Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Engenharia de Software

Implementação de um modelo baseado em visão computacional para detecção de retinopatia diabética em imagens de fundo de olho

Autor: Rômulo Vinícius de Souza

Orientador: Prof. Dr. John Lenon Cardoso Gardenghi

Brasília, DF

2023

Rômulo Vinícius de Souza

Implementação de um modelo baseado em visão computacional para detecção de retinopatia diabética em imagens de fundo de olho

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 UnB Gama - FGA

Orientador: Prof. Dr. John Lenon Cardoso Gardenghi

Brasília, DF

2023

Rômulo Vinícius de Souza
Implementação de um modelo baseado em visão computacional para detecção de retinopatia diabética em imagens de fundo de olho/ Rômulo Vinícius de Souza. – Brasília, DF, 2023-
77 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. John Lenon Cardoso Gardenghi

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2023.

1. retinopatia diabética. 2. rede neural convolucional. I. Prof. Dr. John Lenon Cardoso Gardenghi. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Implementação de um modelo baseado em visão computacional para detecção de retinopatia diabética em imagens de fundo de olho

CDU 02:141:005.6

Rômulo Vinícius de Souza

Implementação de um modelo baseado em visão computacional para detecção de retinopatia diabética em imagens de fundo de olho

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.

Trabalho aprovado. Brasília, DF, 20 de dezembro de 2023:

Prof. Dr. John Lenon Cardoso Gardenghi
Orientador

Prof. Dr. Glauco Vitor Pedrosa
Convidado 1

Prof. Dr. Tiago Alves da Fonseca
Convidado 2

Brasília, DF
2023

Agradecimentos

Agradeço aos meus amigos que me apoiaram durante a execução do trabalho, a minha mãe, que ajudou na revisão do texto e ao meu orientador, pela paciência e pelo direcionamento.

Resumo

Atualmente, diversos estudos vêm sendo realizados com a finalidade de automatizar o processo de identificação de doenças. Com a automatização, pode-se auxiliar os profissionais de saúde, visando diagnósticos mais precisos e eficientes dos pacientes. Dentre as diversas abordagens existentes para automatização desse processo, pode-se citar a visão computacional e a inteligência artificial. Esses são temas em ascensão, que são bastante utilizados em campos variados, como reconhecimento de objetos em imagens e vídeos. Ambos os temas são abordados de forma recorrente dentro da medicina, objetivando tanto a detecção quanto a classificação de doenças através de imagens. Este trabalho realiza um estudo de um método de detecção de retinopatia diabética (uma complicação resultante da diabetes), através de imagens do fundo do olho, visando automatizar o processo de detecção da doença, através do uso de uma rede neural convolucional.

Palavras-chaves: diabetes. retinopatia diabética. *deep learning*. rede neural convolucional.

Abstract

Nowadays, several studies are being carried out with the purpose of automating the process of diseases diagnosis. This automation assists healthcare professionals in obtaining more accurate and efficient diagnoses of patients. Among all the existing approaches to automating this process, we can mention the computer vision and artificial intelligence. Theses subjects are continuously growing in popularity and being used in various fields, such as objects recognition in images and videos. Both subjects are covered recurrently within medicine, with the main objective of classification of diseases through images. This work conducts a study of a method of detection of diabetic retinopathy (a complication that results from diabetes), through eye fundus images, with the purpose of automating the detection process of the disease, through the use of a convolutional neural network.

Key-words: diabetes. diabetic retinopathy. deep learning. convolutional neural network.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ANN	*Artificial Neural Network*
AUC	*Area Under the Curve*
CLAHE	*Constrast Limited Adaptative Histogram Equalization*
CNN	*Convolutional Neural Network*
DL	*Deep Learning*
FCNN	*Fully Connected Neural Network*
HOG	*Histogram of Oriented Gradients*
IA	Inteligência Artificial
ML	*Machine Learning*
MLP	*Multilayer Perceptron*
MSR	*Mean Squared Error*
ROC	*Receiver Operating Characteristic*
SGD	*Stochastic Gradient Descent*
SURF	*Speeded-up Robust Features*
SVG	*Support Vector Machine*
TCC	Trabalho de Conclusão de Curso

Sumário

1 Introdução

A diabetes é uma doença crônica grave, de longa duração e causadora de um grande impacto nas vidas e no bem-estar de indivíduos ao redor do mundo. Ela está entre as principais causas de morte em adultos (SAEEDI et al., 2019). Todos os tipos de diabetes podem resultar em complicações em diversas partes do corpo e aumentar o risco de uma morte prematura. Além disso, a diabetes e suas complicações ocasionam um grande impacto econômico às pessoas com a doença e a seus familiares, aos sistemas de saúde e à economia nacional (TIME, 2016). Estima-se que em 2017 causou quatro milhões de mortes ao redor do mundo, o que resultou em gastos em torno de 727 bilhões de dólares (SAEEDI et al., 2019).

Com o aumento de doenças crônicas e o envelhecimento da população, fez-se necessário cada vez mais avanços na ciência e tecnologia. Dentre as diversas evoluções recentes, pode-se citar a inteligência artificial (IA), que está em constante ascenção. A inteligência artificial foi desenvolvida com o objetivo de criar máquinas com a capacidade de imitar a inteligência humana e aprender. O uso de inteligência artificial e os seus ramos abrangem diversos campos de estudo, como medicina, biomanufatura, modelagem molecular, diagnóticos automatizados e outros (AKAY; HESS, 2019).

Uma das complicações da diabetes que afeta a população é a retinopatia diabética, que está entre as maiores causadoras de cegueira no mundo. O aumento da expectativa de vida da população, o estilo de vida precário e outros fatores contribuem para o aumento contínuo do número de pessoas com a doença. Apesar disso, a retinopatia pode ser tratada de forma efetiva, caso seja identificada nos estágios iniciais (PRATT et al., 2016).

1.1 Justificativa

O diagnóstico manual da retinopatia está propenso a erros. Com o envelhecimento da população, a procura por especialistas da doença está cada vez maior. Os métodos automatizados para detecção de retinopatia diabética são mais rápidos e baratos, o que os tornam mais eficientes que diagnósticos manuais (SCOTLAND et al., 2010). A proposta deste trabalho é estudar e implementar um método de detecção de retinopatia diabética através de imagens da retina, com o uso de redes neurais.

1.2 Objetivos

O objetivo geral deste trabalho é estudar e implementar o método de classificação de imagens do fundo do olho para detecção de retinopatia diabética, com base no trabalho de Gama, Coelho e Baffa (2020).

Os objetivos específicos deste trabalho, fortemente apoiados no desenvolvimento proposto por Gama, Coelho e Baffa (2020), são:

- implementar o método de classificação de imagens para detecção de retinopatia diabética;
- realizar experimentos com um conjunto de dados;
- melhorar o método implementado ou propor outro método.

1.3 Estrutura do Trabalho

Este trabalho será dividido em 4 capítulos. O Capítulo 1 apresenta uma introdução a que se refere o trabalho. O Capítulo 2 apresenta conceitos importantes para o entendimento geral do trabalho. O Capítulo 3 apresenta os passos necessários para a preparação e a execução da rede neural avaliada. O Capítulo 4 apresenta os resultados obtidos após execução da metodologia proposta. O Capítulo 5 mostra as considerações finais do trabalho após a análise de resultados.

2 Referencial Teórico

Neste capítulo, serão apresentados conceitos importantes para o entendimento geral do método proposto e trabalhos correlatos.

2.1 Diabetes

Diabetes é uma doença crônica que ocorre quando o pâncreas não produz insulina (hormônio regulador de glicose no sangue) suficiente ou quando o corpo não consegue utilizar a insulina produzida de forma eficiente, sendo esses dois casos categorizados como diabetes tipo 1 e diabetes tipo 2, respectivamente. O aumento de glicose no sangue (uma característica comumente apresentada por diabetes não controlada) pode, no decorrer do tempo, resultar em sérias lesões em alguns órgãos, como os olhos, o coração, os vasos sanguíneos, dentre outros. Dentre as lesões que podem ocorrer nos olhos, pode-se citar a retinopatia diabética, que é o caso de estudo deste trabalho (TIME, 2016).

2.1.1 Retinopatia Diabética

A retinopatia diabética é uma complicação microvascular específica da diabetes, resultado da acumulação de danos nos capilares da retina, no decorrer do tempo (TIME, 2016). A retinopatia é considerada uma dos maiores causadores de cegueira em pessoas em idade ativa. A predominância da retinopatia diabética aumenta conforme a duração da diabetes. A grande maioria daqueles afetados por diabetes do tipo 1 e mais de 60% daqueles afetados por diabetes do tipo 2 possuem alguma retinopatia após 20 anos (MOHAMED; GILLIES; WONG, 2007).

A diabetes é conhecida por afetar muitos tipos de células da retina, mas, devido à vasculatura interna da retina ser amplamente visível, a classificação da retinopatia é baseada na severidade das lesões vasculares. O método de identificação de retinopatia comumente utilizado baseia-se na análise direta de características anatômicas da retina e no número de lesões microvasculares detectáveis fotograficamente (STITT et al., 2016). A Figura 1 mostra uma retina com retinopatia diabética, retirada da base de dados Decencière et al. (2014).

Devido ao grande número de pacientes com diabetes no mundo, o processo de identificação manual pode ser caro e demorado. Além disso, o diagnóstico da gravidade da retinopatia e a detecção precoce da doença são subjetivos, variando substancialmente entre especialistas treinados. A fim de mitigar esses problemas, foi proposto o uso de soluções automatizadas para realização do diagnóstico através das imagens do fundo do

olho (GARGEYA; LENG, 2017). Dentre as diversas soluções propostas, pode-se citar o uso de aprendizado de máquina através de redes neurais.

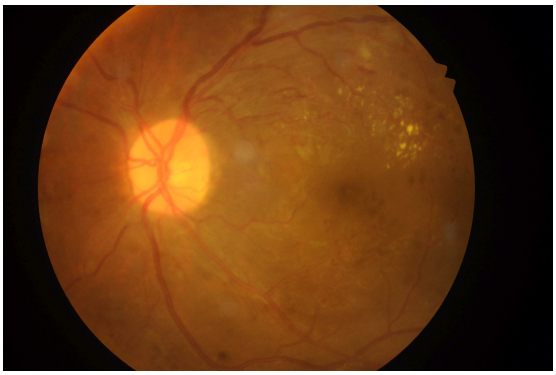


Figura 1 – Retina com retinopatia

Fonte: Base de dados Decencière et al. (2014)

2.2 Visão Computacional

Os seres humanos são capazes de compreender o mundo ao seu redor em uma perspectiva de três dimensões. A visão computacional tem como objetivo desenvolver formas matemáticas de se obter essa percepção de dimensões e aparência de objetos através de imagens. Já existem aplicações capazes de acompanhar o movimento de pessoas, identificar pessoas através de uma combinação de características do rosto, das roupas, dentre outros.

Apesar disso, obter-se a mesma precisão que humanos em computadores é um grande desafio, quando se diz respeito a interpretar uma imagem, pois, enquanto humanos realizam essa tarefa de forma mais intuitiva, expressar essa complexidade de informação presente no mundo para modelos físicos e probabilísticos é uma tarefa árdua (SZELISKI, 2022).

A visão computacional avançou de aplicação de abordagens clássicas de reconhecimento e técnicas de processamento de imagem, como na Figura 2, para aplicações avançadas capazes de entender imagens, construir modelos baseados na visão e criar seu próprio sistema com capacidade de aprendizado. A capacidade de aprendizado é um dos temas que mais recebem atenção, na atualidade, e pode ser obtido através da integração de técnicas de aprendizagem de máquina no ramo da visão computacional (SEBE, 2005). Um exemplo de uma aplicação avançada, com o aproveito dessa integração entre as áreas, pode ser observado na Figura 3, a qual mostra identificação de objetos em imagens.

O número de aplicações de visão computacional é grande, mas grande parte dessas aplicações realizam os mesmos passos para realizar todo o processamento da imagem: obtenção dos dados de entrada (imagens, vídeos); etapas de preparação das imagens (nor-

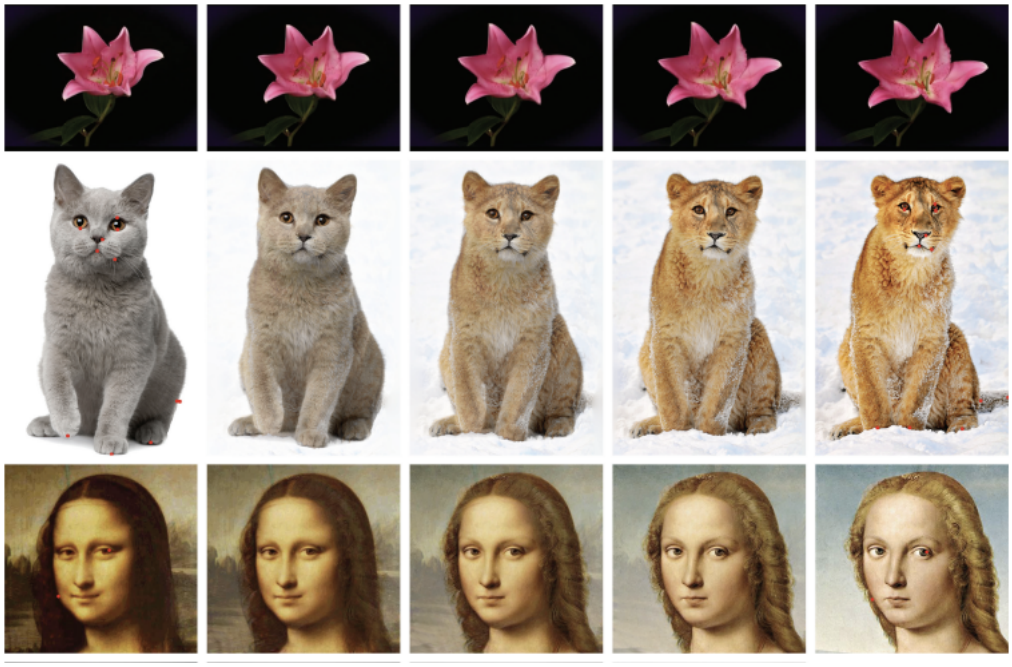


Figura 2 – Exemplo de transformações de imagem por similaridade

Fonte: Liao et al. (2014)

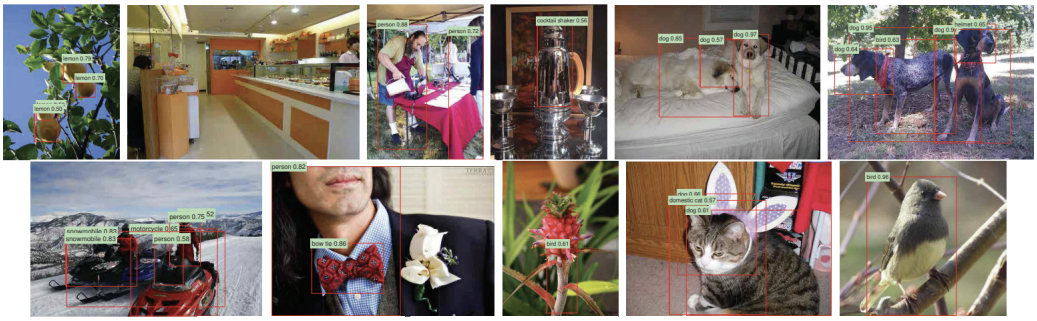


Figura 3 – Exemplo de identificação de objetos por uma R-CNN

Fonte: Girshick et al. (2015)

malização, transformações de cores); extração de características e, por fim, o aprendizado, seja este uma predição ou uma classificação (ELGENDY, 2020).

