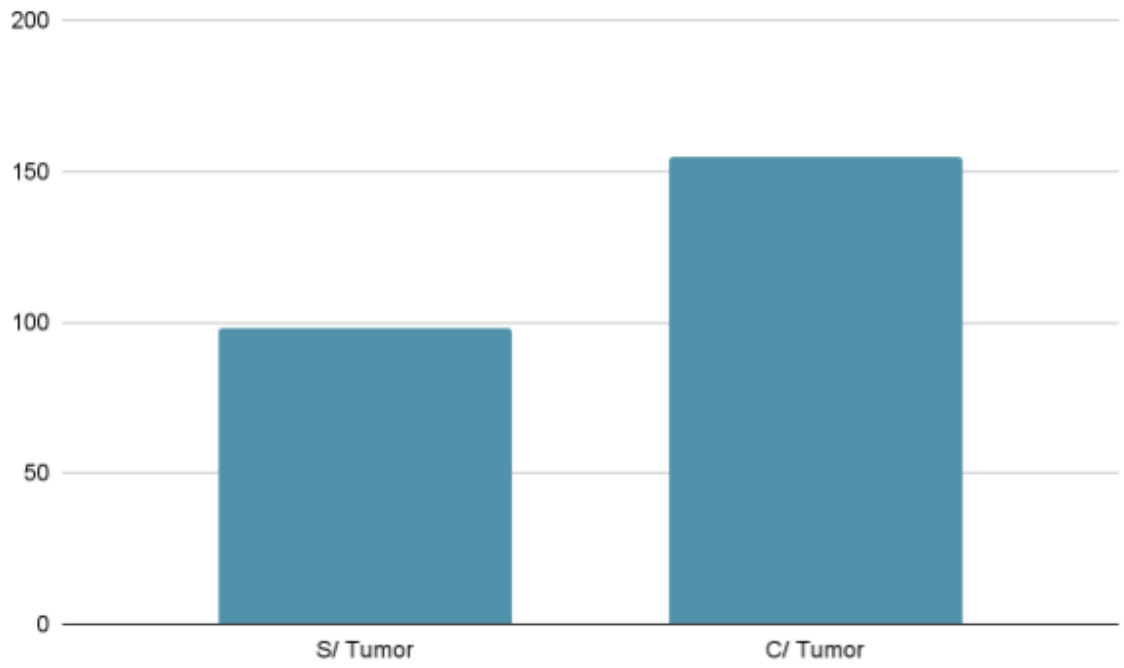


(c) Imagem com técnica Flair

Fonte: (CHAKRABARTY, 2019)

O *dataset* em questão é formado por 253 imagens, sendo elas divididas em imagens com a presença de tumores e sem a presença de tumores.

Figura 11 – Representação Subgrupos Label *Dataset*



Fonte: Autoria Própria

3.3 PRÉ-PROCESSAMENTO DOS DADOS

Após a importação das bibliotecas e antes da construção do modelo foram feitos no código, apêndice A, alguns tratamentos na imagem com o intuito de transformar o dado em uma entrada aceitável pra função *model.fit*. Essa função faz parte da biblioteca *TensorFlow.Keras* e foi utilizada no treinamento do modelo.

Os procedimentos utilizados nessa etapa foram: coleção de dados, definição das constantes, redimensionamento das imagens, separação do *dataset* entre dados de treinamento e dados de teste, e *one hot encoding*. Não se fez necessário adicionar label as imagens, pois o *dataset* utilizado foi disponibilizado no *Kaggle* (CHAKRABARTY, 2019) já com as rótulos, "yes" para imagens com tumor e "no" para imagens sem tumor.

Algumas contantes precisam ser definidas antes mesmo da utilização das funções de pré-processamento, essas contantes são o **batch-size** e o **image-size**. Para o **batch-size** foi escolhido de forma empírica o valor 32. Na seção 4 foram explanados os dados que embasam essa decisão. Quanto ao valor da constante **image-size**, foi escolhido $256x256$ pixels.

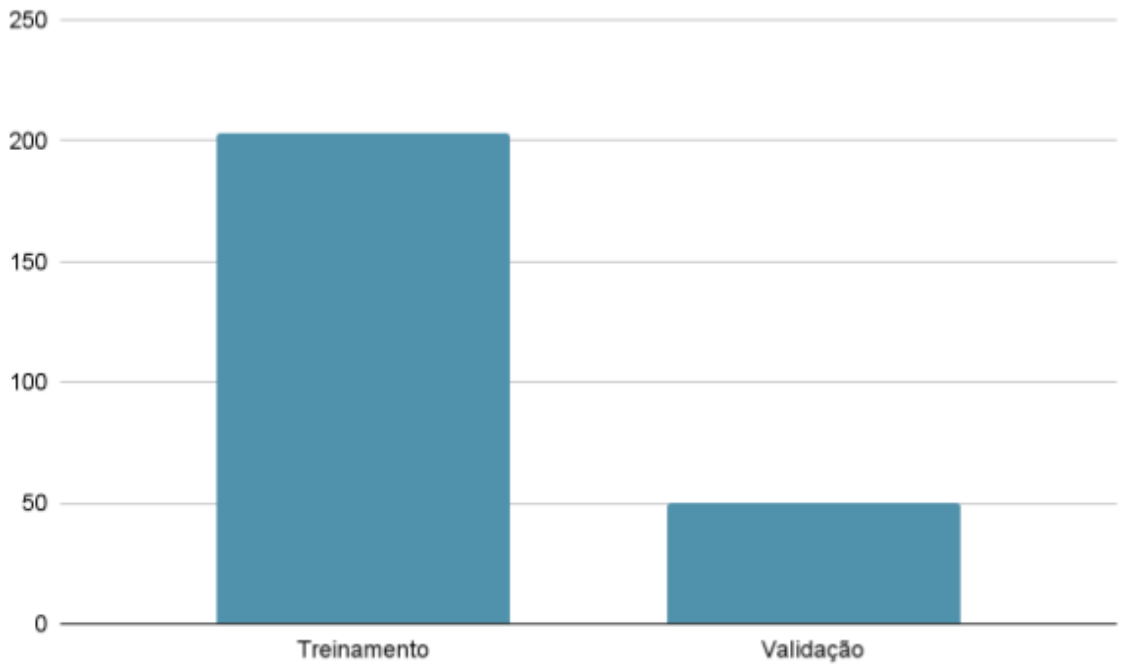
Seguindo com a etapa de pré-processamento, foi utilizada a função *tf.keras.preprocessing.image-dataset-from-directory()* da biblioteca *TesorFlow.keras*. Ela foi responsável por realizar o redimensionamento das imagens, e o particionamento do banco de dados em, dados para teste e dados para treinamento e o *one hot encoding*. As constantes definidas anteriormente foram

empregadas na configuração desta função.

