Universidade de Brasília – UnB

Faculdade de Ciências e Tecnologias em Engenharia – FCTE

Engenharia de Software

Uso de aprendizado de máquina para detecção de glaucoma em exame de tomografia de mácula

Autor: Lorenzo de Lima Alves dos Santos

Orientador: Prof. Fabiano Araujo Soares, Dr.

Brasília, DF

2025

Lorenzo de Lima Alves dos Santos

Uso de aprendizado de máquina para detecção de glaucoma em exame de tomografia de mácula

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília – UnB

Faculdade de Ciências e Tecnologias em Engenharia – FCTE

Orientador: Prof. Fabiano Araujo Soares, Dr.

Brasília, DF

2025

Lorenzo de Lima Alves dos Santos

Uso de aprendizado de máquina para detecção de glaucoma em exame de tomografia de mácula/ Lorenzo de Lima Alves dos Santos. – Brasília, DF, 2025-

64 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Fabiano Araujo Soares, Dr.

Trabalho de Conclusão de Curso – Universidade de Brasília – UnB
Faculdade de Ciências e Tecnologias em Engenharia – FCTE , 2025.

1. Rede neural. 2. Aprendizado de máquina. I. Prof. Fabiano Araujo Soares, Dr.. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Uso de aprendizado de máquina para detecção de glaucoma em exame de tomografia de mácula

CDU 02:141:005.6

Lorenzo de Lima Alves dos Santos

Uso de aprendizado de máquina para detecção de glaucoma em exame de tomografia de mácula

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Prof. Fabiano Araujo Soares, Dr.
Orientador

Prof. Adson Ferreira da Rocha, Dr.
Convidado 1

Prof. Cristiano Jacques Miosso, Dr.
Convidado 2

Brasília, DF
2025

Este trabalho é dedicado a todos os alunos que, no meio de um mar de ansiedade e inseguranças, ousaram buscar mais.

Agradecimentos

Agradeço a meus pais o apoio e o suporte durante minha vida acadêmica; a meus amigos, os quais me ajudaram a descontrair e relaxar; aos meus colegas de curso, cujo apoio e parcimônia jamais serão esquecidos, e a meus professores e monitores, os quais me abriram as portas para um mundo de tecnologia desconhecido por mim.

"As pessoas costumam dizer que a motivação não dura sempre. Bem, nem o efeito do banho, por isso recomenda-se diariamente."

Zig Ziglar

Resumo

Glaucoma é uma condição médica caracterizada por lesões no nervo óptico, responsável por enviar os impulsos elétricos advindos da retina para o cérebro, causando diversos sintomas como dor, visão embaçada e até cegueira. O glaucoma é uma das principais causas de perda irreversível da visão em todo o mundo. Não existe cura para a doença, mas o tratamento, quando realizado nos estágios iniciais, permite o retardamento dos sintomas e uma boa qualidade de vida. Devido ao fato do diagnóstico estar atrelado ao processo de análise de tomografias por um especialista, tal procedimento poderia ser replicado por um computador treinado para detectar estágios iniciais de glaucoma por meio de imagens. Por isso, foi decidido utilizar a tecnologia das redes neurais, estruturas computacionais que, além de diversas outras capacidades, permitem que a máquina aprenda e detecte padrões em imagens, como foco neste trabalho. Na **introdução**, são mostradas as definições principais relacionadas ao glaucoma, o que é, como aparece, o que representa, por que se importar, e *Artificial Neural Networks* (ANNs) e suas definições relacionadas ao processo de aprendizagem e classificação. No **Referencial Teórico** são aprofundados os conceitos e técnicas, além de apresentar as funções de ativação e sua relevância, como e quando são melhores de usar. No capítulo de nome **Implementações e Avaliações Preliminares**, foi feito um exemplo de uma rede neural, usando da linguagem de programação Python, em funcionamento e foi observado que a taxa de aprendizado e número de iterações afetam consideravelmente a predição. É depois apresentado o capítulo **Método Proposto para Detecção de Glaucoma: Implementação e Avaliação**, o qual apresenta os resultados do estudo sobre tentar usar uma rede neural para realizar aprendizado e predição com imagens de pacientes e suspeitos de glaucoma,com parte dedicada a apresentar estudo feito. As **Considerações Finais** encerram o documento, abordando os principais pontos do trabalho.

Palavras-chave: Glaucoma; Redes neurais; ferramenta de auxílio a diagnóstico; Detecção de padrões; Aprendizado de máquina;

Abstract

Glaucoma is a medical condition that results in damage to the optic nerve, responsible for transmitting electrical signals from the retina to the brain, leading to symptoms including pain, blurred vision, and potential blindness. Glaucoma is one of the leading causes of irreversible vision loss worldwide. There is no cure for the disease, but treatment, when carried out in the early stages, can slow down the symptoms and ensure a good quality of life. As a specialist's analysis of CT scans to diagnose is a linked process, a computer trained to identify early glaucoma stages via images could replicate this procedure. Consequently, the decision was made to utilise neural network technology, which comprises computational structures capable of learning and detecting patterns in images, as the primary focus of this work. The **Introduction** gives a general overview of the main ideas connected to glaucoma, including its characteristics, symptoms, consequences, and importance, along with the key principles of Artificial Neural Networks (ANNs) related to learning and categorization processes. The **Theoretical Framework** delves deeper into the concepts and techniques, in addition to presenting the activation functions and their relevance and how and when they are best to use. In the chapter named **Implementations and Preliminary Evaluations**, an example of a neural network was created, using the Python programming language, in operation, and it was observed that the learning rate and number of iterations considerably affect the prediction. Next, the chapter **Proposed Method for Glaucoma Detection: Implementation and Evaluation** presents the results of the study on attempting to use a neural network to perform learning and prediction with images of patients and suspected glaucoma cases, with a section dedicated to presenting the study. The **Final Considerations** conclude the document, addressing the main points of the work.

Key-words: Glaucoma; Neural Network; diagnostic aid tool; Pattern Recognition; Machine Learning;

Lista de ilustrações

Figura 1 – Infográfico sobre o funcionamento do olho (linhas tracejadas representam a luz atravessando o cristalino e formando a imagem da árvore de cabeça para baixo na retina) - Fonte: biologianet.com. 23
Figura 2 – Comparação entre olho normal, à esquerda, e com glaucoma, à direita. - Fonte: Dataset Ocular Disease Recognition em kaggle.com 24
Figura 3 – Exame de OCT, com a imagem da mácula apresentada no canto superior esquerdo, a tomografia de nervo óptico na parte superior do centro do resultado, e as mediçoes de setores da retina e grossura ao redor na parte lateral direita e inferior esquerda - Fonte: Ortolan Oftalmologia. . 29
Figura 4 – Rede neural simplificada genérica para demonstração, com uma camada de entrada, uma camada interna que aplica pesos e vieses, e uma camada de saída, com cinco nós registrando valores de peso e erro final. 30
Figura 5 – Gráfico da função ReLU. 32
Figura 6 – Gráfico da função *softmax*. 33
Figura 7 – Gráfico da função sigmoide. 34
Figura 8 – Demonstração das imagens do dataset MNIST, com imagens de números escritos de forma cursiva em preto e branco. 36
Figura 9 – Representação visual do modelo VGG 16 com dezesseis camadas convolucionais seguidas de três camadas densas para processamento de pesos - fonte: (BOESCH, 2021) 42
Figura 10 – Representação visual do modelo ResNet50, representando o primeiro bloco com convolução, normalização, Função Relu e *max pool*, seguido por uma camada densa para saída - fonte: Medium.com 43
Figura 11 – Gráfico de acurácias do VGG 16 por dezenas de épocas. 45
Figura 12 – Gráfico de acurácias do ResNet50 por dezena de épocas. 45

Lista de tabelas

Tabela 1 – Medições de acurácia em função da taxa de aprendizagem com 100 iterações 39
Tabela 2 – Distribuição do dataset de treinamento. 42
Tabela 3 – Métricas para VGG 16 e ResNet50. 47
Tabela 4 – Matriz de confusão da arquitetura VGG 16 gerada com teste de 80 imagens do dataset. 47
Tabela 5 – Matriz de confusão da arquitetura ResNet50 gerada com teste de 80 imagens do dataset. 47

Lista de abreviaturas e siglas

Fig.	Figura
ReLU	*Rectified Linear Unit*
CNN	*Covolutional Neural Network*
OCT	Tomografia de Coerência Óptica
PIO	Pressão Intraocular
ANN	*Artifical Neural Network*
PNG	*Portable Network Graphic*
JPEG	*Joint Photographic Experts Group*
MNIST	*Modified National Institute of Standards and Technology*
TCC	*Trabalho de Conclusão de Curso*
VGG	*Visual Geometry Group*
API	*Application Programming Interface*
GPU	*Graphics Processing Unit*
α	Letra grega alfa
Σ	Somatório
TP	*True Positive*
FP	*False Positive*
TN	*True Negative*
FN	*False Negative*

Sumário

1 INTRODUÇÃO 23
1.1 Contextualização 23
1.2 Questão de Pesquisa 25
1.3 Justificativa 26
1.4 Objetivos e detalhamento 26
1.4.1 Objetivos 26
1.5 Materiais e Métodos 26

2 REFERENCIAL TEÓRICO 28
2.1 Tipos de glaucoma 28
2.2 Exame de Tomografia 28
2.3 Redes neurais 29
2.3.1 Tipos de redes neurais 31
2.3.2 Função ReLU 31
2.3.3 Função *softmax* 32
2.3.4 Função Sigmoide 33
2.4 Propagação de pesos e vieses 34
2.4.1 Propagação direta 34
2.4.2 Retropropagação do erro 34

3 IMPLEMENTAÇÕES E AVALIAÇÕES PRELIMINARES 36
3.1 Dataset MNIST e prova de conceito 36
3.2 Acurácia 38

4 MÉTODO PROPOSTO PARA DETECÇÃO DE GLAUCOMA: IMPLEMENTAÇÃO E AVALIAÇÃO 40
4.1 Metodologia 40
4.2 Requisitos 40
4.3 Aplicação e desenvolvimento 41
4.3.1 Amostra 41
4.3.2 Modelos 42
4.3.2.1 VGG 16 42
4.3.2.2 ResNet50 43
4.3.3 Ambiente e ferramentas 44
4.3.4 Limitações e ajustes 44
4.3.5 Medições 46

4.3.5.1 Resultados Obtidos 47
4.3.6 Discussão 47
4.3.6.1 Limitações do Dataset e Generalização 47
4.3.6.2 Infraestrutura Computacional e Ajustes de Treinamento 48
4.3.6.3 Análise dos Modelos e Overfitting 48
4.3.6.4 Avaliação de Métricas e Erros 48
4.3.6.5 Sugestões para Trabalhos Futuros 48
4.3.6.6 Conclusão da Discussão 49

5 CONSIDERAÇÕES FINAIS 50

REFERÊNCIAS 51

APÊNDICE A – PRIMEIRO APÊNDICE 54

APÊNDICE B – SEGUNDO APÊNDICE 58

APÊNDICE C – TERCEIRO APÊNDICE 61

APÊNDICE D – QUARTO APÊNDICE 64

1 Introdução

Este capítulo pretende abordar a patologia oftalmológica denominada como glaucoma e a tecnologia das redes neurais. Neste sentido, será conceituado o que é o glaucoma, a origem do seu nome, os motivos que permitem considerá-lo como uma patologia grave. Aliada a esta conceituação, o capítulo tratará da tecnologia das redes neurais e sua definição, o propósito a que se destina tal tecnologia, seu funcionamento, tipologias existentes e suas relevâncias, e de que maneira esse trabalho pretende entrelaçar essas duas áreas de conhecimento: saúde e tecnologia.

1.1 Contextualização

De acordo com a definição do Hospital Albert Einstein (BIONDI, 2022), glaucoma é uma doença ocular que se desenvolve devido a mudanças no nervo óptico, que o torna mais frágil. De forma simplificada, o olho humano é composto de estruturas que permitem a imagem vista se formar no fundo do olho e ser levada ao cérebro por meio do nervo óptico, no que é chamado de retina, como mostra a figura abaixo.