2.2.1 Dados de Entrada

Para visão computacional, os dados de entrada podem ser imagens ou vídeos. Uma imagem pode ser definida como uma função de duas variáveis (x,y), resultando em uma área de duas dimensões, em que cada ponto (x,y) representa um pixel da imagem. Dessa forma, toda imagem é formada por um conjunto de pixels, e cada valor de pixel representa a intensidade de luz presente naquele ponto da imagem. Imagens em escala de cinza possuem um valor para cada pixel, como pode ser observado na Figura 4b, enquanto imagens RGB possuem três valores para representarem cada pixel, correspondentes às

(a) (b)

Figura 4 – Imagem em escala de cinza e sua representação em pixels

intensidades de vermelho, verde e azul. Outra forma de visualizar uma imagem com os três canais (RGB) é pensar em três matrizes com a mesma quantidade de pixels entre elas, sendo que cada matriz equivale a um dos três canais (ELGENDY, 2020).

2.2.2 Preparação das Imagens

Técnicas de pré-processamento são muito utilizadas para melhorar a eficiência de modelos de visão computacional. O artigo de Pal e Sudeep (2016) mostra o impacto da etapa de pré-processamento na acurácia de três modelos diferentes de redes neurais convolucionais. No artigo, foram aplicadas três diferentes técnicas para a preparação das imagens: normalização, cálculo do desvio padrão e análise de componente zero. Existe uma grande variedade de filtros e funções que podem ser aplicados para destacar certas características e melhorar a qualidade da imagem. No artigo de Setiawan et al. (2013), é mostrada a aplicação de uma equalização adaptativa do histograma limida por contraste (*Contrast Limited Adaptative Histogram Equalization* - CLAHE), para aprimorar as cores das retinas. Na Figura 5b, podemos observar a imagem de Figura 5a após ser aplicada a equalização do histograma.

2.2.3 Extração de Características

Nessa etapa, são extraídas das imagens as características relevantes para a classificação. Nos modelos tradicionais de aprendizagem de máquina, há a atuação conjunta entre o responsável pela aplicação e pessoas especializadas na identificação de determinadas características. Alguns algoritmos tradicionais utilizados para extração de características são: histograma de gradientes orientados (*Histogram of Oriented Gradients* - HOG), transformação de recursos invariantes em escala (*Scale-invariant Feature Transform* - SIFT), recursos robustos acelerados (*Speeded-up Robust Features* - SURF). Nos modelos mais modernos, também chamados de aprendizado profundo, a extração das características não é

(a) (b)

Figura 5 – Imagem em escala de cinza e sua resultante após aplicação do CLAHE

realizada de forma manual. Nesses modelos, o processo de extração ocorre de forma automática, com a utilização de redes neurais. As redes extraem características e aprendem a importância de tal para o resultado de saída (ELGENDY, 2020).

2.2.4 Predição ou Classificação

A última etapa do processo geral é a classificação. Aqui podem ser usados modelos de aprendizagem tradicionais, como a máquina de vetores de suporte (*Support Vector Machine* - SVM) ou as redes neurais artificiais (*Artificial Neural Network* - ANN). Também podem ser usados modelos mais recentes, como redes neurais convolucionais (*Convolutional Neural Network* - CNN). Quando comparadas, as redes neurais convolucionais se mostram mais eficientes, no contexto de identificação/classificação de imagens, do que as redes neurais artificiais. No estudo feito por Shah et al. (2023), foi realizada uma comparação entre ambas as redes para a detecção de câncer de pele.

2.3 Aprendizado de Máquina

Com o advento da inteligência artifical, foram solucionados problemas que poderiam ser descritos através de modelos matemáticos formais. Com isso, o verdadeiro desafio passou a ser solucionar problemas fáceis para humanos realizarem, mas difíceis para serem descritos formalmente (problemas resolvidos intuitivamente), como reconhecimento de faces em imagens. Para isso foi, então, criado o aprendizado de máquina (*Machine Learning*) (GOODFELLOW; BENGIO; COURVILLE, 2016).

Aprendizado de máquina é um ramo da inteligência artificial que constrói um modelo matemático baseado em um conjunto de dados, com a finalidade de realizar predições ou decisões sem ter sido programado para realizar tal tarefa (ZHANG, 2020). Esse conjunto de dados pode ser especificado pelo homem, com o propósito de realizar

o aprendizado, ou pode ser obtido através de interações com o ambiente em que se está executando a máquina. Em ambos os casos, a qualidade e o tamanho dos dados são cruciais para o sucesso das predições realizadas pela máquina (MOHRI; ROSTAMIZADEH; TALWALKAR, 2018).

Técnicas de aprendizado de máquina convencionais eram limitadas a processar os dados em sua forma bruta. Durante décadas, a construção de sistemas de aprendizado de máquina requeria um domínio considerável no assunto para realizar a extração das características de um dado bruto e transformá-las em uma representação interna adequada para detecção ou classificação (LECUN; BENGIO; HINTON, 2015).

2.3.1 Rede Neural Artificial

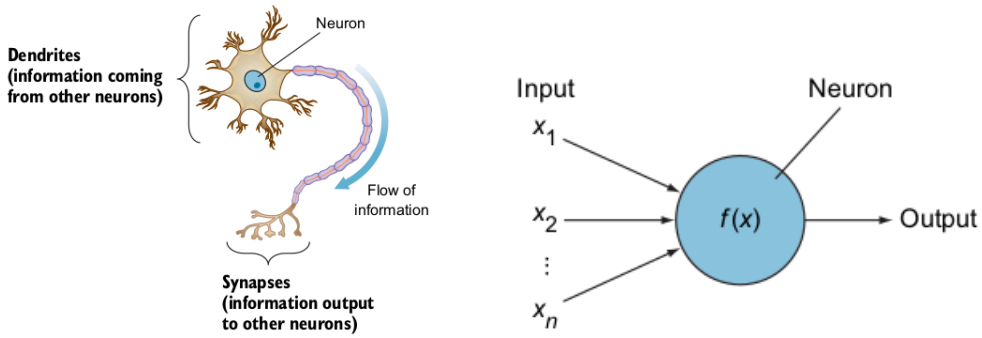
As redes neurais artificiais surgiram com o intuito de replicar a capacidade de aprendizado do cérebro humano, recriando a forma como são feitos os processamentos de sinais pelos neurônios do cérebro. Aplicando-se algoritmos que simulam os processos de neurônios verdadeiros, é possível criar uma rede capaz de aprender a resolver diversos problemas (KROGH, 2008).

2.3.1.1 *Perceptron*

O *perceptron* é unidade básica da rede neural artificial, ou a ANN mais simples, composta por um único neurônio. Conceitualmente, o *perceptron* atua de forma semelhante a um neurônio biológico. Um neurônio biológico recebe sinais através de seus dendritos, modula esse sinal e, então, o manda para o próximo neurônio através de sinapses, caso o valor do sinal ultrapasse certo limite. O neurônio artificial recebe vários valores de entrada, com pesos atribuídos a cada um deles, realiza a soma de todos esses valores, aplica uma função de ativação, verifica se o resultado obtido está de acordo com o esperado e, então, calcula-se o erro. Dessa forma, os pesos são ajustados de acordo com o erro e o resultado é mandado para o próximo neurônio. Esse processo se repete entre vários neurônios (ELGENDY, 2020). Na Figura 6b, é possível comparar o funcionamento entre um neurônio artificial e um biológico, que está representado na Figura 6a.

2.3.1.2 *Multilayer Perceptron*

A rede neural artificial consiste de camadas compostas por neurônios, por isso também é chamada de *Multilayer Perceptron*. Dentre essas camadas, existem a camada de entrada (*input*), as camadas escondidas (*hidden layers*) e a camada de saída (*output*). A arquitetura de uma ANN está representada em Figura 7. A camada de entrada é onde entram as características extraídas, em forma de vetor (*feature vector*). As camadas escondidas são responsáveis por fazer o processo do somatório de pesos e aplicação de função de ativação entre todos os *perceptrons*. Nessa camada, é onde ocorre o reconhecimento de



(a) Neurônio biológico (b) Neurônio artificial

Figura 6 – Representação de um neurônio artificial, baseado em um neurônio biológico.

Fonte: Elgendy (2020)

padrões entre as características extraídas. Todos os neurônios de uma camada se comunicam com todos os neurônios da próxima camada. Essa configuração, de nós conectados, resultou em ainda outro nome para denominar essa rede neural, chamado rede neural totalmente conectada (*Fully Connected Neural Network* - FCNN). A camada de saída é onde ocorre a resposta/predição para o problema esperado (WANG; WANG, 2003).

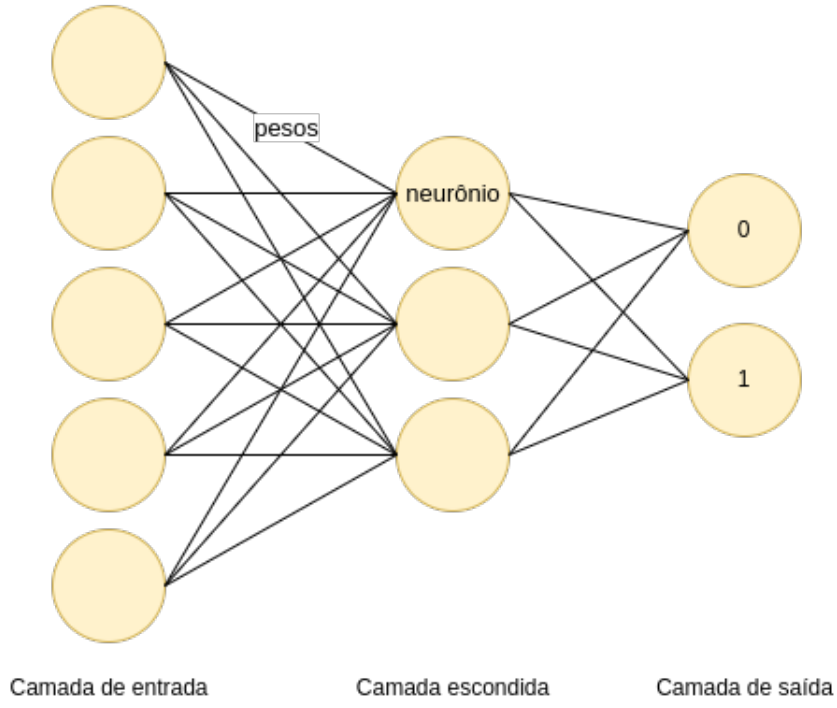
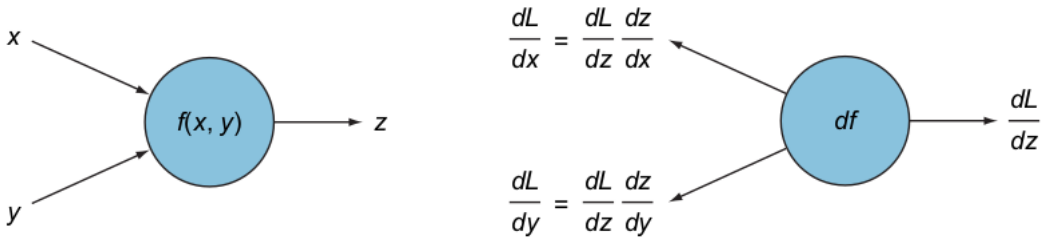


Figura 7 – Arquitetura rede neural artificial

2.3.1.3 Função de Ativação

A função de ativação é responsável por introduzir uma não linearidade à rede neural, transformando uma combinação linear de uma média ponderada em um modelo não linear. Ela é adicionada após cada neurônio, para determinar se este será ou não ativado



(a) *Feedforward* (b) *Backpropagation*

Figura 8 – Sentido de propagação de dados e erros na rede.

Fonte: Elgendy (2020)

(GUPTA et al., 2013). Existem diversos tipos de função de ativação, cada um usado para um propósito diferente. A função de transformação linear (função identidade), por exemplo, mantém o sinal de saída do neural idêntico ao de entrada, o que mantém a linearidade da função. A função retificadora (*Rectified Linear Unit* - ReLU) ativa o neurônio apenas se o valor for positivo, mantendo como zero os valores de entrada negativos. Essa função apresenta bons resultados quando utilizada nas camadas escondidas da rede. Outro exemplo é a função sigmoid, que é muito utilizada na camada de saída em classificações binárias, pois ela é capaz de prever a probabilidade de uma classe dentre duas possibilidades (ELGENDY, 2020).

2.3.1.4 *Feedforward*

O processo dos dados de entrada serem transmitidos por todos os neurônios, realizando os cálculos dos pesos, é chamado de *feedforward*. O nome *feedforward* foi atribuído pois um dado de entrada sempre resulta em um mesmo grupo de dado de saída. Esse processo é feito de forma unilateral, sem a presença de ciclos ou retroalimentação durante o processo, como pode ser observado na Figura 8a (GUPTA et al., 2013).

2.3.1.5 Funções de Erro

Durante o processo de aprendizado, o dado de saída gerado pela rede, também conhecido como *label*, pode não ser igual ao esperado. A rede precisa de alguma forma de reajustar seus pesos para que os dados de saída sejam mais próximos dos esperados, e que o erro total convirja para o menor valor possível. As funções de erro são responsáveis por estimar o quão distante a rede está do resultado real. O decaimento das métricas de erro (*loss* e *val_loss*) sinalizam que a rede neural está melhorando e, se o valor de erro chegasse a zero, sinalizaria a rede ideal (KROGH, 2008). Alguns exemplos de funções de erro são: erro quadrático médio (*Mean Squared Error* - MSE), utilizada em problemas de regressão que requerem que a saída seja um valor real; entropia cruzada (*Cross-entropy*), utilizada em problemas de classificação, pois consegue quantificar a diferença entre duas

distribuições de probabilidade (ELGENDY, 2020).