No particionamento foi selecionado 80% das imagens para treinamento e 20% para teste. Existe uma convenção comum em casos de RNC como este, que o banco de dados é subdividido com essas porcentagens 80/20. Um ponto positivo de seguir essa convenção, seria que os resultados obtidos nesse treinamento podem ser facilmente comparado com diferentes estudo e experimentos, vendo que existe uma compatibilidade da estrutura de treinamento.

A estratégia *one hot encoding* citada acima transforma as labels categóricas "yes" e "no" em valores numéricos, para facilitar o treinamento do modelo. O formato de saída do dado é do tipo *tf.data.Dataset*, cada elemento dessa classe é composto pela imagem e pelo rótulo, e esse tipo de dado equivale ao tipo usado na função de treinamento *model.fit()*.

Figura 12 – Representação Divisão *Dataset* nos Subgrupos Treinamento e Validação



Fonte: Autoria Própria

3.4 MODELOS RNC APLICADOS NO TRABALHO

Nesse trabalho foi utilizada a ferramenta *transfer learning*, visando contornar a problemática do banco de dados reduzido, possuindo apenas 253 imagens, além da possível complicação de demanda de recursos, que seria necessário caso fosse ser treinado o modelo de aprendizado profundo do zero.

A metodologia do *transfer learning*, como já abordada com mais detalhes na seção 2.4, permite a reutilização dos pesos de modelos pré-treinados. Neste trabalho foi escolhido para uso

o modelo VGG19, pré-treinado com o banco ImageNet. A arquitetura do VGG19 foi descrita na seção 2.4.1.

Para utilizar o modelo escolhido, é necessário fazer o *download* dos seus pesos. Isso foi realizado aqui, através da função *keras.applications.vgg19.VGG19()*, que possui alguns parâmetros fundamentais para a aplicação: *input shape*, *include top* e *weights*. Para o *input shape*, foi definido o valor $(256, 256, 3)$, que se alinha com o valor da constante **image-size** de $256x256$ pixels. O parâmetro *include top* foi configurado como "False", removendo a camada de classificação originalmente utilizada no *dataset ImageNet* e permitindo a substituição dessa camada por uma nova, treinada para a classificação das labels do *dataset* de tumores. Já o parâmetro *weights* define com qual banco de dados o modelo escolhido será pré-treinado, foi selecionado o ImageNet.

Na sequência da construção do modelo, foi utilizada a camada de *Global Average Pooling*, cujo papel é preparar o tensor de saída da VGG19 para a camada densa a ser adicionada. Com o parâmetro *include top* definido como "False", o modelo mantém apenas as camadas convolucionais da VGG19, resultando em um tensor de saída tridimensional. A camada de *Global Average Pooling* realiza uma média das características, transformando-as em um vetor adequado para a etapa de classificação final.

Após extrair o mapa de características e redimensionar a saída, o próximo passo nesse sistema é passar pela função de ativação ReLU. O objetivo de acrescentar uma função de ativação é que só a operação de convolução em si não entrega a capacidade de aprender a discernir padrões complexos. A operação convolucional é linear, como pode ser vista na figura 5, sendo necessária a adição de uma não-linearidade para melhor representar situações complexas. Em específico a função ReLU, como explicada na seção 2.3.1, irá evitar problemas de saturação do gradiente.

Entre as camadas da função ReLU, que foram duas, foram aplicadas camadas de Dropout. O método Dropout tem o objetivo de regularizar o modelo, desativando de forma aleatória uma fração de neurônios. É atribuído um parâmetro para a função Dropout que representa a taxa de regularização do modelo. O método foi utilizado considerando uma taxa de 0,35, definida de forma empírica com base na acurácia alcançada na validação do modelo. Esse processo evita um "vício" nos pesos para que o modelo em questão não se adequa de forma exacerbada aos dados de treinamento e no momento que for validado com novos dados não obtenha sucesso na classificação. Esse processo explanado, conhecido como *"Overfitting"*, foi abordado na seção 2.3.3. Também é importante deixar claro que o dropout é somente ativado na etapa de treinamento. Durante a validação todos os neurônios da rede estarão ativos.

De uma forma simplificada, acima foram descritas as camadas do *transfer learning* com VGG19 assim como as camadas de "adaptação" do VGG19 anteriormente treinado com o ImageNet para o *dataset* aqui utilizado. Para finalizar a descrição do modelo, para a última camada densa foi escolhida a função *Sigmoid*, 2.3.2. Comumente empregada em modelos de classificação binária, a função *Sigmoid* irá converter as saídas das previsões das camadas anteriores em um valor no intervalo de 0 e 1.

A figura 13 abaixo representa um sumário feito com a função *Summary()* que exibe todas as camadas do modelo após o encaixe com o VGG19.

Figura 13 – Sumario arquitetura modelo RNC com VGG19

Layer (type)	Output Shape	Param #	Connected to
input_layer_1 (InputLayer)	(None, 256, 256, 3)	0	–
get_item (GetItem)	(None, 256, 256)	0	input_layer_1[0][0]
get_item_1 (GetItem)	(None, 256, 256)	0	input_layer_1[0][0]
get_item_2 (GetItem)	(None, 256, 256)	0	input_layer_1[0][0]
stack (Stack)	(None, 256, 256, 3)	0	get_item[0][0], get_item_1[0][0], get_item_2[0][0]
add (Add)	(None, 256, 256, 3)	0	stack[0][0]
vgg19 (Functional)	(None, 8, 8, 512)	20,024,384	add[0][0]
global_average_pooling2d (GlobalAveragePooling2D)	(None, 512)	0	vgg19[0][0]
dense (Dense)	(None, 128)	65,664	global_average_poolin…
dropout (Dropout)	(None, 128)	0	dense[0][0]
dense_1 (Dense)	(None, 64)	8,256	dropout[0][0]
dropout_1 (Dropout)	(None, 64)	0	dense_1[0][0]
dense_2 (Dense)	(None, 1)	65	dropout_1[0][0]

Total params: 20,098,369 (76.67 MB)
Trainable params: 73,985 (289.00 KB)
Non-trainable params: 20,024,384 (76.39 MB)

Fonte: Autoria Própria

Após a definição das camadas foi compilado o modelo. A compilação é a etapa onde serão definidas as configurações de treinamento, além da seleção do método utilizado para fazer a otimização e cálculo da perda. Também é definida a forma de métrica, que neste trabalho foi selecionada a acurácia, será falado mais sobre na seção 4 análises de validação.