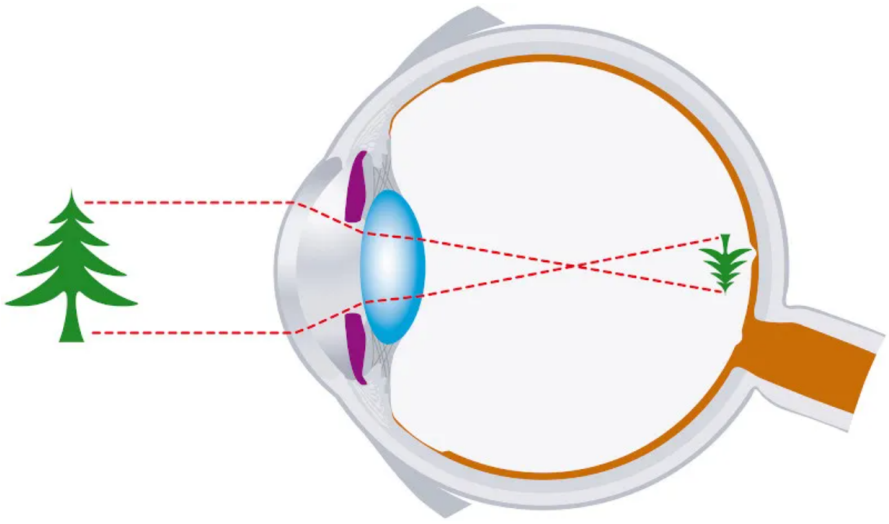

Figura 1 – Infográfico sobre o funcionamento do olho (linhas tracejadas representam a luz atravessando o cristalino e formando a imagem da árvore de cabeça para baixo na retina) - Fonte: biologianet.com.

Glaucoma, em si, significa "esverdeado"em grego antigo, e era chamado assim pois se dizia que a cor do olho mudava para tons da cor do mar, sintoma associado com estágios mais avançados da doença em que ocorre a cegueira. Devido a algum motivo ou causa, o nervo óptico sofre danos progressivos, fazendo com que as imagens captadas pela retina deixem de ser transmitidas ao cérebro de maneira completa (a partir da imagem na córnea

demonstrada na figura acima), e caso não seja tratado, o nervo pode ser danificado a ponto de se tornar inutilizável. Entre as causas associadas ao glaucoma estão fatores genéticos, como casos em que a criança nasce com a doença por herdar a condição da mãe durante a gestação, além de influências ambientais.

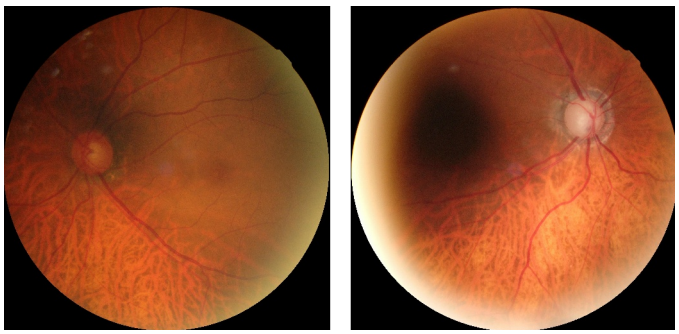

Figura 2 – Comparação entre olho normal, à esquerda, e com glaucoma, à direita. - Fonte: Dataset Ocular Disease Recognition em kaggle.com

A figura acima representa as máculas (termo usado em contexto médico que se refere à zona central da retina, onde a concentração de células cones é elevada) de uma pessoa de 31 anos diagnosticada com glaucoma no olho direito. Neste caso, detectar a existência de uma anormalidade com o fundo do olho direito é trivial, e caso o observador seja bem versado nos tipos de doença que afetam essa região e como se manifestam nessas imagens, até existe a possibilidade de reconhecer o mal como glaucoma. Entretanto, além de ser inviável financeiramente para uma clínica treinar uma pessoa só para isso, essa imagem pode ser afetada por diversas outras doenças, algumas delas como diabetes ou cataratas, dificultando enxergar detalhes na retina.

O glaucoma muitas vezes é relacionado ao aumento de pressão intraocular, e por mais que haja algum sentido por detrás, é errado afirmar que o primeiro é causado pelo segundo. Explicando de forma simples, o olho humano é um órgão que possui um líquido transparente, doravante chamado de humor aquoso, cuja drenagem ocorre pelo canal de Schlemm e a via uveoscleral (segundo o Manual Merck Sharp and Dohme (RHEE, 2023)). Caso o escoamento do humor não ocorra ou seja realizado de forma insuficiente, a pressão intraocular se elevará, o que pode resultar em lesões no nervo óptico, o que pode causar glaucoma. O diagnóstico muitas vezes é derivado de exames como tonometria (exame realizado para medir a Pressão Intraocular, ou PIO) e a Tomografia de Coerência Óptica (OCT), um exame que permite o oftalmologista (ou clínico) examinar a imagem da camada fibrosa do nervo da retina e procurar por deformações e/ou lesões.

Redes neurais são um tipo de modelo computacional relacionado a *Machine Learning* (aprendizado de máquina em inglês), isto é, um campo de estudo da área da computação voltado a ensinar máquinas a como aprender sem ser necessariamente programada explicitamente (BROWN, 2021). Uma das características extremamente interessante de

redes neurais é a capacidade de encontrarem padrões não lineares em um grupo de dados (um exemplo clássico é ter os dados como os *pixels* de uma imagem e o padrão não linear que se busca identificar sendo um gato ou cachorro na imagem), empregando de funções não lineares para realizar cálculos em cima dos dados recebidos. Como redes neurais artificiais são elaboradas para funcionarem de forma análoga ao cérebro, o elemento básico que compõe as redes, recebendo e transformando informações passadas por outro é chamado de neurônio (sendo o agrupamento chamado de camada ou *layer*), e uma rede pode ter vários destes apenas para receber a entrada de dados, com suas camadas internas possuindo ainda mais desses neurônios.

Como um grupo de neurônios pode conter mais informações relevantes que outras, dependendo do que está sendo analisado, é comum aplicar a esses neurônios um peso, e para permitir mais complexidade no cálculo, é adicionado a cada neurônio um valor de viés. Os valores de peso e viés são arbitrários, ou seja, não há uma regra para qual deve ou não ser, e os "melhores"serão aqueles que derem a permissão mais acurada possível sem superajustar às imagens de treino, e para descobrir esses valores ideais são feitos vários treinamentos com imagens cujo rótulo de predição já é conhecido, até que a taxa de acerto desejada seja encontrada. Este processo de tentativa e erro para encontrar os pesos e viés corretos de cada nódulo é o que se chama habitualmente de "treinar a rede".

O uso de ANNs para a classificação de imagens possui aplicações em outras áreas hospitalares, justamente pela facilitação de diagnósticos com taxas de erro mínimas. No entanto, quando se trata do uso desta tecnologia no contexto de detecção de glaucoma, o tamanho de literatura e de aplicações é ligeiramente menor, por mais que haja uma possibilidade de integração. Em um cenário ideal, uma rede neural devidamente treinada o suficiente permitiria a criação de uma rotina de pré-diagnóstico: o paciente teria suas imagens já coletadas periodicamente em exames de rotina analisada pelo modelo, classificada, e repassada para um especialista em casos positivos. Dessa forma, o tratamento poderá ser iniciado em estágios iniciais, aumentando a qualidade de vida do paciente a longo prazo.

1.2 Questão de Pesquisa

Levando-se em consideração os contextos apresentados acima, a questão levantada que motiva a execução deste trabalho é:

O quão viável é o usi de aprendizado de máquina para auxiliar no diagnóstico de glaucoma utilizando-se de imagens advindas de OCT, mais especificamente tomografia de mácula?

1.3 Justificativa

Em 2021, um estudo realizado pela Lancet Global Health (STEINMETZ et al., 2021) apontou que, no ano anterior a segunda maior causa de cegueira no mundo são as variações de glaucoma, e o diagnóstico em estágios iniciais permite uma arrefecimento na severidade das lesões no nervo, e como o aprendizado de máquina já é utilizado na área da saúde para classificação e diagnóstico, a utilização de uma ANN não é uma extrapolação das capacidades do algoritmo, mas sim uma possibilidade de exercitar os conhecimentos sobre a construção de redes neurais e tratamento de imagens, além de uma exposição da lógica por detrás desses algoritmos para aqueles interessados no tema.

1.4 Objetivos e detalhamento

1.4.1 Objetivos

Desenvolver uma rede neural com a capacidade de receber imagens de tomografias originalmente em formatos *Portable Network Graphic* (PNG) e *Joint Photographic Experts Group* (JPEG) que serão pré-processadas, tendo a capacidade de predizer com uma certa acurácia se aquela imagem é de um um paciente de glaucoma. O programa deverá ter múltiplas iterações para testes, estes realizados com um banco de imagens já classificado em participantes com glaucoma e participantes saudáveis, assim calculando os pesos e *bias* de cada neurônio que serão usados para a predição nos casos de validação.

1.5 Materiais e Métodos

A metodologia que será usada no contexto de pesquisa é uma versão adaptada de pesquisa exploratória,cuja função é preencher lacunas de informações que estão faltando em um estudo (SERASA, 2023). Existem diferentes formas de pesquisa exploratória categorizadas pela forma de obtenção de dados, mas levando em consideração o escopo do trabalho e a temática, a pesquisa realizada será uma mistura de bibliográfica, ou seja, comparação e análise de diversos estudos, e documental, caracterizada por dados e informações que ainda não foram tratados de forma científica ou analiticamente.

No que se refere a desenvolvimento de software, serão usados elementos de metodologias ágeis (metodologias que produzem entregas mais rápidas, frequentes e flexíveis, pois dividem o desenvolvimento de um projeto em fases, cada uma com uma meta estabelecida (ALGAR, 2018)), mais especificamente, as partições do projeto serão desenvolvidas em um período de tempo chamado de *sprint* (em torno de 1-2 semanas) e por fim, serão aderidas práticas da metodologia *Extreme Programming* de escrever o código o mais sim-

ples e claro possível e de refatorar continuamente para simplificar o código mesmo que seja necessário alterar um processo que funcione.

De forma enumerada e resumida, as tarefas a serem realizadas nesse projeto serão:

1. Separar e compatibilizar as imagens do conjunto de dados em grupo de treinamento e grupo de testes;
2. Testar arquiteturas compatíveis variando: número de camadas, funções de ativação, critérios de parada etc.;
3. Fazer o treinamento da rede em ambiente apropriado;
4. Testar a rede com o subgrupo de testes composto de forma aleatória com imagens do conjunto;
5. Relatar resultados encontrados e análise por meio de métricas adotadas para discussão;

2 Referencial Teórico

2.1 Tipos de glaucoma

O glaucoma é percebido principalmente quando há lesões no nervo ótico, mas para ser capaz capaz de identificar e diagnosticar esse mal, é necessário conhecer as diversas formas, em casos infantis e adultos. A maioria desses tipos pode ser agrupada em duas categorias (CERQUETANI, 2024):

- Glaucoma de ângulo aberto: Tipo mais comum da doença, causado por um aumento de pressão intraocular devido à falta de escoamento do humor aquoso, trazendo uma perda gradativa da visão periférica e possui esse nome porque não afeta o ângulo entre íris e a córnea, permitindo ver os canais de drenagem;

- Glaucoma de ângulo fechado: Nessa categoria, os canais de escoamento são bloqueados, de maneira súbita ou crônica, causando muita dor e necessitando de cuidados imediatos para impedir lesões mais graves no nervo;

O glaucoma também pode ocorrer devido a acidentes, má formações congênitas, outros males como diabetes, ou até mesmo uso de medicamentos como corticoesteroides, além de existir uma variação em que a PIO não altera, tornando esse tipo de glaucoma mais difícil de detectar. Todas essas formas de glaucoma possuem suas peculiaridades, mas a lesão do nervo ainda é elemento presente em todas, e seu reconhecimento por meio de imagens ainda bastante efetivo;

2.2 Exame de Tomografia

A imagem a seguir é um relatório de exame de tomografia de coerência óptica, também conhecido como tomografia de mácula, e exemplifica os dados gerais que são obtidos por esse tipo de exame, como as camadas de fibras nervosas e as de células ganglionares, as quais podem estar reduzidas além do normal, criando suspeita de glaucoma. O exame também permite a visualização de cortes transversais da retina e a análise quantitativa dos setores retinianos, como os valores apresentados no relatório (em micrômetros), fornecendo parâmetros objetivos para comparação em exames futuros e possibilitando o monitoramento da progressão da doença. Normalmente esses dados devem ser levados a um especialista de confiança, o qual irá julgar a condição atual dos olhos comparando com resultados de outros exames para identificar possíveis anormalidades.

