

**IZABEL CRISTINA VIEIRA GOMES FERRARI
LUCAS KENDI FATTORE HIRANO**

**RESTAURAÇÃO DE RETRATOS POR INPAINTING
BASEADO EM APRENDIZADO DE MÁQUINA**

São Paulo
2018

**IZABEL CRISTINA VIEIRA GOMES FERRARI
LUCAS KENDI FATTORE HIRANO**

**RESTAURAÇÃO DE RETRATOS POR INPAINTING
BASEADO EM APRENDIZADO DE MÁQUINA**

Trabalho apresentado à Escola Politécnica da
Universidade de São Paulo para obtenção do
título de Engenheiro de Computação.

São Paulo
2018

**IZABEL CRISTINA VIEIRA GOMES FERRARI
LUCAS KENDI FATTORE HIRANO**

**RESTAURAÇÃO DE RETRATOS POR INPAINTING
BASEADO EM APRENDIZADO DE MÁQUINA**

Trabalho apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do título de Engenheiro de Computação.

Área de Concentração:
Engenharia de Computação e Sistemas Digitais

Orientador:
Prof. Dr. Edson Satoshi Gomi

São Paulo
2018

Autorizo a reprodução e divulgação total ou parcial deste trabalho, por qualquer meio convencional ou eletrônico, para fins de estudo e pesquisa, desde que citada a fonte.

Catalogação-na-publicação

Ferrari, Izabel Cristina Vieira Gomes
Restauração de Retratos por Inpainting Baseado em Aprendizado de
Máquina / I. C. V. G. Ferrari, L. K. F. Hirano -- São Paulo, 2018.
59 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São
Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1.Processamento de imagens 2.Restauração de fotografias digitais
3.Inpainting 4.Aprendizado de máquina I.Universidade de São Paulo. Escola
Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais
II.t. III.Hirano, Lucas Kendi Fattore

AGRADECIMENTOS

Ao professor Edson Satoshi Gomi, pela orientação e acompanhamento durante todo o projeto.
E a todos que colaboraram direta ou indiretamente na execução deste trabalho.

RESUMO

Atualmente, a restauração de retratos danificados pelo tempo requer a manipulação de softwares de edição de imagens digitais ou a contratação de um serviço profissional. Para tornar esse procedimento mais acessível a usuários leigos, o projeto propõe automatizar o processo de restauração de retratos digitais danificados por vincos ou pequenos rasgos. Para isso, o grupo desenvolveu um algoritmo que identifica automaticamente os trechos danificados da imagem e os mapeia em uma máscara binária. Em seguida, são utilizados dois métodos de *inpainting* para restaurar as regiões mapeadas. O primeiro algoritmo é empregado para preencher as regiões danificadas do rosto, e faz uso de uma arquitetura de redes neurais gerativas. O segundo método é aplicado sobre as demais regiões da imagem, e é baseado nas heurísticas de restauração de imagens por artistas profissionais. Por fim, o rosto e o fundo reconstruídos são combinados para gerar o retrato restaurado. Todo esse processo é executado em uma aplicação *web* em que o usuário precisa apenas fornecer a imagem a ser restaurada. Na avaliação dos resultados obtidos, as máscaras geradas automaticamente são capazes de contemplar a maior parte dos danos reais da imagem. No entanto, também englobam regiões que não estão danificadas. Apesar disso, as restaurações obtidas com os métodos de *inpainting* apresentam preenchimentos adequados ao contexto, com algumas perdas de detalhes e uma certa distorção e/ou falta de nitidez em estruturas únicas, como os olhos. No geral, o projeto cumpre a proposta de automação e faz uso dos melhores resultados possíveis que os métodos de *inpainting* atuais conseguem fornecer.

Palavras-chave: Processamento de imagens. Restauração. *Inpainting*. Aprendizado de máquina.

ABSTRACT

Nowadays, the restoration of damaged portraits due to the passing of time requires the usage of an image manipulation software or a professional restoration service. In order to make this procedure more accessible to lay users, the project intends to automate the process of damaged portraits' restoration, which contain wrinkles or small rips. Therefore, the group developed an algorithm that identifies damaged regions of an image automatically and maps them into a binary mask. After that, two inpainting methods are employed to fill in the mapped regions. The first one fills in the damaged regions of the face, and uses an generative neural network architecture. The second method is applied to the other regions of the image and it is based on image restoration heuristics employed by professional artists. Finally, both the face and background reconstructed are combined to generate the restorated portrait. The process is executed on a web application in which the user needs to supply only the image to be restorated. In the evaluation of the results, the masks that were generated automatically are able to contemplate most of the real damage present on the image. However, they also encompass portions of the image that are not damaged. In spite of that, the restorations obtained with both inpainting methods presented a filling adequate to its context, with the loss of some details and certain distortion and/or lack of sharpness in unique structures, such as the eyes. Overall, the project accomplishes its automation goal and uses the best results that the state-of-art inpainting methods are able to provide.

Keywords: Image processing. Restoration. Inpainting. Machine learning.

LISTA DE FIGURAS

1	Imagen que sofrerá uma dilatação (esquerda) e operador da dilatação (direita). Fonte: Elaborado pelos autores (2018)	17
2	Sobreposição dos pixels da imagem a ser dilatada com seu operador de dilatação (em verde) e região dilatada resultante do processo (em cinza). Fonte: Elaborado pelos autores (2018)	18
3	Imagen original ilustrada em branco junto aos novos pixels resultantes do processo de dilatação, indicados em cinza (esquerda). Imagem final gerada pelo processo de dilatação (direita). Fonte: Elaborado pelos autores (2018)	18
4	Processo de erosão sobre a Figura 3 (direita) com o operador da Figura 1 (direita). A intersecção entre os pixels da imagem e seu operador são destacados em verde e o resultado da erosão é indicado em azul. Fonte: Elaborado pelos autores (2018)	19
5	Intersecção da imagem original com o operador de erosão, ilustrada em verde, junto aos pixels resultantes do processo de erosão, indicados em azul (esquerda). Imagem final gerada pelo processo de erosão (direita). Fonte: Elaborado pelos autores (2018)	19
6	Imagen com duas regiões conectadas por um pixel. Fonte: Elaborado pelos autores (2018)	20
7	Erosão sobre a Figura 6 com o operador da Figura 1 (direita), com a intersecção dos pixels do operador com a imagem original indicados em verde e os pixels resultantes da erosão indicados em azul (esquerda). Imagem final gerada pelo processo de erosão (direita). Fonte: Elaborado pelos autores (2018)	20
8	Dilatação sobre a Figura 6 com o operador da Figura 1 (direita), com a intersecção dos pixels do operador com a imagem original indicados em verde e os pixels resultantes da dilatação indicados em cinza (esquerda). Imagem final gerada pelo processo de dilatação (direita). Fonte: Elaborado pelos autores (2018)	21
9	Imagen com duas regiões distintas, sendo que cada região contém descontinuidades. Fonte: Elaborado pelos autores (2018)	21

10	Dilatação sobre a Figura 9 com o operador da Figura 1 (direita), com a intersecção dos pixels do operador com a imagem original indicados em verde e os pixels resultantes da dilatação indicados em cinza (esquerda). Imagem final gerada pelo processo de dilatação (direita). Fonte: Elaborado pelos autores (2018) . . .	22
11	Erosão sobre a Figura 10 com o operador da Figura 1 (direita), com a intersecção dos pixels do operador com a imagem original indicados em verde e os pixels resultantes da erosão indicados em azul (esquerda). Imagem final gerada pelo processo de erosão (direita). Fonte: Elaborado pelos autores (2018)	22
12	Identificação de vasos sanguíneos num globo ocular utilizando o detector de picos. Fonte: <i>Ridge and tree feature detection on images</i> (YURIN; LEVASHOV, 2013)	24
13	Diagrama do fluxo de dados em um nó de uma rede neural artificial. As entradas e seus respectivos pesos são submetidos a uma função afim seguida de uma função de ativação e, por fim, a saída do nó é gerada. Adaptado de: <i>A Beginner's Guide to Neural Networks and Deep Learning</i> (Autor desconhecido, 2018) . .	26
14	Estrutura de uma rede neural artificial totalmente conectada, composta de uma camada de entrada, uma camada de saída e apenas uma camada intermediária. Adaptado de: <i>A Beginner's Guide to Neural Networks and Deep Learning</i> (Autor desconhecido, 2018)	27
15	Ilustração de uma rede neural profunda (esquerda) e uma rede neural convolucional (direita). Adaptado de: <i>Convolutional Neural Networks (CNNs / Conv-Nets)</i> (KARPATY, 2015)	27
16	Representação das operações realizadas pelo gerador de uma GAN sobre uma entrada z para gerar a saída $G(z)$. Adaptado de: <i>Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks</i> (RADFORD; METZ; CHINTALA, 2015)	30
17	Diagrama do processo de treinamento de uma GAN. Adaptado de: <i>GAN — Wasserstein GAN & WGAN-GP</i> (HUI, 2018)	31
18	Diagrama do processo de treinamento de uma WGAN. Adaptado de: <i>GAN — Wasserstein GAN & WGAN-GP</i> (HUI, 2018)	32
19	Arquitetura da rede generativa com atenção contextual. Adaptado de: <i>Generative Image Inpainting with Contextual Attention</i> (YU et al., 2018b)	33