2.3.1.6 Algoritmos de Otimização

Algoritmos de otimização são usados em conjunto com as funções de erro, com o objetivo de obter o menor erro possível. Caso a rede neural tentasse calcular todas as probabilidades, variando os pesos em todas as combinações de nós possíveis, seria necessário muito tempo para que ela realizasse todo o processo, dependendo do número de nós da rede. Dessa forma, os algoritmos de otimização são responsáveis por melhorar a função de erro, atualizando os possíveis pesos aplicados aos nós, em busca do melhor resultado (ELGENDY, 2020). Alguns exemplos de algoritmos de otimização são: método do gradiente estocástico (*Stochastic Gradient Descent* - SGD), o qual escolhe pontos aleatórios para percorrer o gradiente descendente; Adadelta, que é um metodo que se adapta dinamicamente no decorrer do tempo e demanda pouco recurso computacional; estimativa de momento adaptativo (*Adaptive Moment Estimation* - ADAM), que é popular por aplicar uma taxa de aprendizado adaptativa e com bom desempenho (BAE; RYU; SHIN, 2019).

2.3.1.7 *Backpropagation*

A retropropagação do erro (*backpropagation*) é uma técnica utilizada em *feedforward* ANNs. Em *feedforward* ANNs, os sinais são enviados sempre em uma única direção, para frente. Dessa forma, é necessário um método para que o erro obtido no decorrer da rede seja analisado e aproveitado na próxima execução, o *backpropagation*. Com esse método, os erros obtidos durante a época (execução) são propagados no sentido contrário dos sinais, sendo calculado a derivada do erro em relação a cada peso, para cada nó, até retornar ao começo da rede, onde os pesos são reajustados para a próxima execução (época). Na Figura 8b, é possível observar o sentido de propagação dos erros (ELGENDY, 2020).

2.4 Aprendizado Profundo

Aprendizado profundo (*Deep Learning*) é um ramo do aprendizado de máquina, o qual difere principalmente na forma de extração das características de um dado de entrada, independente do tipo (EDUCATION, 2020).

Entende-se por aprendizado de representação um conjunto de métodos que permitem uma máquina a receber um dado bruto e, automaticamente, descobrir as características necessárias para detecção ou classificação. Aprendizado profundo consiste em métodos de aprendizado de representação com muitos níveis/camadas (começando pelo dado bruto). A cada camada, são extraídas novas características e gerada uma nova representação do dado para a próxima camada. Pode-se dizer que o dado é transformado

a cada camada, e a composição dessas transformações permite o aprendizado de funções complexas. O principal aspecto do aprendizado profundo é que as formas de representação não são definidas pelo homem, mas sim aprendidas através do próprio dado de entrada, utilizando um procedimento de aprendizado com propósito geral (LECUN; BENGIO; HINTON, 2015).

2.4.1 Rede Neural Convolucional

Rede neural convolucional, conhecida como *Convolutional Neural Network* (CNN), é uma das redes neurais mais famosas dentro do *Deep Learning*. Seu nome foi dado devido à operação matemática linear entre matrizes, chamada convolução. A CNN é composta por diversas camadas, como a camada de convolução, a camada de *pooling* e a camada totalmente conectada. A CNN tem um excelente desempenho em problemas de aprendizado de máquina, especialmente aqueles relacionados a imagens, à visão computacional e ao processamento de linguagem natural (ALBAWI; MOHAMMED; AL-ZAWI, 2017).

As CNNs estão em uma das etapas do processo principal de visão computacional. Elas foram desenvolvidas com o objetivo de realizar a extração de características automaticamente, sem a atuação humana, conforme citado na Seção 2.2.3.

Uma das grandes vantagens da CNN com relação às outras redes neurais é a redução do número de parâmetros (ALBAWI; MOHAMMED; AL-ZAWI, 2017). Ao contrário das outra redes neurais, a CNN é capaz de lidar com dados de entrada com mais de uma dimensão, como duas dimensões para processar uma imagem ou três dimensões para processar um vídeo (LECUN; BENGIO; HINTON, 2015).

2.4.1.1 Camadas de Convolução

Na matemática, convolução é uma integral que expressa a quantidade de sobreposição de uma função à medida que ela é deslocada sobre outra função. No contexto das CNNs, a primeira função é uma imagem de entrada, a segunda função é um filtro de convolução, também chamado de kernel, e a imagem resultante após a aplicação do filtro é o resultado da convolução. As camadas de convolução são as camadas principais de uma rede neural convolucional. Elas atuam como uma janela que percorrem a imagem, pixel por pixel, para extrair características relevantes que representam os objetos na imagem (O'SHEA; NASH, 2015).

Enquanto rede neural artificial calcula a média ponderada de todos os valores de entrada e cada um dos nós da rede, a CNN realiza um processo semelhante, de forma espacial. Esse cálculo de pesos é feito em uma forma matricial. O filtro, que é uma matriz, desliza sobre a imagem, deslocando-se uma unidade predeterminada (*stride*). A cada posição que o filtro é deslocado, é realizada uma operação de multiplicação pixel a pixel entre

a região da imagem original e o filtro. Os valores resultantes são somados, resultando em um novo pixel na imagem resultante. Ao final da convolução, a imagem resultante terá suas dimensões de largura e altura reduzidas, dependendo do valor dos *strides* e dos *paddings*. Essa imagem resultante é chamada de mapa de característica (*feature map*) (ELGENDY, 2020).

2.4.1.2 *Strides* e *Padding*

Strides são as unidades deslocadas pelo filtro, durante a convolução. O *padding* é utilizado para realizar o preenchimento das bordas da imagem com algum valor escolhido. Um dos casos de uso de *padding* é manter o mesmo tamanho da imagem após a convolução. A Figura 9 ilustra como é o processo de convolução e a Figura 10 mostra como o kernel se desloca em uma imagem.

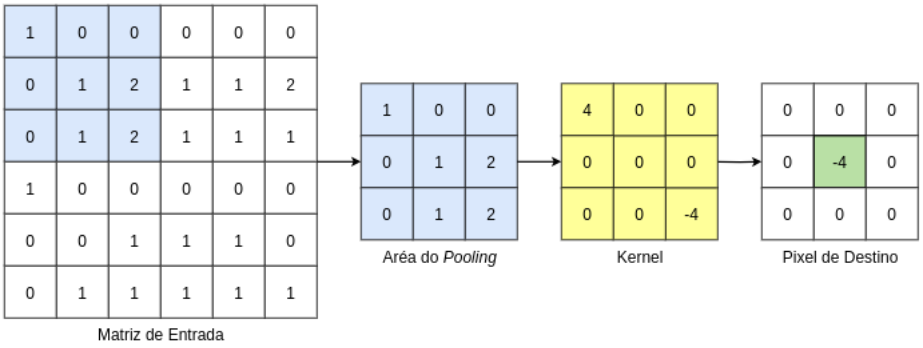


Figura 9 – Convolução em uma imagem

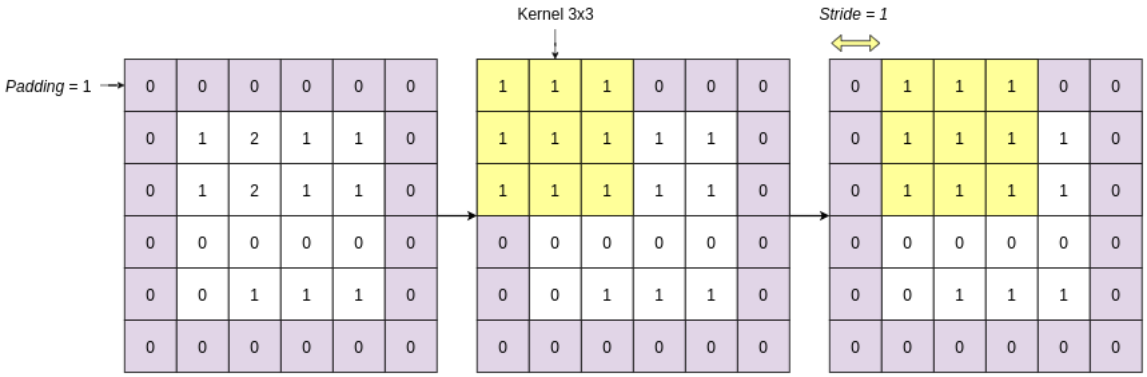


Figura 10 – Movimentação do kernal em uma imagem

2.4.1.3 Camadas de *Pooling*

As camadas de *pooling* são utilizadas para reduzir as dimensões dos mapas de características gerados pelo kernel. É um processo semelhante à convolução, em que o kernel percorre as matrizes, porém com um *stride* maior, resultando em uma amostra reduzida. O número de *feature maps* é mantido, o que não altera a profundidade da rede, mas é reduzido o total de parâmetros da rede. É necessário cuidado ao se utilizar das

camadas de *pooling*, pois redução de parâmetros pode implicar em redução de informação, impactando o desempenho da rede (O'SHEA; NASH, 2015).

2.4.1.4 Camadas Totalmente Conectadas

As últimas camadas da CNN são as camadas totalmente conectadas. Conforme explicado na Seção 2.2.3, elas são compostas por grupamentos de neurônios e serão responsáveis pelo resultado de saída, seja ele uma classificação ou predição. Entre as camadas de convolução e as camadas totalmente conectadas, há uma camada chamada *flatten*. Essa camada é responsável por transformar os mapas de características, que são matrizes de duas dimensões, em um vetor de uma dimensão. Esse vetor, por fim, é utilizado para alimentar a primeira camada da ANN (ELGENDY, 2020).

2.4.1.5 *Overfitting* e *Underfitting*

Overfitting e *Underfitting* são os principais fatores que ocasionam um baixo desempenho em aprendizado de máquina. *Underfitting* ocorre quando o modelo treinado falha em representar os dados durante o treinamento, o que indica que o modelo é muito simples para entender o problema em questão. *Overfitting*, por outro lado, ocorre quando o modelo entende muito bem os dados, o que pode ser interpretado como o modelo memorizando os possíveis resultados, e não aprendendo a generalizar uma solução para o problema em questão (ELGENDY, 2020).

2.5 Trabalhos Correlatos

Nos dias atuais, muitos estudos são feitos a respeito da inteligência artifical, visando sempre o aperfeiçoamento das técnicas para criação das bases de dados e para processamento destas. Esta seção descreve alguns dos estudos realizados, focados na identificação de retinopatia diabética.

O trabalho de Safi et al. (2018) realiza um estudo detalhado sobre o uso de *deep learning* no contexto de retinopatia diabética. O estudo apresenta as características necessárias a serem extraídas da retina para identificação da doença, bases de dados populares em estudos semelhantes, possíveis métricas a serem utilizadas ao se criar um modelo, entre outros passos necessários para detectar a doença através de redes neurais convolucionais. O estudo também se destaca por descrever técnicas de pré-processamento usadas por outros estudos, como a extração do canal verde, normalização de dados, redução de ruídos através de aplicação de filtro Gaussiano. Além disso, também são dispostas métricas coletadas de algumas redes neurais populares em algumas das bases de dados previamente listadas.

No estudo feito por Hemanth, Deperlioglu e Kose (2020), foi proposta uma rede neural convolucional para detecção de retinopatia, utilizando a base de dados do Decencière et al. (2014). Nesse estudo, foi avaliado o desempenho da rede proposta a cada etapa de um pré-processamentos na base de dados. No primeiro momento, as imagens foram redimensionadas para 150x225 pixels. Depois, as imagens foram separadas nos três canais R, G e B e, então, foram aplicadas as técnicas de equalização de histograma e CLAHE em cada um dos canais. No final, os canais foram reagrupados. A rede apresentou as acurácias de 90,75%, 94,33% e 97% para os estágios de redimensionamento, aplicação da equalização do histograma, aplicação do CLAHE, respectivamente.

O método proposto por Pratt et al. (2016) se destaca por utilizar um modelo de uma CNN para apresentar uma classificação não binária sobre retinopatia diabética. Essa classificação pode ser definida como cinco estágios: sem retinopatia, retinopatia branda, moderada, severa ou proliferativa. Para treinar o modelo, foi utilizada uma base de dados com 80000 imagens obtidas no *Kaggle*. A padronização das imagens da base de dados foi relativamente simples, apenas realizando uma normalização das cores com a biblbioteca OpenCV e o redimensionamento das imagens para 512x512 *pixels*. Já o método utilizado para o treino foi de grande complexidade. Inicialmente, um conjunto de 10290 imagens foi utilizado até obter um nível de aprendizado considerável. Essa etapa inicial totalizou cerca de 350 horas de treinamento e, somente então, foi realizado o treinamento com a base de dados em sua totalidade, usando técnicas de aumento de dado (PRATT et al., 2016).

Para validação dos resultados, foram separadas 5000 imagens da base de dados. A execução dessas imagens na rede neural teve duração de 188 segundos. Foi definido como acurácia o número de pacientes classificados em um dos cinco estágios corretamente, obtendo o valor de 75%. Além disso, foi obtido 95% de especificidade e 30% de sensitividade (PRATT et al., 2016).

3 Metodologia

Neste capítulo, será apresentada a metodologia proposta para a execução do trabalho. A metodologia proposta se baseia no artigo apresentado por Gama, Coelho e Baffa (2020).

3.1 Base de Dados

Para o treinamento do modelo e a relização dos testes, foram escolhidas duas bases de dados, IDRiD (PORWAL et al., 2018) e MESSIDOR (DECENCIèRE et al., 2014).

3.1.1 IDRiD

A base de dados IDRiD consiste em 516 imagens, classificadas manualmente por oftalmologistas e divididas em dois grupos: retinas que apresentam retinopatia diabética e retinas saudáveis. As imagens são feitas de cortes em regiões aleatórias da retina e estão em dimensões variadas (PORWAL et al., 2018).

3.1.2 MESSIDOR

A base de dados MESSIDOR é composta por 1200 imagens, obtidas através de 3 departamentos de oftalmologia, por meio de câmeras com a mesma configuração. Dessas imagens, 800 foram realizadas com a pupila do paciente dilatada, enquanto 400 foram realizadas com a pupila sem dilatação. A base de dados é dividida em 3 conjuntos com 400 imagens cada, um conjunto para cada departamento. Nessa base, as imagens são apresentadas com graus de retinopatia (DECENCIèRE et al., 2014). Ao contrário da base Porwal et al. (2018), as imagens foram realizadas sem um corte específico, sendo possível visualizar a retina em sua totalidade. As dimensões das imagens variam entre 1440x960, 2240x1488 ou 2304x1536 *pixels*.

3.2 Recursos Utilizados

3.2.1 Infraestrutura

Para desenvolvimento do método proposto, foi utilizado um notebook Dell Alienware m16, com um processador 13ª geração Intel® Core™ i9-13900HX, com 24 núcleos e 32 threads, placa de vídeo NVIDIA® GeForce® RTX™ 4070, 8GB GDDR6 e 32GB DDR5 de memória RAM.