Depois da preparação da RNC foi aplicado o *dataset* escolhido pela função *model.fit*, onde a RNC será treinada, terá seus mapas de características calculados após a passagem das matrizes *kernel* pelas imagens, seus pesos calculados com base nos gradientes informados pela função de otimização e de perda. Esse processo irá ocorrer de acordo com o número de épocas selecionado, que foi 30.

As etapas de retro programação e cálculo de erro são o que garante o aprendizado do algoritmo. Esses dois processos trabalham em conjunto, ajustando os pesos e melhorando a acurácia a cada época. No cálculo de erro comparamos o valor da predição com o valor da label, em um treinamento de uma RNC com aprendizado supervisionado, onde o dataset de treinamento deve ser catalogado. Para fazer essa comparação podemos escolher na configuração qual função de cálculo de erro vamos utilizar, ou *loss function*. Neste trabalho optamos pela

BinaryCrossentropy() da biblioteca Keras.io. Essa função é aplicada para casos de classificação binária, avaliando a diferença entre os valores de predição e de label a partir da equação abaixo:

$$loss = -\frac{1}{N}\sum_{i=1}^{N}[y_i \cdot log(p_i) + (1 - y_i) \cdot log(1 - p_i)] \tag{3}$$

- y_i é o valor da label (0 ou 1) da amostra i.
- p_i é a previsão em probabilidade da label.
- N é o número de amostras.

A primeira parte do somatório $y_i \cdot log(p_i)$ calcula o valor da penalidade quando a label é 1, com base na previsão. E a segunda parte $(1 - y_i) \cdot log(1 - p_i)$ faz o mesmo cálculo para quando a label é 0.

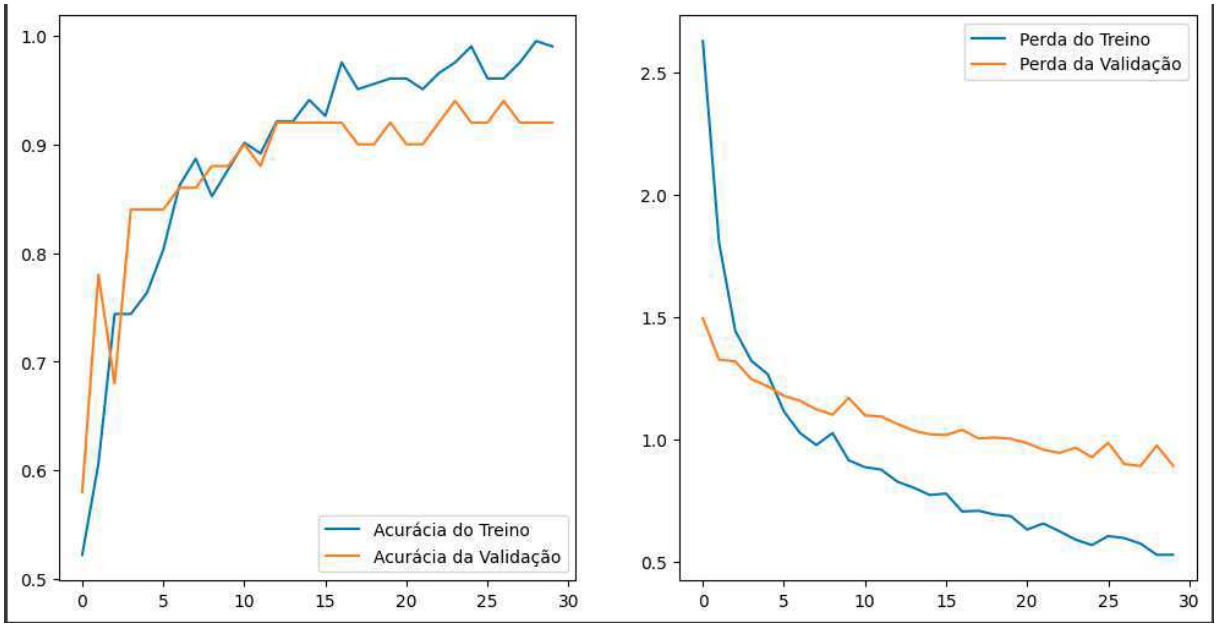
E para o cálculo da retro programação, ou *backpropagation*, onde de fato serão ajustados os pesos com os valores que obtivemos na função *loss* para otimização da acurácia, aqui usamos a função de otimização Adam(1e-3), também da biblioteca Keras.io. Ela trabalha adaptando individualmente o *learning rate* de cada parâmetro, operando de forma eficaz em dados com a presença de ruídos. Essa etapa de retro programação será a fase onde vamos ajustar os nossos pesos para minimizar o valor de *loss* que obtivemos com a função de perda, e para nos guiar em qual direção vamos executar esse ajuste utilizamos os gradientes calculados a partir da função de otimização.

Para finalizar essa breve explicação sobre CNN será abordada a etapa de inferência, onde os dados que ainda não são conhecidos pela RNC que foi treinada, para avaliar se a acurácia alcançada ao final do treinamento que passou por todas as épocas definidas, está realmente condizente com a realidade, ou se o algoritmo apenas memorizou as imagens e suas labels, diagnosticando um *overfiting*.

4 ANÁLISE E DISCUSSÃO DOS RESULTADOS

Neste capítulo serão abordadas as principais métricas e métodos utilizados para comprovar e validar o modelo apresentado na seção 3. No momento, o modelo já passou pelas etapas de treinamento e é avaliada a qualidade de sua resposta diante de dados que são desconhecidos para ele até o momento. Foi mencionado na seção 3.3, que durante a etapa de pré-processamento dos dados o *dataset* foi subdividido em dois grupos o grupo de *dados de treinamento* forneceu os valores de *acurácia do treino* e *perda do treino*, que refletem o desempenho do algoritmo apenas com relação às imagens que ele está tendo contato de forma repetida, mais especificamente por 30 vezes consecutivas, que equivale ao valor de *épocas*. Aqui serão exibidos em contraste com esses resultados os valores de *acurácia de validação* e *perda de validação*, calculados em cima dos *dados de validação*, que como mencionados anteriormente, representam 20% do *dataset*.