20	Componentes utilizados na camada de atenção contextual para identificação dos pixels mais relevantes da imagem. Adaptado de: <i>Generative Image Inpainting with Contextual Attention</i> (YU et al., 2018b)	33
21	Diagrama do processo de restauração automática de retratos proposto neste trabalho. Fonte: Elaborado pelos autores (2018)	39
22	Exemplo de retrato danificado por vincos. Fonte: <i>Old photo restoration and repair</i> (CORBIN, 2012)	40
23	Exemplo de identificação do rosto com o algoritmo de Viola-Jones. Adaptado de: <i>Old photo restoration and repair</i> (CORBIN, 2012)	41
24	Exemplo de identificação dos olhos com o algoritmo de Viola-Jones. Adaptado de: <i>Old photo restoration and repair</i> (CORBIN, 2012)	41
25	Imagen original danificada (esquerda), máscara gerada utilizando um limite dinâmico (centro) e máscara gerada usando um detector de bordas (direita). Adaptado de: <i>Old photo restoration and repair</i> (CORBIN, 2012)	42
26	Combinação das máscaras exibidas na Figura 25. Adaptado de: <i>Old photo restoration and repair</i> (CORBIN, 2012)	43
27	Rosto detectado pelo algoritmo de Viola-Jones (esquerda), máscara gerada (centro) e máscara retangularizada (direita). Adaptado de: <i>Old photo restoration and repair</i> (CORBIN, 2012)	44
28	Imagen original (esquerda) e máscara do fundo (direita). Adaptado de: <i>Old photo restoration and repair</i> (CORBIN, 2012)	45
29	Imagen restaurada (esquerda), máscara final sem limiarização (centro) e máscara final limiarizada (direita). Adaptado de: <i>Old photo restoration and repair</i> (CORBIN, 2012)	46
30	Rosto original (esquerda) e sua restauração final (direita). Adaptado de: <i>Old photo restoration and repair</i> (CORBIN, 2012)	47
31	Imagen original (esquerda) e restauração do fundo com algoritmo baseado em heurísticas de <i>inpainting</i> disponível na biblioteca OpenCV (direita). Adaptado de: <i>Old photo restoration and repair</i> (CORBIN, 2012)	48

32	Imagen original (esquerda) e imagem obtida pela união do rosto e fundo restaurados (direita). Adaptado de: <i>Old photo restoration and repair</i> (CORBIN, 2012)	49
33	Imagen original (esquerda) e restauração final obtida após <i>inpainting</i> nas regiões mapeadas pelo detector de pico utilizando o algoritmo disponível no OpenCV (direita). Adaptado de: <i>Old photo restoration and repair</i> (CORBIN, 2012) . . .	50
34	Imagens originais danificadas. Fonte: (a) <i>Old photo restoration and repair</i> (CORBIN, 2012) (b) <i>Photo Restoration Services</i> (CROSSWAYS, 2017) (c) <i>Copy & Restoration of Old Photos</i> (NEW SOMES STUDIO, 2018) (d) <i>Photo Restoration Service</i> (BIG PIXEL GRAPHICS, 2018)	52
35	Restaurações obtidas automaticamente usando o processo desenvolvido neste projeto. Adaptado de: (a) <i>Old photo restoration and repair</i> (CORBIN, 2012) (b) <i>Photo Restoration Services</i> (CROSSWAYS, 2017) (c) <i>Copy & Restoration of Old Photos</i> (NEW SOMES STUDIO, 2018) (d) <i>Photo Restoration Service</i> (BIG PIXEL GRAPHICS, 2018)	52
36	Restaurações obtidas pelos mesmos algoritmos de <i>inpainting</i> utilizados neste projeto, mas com máscaras criadas manualmente. Adaptado de: (a) <i>Old photo restoration and repair</i> (CORBIN, 2012) (b) <i>Photo Restoration Services</i> (CROSSWAYS, 2017) (c) <i>Copy & Restoration of Old Photos</i> (NEW SOMES STUDIO, 2018) (d) <i>Photo Restoration Service</i> (BIG PIXEL GRAPHICS, 2018)	53
37	Restaurações realizadas por artistas profissionais. Fonte: (a) <i>Old photo restoration and repair</i> (CORBIN, 2012) (b) <i>Photo Restoration Services</i> (CROSSWAYS, 2017) (c) <i>Copy & Restoration of Old Photos</i> (NEW SOMES STUDIO, 2018) (d) <i>Photo Restoration Service</i> (BIG PIXEL GRAPHICS, 2018)	53

SUMÁRIO

1	Introdução	13
1.1	Motivação	13
1.2	Contextualização	13
1.3	Objetivo	14
1.4	Metodologia	14
1.5	Organização do trabalho	15
2	Aspectos conceituais	16
2.1	Imagens digitais	16
2.1.1	Definição	16
2.1.2	Operações morfológicas em processamento de imagens	17
2.1.2.1	Dilatação	17
2.1.2.2	Erosão	18
2.1.2.3	Abertura	19
2.1.2.4	Fechamento	21
2.1.3	Segmentação de imagens	22
2.1.3.1	Limiarização	22
2.1.3.2	Detecção de bordas	23
2.1.3.3	Detecção de picos	23
2.2	Reconstrução de imagens	24
2.2.1	Heurísticas de reconstrução manual de imagens	24
2.2.2	Algoritmo baseado nas heurísticas de reconstrução de imagens	25
2.3	Aprendizado de máquina	25
2.3.1	Rede neural artificial	25

2.3.2	Rede neural convolucional	27
2.3.2.1	Etapa de convolução	28
2.3.2.2	Etapa de detecção	29
2.3.2.3	Etapa de agregação	29
2.3.3	Rede adversária generativa	30
2.3.4	Rede adversária generativa de Wasserstein	31
2.3.5	Inpainting de imagens usando rede generativa com atenção contextual .	32
2.3.6	Algoritmo de Viola-Jones	34
3	Tecnologias utilizadas	35
3.1	Linguagem Python	35
3.2	Bibliotecas de código aberto	35
3.2.1	OpenCV	35
3.2.2	NumPy	36
3.2.3	TensorFlow	36
3.3	Flask	36
3.4	Heroku	36
4	Especificação dos requisitos do sistema	38
5	Projeto e implementação	39
5.1	Identificação do rosto	40
5.2	Identificação dos olhos	41
5.3	Criação da máscara binária	42
5.3.1	Criação da máscara do rosto	43
5.3.2	Criação da máscara do fundo	44
5.3.3	Criação da máscara final	45
5.4	Restauração do retrato	46

5.4.1	Restauração do rosto	46
5.4.2	Restauração do fundo	47
5.4.3	Restauração da imagem final	48
5.5	Aplicação <i>web</i>	50
6	Testes e avaliação	51
6.1	Métricas de avaliação	51
6.2	Testes	51
6.3	Resultados	52
7	Conclusões	55
7.1	Conclusões do projeto	55
7.2	Contribuições	55
7.3	Perspectivas de continuidade	56
Referências		57

1 INTRODUÇÃO

Neste capítulo são abordadas as questões introdutórias do projeto. Na seção 1.1 é apresentada a motivação por trás da realização deste trabalho e, em seguida, na seção 1.2, o trabalho é contextualizado em meio aos trabalhos semelhantes existentes. Na seção 1.3 são delimitados os objetivos deste trabalho e na seção 1.4 é apresentada a metodologia empregada na execução do projeto. Por fim, na seção 1.5 é apresentada a organização dos demais capítulos deste trabalho.

1.1 Motivação

Atualmente, apesar dos avanços em inteligência artificial e automação de processos, muitas tarefas sistemáticas são feitas manualmente. Esse é o caso da restauração de fotos antigas deterioradas pelo tempo ou pela manipulação inadequada delas. Fotos nesse estado apresentam, em geral, rasgos e manchas, que comprometem a legibilidade da fotografia.

Para que a imagem deteriorada recupere seus traços originais, muitas famílias contratam serviços de restauração de fotos. Nesse serviço, é fornecida ao restaurador a fotografia digitalizada, sobre a qual são aplicadas manipulações gráficas com o auxílio de softwares específicos de edição de imagens. O resultado é uma foto digital cujas imperfeições são substituídas por um preenchimento que faz sentido no contexto da imagem em questão.

Esse processo é lento e, consequentemente, custoso do ponto de vista financeiro. Além disso, exige pouca criatividade artística do restaurador. Com isso, realizar a restauração de fotografias de forma automatizada reduziria expressivamente o tempo gasto, assim como os custos envolvidos.

1.2 Contextualização

Comercialmente, não existem serviços que realizam a restauração de danos em fotografias de forma totalmente automática. Os softwares existentes no mercado requerem que o usuário selecione manualmente as áreas da fotografia que estão danificadas. A partir dessa seleção

manual, o software consegue preencher automaticamente tais áreas com base nas informações disponíveis na própria imagem.

No meio acadêmico, existem trabalhos semelhantes em duas vertentes distintas. A primeira vertente visa automatizar a criação de máscaras. As máscaras são imagens binárias que indicam quais áreas da imagem contém a informação de interesse (que no caso, são as áreas que contém danos). Nessa vertente, destaca-se o método de detecção de regiões danificadas cujas bordas são pouco aparentes (ZHANG et al., 2016).

A outra vertente de pesquisa tem como objetivo realizar o processo de identificação e restauração sem empregar máscaras binárias. Para isso, é utilizada uma arquitetura de redes neurais artificiais que recebe de entrada a imagem danificada e produz como saída a imagem restaurada. No entanto, sofre de uma expressiva limitação: essa arquitetura não é capaz de gerar uma restauração adequada quando a região danificada contém estruturas únicas, tais como olhos e bocas (LIU; PAN; SU, 2017).