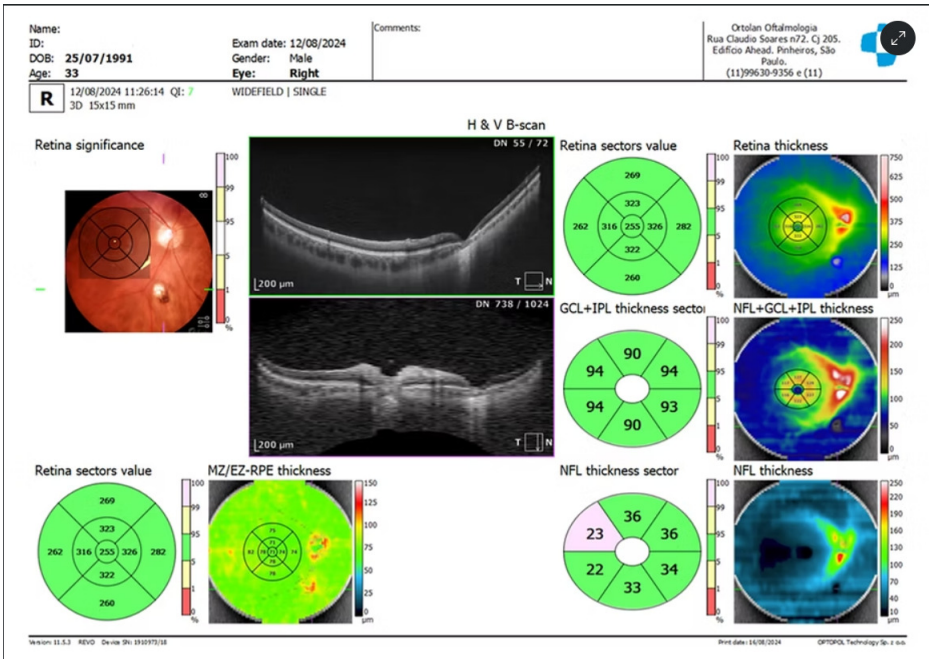

Figura 3 – Exame de OCT, com a imagem da mácula apresentada no canto superior esquerdo, a tomografia de nervo óptico na parte superior do centro do resultado, e as mediçoes de setores da retina e grossura ao redor na parte lateral direita e inferior esquerda - Fonte: Ortolan Oftalmologia.

Existem duas imagens principais que são geradas nesse documento, a em preto e branco, também chamada de tomografia de nervo, que foca no afundamento do nervo óptico, permitindo análise especial e diferenciada da estrutura ocular, e a colorida, a qual mostra o fundo da retina, sendo essa escolhida para o treinamento de redes neurais, pois é que tem mais detalhes das duas e a mais disponível em repositórios públicos.

2.3 Redes neurais

Redes neurais artificiais são definidas como sistemas computacionais distribuídos compostos de unidades de processamento simples, densamente interconectadas. Essas unidades, conhecidas como neurônios artificiais, tendo o agrupamento desses a designação de camada, computam funções matemáticas (FACELI et al., 2021).As redes neurais levam esse nome pelo fato de que foram desenvolvidas com o propósito de simular o funcionamento do cérebro humano: diferentes neurônios se comunicam por meio de sinapses e processam a informação que recebem e passam adiante. Inspiradas nesse comportamento biológico, as redes neurais artificiais utilizam unidades chamadas neurônios artificiais, organizadas em camadas, para realizar o processamento de dados. A Figura 4 mostra um exemplo rudimentar de uma rede neural artificial com duas camadas de processamento, ilustrando como as informações fluem da camada de entrada, passam por camadas intermediárias (também chamadas de camadas ocultas) e seguem até a saída:

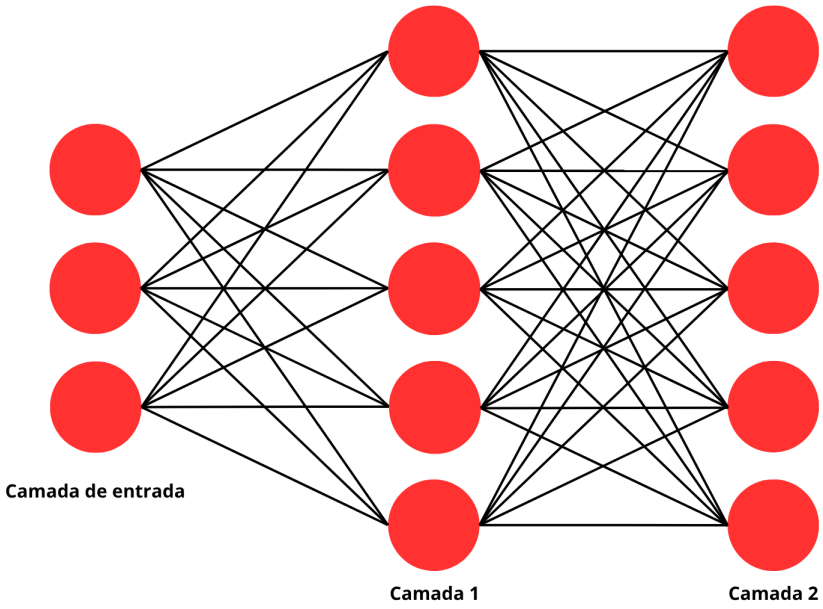

Figura 4 – Rede neural simplificada genérica para demonstração, com uma camada de entrada, uma camada interna que aplica pesos e vieses, e uma camada de saída, com cinco nós registrando valores de peso e erro final.

Como se pode observar na imagem acima, a informação é recebida em uma camada de neurônios (normalmente esses dados são valores numéricos), que enviará para uma camada oculta, a qual envia para a camada 2, a camada final neste exemplo.É importante destacar que as conexões realizadas pelos neurônios não são iguais, e para refletir o quão relevante é a influência da mensagem de um neurônio para o outro, é dito que existe um peso ponderado para aquela sinapse lógica (pode também haver parâmetros somados ao resultado para ajuste fino e predições em dados pobres em variação - são chamados de vieses (INFORMAÇÃO, 2023)). O processamento que os neurônios realizam sob os dados não passa de aplicar uma função matemática em cima desses valores ponderados e enviá-los. Neste momento devem surgir as dúvidas: mas o que é essa função e qual utilizar?

Primeiramente, essas funções são conhecidas como funções de ativação, e servem para conferir uma capacidade não-linear ao processamento realizado pelas redes neurais (CECCON, 2020). A linearidade no contexto de redes neurais é referente ao relacionamento entre dados que influenciam entre si, ou seja, se um dado influencia de forma clara e precisa em outro, então essa relação pode ser escrita usando uma função linear. O problema surge quando se observa o mundo real e percebe-se que a influência de uma informação em outra dificilmente é algo claro e preciso, demonstrando a insuficiência das funções desse tipo. Logo, é sempre interessante buscar o uso de funções não lineares para ativação desses dados, levando em conta relações complexas entre dados.

Ainda assim, o número de funções não lineares ainda é extremamente grandioso,

e por isso é necessário compreender o que cada uma delas oferece e permite de processo e entendimento dos dados. Entre as mais conhecidas estão a ReLu, a *softmax* e a sigmoide.

2.3.1 Tipos de redes neurais

A tecnologia de redes neurais é algo utilizado e diversas áreas com diferentes propósitos, e, com o passar do tempo, necessidades surgiram. Como forma de resposta, foram criadas versões especializadas de redes neurais para cada situação. Um dos exemplos são as CNNs, ou redes neurais convolucionais, que são um subconjunto das ANNs, as quais são as redes em que são aplicados filtros de convolução, os quais permitem a detecção de padrões já conhecidos pelo desenvolvedor, e o processo de *pooling*, nos quais são observados os dados obtidos por meio da filtragem e uma quantidade desses dados é reduzida para um valor sumário (no caso de *maxpooling*, seria o maior valor), diminuindo o tamanho dos dados a serem treinados na rede (ALVES, 2018).

De maneira simplificada, CNNs são muito recomendadas para treinamento de classificação de imagens, pois os filtros são usados para encontrar características importantes, e o *pooling* vai justamente olhar partes da imagem e salvar apenas os pontos que foram identificados como com características relevantes pelos filtros. Dentre as CNNs, algumas arquiteturas se destacam:

- LeNet-5: Uma das primeiras CNN desenvolvidas, usa de convolução intercalada com *sampling* para reduzir o tamanho do conjunto de dados que são usados para classificação (LECUN et al., 1998);
- AlexNet: CNN desenvolvida em 2012, utiliza da função ReLU para ativação, usa de convolução seguida de normalização seguida de *pooling* (KURAMA, 2024);
- ResNet: Arquitetura baseada no conceito de mapeamento de identidade de blocos e "atalhos"na rede neural, transformando-a um uma rede residual. (HE et al., 2016);

2.3.2 Função ReLU

Essa função é uma das mais utilizadas em diversos contextos, não apenas pela sua facilidade de implementação em ambientes de programação, mas também devido à simplicidade intrínseca da sua estrutura e ao fato de ser extremamente fácil de compreender e explicar. Sua popularidade se deve, em grande parte, à combinação desses fatores, que a tornam uma escolha comum tanto para iniciantes quanto para desenvolvedores experientes. A Figura 2 ilustra de forma clara e precisa a representação gráfica dessa função, destacando sua forma característica e como ela se comporta em diferentes cenários. Essa representação visual ajuda a consolidar o entendimento da função, reforçando sua natureza acessível e intuitiva.

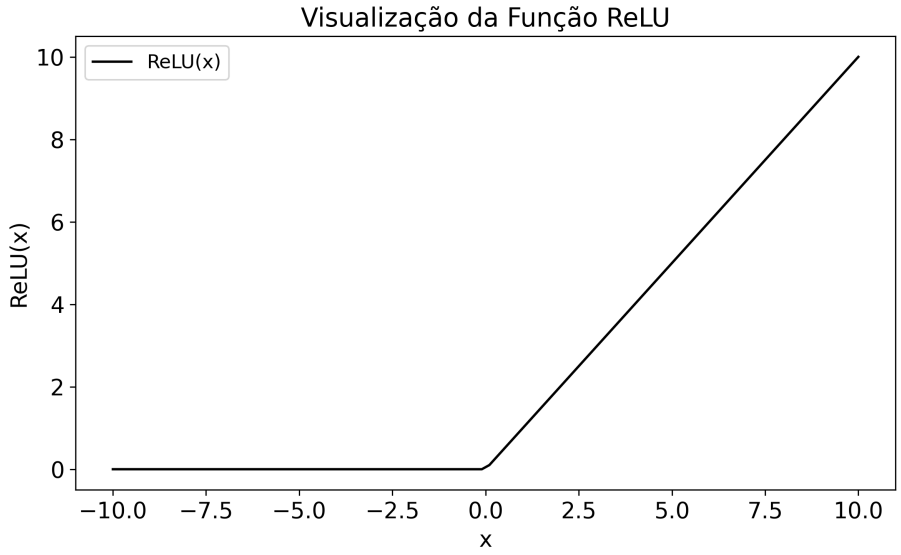

Figura 5 – Gráfico da função ReLU.

Essa função permite que o valor de X oscile entre 1 e infinito positivo, e é recomendada para lidar com o problema do gradiente desvanecente (erros se tornam pequenos demais para serem computados) e quando velocidade computacional é um fator a se considerar. Teríamos então:

$$f(x) = \begin{cases} x & \text{se } x > 0 \\ 0 & \text{caso contrário} \end{cases} \tag{2.1}$$

em que X será o valor da entrada na camada com essa função e f(x) a saída do nó pós ativação.

2.3.3 Função *softmax*

A função *softmax*, também conhecida como função exponencial normalizada, é amplamente utilizada especialmente na camada de saída de redes neurais voltadas para tarefas de classificação. Isso se deve ao fato de que ela tem a capacidade de transformar um vetor contendo N valores reais — geralmente representando os escores ou ativações produzidos pela rede — em uma distribuição de probabilidades com N componentes, onde cada valor resultante estará no intervalo entre 0 e 1, e a soma de todas as probabilidades será igual a 1. Dessa forma, a *softmax* permite interpretar a saída da rede como uma estimativa da probabilidade de pertencimento a cada uma das classes possíveis. Abaixo, apresenta-se o gráfico que ilustra o comportamento da função.