Figura 14 – Gráficos com curva de acurácia e perda nos cenários de treino e validação - 30 épocas



Fonte: Autoria Própria

Foram utilizados acima duas métricas para analisar o desempenho: *acurácia* e *perda*. Segundo a fonte (DEEPAI, 2021), a *acurácia* quantifica o número de previsões corretas com relação ao total de previsões realizadas. Ela é definida em conjunto com a *taxa de erro*, que quantifica o valor de previsões incorretas. A fórmula matemática para calcular ambas é:

$$taxa\_de\_erro = \frac{(FP + FN)}{(PV + NV + FP + FN)} \tag{4}$$

$$acuracia = 1 - taxa\_de\_erro \tag{5}$$

- FP número de previsões com resultados falso positivo.
- FN número de previsões com resultados falso negativo.
- PV número de previsões com resultados positivo verdadeiro.
- NV número de previsões com resultados negativo verdadeiro.

Já a *Perda* é calculada pela *função de perda*, ela representa o quanto o modelo está desajustado em relação aos dados. Neste trabalho foi utilizada a *função de perda BinaryCrossentropy* da *Keras*, na seção 3 foi abordada a sua representação matemática e mais detalhes sobre seu funcionamento em si.

Fazendo uma análise dos gráficos na figura 14, é possível visualizar um comportamento similar entre as curvas de treinamento e de validação. Esse ponto ressalta que não ocorreu *Overfitting* no algoritmo.

Figura 15 – Tabela com valores das 5 ultimas épocas

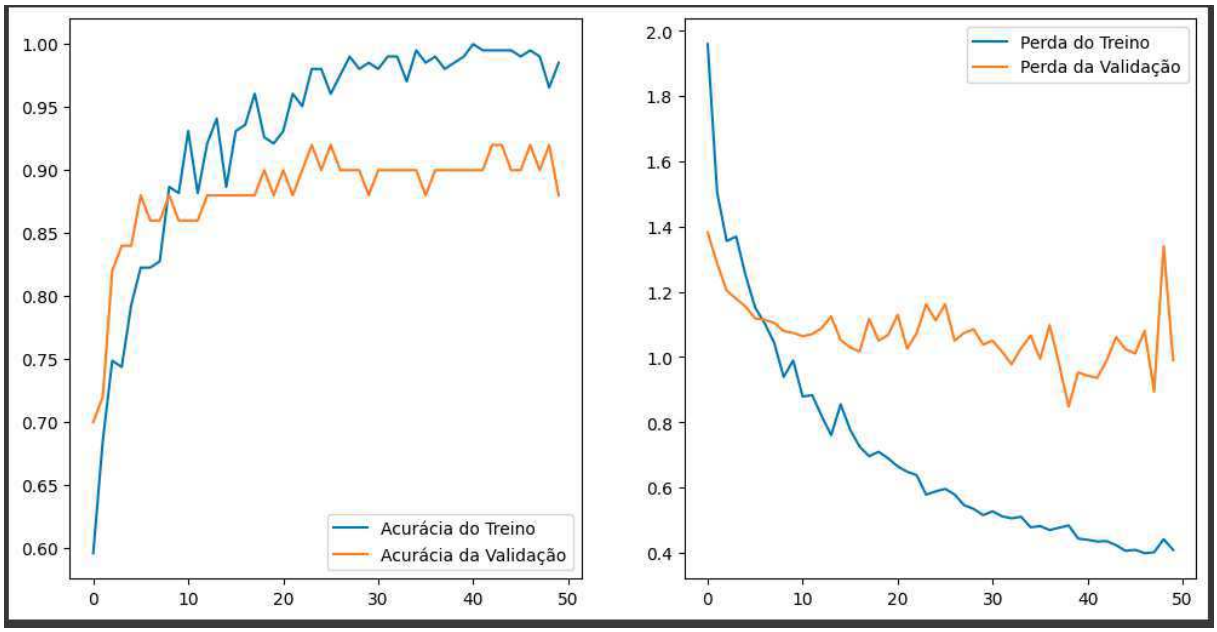
Época	Acurácia Treino	Acurácia Validação	Perda Treino	Perda Validação
30	0.9937	0.9034	0.0629	0.3147
29	0.9937	0.8966	0.0659	0.3187
28	0.9897	0.8966	0.0729	0.3227
27	0.9857	0.8966	0.0800	0.3307
26	0.9818	0.8966	0.0870	0.3387

Fonte: Autoria Própria

Apesar do banco de dados com um volume pequeno de dados, totalizando apenas 253 imagens, foi possível alcançar uma alta acurácia mesmo no banco de validação. Uma das principais técnicas responsáveis por esse resultado foi o *aprendizado por transferência*, como mencionado na seção 3 e 2.4.

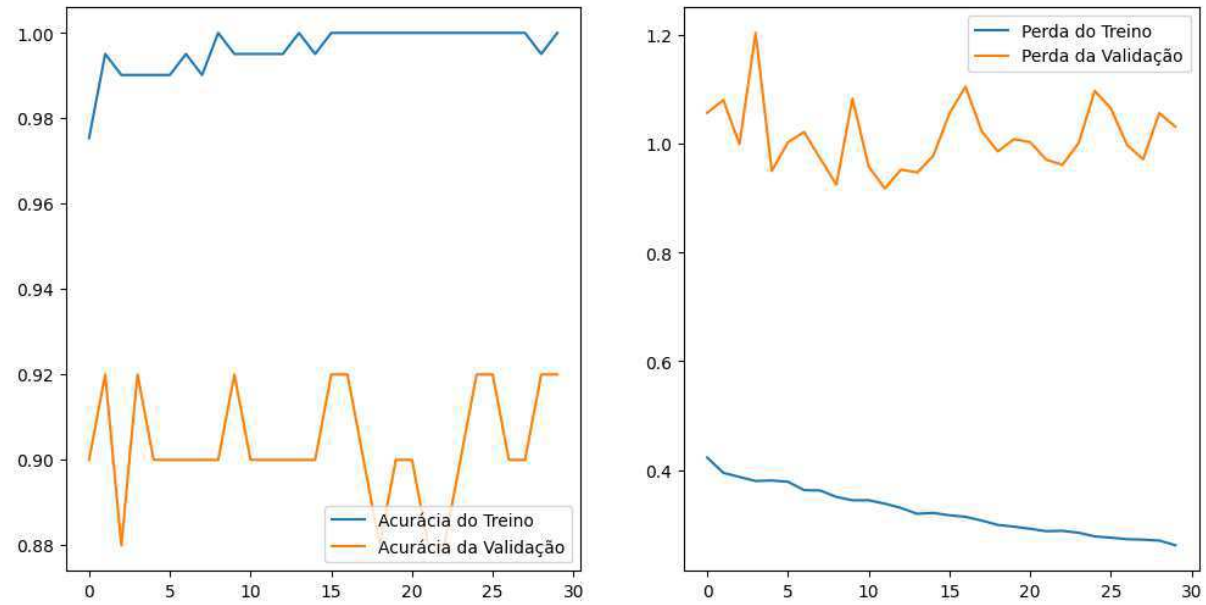
Outro ponto a ser analisado que é uma consequência negativa do banco de dados reduzido, é a instabilidade no gráfico da figura 14. Para problemas de inconsistência na curva de acurácia é recomendado o aumento do número de épocas. Nos gráficos abaixo foi elevado o valor de época para 50 e logo em seguida treinado com mais 30 épocas, totalizando 80 épocas.