3.2.2 Tecnologias Utilizadas

3.2.2.1 Python

O Python é uma linguagem de programação que permite o desenvolvimento e a integração de sistemas de forma rápida. No contexto de inteligência artificial e visão computacional, é uma das linguagens mais populares, pois apresenta muitas ferramentas que são simples de serem utilizadas. Os projetos implementados nesse artigo serão todos feitos no Python, na versão 3.9.18, valendo-se de suas inúmeras bibliotecas.

3.2.2.2 Opencv-python

O opencv-python é uma biblioteca do Python, com os pacotes do OpenCV pré-compilados, a qual é muito utilizada no contexto de visão computacional. Além de ter seu código aberto, possui mais de 2500 algoritmos disponíveis. No projeto, o OpenCV, na versão 4.8.1.78, foi a principal biblioteca utilizada no pré-processamento de imagens.

3.2.2.3 Keras e Tensorflow

O Tensorflow é um *software* de código aberto, uma biblioteca para realizar tarefas de aprendizagem de máquina. O Keras é uma biblioteca do Python de mais alto nível (mais receptiva para humanos), executada em cima do Tensorflow, para criação de redes neurais. Enquanto o Tensorflow é feito em C++, o Keras permite a comunicação com a API do Tensorflow através do próprio Python. No projeto, as duas bibliotecas foram usadas na versão 2.13.1.

3.2.2.4 Matplotlib

O Matplotlib é uma biblioteca para criar visualizações estáticas, animadas ou interativas em Python. No projeto, a biblioteca foi utilizada, na versão 3.8.1, a fim de gerar gráficos auxiliares para análise do problema proposto.

3.2.2.5 Pandas

O Pandas é uma biblioteca do Python que facilita o manuseio de dados relacionais ou categorizados. No projeto, foi utilizada a versão 2.1.2 para realizar a leitura de arquivos do tipo CSV e manipulação dos dados contidos nestes.

3.2.2.6 *Scikit-Learn*

O *scikit-learn* é uma biblioteca utilizada no contexto de análise de dados. Ela disponibiliza diversas ferramentas para manipulação de dados, auxiliando no processo de aprendizado de máquina. No projeto, foi utilizada a versão 1.3.2 para realizar a separação entre bases de dados de treinamento e validação.

3.3 Pré-processamento de imagens

Imagens de fundo de olho podem variar de acordo com vários aspectos, como a idade do paciente, sua etnia ou até mesmo as configurações da câmera utilizada. Essas variações podem resultar em características específicas, como um aspecto borrado em uma imagem, ou uma retina com pigmentação mais escura em outra. Essas características não auxiliam na diferenciação entre uma retina saudável e uma que apresenta retinopatia e acabam por dificultar o processo de classificação (GAMA; COELHO; BAFFA, 2020).

Na Tabela 1, pode-se observar a acurácia média de diferentes arquiteturas de Redes Neurais Convolucionais para a base de dados não pré-processada. Apesar de serem arquiteturas sofisticadas e recentes, elas não apresentaram bons resultados. Dessa forma, será realizado um pré-processamento das imagens, para padronizá-las e destacar as estruturas da retina que são, de fato, relevantes para a detecção da doença.

Tabela 1 – Comparativo entre a acurácia de três diferentes CNNs. Fonte: Gama, Coelho e Baffa (2020)

Arquitetura	**Acurácia Média**	**Imagens Processadas?**
LeNet	55%	Não
VGG	45%	Não
ResNet	51%	Não

O pré-processamento abordado foi o descrito por Gama, Coelho e Baffa (2020), que é composto por 4 etapas: extração do canal verde, equalização adaptativa de historama, cálculo do negativo, recorte da região de interesse.

3.3.1 Extração do Canal Verde

A fim de reduzir os efeitos da cor da retina no processo de busca por padrões, foram separados os canais de cores das imagens e, assim, foi utilizado o canal verde. Ao se utilizar o canal verde, é possível obter mais detalhes sobre as estruturas da retina, como o disco óptico e os vasos sanguíneos, além de destacar anomalias presentes (GAMA; COELHO; BAFFA, 2020).

3.3.2 Equalização Adaptativa de Histograma

Devido ao fato de algumas imagens serem mais escuras ou mais claras que outras, foi realizado um processo de equalização do histograma. A fim de evitar perda de informação na imagem, o método de equalização abordado será o Equalização de Histograma Adaptativa Limitado pelo Contraste (*Contrast-Limited Adaptive Histogram Equalization* – CLAHE). Esse método realiza a equalização baseada em pequenas regiões da imagem,

além de limitar o contraste local, evitando que determinadas regiões fiquem ou claras ou escuras em demasia (GAMA; COELHO; BAFFA, 2020).

3.3.3 Cálculo do Negativo

Ainda para aumentar a homogeneidade da base de dados, foi aplicada a conversão das imagens em seu negativo. Dessa forma, pode-se destacar as anomalias presentes na retina, enquanto as estruturas da retina, como a mácula e os vasos sanguíneos, tornam-se menos evidentes (GAMA; COELHO; BAFFA, 2020).

3.3.4 Recorte da Região de Interesse

As duas bases de dados utilizadas (IDRiD e MESSIDOR) no trabalho possuem diferentes recortes. Assim, com o objetivo de padronizar os cortes das imagens, será realizado um recorte automático da região interior-central da retina, onde geralmente ocorre a presença das anomalias que permitem a detecção da retinopatia (GAMA; COELHO; BAFFA, 2020). Como as imagens possuem dimensões diferentes, houve o cuidado em variar os cortes para cada tipo de imagem. Após o recorte, as imagens ficaram com a resolução de 1100 *pixels* por 800 *pixels*.

3.4 Desenvolvimento da Rede Neural Convolucional

Foi desenvolvida a rede neural convolucional altamente profunda apresentada por Gama, Coelho e Baffa (2020), representada na Figura 11. Essa rede é composta por 3 grupos de convolução, com três camadas cada: duas camadas de convolução 2D, de 64, 128 e 256 filtros de tamanho 3x3 e uma camada de *MaxPooling* de tamanho 2x2. Após passar pelos grupos de convolução, as características são enviadas para uma Rede Neural Totalmente Conectada (*Fully Connected Neural Network* - FCNN) de quatro camadas de 4096 neurônios cada. Nas camadas internas da rede neural, foi utilizada a função de ativação ReLU (*Rectifier Linar Unit*), em conjunto com o algoritmo de ativação Adadelta. A última camada é responsável pela classificação binária e, portanto, foi utilizada a função de ativação Sigmoid (GAMA; COELHO; BAFFA, 2020).

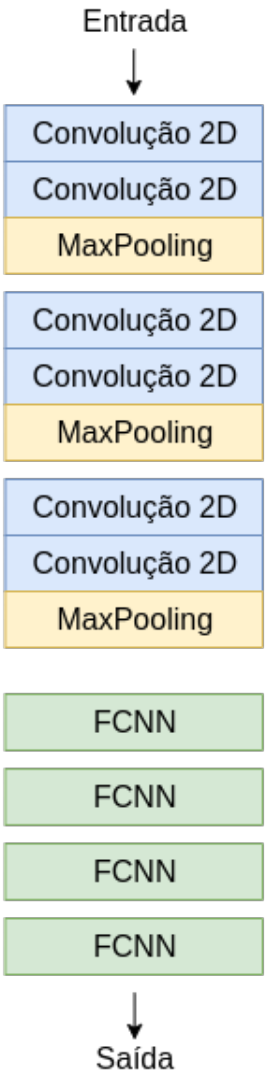


Figura 11 – Arquitetura rede neural convolucional

Fonte: Adaptado de Gama, Coelho e Baffa (2020)

4 Análise de Resultados

Neste capítulo, serão apresentados os resultados obtidos após a implementação da metodologia proposta no Capítulo 3 e a implementação de uma rede neural própria.

4.1 Reprodução do Artigo

4.1.1 Criação da Base de Treinamento e Validação

Primeiramente, foram analisadas as *labels* das bases de dados do Porwal et al. (2018) e do Decencière et al. (2014). As imagens eram classificadas em 5 categorias, variando as severidades de 0 a 4, sendo a severidade 0 considerada uma retina saudável, sem sinais de retinopatia, e a severidade 4 sendo a retina apresentando retinopatia proliferativa. Todas as imagens associadas a *label* 0 foram colocadas em um diretório chamado de saudável, enquanto todas as imagens associadas as demais *labels* foram colocadas em um diretório chamado de retinopatia. Apesar das resoluções das imagens serem diferentes, não há informação no artigo que indiquem que foram usadas de forma separada. Portanto, ambas as bases de dados foram usadas da mesma forma. Além disso, ao analisar o total de imagens usadas por Gama, Coelho e Baffa (2020), viu-se que, para atingir tal número, todas as imagens das bases classificadas como treinamento e teste deveriam ser utilizadas para treinamento e validação. A separação entre treinamento e validação ficou por responsabilidade do Keras, sendo passados os parâmetros de *subset*, como *validation* ou *training* e *validation_split* de 0.3.

4.1.2 Pré-processamento das Imagens

Primeiramente, as imagens no formato TIFF foram convertidas para PNG. A biblioteca Keras não trabalha com imagens TIFF de forma nativa, e no artigo não está explicitado como esse problema foi solucionado. Então, foram replicadas as etapas de pré-processamento descritas por (GAMA; COELHO; BAFFA, 2020), como pode ser visto na figura Figura 12. Os parâmetros utilizados no CLAHE não foram informados no artigo, então, foram mantidos os valores padrão do OpenCV.

4.1.3 Reprodução da Rede Neural Convolucional

Após a reprodução da rede, conforme citada no Capítulo 3, foi obtida a configuração apresentada em Figura 13. Alguns parâmetros da rede não foram informados. Para o tamanho de entrada da imagem, foi escolhida a dimensão de 128x128 pixels, devido à

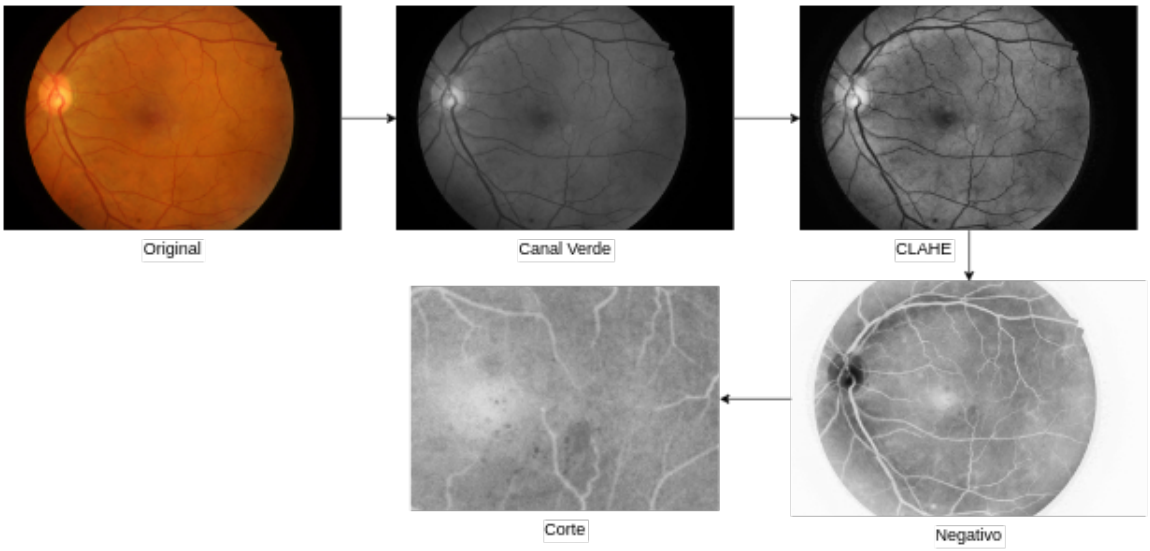


Figura 12 – Exemplo de retina a cada passo

limitação da GPU para a rede. Da mesma forma, foi utilizado *batch_size* de tamanho 8. Para o parâmetro de *padding* nas camadas de convolução, foi escolhido o valor *same*. Para o parâmetro *pool_size* nas camadas de *MaxPooling*, foi mantido o valor padrão do Keras de 2x2. O artigo também não informa qual função de erro foi utilizada, então foi escolhida a função de entropia cruzada binária (*binary cross-entropy*), que é muito utilizada em estudos de classificação binária (ELGENDY, 2020). Como não foi informado o valor da taxa de aprendizado (*learning_rate*) do otimizador Adadelta, foi mantido o valor padrão. A configuração final da rede pode ser vista em Figura 14, com um total de 319931841 parâmetros (pesos) na rede e com tamanho de 1,19GB.