Em outras palavras, seu valor está em receber os dados processados e dizer a qual rótulo esse conjunto de dados mais provavelmente pertence. Têm-se a fórmula

$$\sigma(\mathbf{z})_i = \frac{\exp(z_i)}{\sum_{j=1}^{n} \exp(z_j)} \tag{2.2}$$

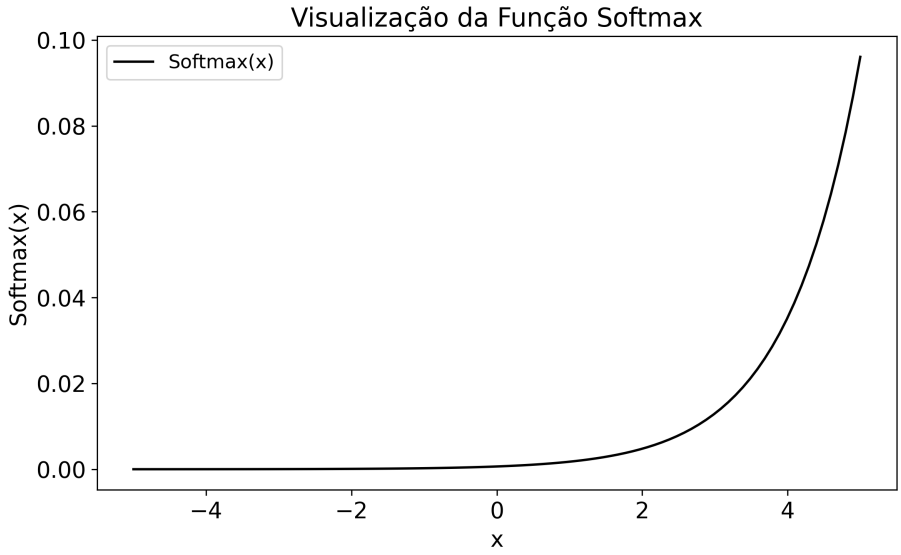

Figura 6 – Gráfico da função *softmax*.

em que z é um elemento de um vetor Z, exp é a função exponencial, i e j seriam os índices dos elementos dentro do vetor de entrada e saída respectivamente, o sigma maiúsculo, o somatório dos exponenciais de 1 a n, e o sigma minúsculo a razão entre a exponencial e o somatório.

2.3.4 Função Sigmoide

Pode ser vista como uma variação ou especialização da função anterior, sendo frequentemente aplicada na última camada de redes neurais, especialmente para tarefas de classificação. Ao contrário de outras funções, a sigmoide possui um domínio restrito entre 0 e 1, o que a torna ideal para problemas de classificação binária, nos quais o objetivo é distinguir entre duas classes distintas. Esse comportamento permite que a saída da função represente uma probabilidade, indicando a confiança do modelo em relação à classe positiva (1) ou negativa (0). Vale a pena destacar que a sigmoide sofre muito com o problema do gradiente desvanecente quando utilizada em camadas ocultas e exige uma capacidade computacional maior. O cálculo da função é descrito como

$$S(x) = \frac{1}{1 + e^{-x}} \tag{2.3}$$

em que x seria o valor de entrada, e o número de Euler, uma constante matemática, e $S(x)$ o valor da saída da função. O gráfico gerado pode ser visualizado na figura abaixo:

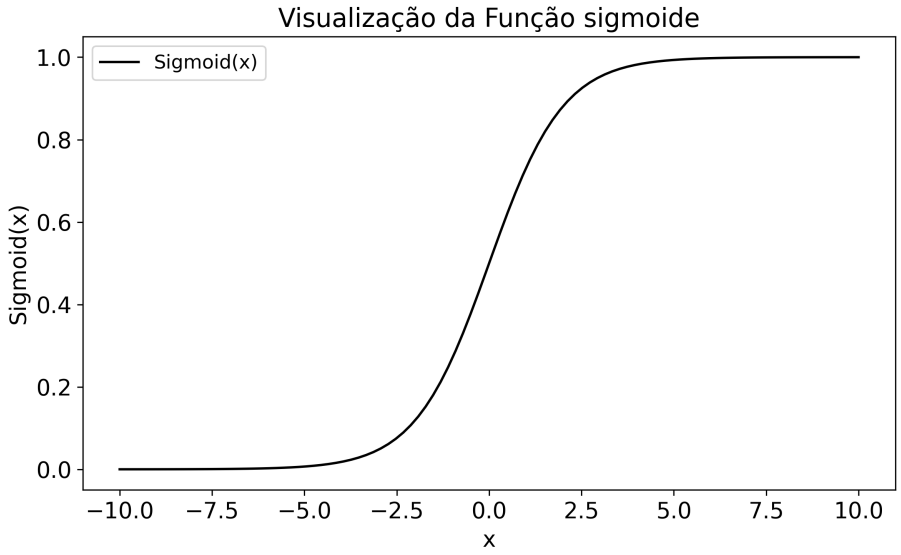

Figura 7 – Gráfico da função sigmoide.

2.4 Propagação de pesos e vieses

Os valores de peso e viés para cada camada são definidos logo no início, quando a rede neural é inicializada, mas como não há como saber de forma precognitiva quais dados a rede receberá nessa nova execução, é necessário treinar o algoritmo para, utilizando de um conjunto de dados de treinamento similares aos dados reais, encontrar os pesos e vieses que garantam um grau de acerto aceitável. Tal processo ocorre ao longo de várias iterações e é guiado pelo algoritmo *back-propagation*, o qual é dividido um duas fases: propagação direta e retropropagação do erro.

2.4.1 Propagação direta

Propagação para frente, ou *Foward Propagation*, é justamente o processo de aplicar o peso e viés correspondentes para cada neurônio e ativar o resultado com uma função de ativação antes de ser enviado para a próxima camada. Tal processo ocorre em todas as camadas e com todos os neurônios, e por mais que pareça algo trivial, ele também é essencial para o aprendizado da rede neural (IATRACKER, 2024).

2.4.2 Retropropagação do erro

Também chamado de *Back Propagation*, é o processo final do algoritmo de aprendizado. Após a propagação direta, é feita a classificação do conjunto de dados de acordo com os rótulos disponibilizados, e naturalmente ocorrerá de um grupo de dados para treinamento ser classificado de forma incorreta. Como é esperado que o algoritmo se ajuste para cada iteração, menos casos como esse surgirão, ocorrendo a segunda parte do aprendizado, na seguinte ordem:

1. É calculada a função de erro para os resultados obtidos;
2. Para cada camada antes da camada de saída, são calculadas as derivadas do erro do peso e viés;
3. Propagando essa derivada dos erros recém calculados, são calculadas as derivadas do erro no peso e viés da camada anterior utilizando da derivada da função de ativação da camada;
4. O terceiro passo é repetido até que sejam calculadas as derivadas dos erros de peso e viés da primeira camada oculta;
5. São calculados novos valores de peso e viés para as camadas usando as derivadas por uma taxa de aprendizado arbitrária;

É importante destacar que existe tanto uma variedade de funções de erro que podem ser aplicadas, dependendo do tipo de predição que ocorrerá (predizer valores contínuos, predição de múltiplas classes etc.) na etapa 1 (ABHISHEK, 2024), quanto de algoritmos voltados a otimização na etapa 5. Depois dessas execuções, com um novo conjunto de pesos e vieses, é chamado o processo de propagação direta e inicia-se uma nova iteração com outro conjunto de dados de treinamento.

3 Implementações e Avaliações Preliminares

3.1 Dataset MNIST e prova de conceito

Com o objetivo de criar uma rede mais simples como prova de conceito, foi decidido usar o dataset MNIST para tal teste. Esse dataset é composto por imagens em preto e branco de 28 x 28 pixels dos dez primeiros números, de 0 a 9, escritos à mão e classificado de acordo com qual é o número na imagem, como mostra a Fig. 2, logo abaixo:

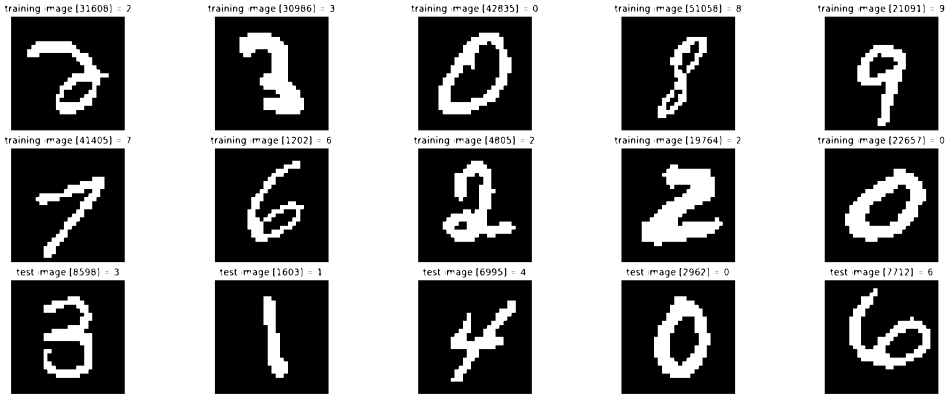

Figura 8 – Demonstração das imagens do dataset MNIST, com imagens de números escritos de forma cursiva em preto e branco.

Como as imagens são consideravelmente pequenas, apenas 784 *pixels* cada, o processo de categorização já foi feito e está disponível para o usuário e existe também um número considerável para conjunto de treinamento e teste (60000 para treinamento e 10000 para testes sendo mais específico), julgou-se que seria um conjunto de dados adequado para tal demonstração (a base específica foi extraída da plataforma do Kaggle). Por fim, decididu-se usar a Linguagem Python, pois os pacotes numpy e pandas facilitaram os cálculos matemáticos e o processo de carregar os dados no algoritmo que se encontra no Apêndice A.

Nas primeiras 26 linhas do programa, são carregados os datasets de treinamento e teste (linhas 6 e 8), em seguida os valores de tom de cada pixel são colocados em um array chamado "data1", o qual será transposto e terá as *labels* retiradas e salvas em um array separado. Por fim, data1 é normalizado na linha 17, assim tornando todos os valores números entre 0 e 1.

A partir da linha 27, é definida a função *init_params*, responsável por inicializar os parâmetros da rede neural. Nessa função, os pesos e vieses para as três camadas são gerados aleatoriamente:

- peso1 e bias1 para a primeira camada (10 neurônios conectados a 784 entradas);
- peso2 e bias2 para a segunda camada (10 neurônios conectados entre si);
- peso3 e bias3 para a terceira camada (10 neurônios de saída).

Logo após, são definidas as funções de ativação:

- *ReLU* na linha 28 (e sua derivada, *ReLU_deriv* na linha 31);
- *Softmax* na linha 34;

Em seguida, a função *forward_prop* realiza a propagação direta dos dados pela rede. Nela, o vetor de entrada é processado:

- Na primeira camada, multiplica-se *peso1* pelo vetor de entrada, soma-se *bias1* e aplica-se a função *ReLU*, gerando *camada1*.
- Na segunda camada, a saída da primeira é processada com *peso2* e *bias2*, seguida da aplicação da função *ReLU* para gerar *camada2*;
- Na terceira camada, *camada2* é transformada com *peso3* e *bias3*, e a função *softmax* é aplicada para obter *camada3*, que representa as probabilidades de cada classe;

A função *one_hot* definida na linha 48 converte os rótulos originais em um vetor codificado no formato *one-hot*, essencial para o cálculo do erro na retropropagação, pois garante que não haverá viés de hierarquia e que as variáveis sejam convertidas para um formato aceitável (GANJI, 2019).

Logo depois, a função *backward_prop* implementa o algoritmo de retropropagação a partir da linha 54. Ela calcula os gradientes dos pesos e vieses a partir do erro entre a saída predita (obtida pela *softmax*) e os rótulos codificados em *one-hot*. Esse erro é propagado para as camadas anteriores utilizando o gradiente da função *ReLU*, permitindo a obtenção dos gradientes:

- *dW3* e *db3* para a terceira camada;
- *dW2* e *db2* para a segunda camada;
- *dW1* e *db1* para a primeira camada.