1.3 Objetivo

O objetivo desse projeto é realizar a restauração automática de retratos danificados. O escopo será limitado à restauração de fotografias digitais que contenham pequenos rasgos ou vincos brancos.

Assim, o projeto consiste de duas etapas principais. A primeira etapa é o desenvolvimento de um método para criação de máscaras binárias, para que os trechos danificados sejam identificados e mapeados automaticamente. A segunda etapa é a utilização de métodos de *inpainting*, que é um processo de reconstrução de partes deterioradas ou perdidas de imagens (BERTALMIO et al., 2000), para reconstruir as áreas danificadas com o melhor preenchimento possível.

1.4 Metodologia

Inicialmente, foi definido como tema de interesse a restauração de imagens digitais. Em seguida, foi delimitado como objetivo realizar a restauração automática de retratos danificados e o escopo foi limitado à restauração automática de retratos com rasgos ou vincos brancos.

Feito isso, foi buscado na literatura quais são os procedimentos de restauração utilizados em casos como esse. Foram selecionadas duas implementações computacionais do processo de *inpainting*, sendo a primeira por difusão de linhas isófotas para a região a ser preenchida (conforme apresentado em 2.2.2) e a segunda, por redes gerativas convolucionais. Na arquitetura

apresentada em 2.3.5, foi encontrada uma solução para o problema de reconstrução de regiões com estruturas únicas, que é o caso dos elementos presentes em um rosto.

Em seguida, a automatização do processo foi realizada. Para isso, um processo que utiliza operações morfológicas em imagens para gerar automaticamente a máscara que é utilizada pelos algoritmos de *inpainting* foi desenvolvido.

Por fim, um servidor foi configurado para receber as requisições de restauração. Para cada requisição, ele cria a respectiva máscara com os danos que ele identifica na imagem e alimenta os algoritmos de *inpainting*. Feita a restauração, o servidor devolve ao usuário a imagem final restaurada.

1.5 Organização do trabalho

As demais seções deste trabalho estão organizadas nos capítulos conforme apresentado a seguir:

- Capítulo 2: apresentam-se os conceitos fundamentais de processamento de imagens, assim como a heurística de restauração empregada por artistas profissionais e as implementações do processo de *inpainting* utilizadas neste projeto.
- Capítulo 3: detalham-se as tecnologias utilizadas na implementação do trabalho. Apresentamos a linguagem de programação escolhida e suas principais bibliotecas. Além disso, explicamos o *framework* utilizado para criar a aplicação *web* e a plataforma escolhida para executá-la.
- Capítulo 4: apresentam-se os requisitos do sistema.
- Capítulo 5: apresenta-se a implementação do projeto.
- Capítulo 6: apresentam-se os critérios de avaliação do nosso sistema, junto aos resultados obtidos e uma breve discussão sobre eles.
- Capítulo 7: discutem-se as conclusões deste projeto e possibilidades de trabalhos futuros.

2 ASPECTOS CONCEITUAIS

Nesse capítulo são discutidos os conceitos necessários para o entendimento do projeto. A seção 2.1 e suas subseções definem imagens digitais, como elas são criadas e operações morfológicas envolvidas. Além disso elas abordam as técnicas de segmentação de imagens que são utilizadas nesse trabalho. A seção 2.2 apresenta o processo de reconstrução de áreas danificadas em imagens, conhecido por *inpainting*, e um algoritmo que realiza o processo computacionalmente. Por fim, a seção 2.3 aborda os conceitos fundamentais de aprendizado de máquina e apresenta a arquitetura de redes neurais artificiais que, mediante treinamento prévio com rostos humanos, consegue reconstruir áreas do rosto perdidas.

2.1 Imagens digitais

2.1.1 Definição

Uma imagem pode ser definida como uma função bi-dimensional continua $f(x,y)$. Nessa função, x e y representam as coordenadas espaciais e o valor f da função representa o valor da escala de cinza da imagem naquela coordenada. Imagens digitais podem ser definidas como imagens nas quais as coordenadas x e y , e os valores da função f estão no domínio discreto. Nesse caso, as coordenadas representam a largura e altura da imagem digital. A coordenada dada por $[x,y]$ é chamada de pixel de modo que a imagem digital pode ser representada por uma matriz onde cada elemento dessa matriz é um pixel (GONZALEZ; WOODS, 2002).

Dependendo do tipo de imagem, cada pixel pode ser representado por um ou mais valores. No caso de imagens na escala de cinza, o pixel é representado por um único valor que varia de 0 a 255, sendo que o 0 representa a cor preta e 255 representa a cor branca. Já para imagens coloridas mapeadas no espectro cromático RGB (do inglês: *Red Green Blue*), a função tem três valores, um para cada cor e cada uma também varia de 0 a 255, podendo representar um total de $255^3 = 1.658.1375$ cores.

2.1.2 Operações morfológicas em processamento de imagens

O processamento de imagens consiste na execução de diversas etapas cuja finalidade é extrair informações relevantes de imagens. Existem diferentes métodos para realizar tal extração, que são escolhidos dependendo de qual característica deseja-se obter da imagem que está sob processamento. Nas seções seguintes são abordadas diferentes técnicas e métodos de processamento utilizados nesse trabalho. Eles têm por finalidade filtrar regiões não desejadas, destacar regiões de interesse, moldar as regiões isoladas e refinar os resultados obtidos.

2.1.2.1 Dilatação

A dilatação é um processo morfológico que faz uma imagem “crescer”. Ele envolve duas regiões distintas, aqui chamadas de A e B . Usualmente, A é a imagem a sofrer o processo de dilatação e B é o elemento estrutural. A operação de dilatação é definida na Equação 2.1 (GONZALEZ; WOODS, 2002).

$$A \oplus B = \{z | [(\widehat{B})_z \cap A] \subseteq A\} \quad (2.1)$$

O espaço dimensional na qual será feita a operação é denominado Z , que para imagens bidimensionais torna-se Z^2 . De maneira simplificada e análoga a uma convolução, o elemento estrutural B “desliza” sobre a imagem A . Caso A tenha um elemento sobreposto com a origem de B , toda a região presente no elemento estrutural B durante esse “deslizamento” é adicionado à imagem A , aumentando-a. A Figura 1 ilustra um exemplo de imagem A e elemento estrutural (também chamado de operador) B :

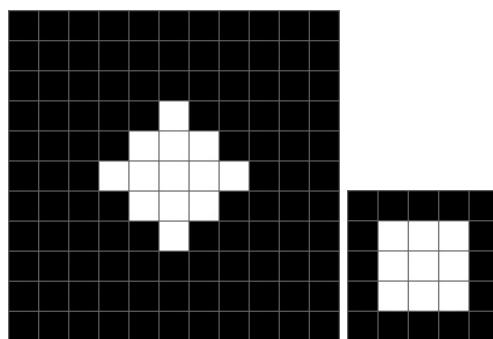


Figura 1: Imagem que sofrerá uma dilatação (esquerda) e operador da dilatação (direita). Fonte: Elaborado pelos autores (2018)

O operador varre a imagem, percorrendo-a em toda a sua extensão. A Figura 2 mostra um exemplo de quando o evento de dilatação ocorre. Nela, tem-se a sobreposição (em verde) da região A (em branco) com a região B (em cinza).

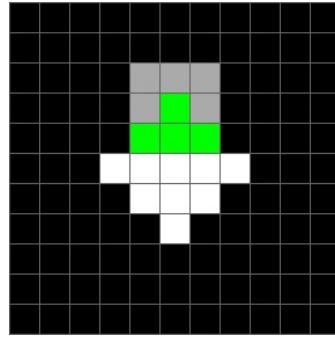


Figura 2: Sobreposição dos pixels da imagem a ser dilatada com seu operador de dilatação (em verde) e região dilatada resultante do processo (em cinza). Fonte: Elaborado pelos autores (2018)

Em seguida, a região B (em cinza) é adicionada à região A e o processo continua por toda a imagem. O resultado final é exibido na Figura 3.

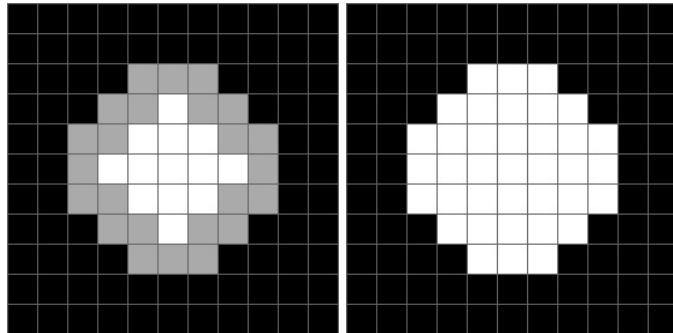


Figura 3: Imagem original ilustrada em branco junto aos novos pixels resultantes do processo de dilatação, indicados em cinza (esquerda). Imagem final gerada pelo processo de dilatação (direita). Fonte: Elaborado pelos autores (2018)

2.1.2.2 Erosão

A erosão é complementar à dilatação e, portanto, seu resultado é uma imagem mais “fina” após o processo. A erosão é formulada conforme apresentado na Equação 2.2 (GONZALEZ; WOODS, 2002).

$$A \ominus B = \{z | (B_z) \subseteq A\} \quad (2.2)$$

Também de maneira análoga à convolução, o processo de erosão consiste em percorrer a imagem A com o operador B . O resultado da erosão é toda a sub-região da região A que contém completamente a região B .