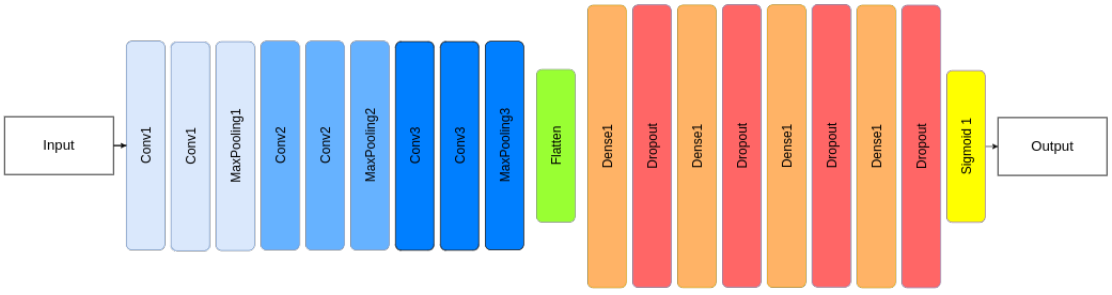
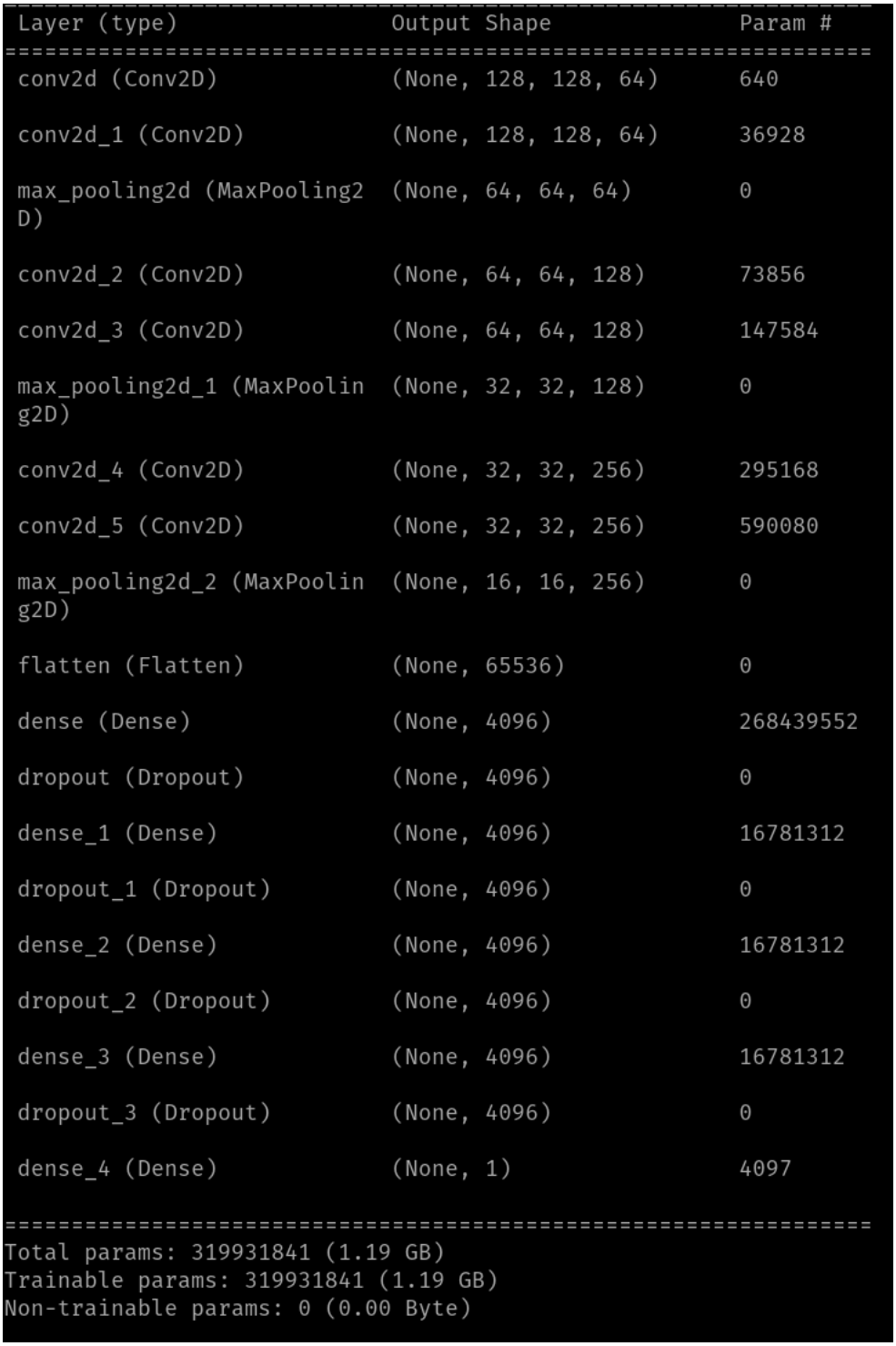


Figura 13 – Arquitetura CNN

4.1.3.1 *Learning Rate* Padrão

Após executar a rede por 30 épocas, foi obtido a acurácia de 57,59%. Apesar do erro ter melhorado no decorrer das épocas, a acurácia do subconjunto de validação se manteve durante todo o processo. As métricas de *loss* e *val_loss* podem ser analisadas em Figura 15, e os picos deste podem representar que a taxa de aprendizado está muito alta.



Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 128, 128, 64)	640
conv2d_1 (Conv2D)	(None, 128, 128, 64)	36928
max_pooling2d (MaxPooling2D)	(None, 64, 64, 64)	0
conv2d_2 (Conv2D)	(None, 64, 64, 128)	73856
conv2d_3 (Conv2D)	(None, 64, 64, 128)	147584
max_pooling2d_1 (MaxPooling2D)	(None, 32, 32, 128)	0
conv2d_4 (Conv2D)	(None, 32, 32, 256)	295168
conv2d_5 (Conv2D)	(None, 32, 32, 256)	590080
max_pooling2d_2 (MaxPooling2D)	(None, 16, 16, 256)	0
flatten (Flatten)	(None, 65536)	0
dense (Dense)	(None, 4096)	268439552
dropout (Dropout)	(None, 4096)	0
dense_1 (Dense)	(None, 4096)	16781312
dropout_1 (Dropout)	(None, 4096)	0
dense_2 (Dense)	(None, 4096)	16781312
dropout_2 (Dropout)	(None, 4096)	0
dense_3 (Dense)	(None, 4096)	16781312
dropout_3 (Dropout)	(None, 4096)	0
dense_4 (Dense)	(None, 1)	4097

Total params: 319931841 (1.19 GB)
Trainable params: 319931841 (1.19 GB)
Non-trainable params: 0 (0.00 Byte)

Figura 14 – Configuração da CNN

4.1.3.2 *Learning Rate* Reduzido

Ao reduzir a taxa de aprendizado da função adadelta de 0,001 para 0,0001, não houve mudança no valor da acurácia, que se manteve constante por 30 épocas. Ao reduzir a taxa de 0,0001 para 0,00001, o valor da acurácia passou a ser 43,39%, enquanto as métricas de erro cresceram.

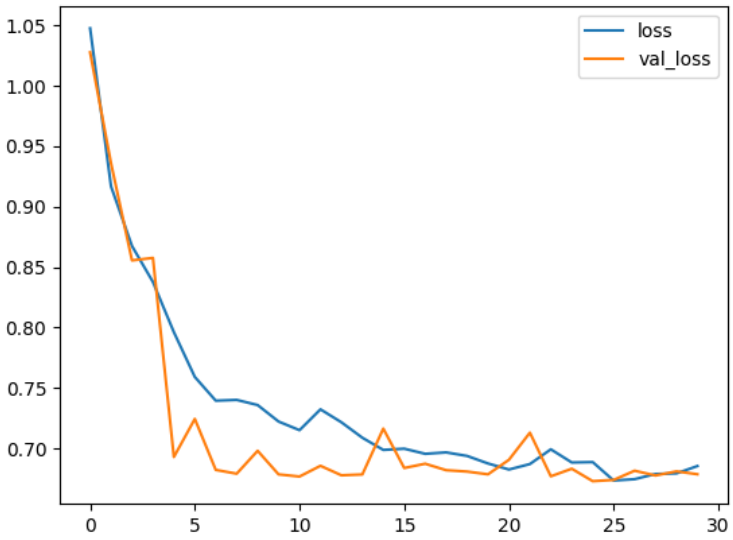


Figura 15 – Gráfico de *loss* e *val_loss* por épocas

4.1.3.3 Ajuste da Base de Dados

Com o propósito de averiguar a maior variável da solução proposta por (GAMA; COELHO; BAFFA, 2020), foi feito o ajuste da base de dados para usar os formatos TIFF e JPEG, sem realizar a conversão. O pré-processamento foi realizado da mesma forma. Para realizar o carregamento e redimensionamento das novas imagens, foi utilizada a biblioteca OpenCV e, então, foi feita a conversão para um *numpy array* com a biblioteca Numpy. As *labels* foram criadas manualmente, como um *numpy array* com valores 0 ou 1, indicando se a imagem era de uma retina saudável ou não. A separação entre subconjuntos de treinamento e validação foi feita com o auxílio da biblioteca *scikit-learn*, de forma que se manteve a proporção de 70% para treinamento e 30% para validação. Com o teste, foi obtido 55% de acurácia na base de validação em 30 épocas.

4.2 Proposta de CNN

Devido à impossibilidade de reproduzir os resultados do artigo, decidiu-se desenvolver uma CNN própria.

4.2.1 Base de Dados

Haja vista que os resultados obtidos através das outras bases de dados não foram os esperados, na Seção 4.1, foi escolhida outra base de dados. As bases do Porwal et al. (2018) e Decencière et al. (2014) possuíam imagens em formatos diferentes e, mesmo juntas, não forneciam uma quantidade satisfatória de imagens para treinamento. Com a

finalidade de ter uma quantidade maior de imagens para treinamento, foi utilizada a base de dados Dugas Jared (2015). Essa base, fornecida pelo *EyePACS*, foi utilizada em uma competição do Kaggle, com a finalidade de deixar público os melhores resultados obtidos, assim melhorando as técnicas de identificação para retinopatia diabética.

A base fornecida pelo *EyePACS* é composta por 35126 imagens, totalizando 37,9GB. As imagens são divididas entre cinco categorias, de acordo com a severidade: saudável, leve, moderada, severa, proliferativa.

4.2.2 Primeira Versão da CNN

Para iniciar os testes, foi feita uma versão simples de uma CNN, composta por três camadas de convolução, com filtros 32, 64, 128, respectivamente. Cada camada de convolução foi seguida por uma camada de *MaxPooling* 2x2. Após as convoluções, foi utilizada a camada de *Flatten* para transformar os dados em uma dimensão, que são mandados para a rede neural totalmente conectada, composta por uma camada densa com 512 neurônios, *Droupout* de 40% e finalizando com uma camada densa com 1 neurônio e função de ativação sigmoid, conforme a Figura 16.

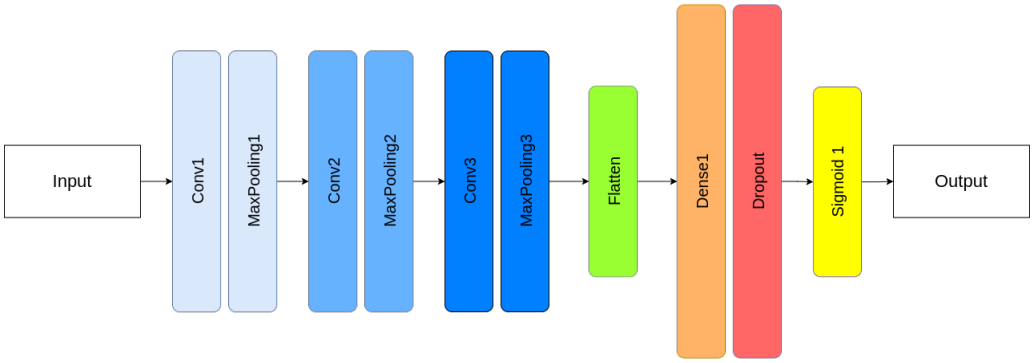


Figura 16 – Primeira versão da CNN

Em cada camada de convolução, foram utilizados *padding "same"*, *strides* (1,1), com a função de ativação Relu(*Rectified Linear Unit*). Em cada camada de *Pooling*, foi usado um kernel de tamanho 2x2. As imagens da camada de entrada possuíam os três canais RGB e tinham o tamanho de 32x32 pixels. O *batch_size* utilizado foi de 128. A base de dados foi separada em 70% para treinamento e 30% para validação. O algoritmo de otimização escolhido foi o Adadelta.

O teste inicial foi realizado com toda a base de dados. Foram executadas 5 épocas, sendo que cada época levou aproximadamente um minuto para ser finalizada. No final do treino, a métrica de erro ainda estava alta, como pode ser visto no gráfico da Figura 17, e a acurácia obtida foi de 73,94% na base de validação.

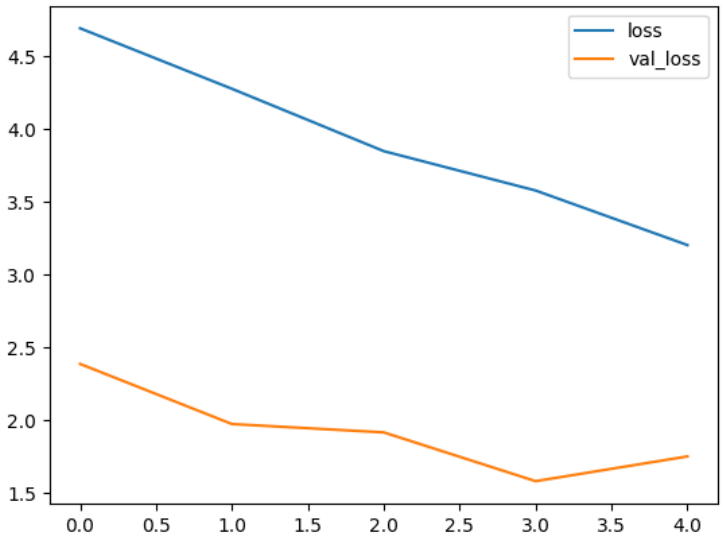


Figura 17 – Gráfico de *loss* e *val_loss* por épocas

4.2.2.1 Testes com Pré-processamento

Foi realizado um teste com a base de dados convertida para escala de cinza, o que resultou em uma acurácia de 73,83% na validação e um teste após a aplicação do negativo na base em escala de cinza, que apresentou uma acurácia de 74,12%. As etapas de pré-processamento estão representadas na Figura 18.

Figura 18 – Conversão RGB para escala de cinza e aplicação do negativo

4.2.3 Aumento de Profundidade da Rede

Foram adicionadas mais camadas de convolução para analisar o comportamento da rede. Cada camada de convolução foi duplicada, possuindo assim três grupamentos de convolução, cada um com duas convoluções e uma de *pooling*, representada na Figura 19. Esse modelo é semelhante a uma CNN popular, a VGG16. A VGG16 já foi bastante utilizada em outros estudos de identificação/classificação de doenças, como no Albashish et al. (2021), em que foi utilizada para identificar câncer de mama. Não houve mudança significativa na acurácia após 20 épocas. Além disso, a métrica *loss* melhorou, enquanto a

métrica *val_loss* se manteve constante após poucas épocas, o que configura uma situação de *overfitting* e pode ser observado na Figura 20.