Figura 16 – Gráficos com curva de acurácia e perda nos cenários de treino e validação - 50 épocas



Fonte: Autoria Própria

Figura 17 – Gráficos com curva de acurácia e perda nos cenários de treino e validação - 80 épocas



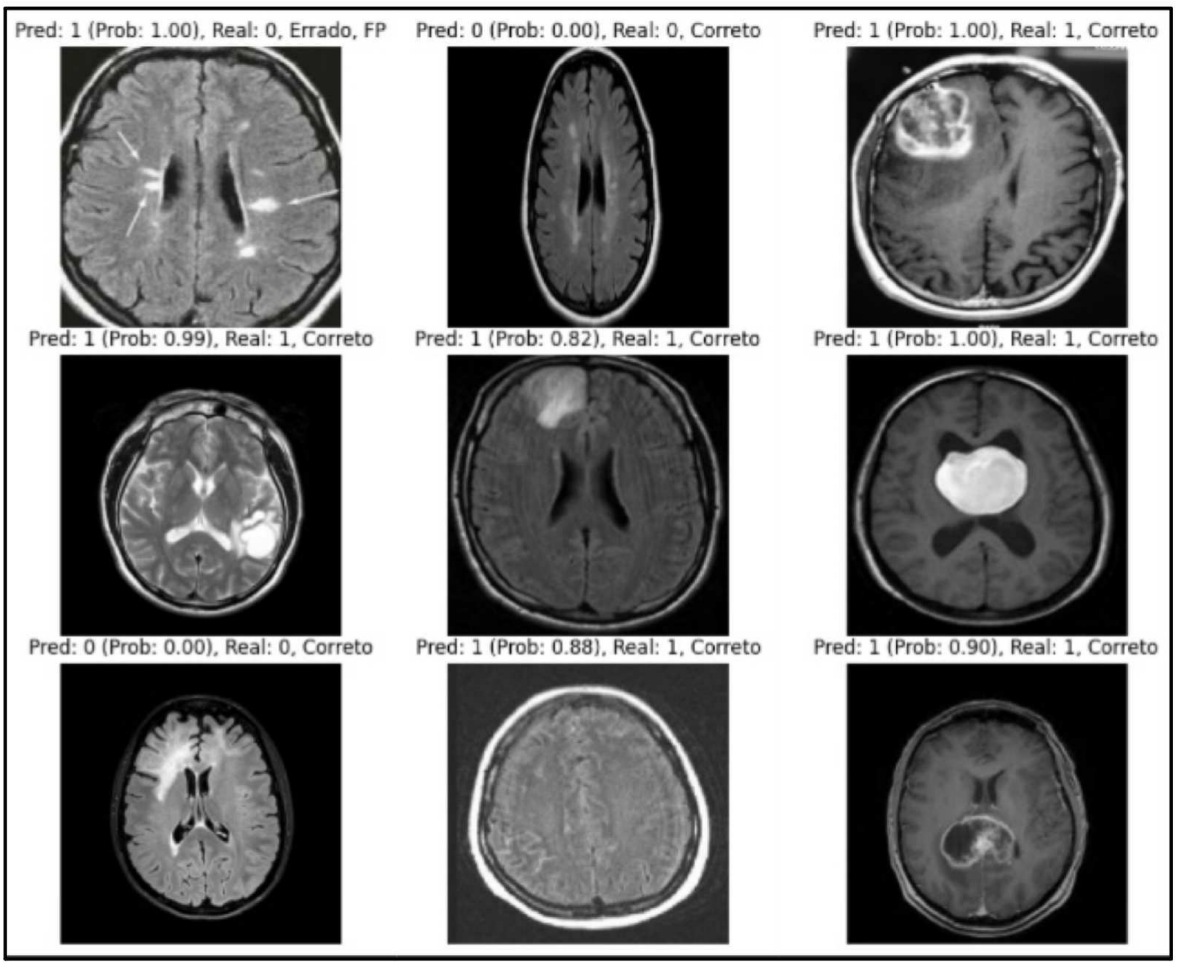
Fonte: Autoria Própria

Adicionar mais camadas de treinamento na RCN não gerou o efeito esperado devido ao volume do *dataset*. Não foram obtidas melhorias na acurácia da validação, apenas na acurácia de treino, e a distância entre essas duas curvas explana que a RNC está super adaptada ao banco

de dados, apesar de uma acurácia de treino boa quando aplicada com imagens novas em uma predição sua performance vai ser baixa.

Segundo o artigo (SIDEY-GIBBONS; SIDEY-GIBBONS, 2019), um ponto que deve ser priorizado ao analisar um algoritmo de AM para aplicação em um contexto médico, é a confiabilidade e precisão, sendo interessante subdividir as previsões que não obtiveram sucesso em Falso Positivo e Falso Negativo. A figura 18 contém um plot de uma amostra do banco de validação após sua previsão com os resultados e labels.

Figura 18 – Imagens de validação com label e previsão



Fonte: Autoria Própria

A figura do canto superior esquerdo é um Falso Positivo, no *dataset* ela está catalogada como que não possui a presença de tumores, e a RNC identificou que sim havia um tumor na imagem. Para compreender melhor as características que foram consideradas pela RNC indicativos do diagnóstico de tumor foi aplicada a figura 19a no algoritmo de Grad-CAM ref.