Para exemplificar o processo, utiliza-se como região A o resultado do exemplo da dilatação apresentado na Figura 3 (direita). A Figura 4 ilustra a interação entre as regiões A e B durante a erosão. A região verde é a intersecção das regiões e o pixel em azul foi marcado pois, quando

centrada nele, a região B está totalmente contida na região A e, portanto, esse pixel faz parte do resultado final da erosão. O processo de erosão completo e seu resultado final é exibido na Figura 5.

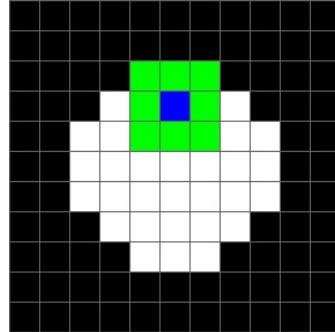


Figura 4: Processo de erosão sobre a Figura 3 (direita) com o operador da Figura 1 (direita). A intersecção entre os pixels da imagem e seu operador são destacados em verde e o resultado da erosão é indicado em azul. Fonte: Elaborado pelos autores (2018)

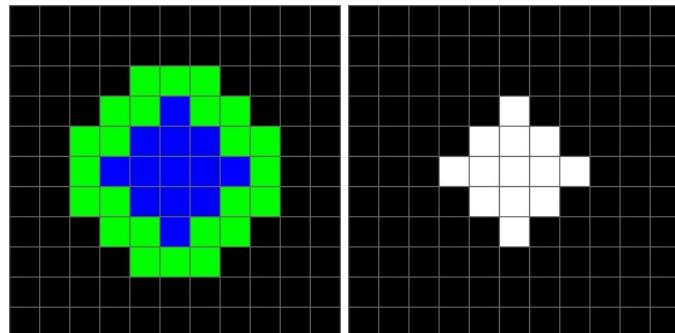


Figura 5: Intersecção da imagem original com o operador de erosão, ilustrada em verde, junto aos pixels resultantes do processo de erosão, indicados em azul (esquerda). Imagem final gerada pelo processo de erosão (direita). Fonte: Elaborado pelos autores (2018)

Como é possível observar, o resultado final retornou a imagem da região A original (apresentada na Figura 1), demonstrando assim a complementaridade das duas operações. No entanto, quando essas operações são aplicadas uma em sequência da outra, o resultado final nem sempre é equivalente à imagem original. Os efeitos de realizar essas operações em sequência são abordados nas próximas seções.

2.1.2.3 Abertura

O processo morfológico conhecido por abertura consiste em realizar uma erosão seguida de uma dilatação na imagem desejada. Ele é definido na Equação 2.3 (GONZALEZ; WOODS, 2002).

$$A \circ B = (A \ominus B) \oplus B \quad (2.3)$$

A operação de abertura remove pequenos ruídos na imagem e deixa seus cantos suavizados, sendo uma boa opção para a remoção de pequenas imperfeições ou regiões indesejadas. A Figura 6 ilustra uma situação na qual duas regiões de interesse foram conectadas através de um pixel indesejado.

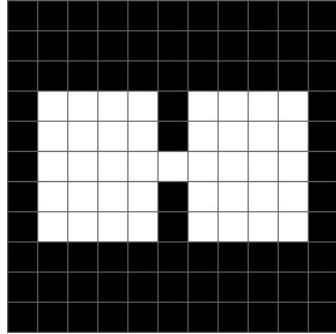


Figura 6: Imagem com duas regiões conectadas por um pixel. Fonte: Elaborado pelos autores (2018)

Considerando o operador dos exemplos anteriores, a Figura 7 ilustra o processo de erosão realizado na imagem da Figura 6. Da mesma forma que no exemplo anterior, os pixels em verde são as regiões que comportam o operador e os pixels em azul são preservados, gerando o resultado final. É possível notar que o pixel indesejado não foi englobado pelo operador durante a operação de erosão. Consequentemente, na próxima etapa, quando é realizada a dilatação da imagem resultante dessa operação, esse pixel não é incluído no resultado final.

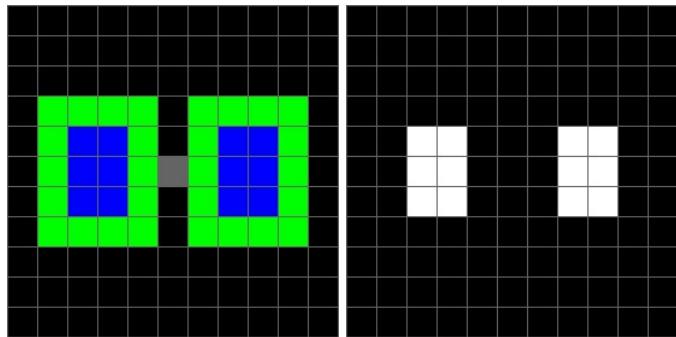


Figura 7: Erosão sobre a Figura 6 com o operador da Figura 1 (direita), com a intersecção dos pixels do operador com a imagem original indicados em verde e os pixels resultantes da erosão indicados em azul (esquerda). Imagem final gerada pelo processo de erosão (direita). Fonte: Elaborado pelos autores (2018)

A Figura 8 ilustra a operação de dilatação e o resultado final obtido. Como é possível observar, o resultado final da operação de abertura removeu o pixel indesejado preservando a forma das regiões originais. Sendo assim, essa operação é ideal para a separação de regiões unidas indesejadamente contanto que o operador utilizado seja escolhido de maneira adequada.

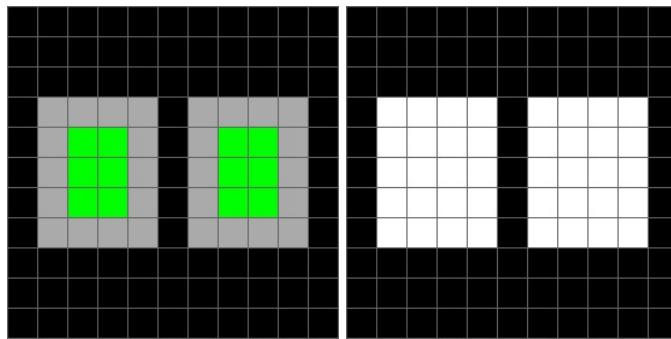


Figura 8: Dilatação sobre a Figura 6 com o operador da Figura 1 (direita), com a intersecção dos pixels do operador com a imagem original indicados em verde e os pixels resultantes da dilatação indicados em cinza (esquerda). Imagem final gerada pelo processo de dilatação (direita). Fonte: Elaborado pelos autores (2018)

2.1.2.4 Fechamento

De maneira análoga ao processo de abertura, o processo de fechamento consiste na execução em sequência das operações mencionadas anteriormente. Neste caso, primeiro é feita uma dilatação e depois uma erosão. A operação de fechamento é formulada conforme a Equação 2.4 (GONZALEZ; WOODS, 2002).

$$A \bullet B = (A \oplus B) \ominus B \quad (2.4)$$

Essa operação, como o próprio nome já diz, “fecha” a imagem. Isso significa que ela é ideal para o preenchimento de suas regiões internas. Ela também serve para realizar a união de regiões separadas indesejadamente. A Figura 9 mostram-se duas regiões com características indesejadas.

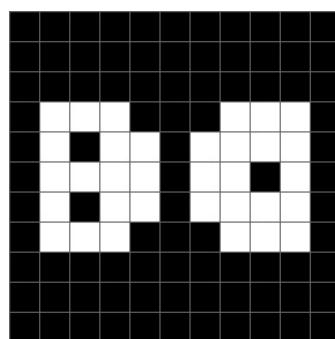


Figura 9: Imagem com duas regiões distintas, sendo que cada região contém descontinuidades. Fonte: Elaborado pelos autores (2018)

Considerando o operador utilizado anteriormente, realiza-se a operação de dilatação cujo resultado é exibido na Figura 10. Observa-se que a dilatação preenche os “buracos” das regiões e também cria uma união entre elas.

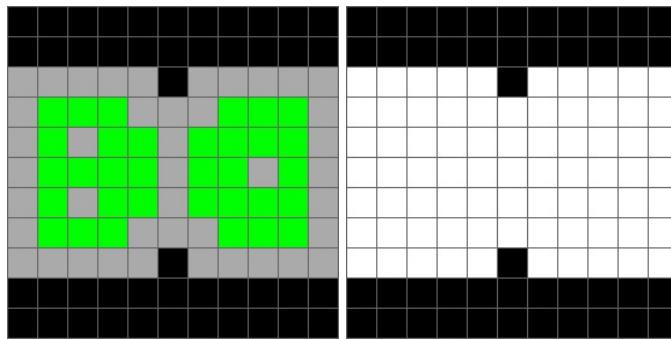


Figura 10: Dilatação sobre a Figura 9 com o operador da Figura 1 (direita), com a intersecção dos pixels do operador com a imagem original indicados em verde e os pixels resultantes da dilatação indicados em cinza (esquerda). Imagem final gerada pelo processo de dilatação (direita). Fonte: Elaborado pelos autores (2018)

Em sequência, realiza-se o processo de erosão, ilustrado na Figura 11, para retornar a imagem ao seu estado original com as alterações desejadas. Como esperado, o processo completo de fechamento preenche os pixels ausentes nas regiões mencionadas e cria uma união entre as duas regiões próximas. Naturalmente, o resultado obtido dependerá da escolha adequada do operador. Regiões a serem preenchidas devem ser menores que o operador, assim como a distância entre as regiões que deseja-se unir.