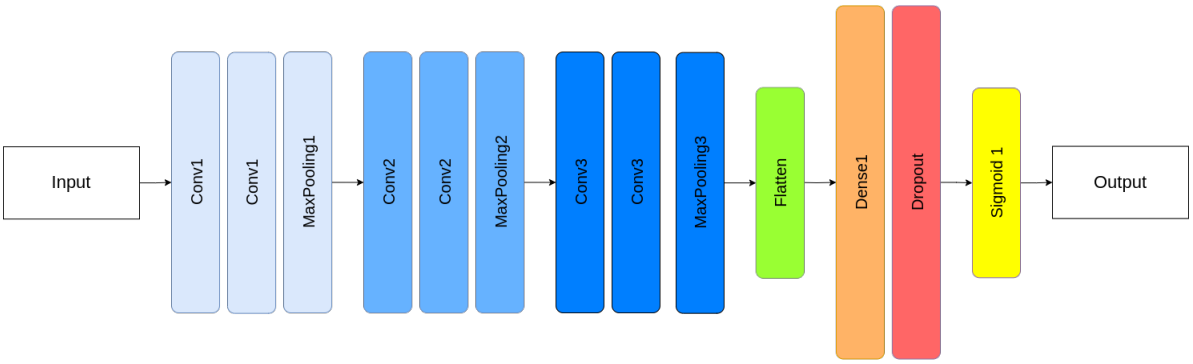


Figura 19 – Segunda versão da CNN

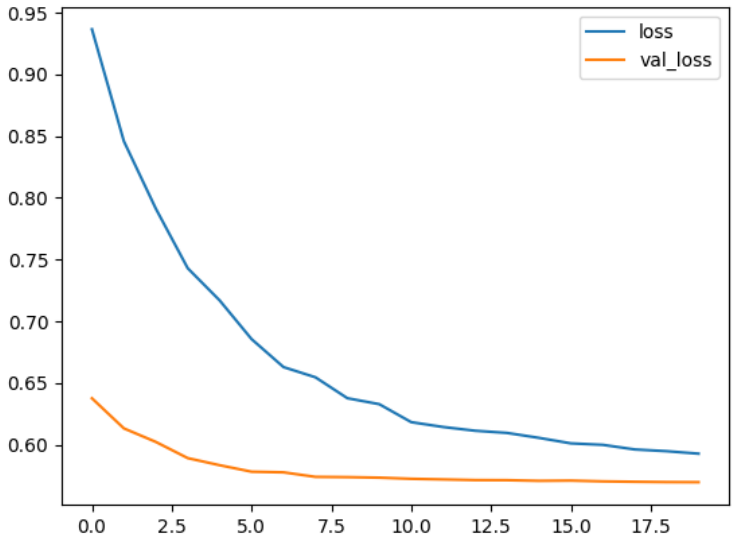


Figura 20 – Gráfico de *loss* e *val_loss* por épocas

4.2.4 Ajuste da Base de Dados

Com a finalidade de minimizar a situação de *overfitting*, a base de dados foi ajustada para uma nova proporção, essa sendo 80% das imagens, utilizadas para treinamento e 20% das imagens, para validação.

4.2.5 Ajustes de Hiperparâmetros

Algumas técnicas foram utilizadas para tentar amenizar a situação de *overfitting*. Dentre essas, podemos citar: aumento do tamanho da imagem de entrada para 128x128 pixels, adição de *dropout* de 30% nas camadas totalmente conectadas, redução do número

de filtros. Nenhuma das tentativas apresentou melhoria, e o tempo gasto para cada teste na base de 35126 imagens se mostrou um grande dificultador. Sendo assim, foi tomada a decisão de reduzir a base de dados para 4000 imagens, composta por 2000 imagens classificadas como saudável e 2000 imagens classificadas como retinopatia. O otimizador foi alterado para o Adam, que é muito popular no contexto de classificação binária e alguns estudos comprovam sua eficácia (BOCK; WEIß, 2019).

4.2.6 Redução da Base de Dados

A redução da base de dados foi feita com o auxílio da biblioteca Pandas, do Python. Imagens que possuem *label* 0, no arquivo de *labels* fornecido pelo Dugas Jared (2015) foram agrupadas como saudável, as demais imagens foram agrupadas como retinopatia. Após a mudança da base de dados, a rede neural se mostrou ineficiente, com uma acurácia de 56,5% na base de validação. A situação de *overfitting* pôde ser percebida desde as primeiras épocas. Dessa forma, foi reduzida a profundidade da rede e o número de neurônios na rede totalmente conectada. As camadas de convolução da rede foram alteradas para a mesma configuração em Figura 16. A camada de *dropout* foi aumentada para 50% e o *learning rate* do otimizador foi reduzido do valor padrão 1e-03 para 1e-05. As imagens voltaram a ser utilizadas no formato RGB. A quantidade de neurônios na camada totalmente conectada foi reduzido para 64.

O baixo valor da acurácia ao reduzir a base incentivou um estudo em cima da base de dados. Ao se verificar a distribuição da base, foi possível perceber que ela estava desbalanceada. Apesar do projeto ser uma classificação binária, as imagens da base são classificadas em 5 categorias, sendo 4 dessas correspondentes à situação de retinopatia diabética. As imagens da base, após a redução, possuíam apenas 2 classes, como pode ser observado na Figura 21.

4.2.7 Balanceamento da Base de Dados

A fim de se abordar o problema identificado, foi feito o balanceamento da base de dados. Mantendo-se a mesma quantidade de imagens (4000), as imagens foram separadas em 2000 imagens classificadas como saudável e 2000 imagens com retinopatia, mas tentando manter o balanceamento entre as classes de retinopatia. Além disso, percebeu-se que a base de dados contém imagens repetidas, uma versão orientada para esquerda, e outra para a direita. Dessa forma, a fim de se aumentar a diversidade da base, houve o cuidado em escolher apenas uma versão de cada imagem para as categorias que possuíam uma quantidade suficiente de imagens (500 imagens para retinopatia e 2000 para saudável). Nas categorias que não possuíam o mínimo necessário, todas as imagens foram selecionadas. A distribuição final pode ser observada na Figura 22. Após o balanceamento, a acurácia na base de dados de validação subiu para 62%.

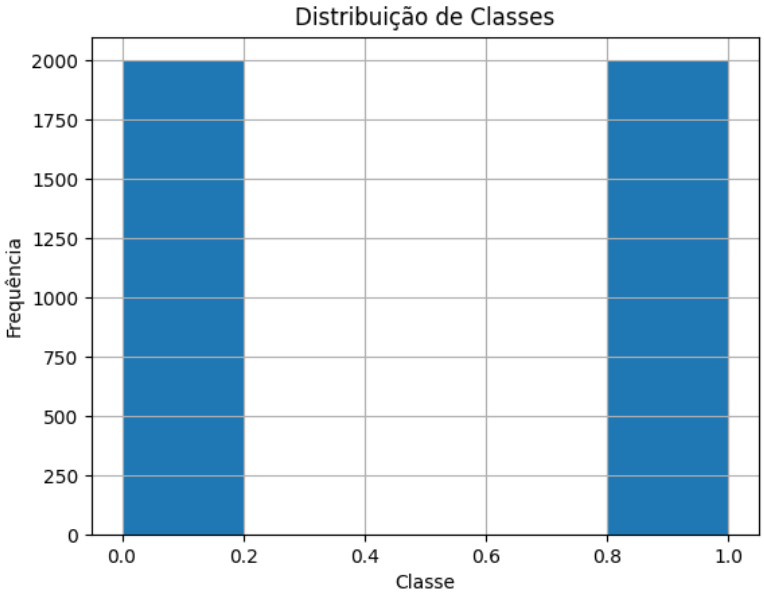


Figura 21 – Distribuição de classes na base de dados

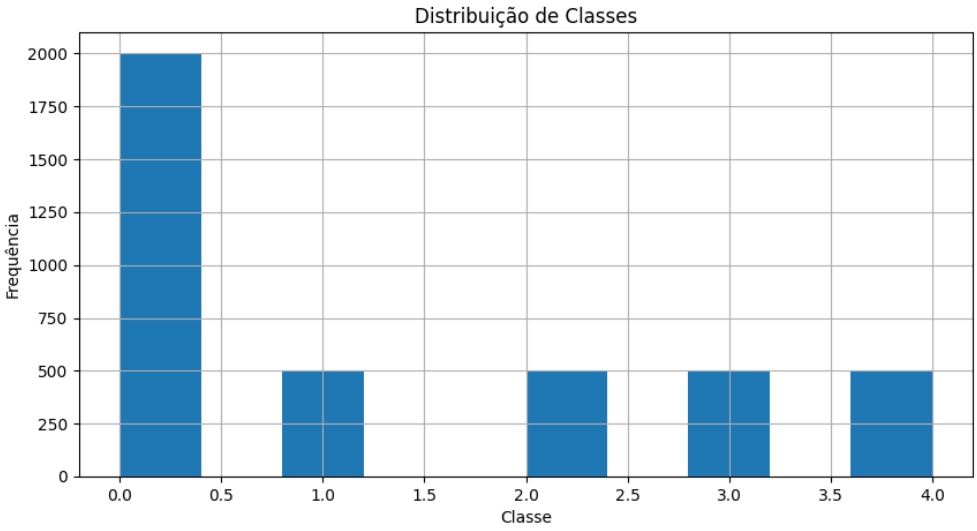


Figura 22 – Distribuição de classes na base de dados após balanceamento

4.2.8 Aplicação de Técnicas de Pré-processamento

Nessa etapa, foram feitos alguns testes com a base após um pré-processamento. Foram repetidos os testes em escala de cinza, negativo. Também foram feitos testes extraindo o canal verde e fazendo uma equalização do histograma, como sugerido por Gama, Coelho e Baffa (2020). Nenhuma dessas técnicas apresentou melhoria nos resultados. A técnica que se mostrou promissora e foi escolhida foi a aplicação de um filtro Gaussiano, representado na Figura 23. Após a aplicação do filtro Gaussiano, a acurácia obtida melhorou para 65,15%. Foram testados alguns ajustes em hiper-parâmetros, mas não apresentaram melhorias. Todos os demais testes foram realizados com as imagens resultantes do filtro.

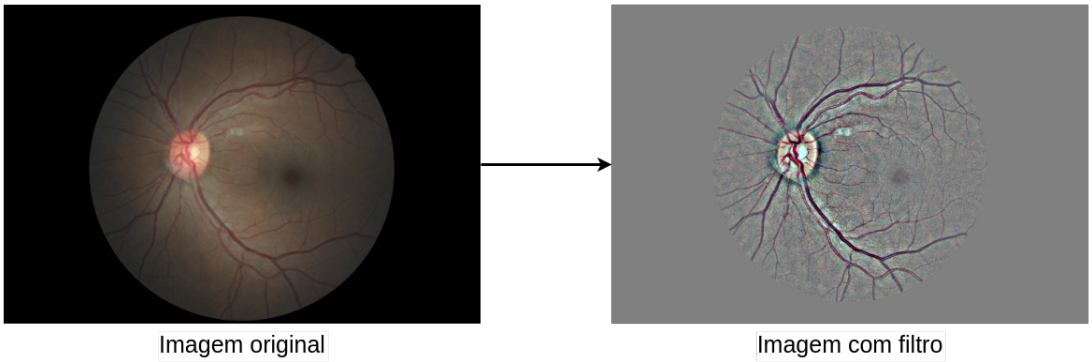


Figura 23 – Exemplo imagem após aplicação de filtro Gaussiano

4.2.9 Normalização de Dados

Normalizar os dados de entrada é uma prática utilizada para acelerar o aprendizado das redes neurais, mantendo o valor de cada pixel na imagem entre 0 e 1 (ELGENDY, 2020). Foi adicionada uma camada para realizar esse redimensionamento no início da rede. Também foi adicionado mais um bloco de convolução, com 256 filtros, e camadas de *BatchNormalization* após cada convolução, resultando na rede da Figura 24. Essa configuração resultou em uma acurácia de 68,05%. Apesar da melhoria, a métrica de *val_loss* não decaiu conforme o esperado, sinalizando o *overfitting* após rodar por 15 épocas. A Figura 25 mostra como as métricas de erro se comportaram no decorrer de 30 épocas.

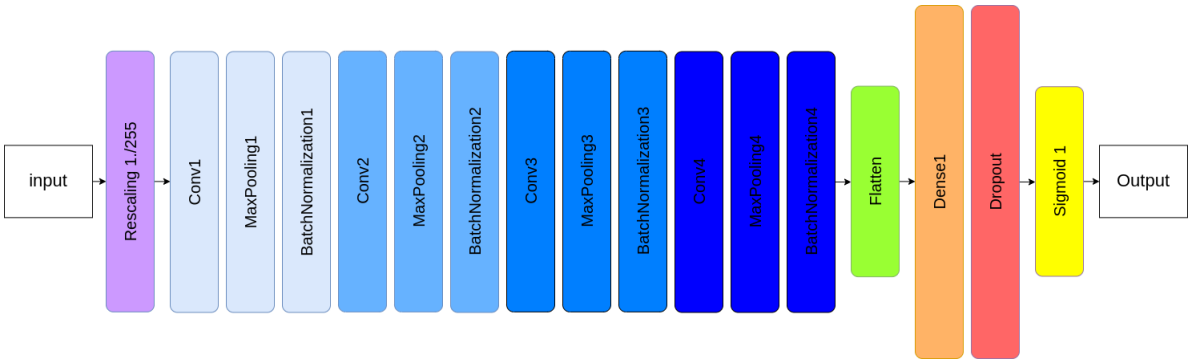


Figura 24 – Terceira versão da CNN

4.2.10 Aumento de Dados

Visando-se melhorar a situação de *overfitting*, novamente, foram aplicadas mais algumas técnicas. Logo antes do primeiro bloco de convolução, foi adicionada uma camada para alterar a orientação das imagens, horizontalmente e verticalmente. Essa escolha foi tomada porque a base não contém todas as imagens de retina na mesma orientação. Apesar dessa técnica não armazenar diferentes imagens, com ela é possível gerar novas instâncias das imagens em tempo de execução (ELGENDY, 2020), o que traz uma maior diversidade para a rede. Caso a rede aprenda a identificar a doença a partir de diferentes

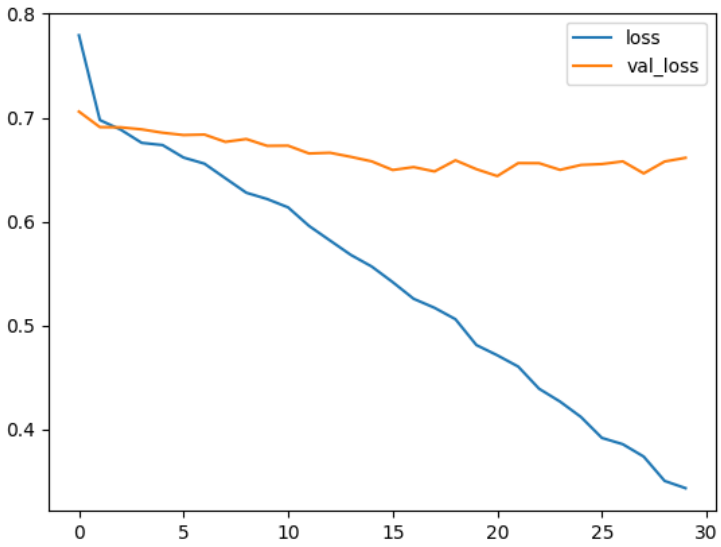


Figura 25 – Gráfico de *loss* e *val_loss* por época

orientações, provavelmente ela vai classificar corretamente alguns dos casos de imagens que estavam em outra orientação.

Foi adicionada mais uma camada de *dropout*, após a camada de *flatten*, pois estas se mostram eficientes quanto a situação de *overfitting* em bases pequenas, como observado por Pasupa e Sunhem (2016). Essas mudanças resultaram em uma nova acurácia de 69,56%. Essa foi a configuração final da rede neural convolucional, e pode ser observada na Figura 26.