Figura 19 – Técnica Grad-CAM

(a) Imagem com label "sem tumor", com previsão FP

Fonte: Autoria Própria

(b) Imagem com resultado FP, e com técnica Grad-CAM

Fonte: Autoria Própria

Na seção 2.5 foi abordada a técnica do *Grad-CAM* com mais profundidade. Na figura 19b é sobreposto um mapa de calor por cima da imagem que foi avaliada na predição, as cores mais quentes informam que no processo de convolução foi identificado que essa região continham características que apontam para o resultado positivo para tumor.

Para facilitar a implementação da técnica do *Grad-CAM*, o modelo aplicado na classificação foi salvo, com todos os seus pesos e gradientes, e separadamente aplicado na arquitetura do *Grad-CAM*. Na seção 2.5 foi explanado que para realizar o mapa de calor é necessário os valores utilizados na classificação, para que o mapa reflita o comportamento do algoritmo, destacando as características (pixels) que levaram a tomada de decisão. Porém, quando aplicados no mesmo momento no código, o algoritmo se torna mais confuso e de difícil análise, no apêndice B é possível observar essa composição.

5 CONCLUSÃO

Este trabalho abordou a análise de imagens de RMC em um classificador binário, para determinação de presença ou ausência de tumor, fazendo uso de RNC.

Durante o trabalho de conclusão de curso todas as etapas foram contempladas, desde o tratamento das imagens, a modulação e treinamento do modelo, até a avaliação da sua acurácia.

A acurácia alcançada no modelo foi de 0.99 em treino e 0.90 em validação. Mesmo com um valor considerado alto, foi abordado formas de se otimizar o modelo, e discutido problemas que foram evitados com técnicas específicas durante a modelagem.

Um dos principais aspectos a ser analisado é o impacto da abordagem de Transferência de Aprendizado, que possibilitou alcançar alta acurácia mesmo com um banco de dados de tamanho modesto. Nesse contexto, destaca-se também a escolha do modelo VGG19. Conforme mencionado na seção 2.4.1, o modelo VGG é amplamente utilizado em estudos de reconhecimento de imagens, sendo as versões VGG16 e VGG19 as mais recentes. Neste trabalho, optou-se pelo VGG19, por possuir mais camadas convolucionais, o que permite uma definição mais precisa e detalhada das características do conjunto de dados.

Apesar do modelo ter sido construído com o auxílio de bibliotecas que disponibilizam funções matemáticas prontas e construídas para uma aplicação direta, neste trabalho foi exposto os cálculos e fundamentações que embasaram essas funções.

O algoritmo aqui desenvolvido pode ser utilizado como ponto de partida para várias linhas de pesquisa. Por identificar em formato de *batch* se uma RMC possui ou não tumor, ele pode ser incluído em pesquisas que trabalham com um grande número de dados e que precisam ser catalogados. A partir dessa identificação, é possível realizar vários estudos em cima da informação. Utilizar um algoritmo para alimentar com seus resultados outro sistema de aprendizado de máquina, que pode trabalhar a área de delimitação do tumor, ou seu tipo, entre várias outras linhas de pensamento.

Gostaria também de ressaltar que um dos principais objetivos deste trabalho é servir de incentivo para outros alunos. Todo o material utilizado aqui é acessível e gratuito, e iniciações em projetos pode ser um fator decisivo, um porta de estímulo para permanência em cursos que exigem grande disponibilidade e comprometimento, como este.

Referências

BARBOSA, L. **Deep Learning, Visão Computacional, Redes Neurais Convolucionais**. 2019. Medium. Acessado em: 05 de outubro de 2024. Disponível em: <https://medium.com/@lucaaslb/deep-learning-visÃčo-computacional-redes-neurais-convolucionais-c21f19f5ec34>. Citado na página 8.

BELLIS, M. **A Guide to Magnetic Resonance Imaging (MRI)**. 2019. Acessado em: 30 de Setembro de 2024. Disponível em: <https://www.thoughtco.com/magnetic-resonance-imaging-mri-1992133>. Citado na página 3.

BLOCH, F. Nuclear induction. **Physical review**, APS, v. 70, n. 7-8, p. 460, 1946. Citado na página 3.

CAMPBELL, M.; HOANE, A. J.; HSU, F.-H. Deep blue. **Artificial Intelligence**, Elsevier, v. 134, n. 1-2, p. 57–83, 2002. Citado na página 5.

CHAKRABARTY, N. **Brain MRI Images for Brain Tumor Detection**. 2019. Acessado em: 29 de Setembro de 2024. Disponível em: <https://www.kaggle.com/datasets/navoneel/brain-mri-images-for-brain-tumor-detection>. Citado 6 vezes nas páginas 7, 15, 16, 17, 31 e 36.

CHOLLET, F. **Deep Learning with Python**. Shelter Island, NY: Manning Publications Co., 2017. Citado na página 5.

DEEPAI. **Accuracy (error rate)**. 2021. Acessado em: 4 de Outubro de 2024. Disponível em: <https://deepai.org/machine-learning-glossary-and-terms/accuracy-error-rate>. Citado na página 22.

DEEPAK, S.; AMEER, P. Brain tumor classification using deep cnn features via transfer learning. **Computers in biology and medicine**, Elsevier, v. 111, p. 103345, 2019. Citado na página 12.

GéRON, A. **Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems**. [S.l.]: O'Reilly Media, 2022. Citado 2 vezes nas páginas 5 e 10.

HU, J.; GU, X.; GU, X. Dual-pathway densenets with fully lateral connections for multimodal brain tumor segmentation. **International Journal of Imaging Systems and Technology**, v. 31, 08 2020. Citado 2 vezes nas páginas 4 e 5.

KERAS, T. **Dropout**. 2023. Acessado em: 23 de Setembro de 2024. Disponível em: <https://keras.io/api/layers/regularization_layers/dropout/>. Citado na página 9.

LECUN, Y. et al. Gradient-based learning applied to document recognition. **Proceedings of the IEEE**, IEEE, v. 86, n. 11, p. 2278–2324, 1998. Citado na página 6.