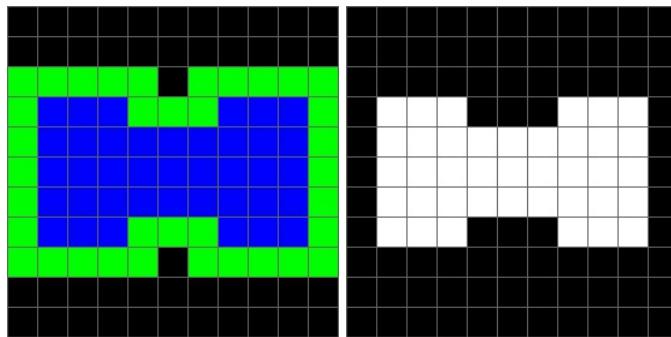


Figura 11: Erosão sobre a Figura 10 com o operador da Figura 1 (direita), com a intersecção dos pixels do operador com a imagem original indicados em verde e os pixels resultantes da erosão indicados em azul (esquerda). Imagem final gerada pelo processo de erosão (direita). Fonte: Elaborado pelos autores (2018)

2.1.3 Segmentação de imagens

2.1.3.1 Limiarização

Dentre as técnicas de segmentação de imagens digitais, existe o processo de limiarização. Ele consiste em definir um limiar que será usado como divisor de regiões da imagem. Uma vez aplicado, as regiões da imagem que apresentarem características de cor abaixo do valor do

limiar serão separadas das regiões cuja as características estiverem acima do limiar. Essas regiões podem então ser manipuladas e utilizadas de diferentes maneiras. O limiar definido pode ter diferentes características dependendo do tipo de imagem com que se está trabalhando e o mapa de cores que ela utiliza.

Para o caso de imagens na escala de cinza, no qual os pixels assumem valores de 0 a 255, o limiar irá dividir a imagem em duas regiões. Essa técnica é utilizada quando se deseja obter uma imagem binária, separando as regiões claras e escuras da imagem. Dada uma imagem $f(x,y)$, uma imagem limiarizada $g(x,y)$ é definida como (GONZALEZ; WOODS, 2002):

$$g(x,y) = \begin{cases} 1, & \text{se } f(x,y) > T \\ 0, & \text{se } f(x,y) \leq T \end{cases} \quad (2.5)$$

onde T é o valor do limiar definido.

2.1.3.2 Detecção de bordas

Outra técnica utilizada na segmentação de imagens é a detecção de bordas. Essa técnica analisa a imagem em busca de regiões onde a transição de cores seja grande o suficiente para ser considerada como uma borda. Normalmente, essas regiões delimitam o contorno de elementos de interesse numa imagem, o que possibilita realizar a segmentação.

Por se tratar da transição de cores em uma imagem, a borda reflete uma taxa de variação nos valores dos pixels presentes. Isso significa que o cálculo da detecção da borda envolve o cálculo do gradiente das regiões na qual se deseja analisar. A matemática completa por trás desse cálculo é extensa e não será abordada em detalhes nesse trabalho. Contudo, o gradiente utilizado pode ser calculado facilmente de acordo com a Equação 2.6 (GONZALEZ; WOODS, 2002):

$$\nabla f = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \quad (2.6)$$

onde ∇f denota o gradiente de uma imagem $f(x,y)$ no local (x,y) . A magnitude $\nabla f = [G_x^2 + G_y^2]^{\frac{1}{2}}$ quantifica a taxa de aumento de $f(x,y)$ na direção de ∇f . Através desses valores, é possível calcular a direção do vetor gradiente e, delimitando valores de referência para a taxa de variação dos pixels, identificar as regiões de borda na imagem.

2.1.3.3 Detecção de picos

O processo descrito na seção 2.1.3.2 é útil para identificar regiões na imagem que apresentam grandes variações de cor. No entanto, o processo descrito nem sempre é suficiente para iden-

tificar todas as regiões desejadas na imagem. Se a transição de cores for suave o suficiente de modo que o gradiente não identifique uma borda, utiliza-se o método conhecido por detector de picos.

O detector de picos analisa a imagem de maneira semelhante ao detector de bordas, porém ele retorna as regiões de máximos e mínimos locais da imagem. Assim ele é capaz de capturar as regiões da imagem que possuem uma transição de cores suave (GONZALEZ; WOODS, 2002). A Figura 12 mostra um exemplo de uma aplicação que utiliza o detector de picos.

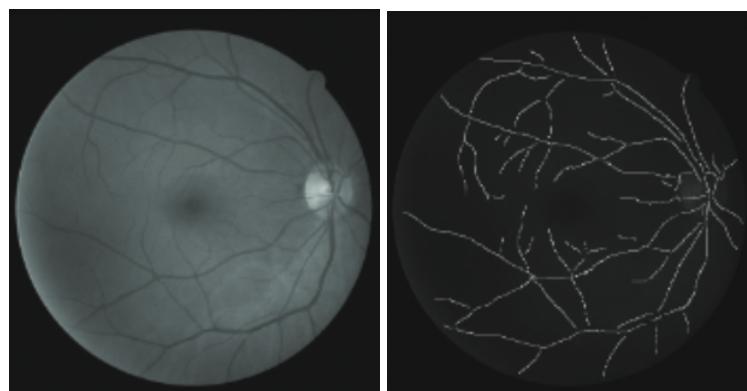


Figura 12: Identificação de vasos sanguíneos num globo ocular utilizando o detector de picos.
Fonte: *Ridge and tree feature detection on images* (YURIN; LEVASHOV, 2013)

2.2 Reconstrução de imagens

2.2.1 Heurísticas de reconstrução manual de imagens

O processo de reconstrução de partes deterioradas ou perdidas de imagens é conhecido por *inpainting*. Tradicionalmente, esse processo é feito de forma manual por restauradores profissionais. A metodologia empregada por eles pode ser descrita pelos seguintes princípios (BERTALMIO et al., 2000):

1. A imagem global determina como preencher a falha, uma vez que o propósito da reconstrução é restaurar a unidade da obra;
2. A estrutura da vizinhança da falha é estendida para dentro da falha (as curvas de nível são desenhadas por meio da prolongação daquelas que se direcionam à fronteira da falha);
3. As diferentes regiões que existem dentro da falha, definidas pelas curvas de nível, são preenchidas com cores que correspondem àquelas da fronteira;

4. Pequenos detalhes são adicionados na região da falha, ou seja, é adicionada textura nessa região.

2.2.2 Algoritmo baseado nas heurísticas de reconstrução de imagens

É possível modelar o processo de *inpainting* por meio de um paralelo entre a função intensidade da imagem e as equações de Navier-Stokes para fluidos incompressíveis em duas dimensões (BERTALMIO; BERTOZZI; SAPIRO, 2001).

Seja ∇ o operador gradiente definido conforme a Equação 2.7.

$$\nabla = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y} \right) \quad (2.7)$$

Seja também Δ o operador laplaciano expresso conforme a Equação 2.8.

$$\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \quad (2.8)$$

A dedução matemática realizada pelos autores na obra mencionada culmina com a solução estacionária da Equação 2.9, que indica que as linhas isófotas são paralelas às curvas de nível do campo ΔI .

$$\nabla^\perp I \cdot \nabla \Delta I \quad (2.9)$$

A partir dessa expressão, os autores propõem um método de *inpainting* cujo princípio é percorrer as bordas das regiões conhecidas para as regiões desconhecidas (considerando que as bordas são contínuas). As linhas isófotas (ou seja, linhas que passam por pontos de mesma intensidade) são extrapoladas, desde que sejam correspondentes aos vetores do gradiente nas bordas da região que está sendo preenchida. Para isso, métodos inspirados nas equações de dinâmica dos fluidos são utilizados.

2.3 Aprendizado de máquina

2.3.1 Rede neural artificial

Uma rede neural artificial é um conjunto de elementos de computação não-linear, conhecidos como nós, organizados em rede (GONZALEZ; WOODS, 2002).

Cada nó recebe um vetor de entrada $x = [x_1, x_2, \dots, x_m]$, seu respectivo vetor de pesos $w = [w_1, w_2, \dots, w_m]$ e um *bias* w_0 . O nó aplica uma função afim (em geral, a soma ponderada dos elementos em x com os pesos em w acrescida do *bias*) e passa o resultado obtido por uma função de ativação. Essa função recebe o resultado da soma ponderada e aplica uma operação

não-linear sobre ela, fornecendo um valor de saída. A Figura 13 ilustra o funcionamento de um nó.

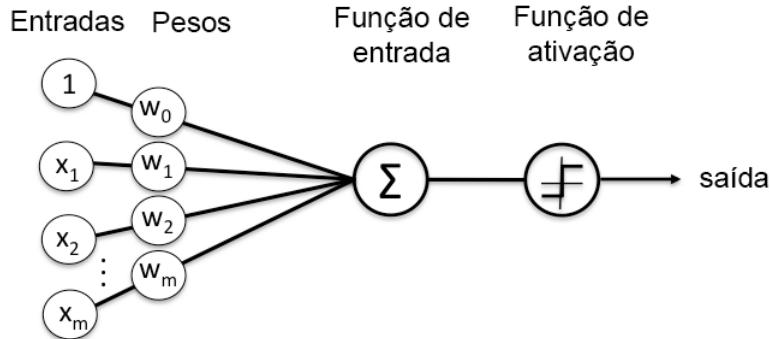


Figura 13: Diagrama do fluxo de dados em um nó de uma rede neural artificial. As entradas e seus respectivos pesos são submetidos a uma função afim seguida de uma função de ativação e, por fim, a saída do nó é gerada. Adaptado de: *A Beginner's Guide to Neural Networks and Deep Learning* (Autor desconhecido, 2018)

Nas implementações de redes neurais modernas, a recomendação padrão é utilizar como função de ativação a unidade linear retificada (do inglês, *rectified linear unit*), mais conhecida por ReLU, expressa conforme a Equação 2.10.

$$g(z) = \max\{0, z\} \quad (2.10)$$

Existem também outras funções de ativação, tal como a *softmax*, formulada conforme a Equação 2.11.