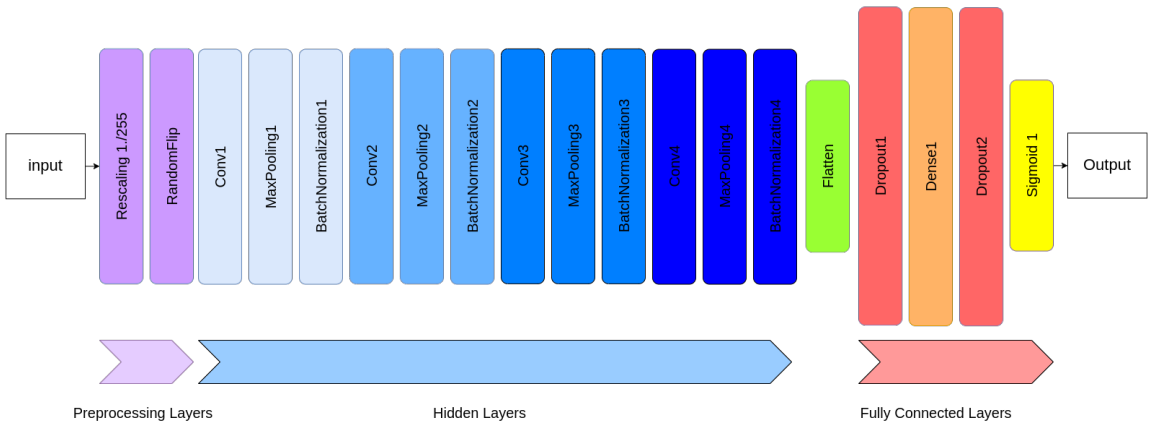


Figura 26 – Versão final da CNN

4.2.11 Aumento de Dimensão das Imagens

Como último teste, na expectativa de que a rede conseguisse extrair algumas novas características, foi aumentado o tamanho da imagem. As imagens foram redimensionadas

para 512x512 pixels. O número de neurônios na camada densa também foi aumentado para 128. Com isso, foi obtido o melhor modelo treinado, com uma acurácia de 72,33% na base de validação em 22 épocas.

4.2.12 Quantidade de Pesos da Rede

O sumário da versão final da CNN pode ser visto na Figura 27. A rede proposta apresentou 33893345 parâmetros (pesos) em sua versão final, totalizando 129,29MB de tamanho.

Layer (type)	Output Shape	Param #
rescaling (Rescaling)	(None, 512, 512, 3)	0
random_flip (RandomFlip)	(None, 512, 512, 3)	0
conv2d (Conv2D)	(None, 512, 512, 32)	416
activation (Activation)	(None, 512, 512, 32)	0
max_pooling2d (MaxPooling2D)	(None, 256, 256, 32)	0
batch_normalization (BatchNormalization)	(None, 256, 256, 32)	128
conv2d_1 (Conv2D)	(None, 256, 256, 64)	8256
activation_1 (Activation)	(None, 256, 256, 64)	0
max_pooling2d_1 (MaxPooling2D)	(None, 128, 128, 64)	0
batch_normalization_1 (BatchNormalization)	(None, 128, 128, 64)	256
conv2d_2 (Conv2D)	(None, 128, 128, 128)	32896
activation_2 (Activation)	(None, 128, 128, 128)	0
max_pooling2d_2 (MaxPooling2D)	(None, 64, 64, 128)	0
batch_normalization_2 (BatchNormalization)	(None, 64, 64, 128)	512
conv2d_3 (Conv2D)	(None, 64, 64, 256)	295168
activation_3 (Activation)	(None, 64, 64, 256)	0
max_pooling2d_3 (MaxPooling2D)	(None, 32, 32, 256)	0
batch_normalization_3 (BatchNormalization)	(None, 32, 32, 256)	1024
flatten (Flatten)	(None, 262144)	0
dropout (Dropout)	(None, 262144)	0
dense (Dense)	(None, 128)	33554560
dropout_1 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 1)	129

Total params: 33893345 (129.29 MB)
Trainable params: 33892385 (129.29 MB)
Non-trainable params: 960 (3.75 KB)

Figura 27 – Quantidade de parâmetros da CNN

5 Considerações Finais

Ao serem realizados os testes da CNN desenvolvida por Gama, Coelho e Baffa (2020), foi verificado que a rede neural, com os parâmetros informados pelo artigo, não atingiu os resultados esperados. Para que sejam feitos testes mais conclusivos, são necessárias algumas informações cruciais, como os tipos de *padding* e *strides* utilizados nas camadas de convolução, os filtros usados nas camadas de *pooling*, o tamanho de entrada das imagens, a função de erro e os parâmetros utilizados no otimizador Adadelta. Além disso, após o desenvolvimento da CNN descrito na Seção 4.2, foi possível verificar como a distribuição da base de dados pode afetar os resultados. Dessa forma, para que sejam obtidos resultados mais precisos, é necessário saber como foi feita a separação entre dados de treinamento e validação, o que não foi informado pelo artigo.

Os resultados obtidos na Seção 4.2 mostram a importância da preparação da base de dados. O uso de uma base de dados bem distribuída, com os pré-processamentos adequados para destacar as características esperadas são de suma importância no processo de aprendizado profundo. Também foi possível concluir que aumentar a profundidade da rede não necessariamente irá aumentar a acurácia obtida. Técnicas como normalização e aumento dos dados se mostraram efetivas em amenizar a situação de *overfitting*.

Durante a execução do trabalho, ocorreram algumas dificuldades. Dentre essas, pode-se destacar a reprodução da rede neural apresentada por Gama, Coelho e Baffa (2020), haja vista que muitos passos necessários para uma reprodução fiel não estavam presentes no artigo.

O tamanho base de dados fornecida pelo Dugas Jared (2015) foi uma limitação. A base, em sua totalidade, ocupava o armazenamento de, aproximadamente, 80GB. Além disso, a base estava dividida em vários arquivos com a extensão ".zip". Apesar de estarem nessa extensão, os arquivos não eram grupos de imagens compactadas em vários arquivos. Os arquivos eram parte de um grande arquivo com todas as imagens, compactado e fracionado em pequenos lotes. Então, só foi possível descompactar as imagens após baixar toda a base de dados e agrupar os lotes de treinamento.

Buscar formas de melhorar a acurácia obtida na rede proposta foi um grande desafio. Não existem estudos que mostrem um caminho certo para se obter os melhores resultados. Cada rede neural é treinada para um objetivo específico, com uma base de dados específica. Uma rede que apresenta bons resultados para uma base de dados pode não se mostrar eficiente para outra base de dados, por mais que ambas pertençam ao mesmo contexto. O aperfeiçoamento de uma rede neural exige que sejam realizados vários testes, variando-se hiperparâmetros, alterando-se a profundidade da rede, manuseando-se

a base de dados, entre outros.

Como complemento do trabalho, algumas técnicas podem ser aplicadas para melhorar a acurácia da rede proposta. Uma ideia interessante é tentar identificar quais das subcategorias, dentro da classe de retinopatia, a rede não está conseguindo identificar. É possível que alguma característica não esteja sendo extraída com sucesso para uma ou duas subcategorias. Identificar qual elemento a rede não consegue identificar pode indicar um caminho a se seguir. Outra abordagem, caso haja recursos, seria fazer o balanceamento da base de dados como um todo, gerando uma grande quantidade de imagens através do aumento de dados.

Referências

AKAY, A.; HESS, H. Deep learning: current and emerging applications in medicine and technology. *IEEE journal of biomedical and health informatics*, IEEE, v. 23, n. 3, p. 906–920, 2019. Citado na página 21.

ALBASHISH, D. et al. Deep cnn model based on vgg16 for breast cancer classification. In: *2021 International Conference on Information Technology (ICIT)*. [S.l.: s.n.], 2021. p. 805–810. Citado na página 48.

ALBAWI, S.; MOHAMMED, T. A.; AL-ZAWI, S. Understanding of a convolutional neural network. In: IEEE. *2017 International Conference on Engineering and Technology (ICET)*. [S.l.], 2017. p. 1–6. Citado na página 32.

BAE, K.; RYU, H.; SHIN, H. Does adam optimizer keep close to the optimal point? *arXiv preprint arXiv:1911.00289*, 2019. Citado na página 31.

BOCK, S.; WEIß, M. A proof of local convergence for the adam optimizer. In: *2019 International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2019. p. 1–8. Citado na página 50.

DECENCIèRE, E. et al. Feedback on a publicly distributed database: the messidor database. *Image Analysis & Stereology*, v. 33, n. 3, p. 231–234, ago. 2014. ISSN 1854-5165. Disponível em: <http://www.ias-iss.org/ojs/IAS/article/view/1155>. Citado 6 vezes nas páginas 23, 24, 35, 37, 43 e 46.

DUGAS JARED, J. W. C. E. *Diabetic Retinopathy Detection*. Kaggle, 2015. Disponível em: <https://kaggle.com/competitions/diabetic-retinopathy-detection>. Citado 3 vezes nas páginas 47, 50 e 55.

EDUCATION, I. C. *Deep Learning*. [S.l.]: <https://www.ibm.com/cloud/learn/deep-learning>, 2020. Acessado: 2021-10-18. Citado na página 31.

ELGENDY, M. *Deep Learning for Vision Systems*. [S.l.]: Manning Publications Co., 2020. Citado 11 vezes nas páginas 25, 26, 27, 28, 29, 30, 31, 33, 34, 44 e 52.

GAMA, I. R.; COELHO, A. M.; BAFFA, M. F. O. Fundus eye images classification for diabetic retinopathy detection using very deep convolutional neural network. In: *Proceedings of XVI Workshop de Visão Computacional*. [S.l.: s.n.], 2020. p. 24–29. Citado 10 vezes nas páginas 13, 22, 37, 39, 40, 41, 43, 46, 51 e 55.

GARGEYA, R.; LENG, T. Automated identification of diabetic retinopathy using deep learning. *Ophthalmology*, Elsevier, v. 124, n. 7, p. 962–969, 2017. Citado na página 24.

GIRSHICK, R. et al. Region-based convolutional networks for accurate object detection and segmentation. *IEEE transactions on pattern analysis and machine intelligence*, IEEE, v. 38, n. 1, p. 142–158, 2015. Citado na página 25.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep learning*. [S.l.]: MIT press, 2016. Citado na página 27.

GUPTA, N. et al. Artificial neural network. *Network and Complex Systems*, v. 3, n. 1, p. 24–28, 2013. Citado na página 30.

HEMANTH, D. J.; DEPERLIOGLU, O.; KOSE, U. An enhanced diabetic retinopathy detection and classification approach using deep convolutional neural network. *Neural Computing and Applications*, Springer, v. 32, p. 707–721, 2020. Citado na página 35.

KROGH, A. What are artificial neural networks? *Nature biotechnology*, Nature Publishing Group US New York, v. 26, n. 2, p. 195–197, 2008. Citado 2 vezes nas páginas 28 e 30.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. *nature*, Nature Publishing Group, v. 521, n. 7553, p. 436–444, 2015. Citado 2 vezes nas páginas 28 e 32.

LIAO, J. et al. Automating image morphing using structural similarity on a halfway domain. *ACM Transactions on Graphics (TOG)*, ACM New York, NY, USA, v. 33, n. 5, p. 1–12, 2014. Citado na página 25.

MOHAMED, Q.; GILLIES, M. C.; WONG, T. Y. Management of diabetic retinopathy: a systematic review. *Jama*, American Medical Association, v. 298, n. 8, p. 902–916, 2007. Citado na página 23.

MOHRI, M.; ROSTAMIZADEH, A.; TALWALKAR, A. *Foundations of machine learning*. [S.l.]: MIT press, 2018. Citado na página 28.

O'SHEA, K.; NASH, R. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015. Citado 2 vezes nas páginas 32 e 34.

PAL, K. K.; SUDEEP, K. Preprocessing for image classification by convolutional neural networks. In: IEEE. *2016 IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*. [S.l.], 2016. p. 1778–1781. Citado na página 26.

PASUPA, K.; SUNHEM, W. A comparison between shallow and deep architecture classifiers on small dataset. In: IEEE. *2016 8th International Conference on Information Technology and Electrical Engineering (ICITEE)*. [S.l.], 2016. p. 1–6. Citado na página 53.

PORWAL, P. et al. *Indian Diabetic Retinopathy Image Dataset (IDRiD)*. IEEE Dataport, 2018. Disponível em: <https://dx.doi.org/10.21227/H25W98>. Citado 3 vezes nas páginas 37, 43 e 46.

PRATT, H. et al. Convolutional neural networks for diabetic retinopathy. *Procedia computer science*, Elsevier, v. 90, p. 200–205, 2016. Citado 2 vezes nas páginas 21 e 35.

SAEEDI, P. et al. Global and regional diabetes prevalence estimates for 2019 and projections for 2030 and 2045: Results from the international diabetes federation diabetes atlas. *Diabetes research and clinical practice*, Elsevier, v. 157, p. 107843, 2019. Citado na página 21.

SAFI, H. et al. Early detection of diabetic retinopathy. *Survey of ophthalmology*, Elsevier, v. 63, n. 5, p. 601–608, 2018. Citado na página 34.

SCOTLAND, G. S. et al. Costs and consequences of automated algorithms versus manual grading for the detection of referable diabetic retinopathy. *British Journal of Ophthalmology*, BMJ Publishing Group Ltd, v. 94, n. 6, p. 712–719, 2010. Citado na página 21.

SEBE, N. *Machine learning in computer vision*. [S.l.]: Springer Science & Business Media, 2005. v. 29. Citado na página 24.

SETIAWAN, A. W. et al. Color retinal image enhancement using clahe. In: IEEE. *International conference on ICT for smart society*. [S.l.], 2013. p. 1–3. Citado na página 26.

SHAH, A. et al. A comprehensive study on skin cancer detection using artificial neural network (ann) and convolutional neural network (cnn). *Clinical eHealth*, Elsevier, 2023. Citado na página 27.

STITT, A. W. et al. The progress in understanding and treatment of diabetic retinopathy. *Progress in retinal and eye research*, Elsevier, v. 51, p. 156–186, 2016. Citado na página 23.

SZELISKI, R. *Computer vision: algorithms and applications*. [S.l.]: Springer Nature, 2022. Citado na página 24.

TIME, G. On diabetes. 2016. Citado 2 vezes nas páginas 21 e 23.

WANG, S.-C.; WANG, S.-C. Artificial neural network. *Interdisciplinary computing in java programming*, Springer, p. 81–100, 2003. Citado na página 29.