Em seguida, a função *update_params* atualiza os parâmetros da rede, subtraindo, de cada peso e viés, o produto da taxa de aprendizado (*alfa*) pelo gradiente correspondente.

As funções *get_predictions* e *get_accuracy* são definidas para, respectivamente, extrair as predições da camada de saída (selecionando o índice com maior valor) e calcular a acurácia do modelo ao comparar as predições com os rótulos reais.

A função *gradient_descent* implementa o processo de treinamento da rede neural. Ela inicia os parâmetros chamando *init_params* e, para cada iteração (no exemplo, 500 iterações), realiza:

- A propagação direta (*forward_prop*);
- O cálculo dos gradientes via retropropagação (*backward_prop*);
- A atualização dos parâmetros (*update_params*).

A cada 10 iterações, o programa imprime o número da iteração e a acurácia obtida com os dados de treinamento, permitindo acompanhar o desempenho do modelo durante o treinamento.

Após o treinamento, as funções *make_predictions* e *test_prediction* são definidas com o intuito de:

- *make_predictions*: processar um conjunto de dados e retornar as predições baseadas na propagação direta.
- *test_prediction*: testar individualmente a predição para uma imagem específica (identificada pelo número passado na primeira posição), exibindo a imagem e comparando a predição com o rótulo real utilizando da biblioteca *matplotlib*.

Da linha 123 a 128, o dataset de teste passa pela mesma transformação que o de treinamento, criando o data2, normalizando os valores e separando os rótulos de cada imagem. Por fim, o conjunto de dados de teste é processado de maneira semelhante ao de treinamento (transposição e normalização), e a função *make_predictions* é utilizada para realizar as predições nesse conjunto. A acurácia do modelo é então calculada e exibida, fornecendo uma avaliação do desempenho final da rede neural.

3.2 Acurácia

Foi utilizada apenas a métrica da acurácia para avaliar a performance da rede, visto que essa métrica representa justamente a razão de acertos pelo tamanho de amostra, tanto de teste quanto de treinamento. A cada iteração, a acurácia está fadada a alterar, e dependendo do valor da taxa de aprendizado passado, ela pode superar os 90% no treino, algo que pode parecer vantajoso, mas quando aplicados os pesos e vieses ao teste,

a acurácia cai consideravelmente, demonstrando um caso de *overfitting*, que é quando a rede se molda demais aos dados de treino e considera ruído como informação relevante, fazendo com que as predições feitas com dados à parte tendam a ter mais erros.

A tabela 1, logo abaixo, mostra os valores de acurácia obtidos no final do treino e nos testes, respectivamente, em função da taxa de aprendizado (representado pelo símbolo α).

Tabela 1 – Medições de acurácia em função da taxa de aprendizagem com 100 iterações

Valor de α	**Final do treino** (%)	**Teste** (%)
0.1	39.15	44.47
0.001	8.76	8.09
1	67.72	55.8
0.5	72.85	79.2

Dois pontos se destacam ao observar a tabela: primeiro, quando o alfa chega no seu valor limite de 1, a acurácia em testes é consideravelmente mais baixa que a de treinamento, possivelmente apontando uma ocorrência de *overfitting*, e segundo, se a taxa for um valor muito baixo, ocorre o oposto: a acurácia dos dois conjuntos é igualmente baixa, já que o modelo não conseguiu se ajustar aos dados (o chamado *underfitting*). O segredo para ter boas métricas é, portanto, encontrar um valor médio que não ajuste demais mas também permita o aprendizado ocorrer a passos maiores.

Finalizando, é necessário destacar que o número de iterações, o qual é pequeno nesse contexto mostrado, também tem influência grande nesse cálculo, pois, nesse caso, ele sendo muito grande, fica mais difícil de perceber *underfitting* e *overfitting*, já que, mesmo com um aprendizado muito lento ou que assume coisas demais sobre os dados, permite a oportunidade de, eventualmente, encontrar bons valores de peso e viés. Entretanto, dependendo de o quão larga ou profunda for a rede neural, seria necessário um tempo absurdamente grande de treinamento para eliminar tais possibilidades, portanto o número de iterações deve também ser estudado junto à taxa de aprendizado.

4 Método Proposto Para Detecção de Glaucoma: Implementação e Avaliação.

4.1 Metodologia

A metodologia apresentada na seção 1.5 deste trabalho explicou as duas metodologias usadas nessa primeira parte do projeto, e, considerando que a segunda parte (TCC 2) ainda haverá mais pesquisa e desenvolvimento, acredita-se que se faz necessário manter as metodologias e apenas adaptar quando ocorrer e caso ocorra algum imprevisto durante o semestre. Portanto, a metodologia será novamente pesquisa exploratória para quaisquer buscar bibliografias necessárias, e uma mescla das *design sprints* com código simples e refatoração mínima.

4.2 Requisitos

Requisitos são especificações de um projeto e são utilizados para guiar o processo de desenvolvimento, mais especificamente assuntos relacionados a funcionalidades (chamados de funcionais), características, qualidades e restrições (chamados de não funcionais)(ROSA, 2021). No que se refere ao TCC 2, os requisitos funcionais serão:

- **RF1**: O programa deve conter uma rede neural, a qual deve ser executada em sua completude sem erros;
- **RF2**: A rede neural deve ser capaz de receber arquivos de imagem para o treinamento e teste;
- **RF3**: O programa deve usar o dataset de treinamento para treinar a rede neural e classificar as imagens desse dataset;
- **RF4**: Deve ser mostrada no final a acurácia obtida no treinamento e no teste;
- **RF5**: O usuário deve ser capaz de editar o caminho para os datasets de treinamento e teste;

Quanto aos requisitos não funcionais:

- **RNF1**: O programa deve ser escrito em alguma linguagem de programação de alto nível;

- **RNF2**: A rede neural deverá ter o nomes de suas funções descritivas da ação realizada;
- **RNF3**: O programa deve ser compilado para execução;
- **RNF4**: Quaisquer mensagens a serem mostradas devem ser exibidas em interface gráfica ou console;
- **RNF5**: As mensagens geradas devem estar em português do Brasil;

4.3 Aplicação e desenvolvimento

Esta seção será voltada a contextualizar os testes de redes neurais feitos. Além disso, também serão apresentados os resultados e medições obtidas, assim como possíveis interpretações a serem feitas e obstáculos encontrados durante esse processo.

4.3.1 Amostra

Uma rede neural aprende utilizando um dataset como alvo de seu estudo, calculando parâmetros e detectando padrões em imagens a ponto de ser capaz o reconhecimento dessas peculiaridades que existem. No caso desse projeto, foi escolhido como banco de imagens um repositório encontrado no site Kaggle, página da internet famosa por conter bases de dados e datasets para treinamento de programas com aprendizado de máquina, chamado de *Ocular Disease Recognition* (Reconhecimento de doenças oculares, em inglês), sendo supostamente montado por discentes da universidade de ciência da computação da universidade de Nankai da China (existe um artigo relacionado à base de dados que detalha os parâmetros medidos, mas a conta que registrou o dataset não aparenta ser de nenhum dos membros relacionados).

Essa base de dados possui imagens 512 por 512 *pixels* de tomografia ótica de pacientes com diversas doenças, como catarata, glaucoma, degeneração macular e outras. Considerando que existe uma variação de idade e gêneros relatados, foi decidido que seriam extraídas dessas imagens o dataset de fotos a serem usadas para treinar a rede neural a seguir. Foram selecionadas as imagens de fundo de retina categorizadas com glaucoma apenas, ou seja, sem outra anormalidade detectada (um total de 200 imagens mais especificamente), e foram selecionadas 200 imagens classificadas como normais de forma aleatória, completando o dataset completo de 400 imagens a serem utilizadas (as imagens são embaralhadas logo na montagem do dataset pelo programa e embaralhadas para a separação entre treinamento e validação).

Montado o grupo de fotos de fundo de retinas, doravante referenciada como dataset de OCT, avançou-se para o próximo passo: a escolha de modelo.

Classe	Esquerda	Direita	Total
Normal	100	100	200
Glaucoma	104	96	200

Tabela 2 – Distribuição do dataset de treinamento.

4.3.2 Modelos

Uma rede neural pode ser organizada de diversas formas, com diversos números de camadas de tratamento, equalização ou saída de dados, e muitas vezes cabe ao responsável pela rede decidir qual a melhor forma ou até como avaliar qual forma de organização deverá ser aplicada. Essas composições, quando se tornam populares, ganham nomes e são chamadas de modelos, normalmente por questões históricas ou por características peculiares.

4.3.2.1 VGG 16

A primeira arquitetura utilizada nesse estudo foi o VGG 16. Criado por volta de 2014 pelo grupo de mesmo nome(KHANDELWAL, 2020), esse modelo foi idealizado para classificação de imagens e chegou a ser utilizado no ILSVRC (*ImageNet Large Scale Virtual Recognition Challenge*) de 2014, uma competição em que várias equipes aplicaram seus modelos a um dataset de diversas imagens coletadas do site Flickr.

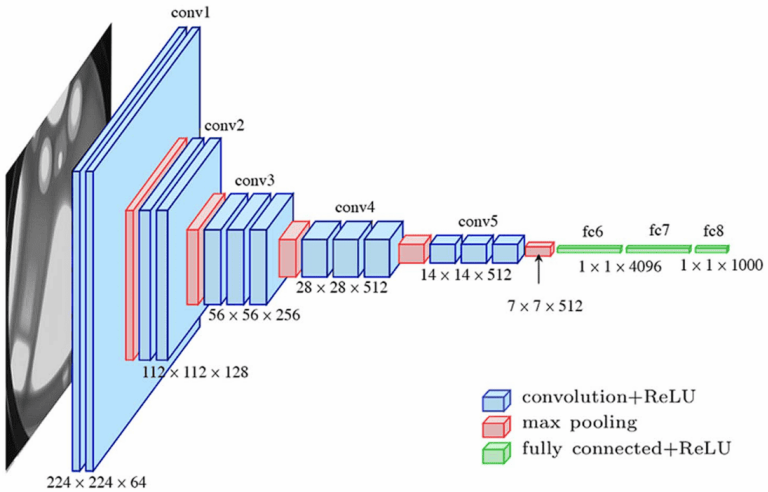

Figura 9 – Representação visual do modelo VGG 16 com dezesseis camadas convolucionais seguidas de três camadas densas para processamento de pesos - fonte: (BOESCH, 2021)

Como mostra a imagem logo acima, a arquitetura VGG se baseia na ideia de blocos de camadas convolucionais, as quais são elementos que vão olhar para uma parte da imagem e aplicar operações de convolução sobre os valores dos pixels na região, interpolados por camadas que realizam a operação *max pooling*, operação que reduz a resolução

da imagem e o escopo de parâmetros contabilizados, encerrando com três camadas conectadas, cujo propósito é reduzir vetor produzido para o tamanho equivalente de classes disponíveis, por exemplo, mil no desafio do ILSVRC. Existem algumas versões diferentes do modelo, como VGG 8 ou 19, mas o número de camadas conectadas sempre será três e oscila apenas o número de blocos e camadas convolucionais.

4.3.2.2 ResNet50

Com o surgimento de arquiteturas como o VGG, as redes neurais encontraram um novo obstáculo: quanto mais profunda a rede mais o gradiente começa a afetar menos as camadas mais superficiais e a acurácia começa a saturar e se torna mais difícil otimizar o processo de treino (POTRIMBA, 2024). Nesse contexto, o ResNet é criado por um time de pesquisadores da Microsoft como uma solução para esses desafios. As chamadas *residual networks* (redes residuais, em inglês) são uma família de CNNs cujas características que as distinguem são os blocos residuais demonstrados na figura abaixo.