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}}, j = 1, 2, \dots, K \quad (2.11)$$

Em seu arranjo mais simples, uma rede neural é denominada *perceptron* e possui apenas uma camada de entrada seguida por uma camada de saída. No entanto, é possível incluir camadas intermediárias, conhecidas por *hidden layers*. Nesse caso, é possível dispor os nós de forma com que aqueles que estiverem em uma mesma camada não compartilhem conexões. Assim, a informação é propagada na rede de forma unidirecional, da entrada para a saída. Nessas redes sem realimentação, é usual utilizar um arranjo de camadas denominado *fully connected layer*, em que o valor de saída de um nó da camada anterior é um dos valores de entrada de todos os nós da próxima camada. A Figura 14 ilustra uma rede totalmente conectada.

Caso a rede tenha mais que uma camada intermediária, ela passa a ser chamada de rede neural profunda e é normalmente conhecida por *deep neural network (DNN)*.

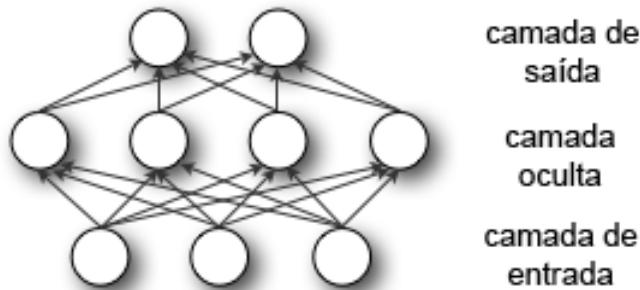


Figura 14: Estrutura de uma rede neural artificial totalmente conectada, composta de uma camada de entrada, uma camada de saída e apenas uma camada intermediária. Adaptado de: *A Beginner's Guide to Neural Networks and Deep Learning* (Autor desconhecido, 2018)

2.3.2 Rede neural convolucional

Rede neural convolucional (também conhecida por *ConvNet*) é uma arquitetura de rede do tipo *feedforward* especializada, utilizada para processamento de dados estruturados matricialmente. Devido a essa característica, essa rede é amplamente empregada em problemas de reconhecimento de objetos em fotos (GOODFELLOW; BENGIO; COURVILLE, 2016).

É importante recordar que uma foto digital possui duas dimensões espaciais: altura e largura. Além disso, possui também uma matriz bidimensional cujo tamanho corresponde às dimensões da imagem. Cada elemento dessa matriz corresponde a um pixel, que é representado por um vetor com três valores (vermelho, verde e azul). Define-se como *canal de cor* uma camada dessa imagem, de forma com que cada camada contenha os valores correspondentes a uma das cores do RGB.

Para lidar com esse tipo de entrada, a arquitetura de uma rede neural convolucional deve aceitar uma entrada tridimensional e sua organização interna também deve ser multidimensional, tal como ilustrado na Figura 15.

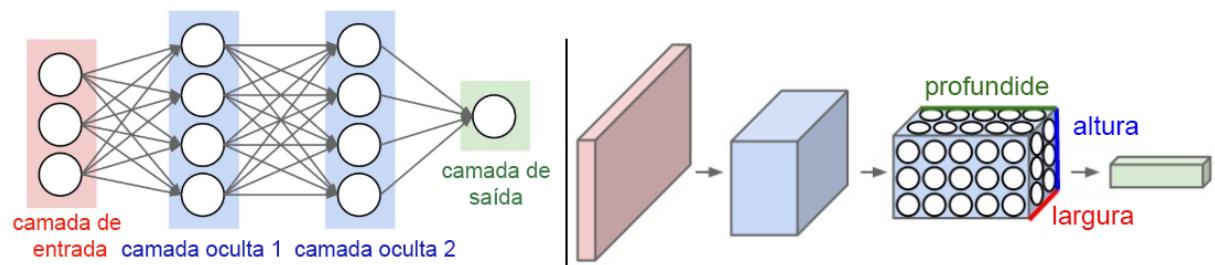


Figura 15: Ilustração de uma rede neural profunda (esquerda) e uma rede neural convolucional (direita). Adaptado de: *Convolutional Neural Networks (CNNs / ConvNets)* (KARPATHY, 2015)

No geral, a arquitetura de uma rede neural convolucional é formada pelas camadas de entrada e

saída e pelo menos uma camada de convolução, que possui três estágios: no primeiro, o estágio de convolução, ela recebe a imagem de entrada e realiza a operação de convolução com um *kernel* (também conhecido por filtro) para gerar a saída, usualmente denominada *feature map* ou *activation map*. Em seguida, as saídas obtidas passam pelo estágio de detecção, em que funções de ativação não-lineares são aplicadas para produzir uma saída que será utilizada pelo último estágio, o de agregação.

Tanto a imagem de entrada quanto o filtro são vetores multidimensionais, que são conhecidos na literatura por *tensores*. É importante mencionar que quando fala-se de convolução no contexto de redes neurais, na verdade fala-se da aplicação de diversas convoluções em paralelo, uma para cada filtro, para extraer diferentes características da imagem de entrada (bordas, orientação, bloco de cores semelhantes, etc.).

2.3.2.1 Etapa de convolução

A operação de convolução conforme sua descrição matemática emprega a inversão do *kernel*, necessária para obter a propriedade comutativa. No entanto, essa propriedade é dispensável no contexto de aprendizado de máquina. Assim, muitas bibliotecas de redes neurais implementam um método denominado *cross-correlation*, similar à convolução sem inversão do *kernel* (GOODFELLOW; BENGIO; COURVILLE, 2016). Seja I o tensor de entrada e K o tensor do *kernel*, então:

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(i + m, j + n)K(m, n) \quad (2.12)$$

O tensor de entrada contém, para cada pixel da imagem de entrada, os seus respectivos valores RGB, e, para cada canal de cor, o tensor do filtro possui parâmetros que serão ajustados por meio do processo de treinamento da rede. Supondo um *kernel* K quadridimensional, sendo o elemento $K_{i,j,k,l}$ a intensidade da conexão entre uma unidade no canal i da saída e a unidade no canal j da entrada, com um *offset* de k linhas e l colunas entre a unidade de saída e de entrada. Suponha que a entrada consiste de uma imagem representada por um tensor V , sendo cada elemento $V_{i,j,k}$ o valor da unidade de entrada no canal i na linha j e coluna k . Suponha uma saída Z no mesmo formato de V . Se Z é produzido a partir da convolução de V por K , sem inversão do *kernel*, então:

$$Z_{i,j,k} = \sum_{l,m,n} V_{i+m-1, j+n-1} K_{i,l, j,m} \quad (2.13)$$

Pode-se ainda desconsiderar algumas posições do *kernel* para reduzir o custo computacional em detrimento de uma extração de características menos detalhada. Para isso, define-se um passo s para que apenas realize-se a amostragem a cada s pixels em cada direção da saída. Com isso,

define-se a convolução com sub-amostragem por:

$$Z_{i,j,k} = c(K, V, s)_{i,j,k} = \sum_{l,m,n} [V_{l,(j-1)s+m,(k-1)s+n} K_{i,l,m,n}] \quad (2.14)$$

Até o momento foram apresentados dois hiper-parâmetros de uma rede convolucional: quantidade de filtros que ela emprega e seu passo. Existe um terceiro hiper-parâmetro essencial para a implementação de uma rede convolucional, que é conhecido por *zero-padding*. Esse parâmetro é responsável por preencher a entrada V com zeros para evitar com que a largura da representação obtida diminua por um pixel em comparação ao tamanho do *kernel* a cada camada.

2.3.2.2 Etapa de detecção

No estágio seguinte, a função de ativação é aplicada sobre os *feature maps* obtidos no primeiro estágio. A função de ativação usualmente empregada é a ReLU, conforme equacionada em 2.10.

2.3.2.3 Etapa de agregação

Por fim, o último estágio é o de agregação, conhecido também por *max pooling*, *downsampling* ou ainda *subsampling*. Esse estágio recebe as saídas não-lineares produzidas pela etapa anterior e, para cada uma, replica para a saída o valor máximo encontrado em uma vizinhança retangular. Dessa forma, apenas os valores que apresentaram maior correlação com uma dada característica serão preservados e juntos formarão um espaço dimensional menor.

Existem outras funções de agregação que também podem ser utilizadas em vez do máximo local, tais como a média da vizinhança retangular, a média ponderada com base na distância até o pixel do centro, etc. Todas têm em comum a propriedade de tornar a representação da imagem invariante a pequenas translações que a imagem de entrada pode sofrer. Essa invariância adquirida é uma propriedade útil quando deseja-se saber se uma determinada característica existe na imagem, e não sua posição exata. Por exemplo, ao determinar se uma imagem contém um rosto, basta saber que no lado esquerdo existe um olho, assim como no lado direito. A localização exata dos olhos não nos interessa, apenas sua posição relativa (GOODFELLOW; BENGIO; COURVILLE, 2016).