MAZZOLA, A. A. Ressonância magnética: princípios de formação da imagem e aplicações em imagem funcional. **Revista Brasileira de Física Médica**, v. 3, n. 1, p. 117–129, out. 2015. Disponível em: <https://www.rbfm.org.br/rbfm/article/view/51>. Citado 2 vezes nas páginas 3 e 4.

PURCELL, E. M.; TORREY, H. C.; POUND, R. V. Resonance absorption by nuclear magnetic moments in a solid. **Physical review**, APS, v. 69, n. 1-2, p. 37, 1946. Citado na página 3.

SELVARAJU, R. R. et al. Grad-cam: visual explanations from deep networks via gradient-based localization. **International journal of computer vision**, Springer, v. 128, p. 336–359, 2020. Citado 2 vezes nas páginas 11 e 12.

SIDEY-GIBBONS, J. A.; SIDEY-GIBBONS, C. J. Machine learning in medicine: a practical introduction. **BMC medical research methodology**, Springer, v. 19, p. 1–18, 2019. Citado na página 25.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. **arXiv preprint arXiv:1409.1556**, 2014. Citado na página 10.

SRIVASTAVA GEOFFREY HINTON, A. K. I. S. R. S. N. Dropout: A simple way to prevent neural networks from overfitting. **Journal of Machine Learning Research**, v. 15, n. 1, p. 30, 2014. Citado na página 9.

TAJBAKHSH, N. et al. Convolutional neural networks for medical image analysis: Full training or fine tuning? **IEEE Transactions on Medical Imaging**, Institute of Electrical and Electronics Engineers (IEEE), v. 35, n. 5, p. 1299–1312, maio 2016. ISSN 1558-254X. Disponível em: <http://dx.doi.org/10.1109/TMI.2016.2535302>. Citado na página 13.

TURING, A. M. Computing machinery and intelligence. **Mind**, v. 59, n. 236, p. 433–460, 1950. Citado na página 5.

Apêndices

APÊNDICE A – Código modelagem RNC com VGG19

Esse código foi executado e desenvolvido no *GoogleColab*. Antes de executá-lo é necessário fazer download como *.zip* do *dataset* (CHAKRABARTY, 2019), e *upload* para o *notebook* do *GoogleColab*. No capítulo 3, foram citadas as especificações utilizadas para configuração do *notebook*.

```
import numpy as np
import pandas as pd
import os
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
import matplotlib.pyplot as plt
from tensorflow.keras import regularizers

# os arquivos não ficam de forma permanente no google colab, logo é nec
# caso voce queira anexar sua conta do drive ao colab tambem é possivel

# from google.colab import drive
# drive.mount('/content/drive')
# !unzip /content/drive/MyDrive/brain_tumor_dataset.zip
!unzip /content/brain_tumor_dataset.zip



diretorio = '/content/brain_tumor_dataset'
dir_yes = '/content/brain_tumor_dataset/yes'
dir_no  = '/content/brain_tumor_dataset/no'

SIZE=256
image_size = (SIZE, SIZE) #simboliza que a imagem será 256 pixels x 256
batch_size = 32

train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    diretorio,
    validation_split=0.2, #fração dos dados que será usada pra validaçã
    subset="training",
```

```
    seed=1337, #número aleatorio q garante que vamos poder reproduzir e
    image_size=image_size,
    batch_size=batch_size,
)
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    diretorio,
    validation_split=0.2,
    subset="validation",
    seed=1337,
    image_size=image_size,
    batch_size=batch_size,
)

base_model = keras.applications.vgg19.VGG19(input_shape=(SIZE,SIZE,3),i
base_model.trainable = False
base_model.summary()

inputs = keras.Input(shape=(SIZE,SIZE,3))
x = keras.applications.vgg19.preprocess_input(inputs)
x = base_model(x, training=False)
x = layers.GlobalAveragePooling2D()(x)
x = layers.Dense(128, activation='relu',kernel_regularizer=regularizers
x = layers.Dropout(0.35)(x)
x = layers.Dense(64, activation='relu',kernel_regularizer=regularizers.
x = layers.Dropout(0.35)(x)

outputs = layers.Dense(1, activation='sigmoid')(x)

model_com_vgg19 = keras.Model(inputs, outputs)
model_com_vgg19.summary()

model_com_vgg19.compile(
    optimizer=keras.optimizers.Adam(1e-3),
    loss="binary_crossentropy",
    metrics=["accuracy"],
)

epochs = 30
history_com_vgg19 = model_com_vgg19.fit(
```

```
    train_ds,
    epochs=epochs,
    validation_data=val_ds
)

def plota_resultados(history,epochs):
  acc = history.history['accuracy']
  val_acc = history.history['val_accuracy']

  loss = history.history['loss']
  val_loss = history.history['val_loss']

  intervalo_epochs = range(epochs)
  plt.figure(figsize=(12, 6))
  plt.subplot(1, 2, 1)
  plt.plot(intervalo_epochs, acc, label='Acurácia do Treino')
  plt.plot(intervalo_epochs, val_acc, label='Acurácia da Validação')
  plt.legend(loc='lower right')


  plt.subplot(1, 2, 2)
  plt.plot(intervalo_epochs, loss, label='Perda do Treino')
  plt.plot(intervalo_epochs, val_loss, label='Perda da Validação')
  plt.legend(loc='upper right')
  plt.show()

plota_resultados(history_com_vgg19,epochs)

#Salvando o modelo para utilizar depois na aplicação do Grad-CAM
model_com_vgg19.save('meu_modelo_com_otimizador_2.keras', include_optim

import matplotlib.pyplot as plt
import numpy as np

# Definir o limiar para predição binária
threshold = 0.5

# Função para exibir as imagens e comparar a label real com a predição
def show_predictions(val_ds, model):
```

```
for images, labels in val_ds.take(1):  # Pega x batchs de imagens e
    predictions = model.predict(images)  # Faz as predições com o m

    # Calcula o número de imagens no batch
    num_images = len(images)

    # Define o layout: número de colunas e linhas ajustado dinamica
    cols = 3
    rows = (num_images // cols) + (num_images % cols > 0)  # Adicio

    # Aumenta o tamanho da figura para ampliar as imagens
    plt.figure(figsize=(30, 30))  # Ajuste o tamanho da figura (20x

    # Loop para exibir as imagens e os resultados
    for i in range(9):
        ax = plt.subplot(rows, cols, i + 1)  # Criar o subplot de a
        plt.imshow(images[i].numpy().astype("uint8"))  # Exibe a im

        # Predição com base no limiar
        pred_prob = predictions[i][0]
        pred_class = 1 if pred_prob > threshold else 0  # 1 para Tu

        # Comparar predição com o rótulo real
        actual_class = labels[i].numpy()

        # Determinar se a predição está correta
        result = "Correto" if pred_class == actual_class else "Erra

        if result == "Errado":
         result += (", FN" if actual_class == 1 else ", FP")



        # Exibir a predição, rótulo real e se foi correto ou errado
        plt.title(f"Pred: {pred_class} (Prob: {pred_prob:.2f}), Rea
        plt.axis("off")  # Remove os eixos

    # Ajustar o layout para evitar sobreposição
    plt.tight_layout()
    plt.subplots_adjust(wspace=0.1, hspace=0.1)
```

```
        plt.show()  # Mostra as imagens e as predições

# Exibir as predições comparadas com as labels reais
show_predictions(val_ds, model_com_vgg19)
```