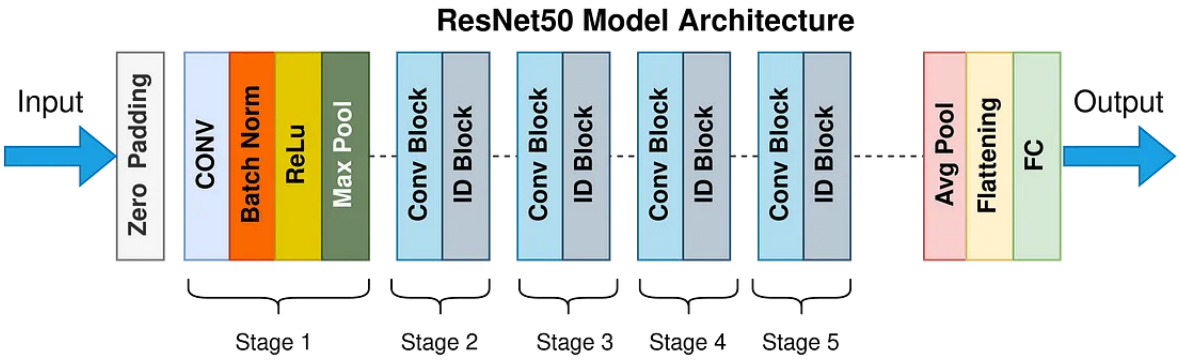

Figura 10 – Representação visual do modelo ResNet50, representando o primeiro bloco com convolução, normalização, Função Relu e *max pool*, seguido por uma camada densa para saída - fonte: Medium.com

O bloco residual é um conjunto de camadas convolucionais e funções de ativação que realizam as mesmas funções em outras arquiteturas, seguidas de uma adição ou concatenação dos dados inseridos como entrada do bloco, a fim de facilitar processamento de gradientes e reforçar os dados obtidos (BRYCE, 2022). Além dos blocos residuais, também haverá camadas densas com o mesmo propósito das camadas densas na arquitetura VGG: processar parâmetros e classificar com base nos pesos finais. A versão apresentada no desafio de 2015 possui 152 camadas, mas a utilizada nesse estudo foi a ResNet 50, com 49 camadas divididas em blocos e uma camada densa, mostrada de forma reduzida na figura 10.

4.3.3 Ambiente e ferramentas

Para aplicar essas arquiteturas ao estudo com imagens de OCT, foi decidido utilizar o ambiente do Google Colab, um ambiente gratuito para escrever códigos e com suporte a diversas bibliotecas python, pois tal plataforma possibilita o uso da biblioteca do Tensorflow, uma biblioteca de código aberto desenvolvida pela própria Google para aprendizado de máquina, e o keras, uma API do Tensorflow voltada para a criação de redes neurais, além de contornar o primeiro obstáculo encontrado: limitação do *hardware* local do redator.

4.3.4 Limitações e ajustes

Considerando o tamanho das imagens do dataset, treinar os parâmetros das camadas convolucionais em sua completude se demonstrou inviável, pois nem mesmo o ambiente online usando a GPU T4 para acelerar o processamento de operações, então foram importados os pesos calculados originalmente no desafio do ImageNet, evitando treinar milhões de parâmetros por época, ciclo em que a rede neural processa todas as imagens uma vez, mas assim limitando quão alta a acurácia pode chegar (ainda ocorre treinamento nas camadas densas, onde a rede aprende a interpretar os valores calculados).Os apêndices B e C apresentam a estrutura completa com os ajustes já implementados.

Ademais, foram feitos testes com os dois modelos para encontrar o valor ideal de épocas. Para isso foram substituídos os trechos 86 - 121, no segundo apêndice, e 95 - 130 , no terceiro apêndice, das redes VGG 16 e Resnet50, respectivamente, pelo trecho apresentado no quarto apêndice.

As medições foram convertidas para os gráficos representados nas figuras 11 e 12.

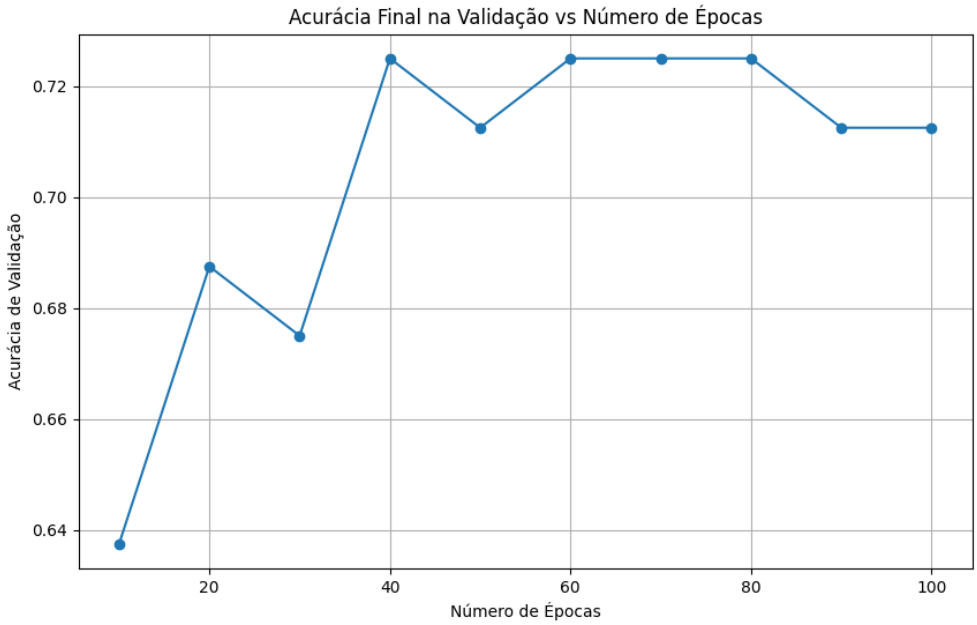

Figura 11 – Gráfico de acurácias do VGG 16 por dezenas de épocas.

Nas medições apresentadas no gráfico 11, das dez medições de acurácia, é perceptível um pico por volta de 60 épocas e uma estabilização por volta de 90 épocas, com 71% de acurácia, muito provavelmente sinais de *overfitting*. Considerando esse padrão observado nos resultados e visando evitar o sobreajuste, foi decidido limitar o treinamento da arquitetura VGG com 16 camadas a 60 épocas, ponto em que o modelo atinge seu melhor desempenho médio antes da degradação associada ao *overfitting*.

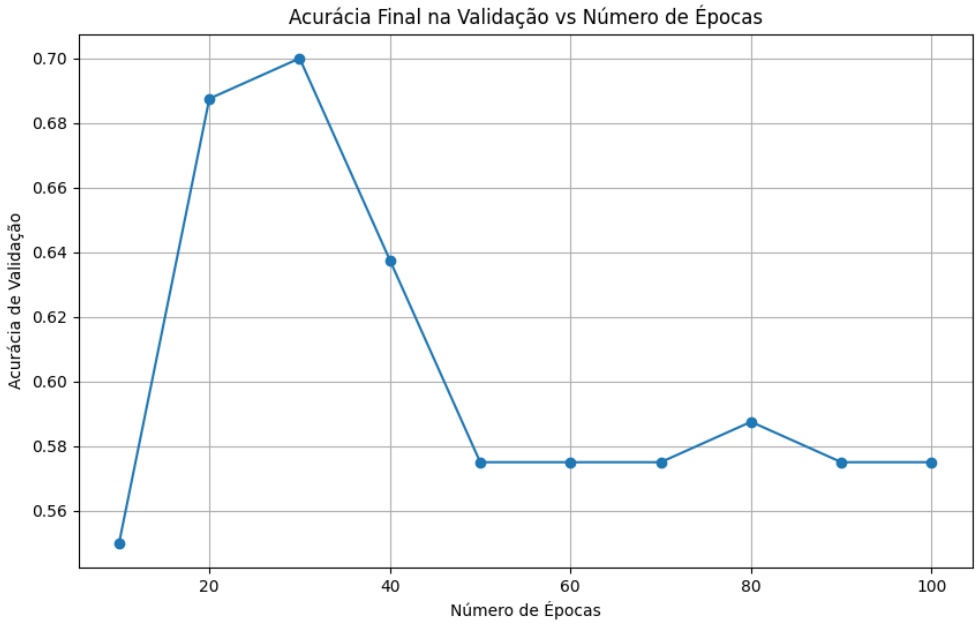

Figura 12 – Gráfico de acurácias do ResNet50 por dezena de épocas.

Nas medições apresentadas no gráfico 12, das dez medições de acurácia, é perceptível um pico por volta de 30 épocas e uma queda vertiginosa para 57% de acurácia, indicando claros sinais de *overfitting*. Considerando esses resultados, foi decidido realizar o treinamento por 30 épocas, pois assim mostraria resultados satisfatórios antes que a rede comece a se ajustar demais ao dataset.

4.3.5 Medições

Com o propósito de avaliar a performance das duas arquiteturas e permitir futuras inferências baseadas nos resultados obtidos, foram selecionadas as seguintes medidas a seguir para serem captadas:

1. **Acurácia:** Representa a proporção de predições corretas em relação ao total de predições, ou seja, quantas das classificações foram corretas. Sua fórmula é

$$\frac{VP + VN}{VP + VN + FP + FN} \tag{4.1}$$

   em que: VP são os verdadeiros positivos (casos corretamente classificados como positivos), VN os verdadeiros negativos (casos corretamente classificados como negativos), FP os falsos positivos (casos negativos classificados incorretamente como positivos) e FN os falsos negativos (casos positivos classificados incorretamente como negativos).

2. **F1-Score:** Representa a média harmônica entre precisão, métrica que representa a taxa de acerto das classificações positivas, e *recall*, conhecida também como sensibilidade, a qual representa a capacidade de um modelo predizer classificações positivas, podendo variar de 0 a 1 e descrito como

$$F_1 = \frac{2 \cdot VP^2}{VP \cdot (2VP + FP + FN)} \tag{4.2}$$

   de forma simplificada;

3. **Variância:** Representa o quanto os resultados de desempenho oscilam, refletindo a estabilidade do modelo, representada pela fórmula

$$\sigma^2 = \frac{1}{n} \sum_{i=1}^{n} (x_i - \mu)^2 \tag{4.3}$$

   em que σ^2 representa variância, n é o número de itens e $(x_i - \mu)^2$ é a dispersão de cada item do conjunto;

4. **Matriz de Confusão:** Demonstração visual detalhada das classificações corretas e incorretas do modelo, separadas por classe, permitindo identificar padrões de erro.

4.3.5.1 Resultados Obtidos

Ao executar as redes neurais (através da compilação dos códigos no segundo e terceiro apêndices), as métricas a seguir são registradas:

Métrica	**VGG 16**	**Resnet50**
Acurácia	0.7125	0.6000
F1-Score	0.7070	0.5997
Variância	0.2400	0.2498

Tabela 3 – Métricas para VGG 16 e ResNet50.

	Predito: glaucoma	**Predito: normal**
Real: glaucoma	34	9
Real: normal	14	23

Tabela 4 – Matriz de confusão da arquitetura VGG 16 gerada com teste de 80 imagens do dataset.

	Predito: glaucoma	**Predito: normal**
Real: glaucoma	25	18
Real: normal	14	23

Tabela 5 – Matriz de confusão da arquitetura ResNet50 gerada com teste de 80 imagens do dataset.

4.3.6 Discussão

Através dos resultados obtidos neste trabalho, são possíveis um análise e identificação de desafios, limitações e potenciais das arquiteturas de redes neurais aplicadas à detecção de glaucoma em exames de tomografia de mácula. A seguir, são destacados os pontos centrais que merecem aprofundamento.

4.3.6.1 Limitações do Dataset e Generalização

A dimensão da base de dados utilizada sempre é um aspecto relevante de apontar. Neste caso, o dataset tem 400 imagens, e isso traz algumas ponderações interessantes de se fazer: por um lado, esse quantidade implica treinamentos mais ágeis e viáveis mesmo em ambientes computacionais modestos. Por outro lado, isso também implica um treinamento não tão robusto e que não permitirá nuances serem detectadas facilmente, algo que poder ser prejudicial dependendo da demanda em contexto. Seria necessária uma base mais robusta e mais diversificada para uma transferência ao mundo real.

4.3.6.2 Infraestrutura Computacional e Ajustes de Treinamento

Mesmo em um ambiente na nuvem, o treinamento completo das redes foi impossibilitado pelo fato de que exige uma quantidade considerável de espaço e velocidade de processamento para calcular milhões de parâmetros centenas de vezes, forçando a importação de pesos e congelamento das principais camadas, técnica conhecida como *Transfer Learning*, permitindo o treinamento das redes com a possibilidade de aprimorar a acurácia das predições. Também foi necessário circunavegar a limitação do uso da GPU T4 que o Google Colab impõe por períodos variáveis criando outro ambiente associado a outra conta do Google. O processo de *Fine Tuning*, um ajuste detalhado da rede, com treinamento parcial da rede e uma ampliação artificial, manipulando os dados do dataset para serem considerados dados adicionais, poderiam trazer ganhos relevantes e um melhor ajuste da rede ao dataset.