2.3.3 Rede adversária generativa

Rede adversária generativa, do inglês *generative adversarial network (GAN)*, foi proposta por Ian Goodfellow em 2014 e é inspirada na teoria dos jogos. Segundo o modelo proposto, a GAN é composta de duas redes profundas: o gerador G e o discriminador D . Fornece-se como entrada para o gerador um ruído z obtido a partir da amostragem de uma distribuição normal ou uniforme. A partir dessa entrada, o gerador cria a imagem $x = G(z)$ empregando múltiplas convoluções transpostas para aumentar a taxa de amostragem de z afim de gerar a imagem x . Esse processo é ilustrado na Figura 16.

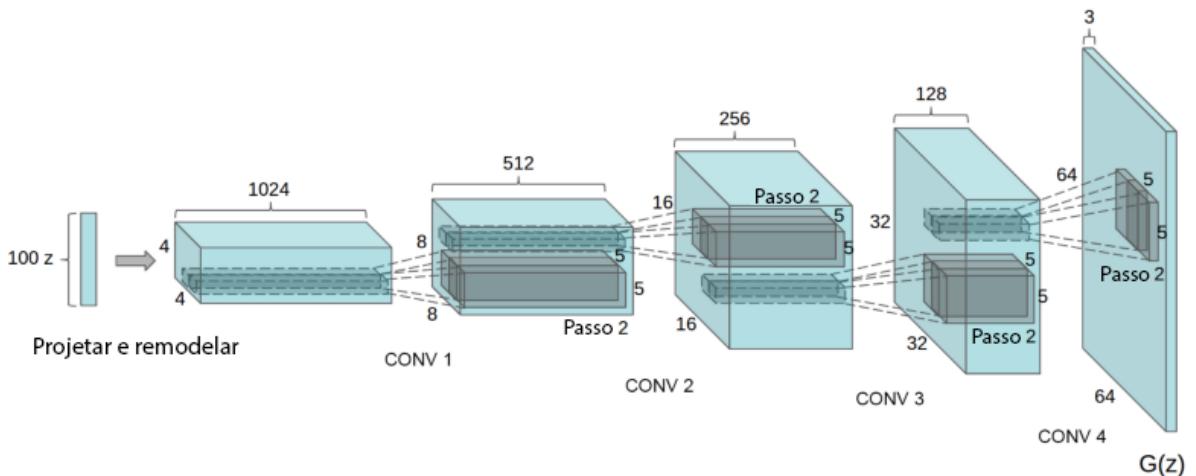


Figura 16: Representação das operações realizadas pelo gerador de uma GAN sobre uma entrada z para gerar a saída $G(z)$. Adaptado de: *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks* (RADFORD; METZ; CHINTALA, 2015)

O discriminador, por sua vez, processa as imagens de treino (reais) e aquelas geradas pelo gerador (falsas) e, para cada imagem analisada, produz uma saída $D(x)$ com a probabilidade de que a imagem x seja real. Dessa forma, $D(x) = 0$ corresponde a uma imagem falsa e $D(x) = 1$, a uma verdadeira. Assim, para a função objetivo do discriminador, usa-se a função da entropia cruzada, comumente utilizada em aprendizado profundo. Essa função descreve a perda entre duas distribuições de probabilidade, indicando o quanto próximo a distribuição atual $q(x)$ está daquela desejada $p(x)$:

$$H(p, q) = - \sum_x p(x) \log q(x) \quad (2.15)$$

O objetivo do discriminador pode então ser formulado conforme a Equação 2.16.

$$\max_D V(D) = E_{x \sim p_{\text{dados}}(x)}[\log D(x)] + E_{z \sim p_z(z)}[\log(1 - D(G(z)))] \quad (2.16)$$

O primeiro termo visa aprimorar o reconhecimento de imagens reais e o segundo, das imagens geradas. Para o gerador, deseja-se que ele produza imagens com o maior valor possível de $D(x)$. Sua função objetivo é formulada conforme apresentado na Equação 2.17.

$$\min_G V(G) = E_{z \sim p_z(z)}[\log(1 - D(G(z)))] \quad (2.17)$$

Com 2.16 e 2.17, define-se a GAN como uma função minimax, expressa pela Equação 2.18.

$$\min_G \max_D V(D, G) = E_{x \sim p_{\text{dados}}(x)}[\log D(x)] + E_{z \sim p_z(z)}[\log(1 - D(G(z)))] \quad (2.18)$$

Com isso, é possível ilustrar o processo de treinamento da rede conforme apresentado na Figura 17. A imagem real e a imagem gerada pelo gerador são fornecidas ao discriminador de forma alternada. O discriminador então gera a probabilidade D de que a imagem seja verdadeira e por fim o custo é calculado e retro-alimentado ao discriminador e gerador, conforme as equações 2.16 e 2.17.

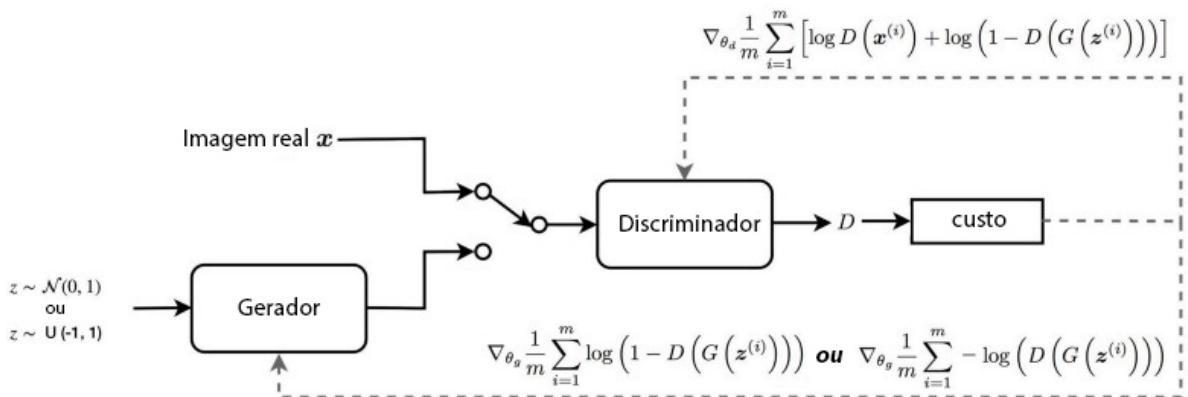


Figura 17: Diagrama do processo de treinamento de uma GAN. Adaptado de: *GAN—Wasserstein GAN & WGAN-GP* (HUI, 2018)

2.3.4 Rede adversária generativa de Wasserstein

A rede adversária generativa de Wasserstein (conhecida por WGAN) foi originalmente proposta no trabalho intitulado “Wasserstein GAN” (ARJOVSKY; CHINTALA; BOTTOU, 2017). Os autores propuseram este novo modelo de rede generativa para mitigar alguns problemas inerentes às GANs originais. Um dos problemas existentes é conhecido por *diminished gradient*, que ocorre quando o discriminador se torna muito bem sucedido em sua tarefa a tal ponto que o gradiente do gerador praticamente zera e passa a não aprender mais. Existe também o problema da não convergência e o de produção limitada de imagens devido ao colapso do gerador.

Com isso, os autores propuseram uma nova função custo junto a algumas alterações estruturais

para solucionar os problemas citados. Os componentes e o processo de treinamento de uma rede gerativa de Wasserstein são ilustrados na Figura 18.

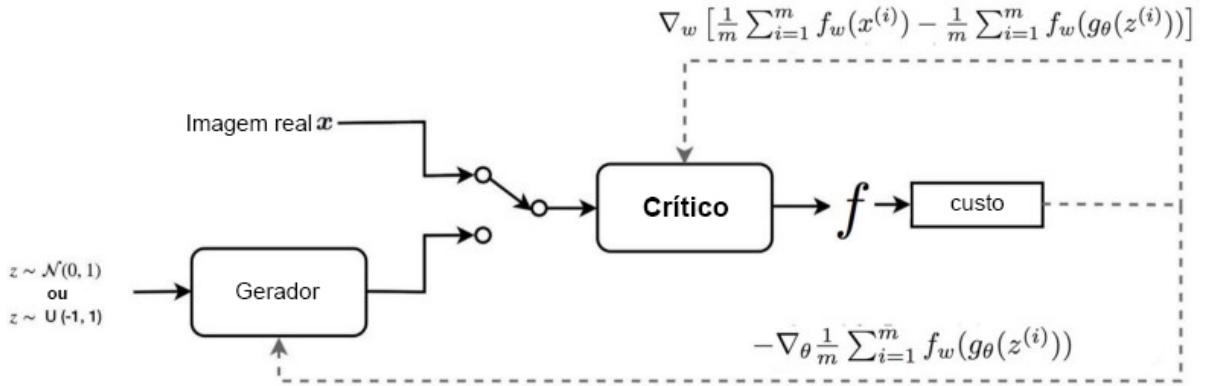


Figura 18: Diagrama do processo de treinamento de uma WGAN. Adaptado de: *GAN — Wasserstein GAN & WGAN-GP* (HUI, 2018)

2.3.5 Inpainting de imagens usando rede gerativa com atenção contextual

Conforme apresentado nas seções 2.3.3 e 2.3.4, a estrutura de redes gerativas é particularmente útil para a criação de novas imagens a partir dos dados de treino. No entanto, para o problema de reconstrução de regiões perdidas da imagem, as redes convolucionais treinadas nessa arquitetura produzem, em geral, imagens com estruturas distorcidas ou texturas embaçadas, inconsistentes com as regiões ao redor.