APÊNDICE B – Código Análise Predição com Grad-CAM

Esse código foi executado e desenvolvido no *GoogleColab*. Antes de executá-lo é necessário fazer download como *.zip* do *dataset* (CHAKRABARTY, 2019) e do modelo que foi treinado no código apresentado no apêndice A, o formato de dado do modelo baixado é *.keras*. Após o download dos dados precisa ser feito o *upload* para o *notebook* do *GoogleColab*. No capítulo 3, foram citadas as especificações utilizadas para configuração do *notebook*.

```
import numpy as np
import pandas as pd
import os
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
import matplotlib.pyplot as plt
from tensorflow.keras import regularizers
from IPython.display import Image
import cv2

!unzip /content/brain_tumor_dataset.zip


diretorio = '/content/brain_tumor_dataset'
dir_yes = '/content/brain_tumor_dataset/yes'
dir_no  = '/content/brain_tumor_dataset/no'

#principais constantes

SIZE=256
IMAGE_SIZE = (SIZE, SIZE)
IMAGE_PATH = '/content/brain_tumor_dataset/yes/Y1.jpg'
BATCH_SIZE = 9
MODEL_PATH = "/content/meu_modelo_com_otimizador_2.keras"
LAST_CONV_LAYER = "block5_conv4"
LAST_CLASSIFIER_LAYER_NAME_FROM_VGG19 = "block5_pool"
LAST_CLASSIFIER_LAYER_NAMES = [
    'global_average_pooling2d_1',
```

```
    'flatten_1',
    'dense',
    'dropout',
    'dense_1',
    'dropout_1',
    'dense_2'
]
TOP_N = 8

train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    diretorio,
    validation_split=0.2,
    subset="training",
    seed=1337,
    image_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
)
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    diretorio,
    validation_split=0.2,
    subset="validation",
    seed=1337,
    image_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
)

display(Image(IMAGE_PATH))

# precisamos construir o image array para fazer o mapa do gradcam

def get_img_array(img_path, image_size):
    img = keras.preprocessing.image.load_img(img_path, target_size=imag
    array = keras.preprocessing.image.img_to_array(img)
    array = np.expand_dims(array, axis=0)
    return array

array = get_img_array(IMAGE_PATH, IMAGE_SIZE)
array.shape #(1, 256, 256, 3)
print(array.dtype)
```

```
#precisamos do modelo carregado

def get_model(model_path):
  model = tf.keras.models.load_model(model_path)

  return model

model = get_model(MODEL_PATH)
model.summary()
model.get_layer('vgg19').summary()

def get_top_predicted_indices(predictions, top_n):
    return np.argsort(-predictions).squeeze()[:top_n]

def make_gradcam_heatmap(
    img_array,
    model,
    last_conv_layer_name,
    last_classifier_layer_name_from_vgg19,
    classifier_layer_names,
    top_n
    ):
  model_vgg19 = model.get_layer('vgg19')
  last_conv_layer = model_vgg19.get_layer(last_conv_layer_name)
  last_conv_layer_model = keras.Model(model_vgg19.inputs, last_conv_lay

  classifier_input = keras.Input(shape=last_conv_layer.output.shape[1:]
  x = classifier_input
  x = model_vgg19.get_layer(last_classifier_layer_name_from_vgg19)(x)

  for layer_name in classifier_layer_names:
      x = model.get_layer(layer_name)(x)
  classifier_model = keras.Model(classifier_input, x)

  last_conv_layer_output = last_conv_layer_model(img_array)
  preds = classifier_model(last_conv_layer_output)
  class_indices = [1]
```

```
heatmaps = []

index = 0

with tf.GradientTape() as tape:
    last_conv_layer_output = last_conv_layer_model(img_array)
    tape.watch(last_conv_layer_output)

    preds = classifier_model(last_conv_layer_output)
    class_channel = preds[0]

grads = tape.gradient(
    class_channel,
    last_conv_layer_output
)


pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))
last_conv_layer_output = last_conv_layer_output.numpy()[0]
pooled_grads = pooled_grads.numpy()


for i in range(pooled_grads.shape[-1]):
    last_conv_layer_output[:, :, i] *= pooled_grads[i]


heatmap = np.mean(last_conv_layer_output, axis=-1)

heatmap = np.maximum(heatmap, 0) / np.max(heatmap)

return heatmap

heatmap = make_gradcam_heatmap(
  get_img_array(IMAGE_PATH, IMAGE_SIZE),
  get_model(MODEL_PATH),
  LAST_CONV_LAYER,
  LAST_CLASSIFIER_LAYER_NAME_FROM_VGG19,
  LAST_CLASSIFIER_LAYER_NAMES,
  TOP_N
```

```
    )

def superimpose_heatmap(image_path, heatmap):
    img = keras.preprocessing.image.load_img(image_path)
    img = keras.preprocessing.image.img_to_array(img)

    # We rescale heatmap to a range 0-255
    heatmap = np.uint8(255 * heatmap)
    heatmap = cv2.applyColorMap(heatmap, cv2.COLORMAP_JET)
    heatmap = keras.preprocessing.image.array_to_img(heatmap)
    heatmap = heatmap.resize((img.shape[1], img.shape[0]))

    heatmap = keras.preprocessing.image.img_to_array(heatmap)
    superimposed_img = cv2.addWeighted(heatmap, 0.4, img, 0.6, 0)
    superimposed_img = np.uint8(superimposed_img)

    return superimposed_img

superimpose_heatmap(IMAGE_PATH, heatmap)
```