4.3.6.3 Análise dos Modelos e Overfitting

Os experimentos evidenciaram diferenças marcantes entre as arquiteturas VGG16 e ResNet50. O modelo VGG16 apresentou acurácia de aproximadamente 71.25%, enquanto a ResNet50 alcançou cerca de 60%, ambos testados especificamente sobre uma partição de validação gerada aleatoriamente, mas igual aos dois, do mesmo dataset. O fenômeno de *overfitting* foi identificado pela disparidade entre as métricas em treinamento e validação, sobretudo quando se elevou o número de épocas, ou seja, a rede se ajustou demais ao dataset e começou a decorar padrões em vez de aprender. Estratégias como regularização (limitar aprendizado, forçando maior generalização) e ampliação artificial dos dados poderiam mitigar este problema.

4.3.6.4 Avaliação de Métricas e Erros

A escolha das métricas foi eficaz em seu propósito, com uso de acurácia, F1-Score, variância e matriz de confusão. Vale ressaltar, porém, que a acurácia pode ser insuficiente para avaliar modelos em cenários reais, onde os custos de falso positivo (indicar glaucoma em paciente saudável) e falso negativo (não detectar glaucoma verdadeiro) são muito diferentes. O cálculo de mais métricas, principalmente por uso das matrizes de confusão, poderia direcionar ajustes finos nos modelos ou mesmo inspirar abordagens híbridas que combinem diferentes arquiteturas ou técnicas de classificação.

4.3.6.5 Sugestões para Trabalhos Futuros

1. Ampliar e diversificar a base de dados, buscando heterogeneidade em equipamentos, tipos de pacientes, de faixas etárias diferentes e com estágios diferentes de glaucoma;

2. Investir em técnicas de *data augmentation* e estratégias de regularização;

3. Avaliar o desempenho sob diferentes métricas, como sensibilidade e especificidade, além da acurácia tradicional;

4. Considerar diferentes abordagens e testes externos para validar a robustez da rede construída.

4.3.6.6 Conclusão da Discussão

Os dois modelos demonstraram, em diferentes níveis de sucesso, o uso das imagens separadas para o auxílio de diagnóstico de glaucoma por meio de imagens de OCT com um nível de confiança minimamente relevante, um resultado satisfatório para um ponto de partida. Os próximos passos para evoluir as redes seriam utilizar outros modelos ou variações dessas arquiteturas e testar métricas mais avançadas que mostrariam algum progresso para ser efetivamente integrados à prática médica.

5 Considerações Finais

Esse trabalho tem o foco de buscar usar de redes neurais para colaborar no diagnóstico de uma doença extremamente severa, glaucoma. O trabalho em si foi idealizado usando uma rede neural artificial com um dataset escolhido e com o objetivo de produzir algo que tenha um valor humanitário e prático ao mesmo tempo.

Ao longo do trabalho, foi mostrado que existe uma possibilidade de aplicar tais conhecimentos de forma a ter resultados que importem e sirvam de primeiro passo a um dia termos uma queda no número de pessoas com cegueira devido à glaucoma. As pesquisas demonstraram que existem diversas formas de configurar uma rede neural, desde a escolha das funções de ativação até quais operações adicionais para conformar o algoritmo ao problema em mãos.

Logo após, foi criada uma rede neural simples para testar esses aprendizados com um conjunto de dados que necessitava de normalização mas possuía os rótulos de suas imagens e foi explicado como esse algoritmo em específico pode ser alterado e quais são as consequências de levar a rede ao extremo ou não incentivar o aprendizado o suficiente. Por fim, é proposto criar um programa que execute um algoritmo de rede neural para uma base de dados mais robusta e que seja relacionada ao tema de glaucoma.

Após isso, é realizado um estudo baseado nos conceitos elaborados anteriormente, tratando da criação de redes neurais utilizando arquiteturas estabelecidas postulando sobre vantagens e desvantagens, e adaptando o código a limitações e obstáculos encontrados. Com métricas calculadas, é discutido como alguns pontos podem ter afetado o aprendizado e sugerindo possíveis soluções que serviriam como guia para um trabalho futuro com mais modelos e equipamento aprimorado.

Referências

ABHISHEK. *Types of Loss Functions in Machine Learning and Their Usage*. 2024. Acesso em: 31 jul. 2025. Disponível em: <https://medium.com/@AIchemizt/types-of-loss-functions-in-machine-learning-and-their-usage-d4594e1529d2>. Citado na página 35.

ALGAR. *Blog: O que são métodos ágeis e como aproveitá-los na sua empresa*. 2018. <https://www.algar.com.br/metodos-ageis/>. Acesso em: 16 fev. 2025. Citado na página 26.

ALVES, G. *Entendendo Redes Convolucionais*. 2018. <https://medium.com/neuronio-br/entendendo-redes-convolucionais-cnns-d10359f21184>. Acesso em: 20 fev. 2025. Citado na página 31.

BIONDI, A. *O que é glaucoma?* 2022. <https://vidasaudavel.einstein.br/o-que-e-glaucoma/>. Acesso em: 14 fev. 2025. Citado na página 23.

BOESCH, G. *Types of Loss Functions in Machine Learning and Their Usage*. 2021. Acesso em: 15 jul. 2025. Disponível em: <https://viso.ai/deep-learning/vgg-very-deep-convolutional-networks/>. Citado 2 vezes nas páginas 15 e 42.

BROWN, S. *Machine learning, explained*. 2021. <https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained>. Acesso em: 15 fev. 2025. Citado na página 24.

BRYCE, P. Vídeo online, *Residual Networks and Skip Connections (DL 15)*. 2022. YouTube. Acessado em: 14 jul. 2025. Disponível em: <https://www.youtube.com/watch?v=Q1JCrG1bJ-A>. Citado na página 43.

CECCON, D. *Funções de ativação: definição, características, e quando usar cada uma*. 2020. <https://iaexpert.academy/2020/05/25/funcoes-de-ativacao-definicao-caracteristicas-e-quando-usar-cada-uma/>. Acesso em: 16 fev. 2025. Citado na página 30.

CERQUETANI, S. *Glaucoma: doença pode causar cegueira*. 2024. <https://nav.dasa.com.br/blog/glaucoma>. Acesso em: 19 fev. 2025. Citado na página 28.

FACELI, K. et al. *Inteligência Artificial - Uma Abordagem de Aprendizado de Máquina*. 2. ed. Rio de Janeiro: LTC, 2021. E-book, p.101. ISBN 9788521637509. Disponível em: <https://integrada.minhabiblioteca.com.br/reader/books/9788521637509/>. Acesso em: 16 fev. 2025. Citado na página 29.

GANJI, L. *One Hot Encoding in Machine Learning*. 2019. <https://www.geeksforgeeks.org/ml-one-hot-encoding/>. Acesso em: 17 fev. 2025. Citado na página 37.

HE, K. et al. *Deep Residual Learning for Image Recognition*. 2016. 772–773 p. Citado na página 31.

IATRACKER. *O que é : Forward Propagation*. 2024. <https://iatracker.com.br/glossario/o-que-e-forward-propagation/>. Acesso em: 17 fev. 2025. Citado na página 34.

INFORMAÇÃO, A. E. D. C. D. T. D. *O que são pesos e vieses na IA?* 2023. <https://pt.eitca.org/artificial-intelligence/eitc-ai-gcml-google-cloud-machine-learning/introduction/what-is-machine-learning/explain-weights-and-biases/>. Acesso em: 16 fev. 2025. Citado na página 30.

KHANDELWAL, V. *The Architecture and Implementation of VGG-16*. 2020. <https://pub.towardsai.net/the-architecture-and-implementation-of-vgg-16-b050e5a5920b>. Acesso em: 13 jul. 2025. Citado na página 42.

KURAMA, V. *A Review of Popular Deep Learning Architectures: AlexNet, VGG16, and GoogleNet*. 2024. <https://www.digitalocean.com/community/tutorials/popular-deep-learning-architectures-alexnet-vgg-googlenet>. Acesso em: 17 jul. 2025. Citado na página 31.

LECUN, Y. et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, v. 86, n. 11, p. 2278–2323, 1998. Citado na página 31.

POTRIMBA, P. *What is ResNet-50?* 2024. <https://blog.roboflow.com/what-is-resnet-50/>. Acesso em: 13 jul. 2025. Citado na página 43.

RHEE, D. J. *Visão geral do glaucoma*. 2023. <https://www.msdmanuals.com/pt/profissional/distÃžrbios-oftalmolÃşgicos/glaucoma/visÃčo-geral-do-glaucoma>. Acesso em: 15 fev. 2025. Citado na página 24.

ROSA Ângela. *Requisitos de software funcionais e não funcionais: o que são?* 2021. <https://softdesign.com.br/blog/requisitos-de-software-funcionais-e-nao-funcionais/#h-o-que-sao-requisitos-de-software>. Acesso em: 20 fev. 2025. Citado na página 40.

SERASA. *Pesquisa exploratória: o que é, e como adota-la ao negócio*. 2023. <https://www.serasaexperian.com.br/conteudos/marketing/pesquisa-exploratoria-para-que-serve-e-como-fazer/>. Acesso em: 15 fev. 2025. Citado na página 26.

STEINMETZ, J. D. et al. Causes of blindness and vision impairment in 2020 and trends over 30 years, and prevalence of avoidable blindness in relation to vision 2020: the right to sight: an analysis for the global burden of disease study. *The Lancet Global Health*, v. 9, n. 2, p. e144–e160, 2021. Citado na página 26.

Apêndices

APÊNDICE A – Primeiro Apêndice

Código criado para a rede neural de classificação do MNIST com três camadas:

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt


dataTrain = pd.read_csv('./mnistcsv/mnist_train.csv')

dataTest = pd.read_csv('./mnistcsv/mnist_test.csv')

data1 = np.array(dataTrain)

m, n = data1.shape

data1 = data1.T
labelTrain = data1[0]
vetorTrain = data1[1:n]
vetorTrain = vetorTrain / 255.

def init_params():
    peso1 = np.random.rand(10, 784) - 0.5  # dez neuronios ligados a 784
        neuronios
    bias1 = np.random.rand(10, 1) - 0.5 # vies de cada neuronio na
       camada 1
    peso2 = np.random.rand(10, 10) - 0.5 # dez neuronios ligados a dez
       neuronios
    bias2 = np.random.rand(10, 1) - 0.5  # vies de cada neuronio na
       camada 2
    peso3 = np.random.rand(10, 10) - 0.5 # dez neuronios ligados a dez
       neuronios
    bias3 = np.random.rand(10, 1) - 0.5  # vies de cada neuronio na
       camada 3
    return peso1, bias1, peso2, bias2, peso3, bias3

def ReLU(Z):
    return np.maximum(Z, 0) # funcao linear retificada

def ReLU_deriv(Z):
    return Z > 0

def softmax(Z):
    exp_Z = np.exp(Z - np.max(Z, axis=0, keepdims=True))
```

```
36      A = exp_Z / np.sum(exp_Z, axis=0, keepdims=True)
37      return A
38
39  def forward_prop(peso1, bias1, peso2, bias2, peso3, bias3, vetor):
40      saida1 = peso1.dot(vetor) + bias1
41      camada1 = ReLU(saida1)
42      saida2 = peso2.dot(camada1) + bias2
43      camada2 = ReLU(saida2)
44      saida3 = peso3.dot(camada2) + bias3
45      camada3 = softmax(saida3)
46      return saida1, camada1, saida2, camada2, saida3, camada3
47
48  def one_hot(label): #one hot encoding
49      one_hot_label = np.zeros((label.size, label.max() + 1))
50      one_hot_label[np.arange(label.size), label] = 1
51      one_hot_label = one_hot_label.T
52      return one_hot_label
53
54  def backward_prop(saida1, camada1, camada2,saida2, camada3, peso2, peso3
        , vetor, label):
55      one_hot_label = one_hot(label)  # Converte os rotulos em formato one
            -hot
56
57      dZ3 = camada3 - one_hot_label
58      dW3 = 1 / m * dZ3.dot(camada2.T)
59      db3 = 1 / m * np.sum(dZ3, axis=1, keepdims=True)
60
61      dZ2 = peso3.T.dot(dZ3) * ReLU_deriv(saida2)
62      dW2 = 1 / m * dZ2.dot(camada1.T)
63      db2 = 1 / m * np.sum(dZ2, axis=1, keepdims=True)
64
65      dZ1 = peso2.T.dot(dZ2) * ReLU_deriv(saida1)
66      dW1 = 1 / m * dZ1.dot(vetor.T)
67      db1 = 1 / m * np.sum(dZ1, axis=1, keepdims=True)
68
69      return dW1, db1, dW2, db2, dW3, db3
70
71  def update_params(peso1, bias1, peso2, bias2,peso3, bias3, dW1, db1, dW2
        , db2,dW3,db3, alfa):
72      peso1 = peso1 - alfa * dW1
73      bias1 = bias1 - alfa * db1
74      peso2 = peso2 - alfa * dW2
75      bias2 = bias2 - alfa * db2
76      peso3 = peso3 - alfa * dW3
77      bias3 = bias3 - alfa * db3
78      return peso1, bias1, peso2, bias2, peso3, bias3
79
```