ZHANG, X.-D. Machine learning. In: *A Matrix Algebra Approach to Artificial Intelligence*. [S.l.]: Springer, 2020. p. 223–440. Citado na página 27.

Apêndices

APÊNDICE A – Algoritmos utilizados para o pré-processamento

```
import cv2
import numpy as np
import tensorflow as tf

from PIL import Image
from pathlib import Path
from utils.crop import crop
from tensorflow.keras import layers


CLASSES = ['retinopatia', 'saudavel']


# Define a function to load and preprocess images using only the green
                                        channel
def load_and_preprocess_images(image_paths, target_size=(128, 128)):
    images = []
    for image_path in image_paths:
        image = Image.open(image_path)
        # Extract the green channel and apply preprocessing steps (e.g.,
                                            resizing and normalization)
        green_channel = image.split()[1]
        green_channel = green_channel.resize(target_size)
        green_channel = np.array(green_channel) / 255.0  # Normalize
                                            pixel values
        images.append(green_channel)
    return np.array(images)


def extract_green_ch(rgb_img):
    green_channel = rgb_img[:,:,1]

    return green_channel


def apply_clahe(img):
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
    cl1 = clahe.apply(img)
```

```
    return cl1


def apply_negative(img):
    negative = cv2.absdiff(img, 255)
    # negative = abs(255-img)
    return negative

crop_mapping_rgb = {
    '960,1440,3': [(80, 880), (170, 1270)],
    '1536,2304,3': [(368,1168), (602, 1702)],
    '1488,2240,3': [(344, 1144), (570, 1670)],
    '2848,4288,3': [(1024, 1824), (1594, 2694)]
}

crop_mapping = {
    '960,1440': [(80, 880), (170, 1270)],
    '1536,2304': [(368,1168), (602, 1702)],
    '1488,2240': [(344, 1144), (570, 1670)],
    '2848,4288': [(1024, 1824), (1594, 2694)]
}


def crop(img):
    shape = ','.join(list(map(lambda x: str(x), img.shape)))
    height, width = crop_mapping[shape]

    cropped_image = img[height[0]:height[1], width[0]:width[1]]
    return cropped_image


def get_paths_and_labels(folder):
    images = []
    labels = []
    for c in CLASSES:
        base_path = Path(folder) / c
        images_paths = base_path.glob('*')

        if c == 'retinopatia':
            lbl = 1
        else:
            lbl = 0

        for p in images_paths:
            images.append(str(p))
            labels.append(lbl)
```

```
    len_ret = len([x for x in labels if x == 1])
    len_normal = len([x for x in labels if x == 0])

    print('Numero de labels com retinopatia: ', len_ret)
    print('Numero de labels saudavel: ', len_normal)

    return images, labels


def gen_augmentation(img):
    data_augmentation = tf.keras.Sequential([
        layers.RandomFlip("horizontal_and_vertical"),
        layers.RandomRotation(0.2),
    ])

    return data_augmentation(img)


def scaleRadius(img,scale):
    k = img.shape[0]/2
    x = img[int(k), :, :].sum(1)
    r=(x>x.mean()/10).sum()/2
    if r == 0:
        r = 1
    s=scale*1.0/r
    return cv2.resize(img,(0,0),fx=s,fy=s)


def get_grayscale(img):
    return cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)


def load_and_process_image(image_path, destination_path):
    image = cv2.imread(image_path)
    # image = extract_green_ch(image)
    # gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # clahe_img = apply_clahe(gray_image)

    print('Saving image in', destination_path)
    cv2.imwrite(destination_path, image)


def gaussian_filter(img, scale=500):
    a = scaleRadius(img, scale)
    b = np.zeros(a.shape)
    x = a.shape[1] / 2
    y = a.shape[0] / 2
```

```
    center_coordinates = (int(x), int(y))
    cv2.circle(b, center_coordinates, int(scale * 0.9), (1, 1, 1), -1, 8
                                        , 0)
    aa = cv2.addWeighted(a, 4, cv2.GaussianBlur(a, (0, 0), scale / 30),
                                        -4, 128) * b + 128 * (1 - b)

    return aa


def load_and_process_image(image_path, destination_path):
    scale = 500
    image = cv2.imread(image_path)

    img = gaussian_filter(image, scale)

    print('Saving image in', destination_path)
    cv2.imwrite(destination_path, img)


def load_and_process_images(images_paths: [Path], target_size=(128, 128)
                                    ):
    images = []
    for idx, path in enumerate(images_paths):
        image = cv2.imread(str(path), cv2.IMREAD_UNCHANGED)
        resized_image = cv2.resize(image, target_size)

        # green_ch_img = extract_green_ch(image)
        # clahe_img = apply_clahe(green_ch_img)
        # negative = cv2.absdiff(clahe_img, 255)
        # cropped = crop(negative)
        # resized_image = cv2.resize(cropped, target_size)

        images.append(np.array(resized_image))

    return images


if __name__ == '__main__':
   images_paths, labels = get_paths_and_labels()
   images = load_and_process_images(images_paths[:10])
```

APÊNDICE B – CNN da reprodução do artigo

```
import time
import numpy as np
import tensorflow as tf

from keras.models import Sequential
from keras.layers import (
    Conv2D,
    MaxPooling2D,
    Flatten,
    Dense,
    Dropout,
    Input,
)
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from process_image_opencv import get_paths_and_labels,
                                 load_and_process_images


AUTOTUNE = tf.data.experimental.AUTOTUNE

DATASET_DIR = 'dataset_processed_mixed_jpg_tiff'

INPUT_SIZE = 128
KERNEL_SIZE = (3, 3)

CONV_1_SIZE = 64
CONV_2_SIZE = 128
CONV_3_SIZE = 256

start = time.time()

model = Sequential()

model.add(Input(shape=(INPUT_SIZE, INPUT_SIZE, 1)))

model.add(Conv2D(CONV_1_SIZE, KERNEL_SIZE, padding='same', activation='
                                 relu'))
model.add(Conv2D(CONV_1_SIZE, KERNEL_SIZE, padding='valid', activation='
                                 relu'))
```

```
model.add(MaxPooling2D())

model.add(Conv2D(CONV_2_SIZE, KERNEL_SIZE, padding='same', activation='
                                    relu'))
model.add(Conv2D(CONV_2_SIZE, KERNEL_SIZE, padding='valid', activation='
                                    relu'))
model.add(MaxPooling2D())

model.add(Conv2D(CONV_3_SIZE, KERNEL_SIZE, padding='same', activation='
                                    relu'))
model.add(Conv2D(CONV_3_SIZE, KERNEL_SIZE, padding='valid', activation='
                                    relu'))
model.add(MaxPooling2D())

model.add(Flatten())

model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.2))

model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.2))

model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.2))

model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.2))

model.add(Dense(1, activation='sigmoid'))

optimizer = tf.keras.optimizers.Adadelta(learning_rate=0.00001)
model.compile(optimizer=optimizer, metrics=['accuracy'], loss='
                                    binary_crossentropy')

########################################################################

print('Starting images loading and processing...')

images_paths, labels = get_paths_and_labels(DATASET_DIR)
images = load_and_process_images(images_paths, target_size=(128, 128))

print('Finished images processing!')

# Convert the lists to numpy arrays
images = np.array(images)
labels = np.array(labels)
```

```
x_train, x_val, y_train, y_val = train_test_split(images, labels,
                                    test_size=0.3, random_state=42)

#############################################################################

history = model.fit(
    x_train,
    y_train,
    batch_size=8,
    epochs=2,
    validation_data=(x_val, y_val)
)

score = model.evaluate(x_train, y_train)
val_score = model.evaluate(x_val, y_val)

print(f'Train loss: {score[0]} / Train accuracy: {score[1]}')
print(f'Test loss: {val_score[0]} / Test accuracy: {val_score[1]}')

end = time.time()
print(f'Time: {end-start}')

pyplot.plot(history.history['loss'], label='loss')
pyplot.plot(history.history['val_loss'], label='val_loss')
pyplot.legend()
pyplot.show()
```

APÊNDICE C – Algoritmo de balanceamento da base de dados

```
import pandas as pd
from process_image_opencv import load_and_process_image_gaussian
from pathlib import Path

NO_DR_SIZE = 2000
OTHER_CAT_SIZE = 1000
SPLIT_VALUE = 0.8

DATASET_PATH = 'dataset_kaggle_raw'
DESTINATION_PATHS = {
    'train': f'{DESTINATION_PATH}/train',
    'val': f'{DESTINATION_PATH}/val',
    'test': f'{DESTINATION_PATH}/test',
}

CLASSES = ['retinopatia', 'saudavel']
dir = Path(DESTINATION_PATH)
dir.mkdir(exist_ok=True)

for k, v in DESTINATION_PATHS.items():
    dir = Path(v)
    dir.mkdir(exist_ok=True)
    for c in CLASSES:
        p = dir / c
        p.mkdir(exist_ok=True)

errors = []


def copy_img(images_names, label, ds_type):
    global errors

    path = Path(DATASET_PATH) / label

    for img in images_names:
        src = path / f'{img}.jpeg'
        dst = Path(DESTINATION_PATHS[ds_type]) / label / f'{img}.jpeg'
        try:
            load_and_process_image_gaussian(str(src), str(dst))
        except Exception as err:
```

```
            print(f'Error found in {ds_type}/{label}!!!!')
            errors.append(err)


if __name__ == '__main__':
    df = pd.read_csv('trainLabels.csv')
    print(df.head(5))

    train_ds_no_dr = []
    train_ds_dr = []
    val_ds_dr = []
    val_ds_no_dr = []

    no_dr = df[(df.level == 0)]
    no_dr = no_dr[0:NO_DR_SIZE*2:2]
    left = no_dr[no_dr.image.str.contains('left')]
    right = no_dr[no_dr.image.str.contains('right')]

    left = [img for _, img in left.image.items()]
    right = [img for _, img in right.image.items()]

    idx = int(len(left)*SPLIT_VALUE)
    left_train = left[0:idx]
    left_val = left[idx:]

    idx = int(len(right)*SPLIT_VALUE)
    right_train = right[0:idx]
    right_val = right[idx:]

    train_ds_no_dr += left_train
    train_ds_no_dr += right_train

    val_ds_no_dr += left_val
    val_ds_no_dr += right_val

    dr = df[(df.level != 0)]
    classes = [1, 2, 3, 4]  # DR severity

    dr_df = pd.DataFrame()
    for c in classes:
        dr_cat = dr[(dr.level == c)]

        step = 2 if len(dr_cat) > OTHER_CAT_SIZE*2 else 1
        size = min(OTHER_CAT_SIZE*step, len(dr_cat))
        dr_cat = dr_cat[0:size:step]

        dr_cat = [img for _, img in dr_cat.image.items()]
```

```
        idx = int(len(dr_cat)*SPLIT_VALUE)
        dr_cat_train = dr_cat[0:idx]
        dr_cat_val = dr_cat[idx:]

        train_ds_dr += dr_cat_train
        val_ds_dr += dr_cat_val

    print('Len train_ds_dr:', len(train_ds_dr))
    print('Len train_ds_no_dr:', len(train_ds_no_dr))

    print('Len val_ds_dr:', len(val_ds_dr))
    print('Len val_ds_no_dr:', len(val_ds_no_dr))

    copy_img(train_ds_dr, 'retinopatia', 'train')
    copy_img(train_ds_no_dr, 'saudavel', 'train')

    copy_img(val_ds_dr, 'retinopatia', 'val')
    copy_img(val_ds_no_dr, 'saudavel', 'val')

    for e in errors:
        print(e)
```

APÊNDICE D – CNN proposta

```
import time
import tensorflow as tf
from matplotlib import pyplot
from tensorflow import keras
from keras.callbacks import ModelCheckpoint, EarlyStopping,
                                    ReduceLROnPlateau
from keras.models import Sequential
from keras.layers import (
    Activation,
    BatchNormalization,
    Conv2D,
    MaxPooling2D,
    Flatten,
    Dense,
    Dropout,
    RandomFlip,
    Rescaling,
)

DATASET_TRAIN = 'dataset_kaggle_split_balanced/train'
DATASET_VAL = 'dataset_kaggle_split_balanced/val'
DATASET_DIR = 'dataset_kaggle_balanced'

print(f'Processing dataset {DATASET_DIR}')

BATCH_SIZE = 8
IMAGE_SIZE = (512, 512)
KERNEL_SIZE = (2, 2)
POOL_SIZE = (2, 2)
BASE = 32

CONV_1_SIZE = BASE * 1
CONV_2_SIZE = BASE * 2
CONV_3_SIZE = BASE * 4
CONV_4_SIZE = BASE * 8

start = time.time()

model = Sequential()

model.add(Rescaling(1./255))
model.add(RandomFlip('horizontal_and_vertical'))
```

```
model.add(Conv2D(CONV_1_SIZE, KERNEL_SIZE, padding='same', input_shape=(
                                    *IMAGE_SIZE, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=POOL_SIZE))
model.add(BatchNormalization())

model.add(Conv2D(CONV_2_SIZE, KERNEL_SIZE, padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=POOL_SIZE))
model.add(BatchNormalization())

model.add(Conv2D(CONV_3_SIZE, KERNEL_SIZE, padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=POOL_SIZE))
model.add(BatchNormalization())

model.add(Conv2D(CONV_4_SIZE, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=POOL_SIZE))
model.add(BatchNormalization())

model.add(Flatten())
model.add(Dropout(0.5))

model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(1, activation='sigmoid'))

model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=1e-05),
    metrics=['accuracy'],
    loss=keras.losses.BinaryCrossentropy()
)

##########################################################################

train_ds = keras.preprocessing.image_dataset_from_directory(
    DATASET_TRAIN,
    image_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    shuffle=True
)

val_ds = keras.preprocessing.image_dataset_from_directory(
    DATASET_VAL,
```

```
    image_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    shuffle=True
)


early_stopping = EarlyStopping(monitor='val_loss', min_delta=0, patience
                                    =20)

checkpointer = ModelCheckpoint(filepath='models/model.temp.xepochs.hdf5'
                                    , verbose=1, save_best_only=True)

plateau = ReduceLROnPlateau(factor=0.5, patience=4, cooldown=2, min_lr=
                                    1e-7)

history = model.fit(
    train_ds,
    epochs=1,
    validation_data=val_ds,
    shuffle=True,
    callbacks=[checkpointer, early_stopping]
)

model.summary()

score = model.evaluate(train_ds)
val_score = model.evaluate(val_ds)

print(f'Train loss: {score[0]} / Train accuracy: {score[1]}')
print(f'Test loss: {val_score[0]} / Test accuracy: {val_score[1]}')

end = time.time()
print(f'Time: {end-start}')

pyplot.plot(history.history['loss'], label='loss')
pyplot.plot(history.history['val_loss'], label='val_loss')
pyplot.legend()
pyplot.show()
```