```
80 def get_predictions(camada3):
81     return np.argmax(camada3, 0)
82
83 def get_accuracy(predictions, label):
84     print(predictions, label)
85     return np.sum(predictions == label) / label.size
86
87 def gradient_descent(vetor, label, alfa, iteracao):
88     peso1, bias1, peso2, bias2, peso3, bias3 = init_params()
89     for i in range(iteracao):
90         saida1, camada1, saida2, camada2, saida3, camada3 = forward_prop
               (peso1, bias1, peso2, bias2, peso3, bias3, vetor)
91         dW1, db1, dW2, db2, dW3, db3  = backward_prop(saida1, camada1,
               camada2,saida2, camada3, peso2, peso3, vetor, label)
92         peso1, bias1, peso2, bias2, peso3, bias3 = update_params(peso1,
               bias1, peso2, bias2,peso3, bias3, dW1, db1, dW2, db2,dW3,db3,
                alfa)
93         if i % 10 == 0:
94             print("iteracao:␣", i)
95             predictions = get_predictions(camada3)
96             print(get_accuracy(predictions, label))
97     return peso1, bias1, peso2, bias2, peso3, bias3
98
99 peso1, bias1, peso2, bias2, peso3, bias3 = gradient_descent(vetorTrain,
       labelTrain, 0.1, 500)
100
101 def make_predictions(vetor,peso1, bias1, peso2, bias2, peso3, bias3):
102     _, _, _,_,_, camada3 = forward_prop(peso1, bias1, peso2, bias2,
           peso3, bias3, vetor)
103     predictions = get_predictions(camada3)
104     return predictions
105
106 def test_prediction(index, peso1, bias1, peso2, bias2, peso3, bias3,):
107     current_image = vetorTrain[:, index, None]
108     prediction = make_predictions(vetorTrain[:, index, None],peso1,
           bias1, peso2, bias2, peso3, bias3)
109     label = labelTrain[index]
110     print("Prediction:␣", prediction)
111     print("Label:␣", label)
112
113     current_image = current_image.reshape((28, 28)) * 255
114     plt.gray()
115     plt.imshow(current_image, interpolation='nearest')
116     plt.show()
117
118 test_prediction(0, peso1, bias1, peso2, bias2, peso3, bias3)
119 test_prediction(1, peso1, bias1, peso2, bias2, peso3, bias3)
```

```
120 test_prediction(2, peso1, bias1, peso2, bias2, peso3, bias3)
121 test_prediction(3, peso1, bias1, peso2, bias2, peso3, bias3)
122
123 data2 = np.array(dataTest)
124 data2 = data2.T
125 labelTest = data2[0]
126 vetorTest = data2[1:n]
127 vetorTest = vetorTest / 255.
128 m2, n2 = data2.shape
129
130 predictionTest = make_predictions(vetorTest, peso1, bias1, peso2, bias2,
        peso3, bias3)
131 print('acuracia: ')
132 print(get_accuracy(predictionTest, labelTest))
```

APÊNDICE B – Segundo Apêndice

Código criado para a rede neural de classificação de imagens de OCT usando modelo VGG 16:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout,
    GlobalAveragePooling2D
from tensorflow.keras.applications import VGG16
import tensorflow.keras.backend as K
from sklearn.metrics import f1_score, accuracy_score,
    confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
import os

# Caminho para as imagens
dataset_path = './treinamento_embaralhado'
image_size = (512, 512)
batch_size_dataset = 32
seed = 42

# Carregar o dataset sem embaralhar para contar
full_dataset_info = tf.keras.utils.image_dataset_from_directory(
dataset_path,
image_size=image_size,
color_mode='rgb',
batch_size=None,
shuffle=False
)

# Contar imagens
total_images = tf.data.experimental.cardinality(
    full_dataset_info).numpy()

# Definir tamanhos
train_size = 320
val_size = 80

# Carregar dataset embaralhado com batches
full_dataset = tf.keras.utils.image_dataset_from_directory(
dataset_path,
image_size=image_size,
color_mode='rgb',
```

```
38          batch_size=batch_size_dataset,
39          shuffle=True,
40          seed=seed
41          )
42
43          train_ds = full_dataset.take(train_size // batch_size_dataset)
44          val_ds = full_dataset.skip(train_size // batch_size_dataset)
45
46          # Normalizacao e one-hot
47          def normalize_img(image, label):
48          return tf.cast(image, tf.float32) / 255.0, label
49
50          def one_hot_label(image, label):
51          return image, tf.one_hot(label, depth=2)
52
53          train_ds = train_ds.map(normalize_img).map(one_hot_label)
54          val_ds = val_ds.map(normalize_img).map(one_hot_label)
55
56          # Performance
57          AUTOTUNE = tf.data.AUTOTUNE
58          train_ds = train_ds.cache().prefetch(buffer_size=AUTOTUNE)
59          val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
60
61          # Modelo VGG16
62          def create_model():
63          base_model = VGG16(
64          include_top=False,
65          weights='imagenet',
66          input_shape=(image_size[0], image_size[1], 3)
67          )
68          base_model.trainable = False
69
70          model = Sequential([
71          base_model,
72          GlobalAveragePooling2D(),
73          Dense(256, activation='relu'),
74          Dropout(0.5),
75          Dense(2, activation='softmax')
76          ])
77
78          model.compile(
79          optimizer='adam',
80          loss='categorical_crossentropy',
81          metrics=['accuracy']
82          )
83
84          return model
```

```
85
86          # Treinamento com 60 epocas
87          K.clear_session()
88          model = create_model()
89
90          history = model.fit(
91          train_ds,
92          epochs=60,
93          validation_data=val_ds,
94          verbose=1
95          )
96
97          # Previsoes
98          y_true = []
99          y_pred = []
100
101         for images, labels in val_ds:
102         preds = model.predict(images)
103         y_pred.extend(tf.argmax(preds, axis=1).numpy())
104         y_true.extend(tf.argmax(labels, axis=1).numpy())
105
106         # Metricas
107         acc = accuracy_score(y_true, y_pred)
108         f1 = f1_score(y_true, y_pred, average='macro')
109         var = np.var(y_pred)
110
111         print(f"\n Avaliacao apos 60 epocas:")
112         print(f" Acuracia: {acc:.4f}")
113         print(f" F1-score (macro): {f1:.4f}")
114         print(f" Variancia: {var:.4f}")
115
116         # Matriz de Confusao
117         cm = confusion_matrix(y_true, y_pred)
118         disp = ConfusionMatrixDisplay(confusion_matrix=cm,
               display_labels=['Classe 0', 'Classe 1'])
119         disp.plot(cmap=plt.cm.Blues)
120         plt.title(" Matriz de Confusao")
121         plt.show()
```

APÊNDICE C – Terceiro Apêndice

Código criado para a rede neural de classificação de imagens de OCT usando modelo Resnet50:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D,
    Flatten, Dense, Dropout, Input
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import tensorflow.keras.backend as K
import os
from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import GlobalAveragePooling2D
from sklearn.metrics import f1_score, accuracy_score,
    confusion_matrix, ConfusionMatrixDisplay


# Caminho para as imagens
dataset_path = './treinamento_embaralhado'

image_size = (512, 512)
batch_size_dataset = 32 # Use um tamanho de batch para carregar
    o dataset
seed = 42


full_dataset_info = tf.keras.utils.image_dataset_from_directory(
dataset_path,
image_size=image_size,
color_mode='rgb',
batch_size=None, # Temporariamente None para contar imagens
shuffle=False
)

total_images = tf.data.experimental.cardinality(
    full_dataset_info).numpy()

train_size = 320
val_size = 80

# separando treino/validacao
full_dataset = tf.keras.utils.image_dataset_from_directory(
```

```
37          dataset_path,
38          image_size=image_size,
39          color_mode='rgb',
40          batch_size=batch_size_dataset, # Agora usamos batch_size para o
                dataset
41          shuffle=True,
42          seed=seed
43          )
44
45
46          train_ds = full_dataset.take(train_size // batch_size_dataset)
47          val_ds = full_dataset.skip(train_size // batch_size_dataset)
48
49          def normalize_img(image, label):
50          return tf.cast(image, tf.float32) / 255.0, label
51
52          train_ds = train_ds.map(normalize_img)
53          val_ds = val_ds.map(normalize_img)
54
55          def one_hot_label(image, label):
56          return image, tf.one_hot(label, depth=2)
57
58          train_ds = train_ds.map(one_hot_label)
59          val_ds = val_ds.map(one_hot_label)
60
61
62          AUTOTUNE = tf.data.AUTOTUNE
63          train_ds = train_ds.cache().prefetch(buffer_size=AUTOTUNE)
64          val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
65
66
67          def create_model():
68          base_model = tf.keras.applications.ResNet50(
69          include_top=False,
70          weights='imagenet',
71          input_shape=(512,512, 3),
72          pooling='avg',
73          classes=2
74          )
75
76          for layer in base_model.layers:
77          layer.trainable = False
78
79          model = Sequential([
80          base_model,
81          Flatten(),
82          Dense(256, activation='relu'),
```

```
83          Dense(2, activation='softmax')
84          ])
85
86          model.compile(
87          optimizer='adam',
88          loss='categorical_crossentropy',
89          metrics=['accuracy']
90          )
91
92          return model
93
94          K.clear_session()
95          model = create_model()
96
97          history = model.fit(
98          train_ds,
99          epochs=30,
100         validation_data=val_ds,
101         verbose=1
102         )
103
104         # Previsoes
105         y_true = []
106         y_pred = []
107
108         for images, labels in val_ds:
109         preds = model.predict(images)
110         y_pred.extend(tf.argmax(preds, axis=1).numpy())
111         y_true.extend(tf.argmax(labels, axis=1).numpy())
112
113         acc = accuracy_score(y_true, y_pred)
114         f1 = f1_score(y_true, y_pred, average='macro')
115         var = np.var(y_pred)
116
117         print(f"\n Avaliacao apos 60 epocas:")
118         print(f" Acuracia: {acc:.4f}")
119         print(f" F1-score (macro): {f1:.4f}")
120         print(f" Variancia: {var:.4f}")
121
122         cm = confusion_matrix(y_true, y_pred)
123         labels = ['glaucoma', 'normal']
124         disp = ConfusionMatrixDisplay(confusion_matrix=cm,
               display_labels=labels)
125         disp.plot(cmap=plt.cm.Blues)
126         plt.title(" Matriz de Confusao")
127         plt.show()
```

APÊNDICE D – Quarto Apêndice

Código criado gerar as figuras 11 e 12:

```
epocas_list = list(range(10, 110, 10))
val_accuracies = []

for epocas in epocas_list:
K.clear_session()
model = create_model()

history = model.fit(
train_ds,
epochs=epocas,
validation_data=val_ds,
verbose=0
)

_, val_acc = model.evaluate(val_ds, verbose=0)
val_accuracies.append(val_acc)

print(f"Treinamento com {epocas} epocas concluido. Acuracia final na validacao: {val_acc:.4f}")


plt.figure(figsize=(10,6))
plt.plot(epocas_list, val_accuracies, marker='o')
plt.title('Acuracia Final na validacao vs Numero de epocas')
plt.xlabel('Numero de epocas')
plt.ylabel('Acuracia de validacao')
plt.grid(True)
plt.show()
```