Argumenta-se que os resultados embaçados ou distorcidos são fruto da incapacidade das redes convolucionais em obter informações relevantes de regiões espacialmente distintas, e utilizá-las no treino. Essa hipótese motivou a criação de uma arquitetura ilustrada na Figura 19. Nela, a imagem original e sua máscara são passadas por uma rede convolucional simples, treinada para gerar um rascunho da imagem que falta. Em seguida, esse primeiro preenchimento é passado por uma rede convolucional com atenção contextual e, paralelamente, por uma rede convolucional simples. As saídas de ambas são agregadas e geram a imagem final (YU et al., 2018b).

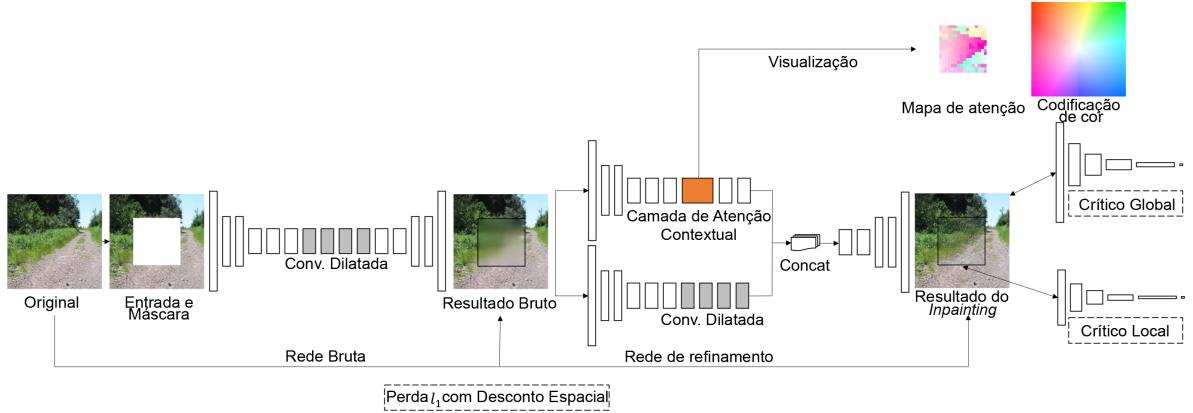


Figura 19: Arquitetura da rede generativa com atenção contextual. Adaptado de: *Generative Image Inpainting with Contextual Attention* (YU et al., 2018b)

A atenção contextual consiste em utilizar as características das regiões conhecidas como filtros convolucionais para processar as regiões geradas. Seus componentes são ilustrados na Figura 20. Primeiramente, são extraídos regiões de dimensões 3x3 pixels da imagem, que são utilizadas como filtros convolucionais. Para que seja possível parear as regiões do primeiro plano (*foreground*, que corresponde ao trecho que foi preenchido na etapa anterior pela primeira rede convolucional) às regiões do segundo plano (*background*, que contém as informações da imagem original), utiliza-se o produto interno normalizado. Seu resultado representa a semelhança entre os trechos do primeiro e segundo plano. Nessa arquitetura, o processo descrito é implementado por meio de uma convolução. Em seguida, essa semelhança é ponderada com uma função *softmax* (apresentada na Equação 2.11) ao longo de toda a imagem de fundo, obtendo uma pontuação para cada pixel. Por fim, essa pontuação é empregada no momento em que os trechos extraídos da imagem de fundo são utilizados como filtros deconvolucionais para reconstruir o primeiro plano.

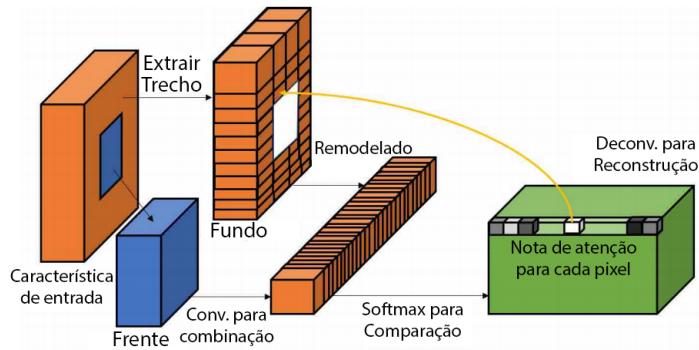


Figura 20: Componentes utilizados na camada de atenção contextual para identificação dos pixels mais relevantes da imagem. Adaptado de: *Generative Image Inpainting with Contextual Attention* (YU et al., 2018b)

2.3.6 Algoritmo de Viola-Jones

O Algoritmo de Viola-Jones (VIOLA; JONES, 2001), mais conhecido por Haar Cascade, é um algoritmo de busca de objetos em imagens. Para caracterizar um objeto, o Haar Cascade utiliza Haar Features, que são máscaras que capturam variações de luminosidade (principalmente nas bordas) dos diversos componentes de uma imagem, em diferentes amplitudes e direções. Os valores que definem um determinado objeto são aprendidos por um algoritmo de aprendizado de máquina denominado AdaBoost. Esse algoritmo é treinado com imagens positivas (ou seja, imagens que contém o objeto de interesse) e negativas. Por fim, gera diversos classificadores, um para cada Haar Feature.

O Algoritmo de Viola-Jones utiliza esses classificadores em cascata para encontrar o objeto de interesse na imagem. Para isso, define uma janela e redimensiona as máscaras para o tamanho definido. Em cada janela, o algoritmo seleciona e executa um dos classificadores com base nos valores dos pixels da imagem sob aquela janela. Se o classificador retorna falso, o algoritmo segue para a próxima janela. Se acabaram as janelas, ele conclui que o objeto buscado não existe na imagem. Caso contrário, o algoritmo passa para o próximo classificador e repete o processo até passar por todos os classificadores e retorna o objeto identificado.

3 TECNOLOGIAS UTILIZADAS

Este capítulo aborda brevemente as tecnologias que foram empregadas na realização desse trabalho. As seções 3.1 a 3.4 descrevem as linguagens, bibliotecas, *frameworks* e plataforma utilizados.

3.1 Linguagem Python

Foi escolhida a linguagem de programação Python para o desenvolvimento do software. Python é uma linguagem de programação interpretada e orientada a objetos. Seu código é aberto e atualmente conta com diversas bibliotecas e arcabouços para auxiliar no desenvolvimento das mais diversas aplicações.

3.2 Bibliotecas de código aberto

3.2.1 OpenCV

OpenCV (Open Source Computer Vision Library) é uma biblioteca de código aberto desenvolvida para visão computacional e aprendizado de máquina. Contando com mais de 2500 algoritmos, o OpenCV é a biblioteca de escolha de grandes empresas como Google, Microsoft, Intel e IBM. Através do uso de seus algoritmos, é possível fazer a identificação de objetos e rostos em imagens, filtrar informações em imagens, rastrear elementos de interesse na imagem, dentre outros. Esse trabalho usa extensivamente os recursos do OpenCV. O processamento da imagem a ser restaurada e a geração da máscara binária são realizados com métodos e funções presentes nessa biblioteca.

3.2.2 NumPy

Uma vez que o trabalho é realizado com imagens, que são tratadas como matrizes quando processadas computacionalmente, foi necessário usar uma biblioteca que permita a manipulação desse tipo de dado. De acordo com sua documentação, o NumPy é um importante pacote para computação científica com Python. Dentre suas capacidades, ele possui um objeto vetorial N-dimensional, funções sofisticadas para o trabalho com vetores e diversas capacidades de álgebra linear. A grande motivação para o uso da biblioteca NumPy para esse trabalho foi a presença de métodos e objetos específicos para manipulação de vetores multidimensionais. Seus métodos permitem a fácil edição de grandes matrizes de maneira rápida e eficiente.

3.2.3 TensorFlow

TensorFlow é uma biblioteca de código aberto voltada para o desenvolvimento de aplicações que utilizem aprendizado de máquina. Por possuir uma API de fácil uso, ele permite o desenvolvimento em alto nível, propiciando agilidade e eficiência. Como mencionado anteriormente, tensores são vetores multidimensionais e é desse conceito que se origina o nome da biblioteca. Seu trabalho com tensores torna o TensorFlow uma excelente biblioteca para se projetar redes neurais artificiais.

3.3 Flask

Flask é um *micro-framework web* escrito em Python. Ele possui um núcleo simples, sem camada de abstração do banco de dados, validação de formulários ou afins. No entanto, ele suporta extensões que possuem essas funcionalidades, possibilitando integrá-las ao projeto. Seu uso é recomendado para a criação rápida de aplicações *web* simples, especialmente no caso delas executarem processos em Python.

3.4 Heroku

O Heroku é uma Plataforma como Serviço (PaaS), ou seja, é um serviço que provê ao cliente um ambiente para executar a aplicação. No caso do Heroku, esses ambientes são containers denominados dynos, que executam o sistema operacional Linux. Eles são isolados um dos outros, apesar de compartilharem a mesma infraestrutura computacional subjacente. É permitido ao cliente escalar seu dyno horizontalmente, adicionando mais dynos para permitir múltiplas

requisições HTTP simultâneas, por exemplo, ou verticalmente, aumentando a memória RAM disponível em seu container.

Cada dyno contém um sistema de arquivos efêmero, com uma cópia do código mais recente implantado pelo cliente. Durante sua execução, o dyno pode utilizar esse sistema de arquivos para escrita. No entanto, todos os arquivos serão descartados no momento em que o container é reiniciado, o que ocorre aproximadamente uma vez ao dia, devido às suas rotinas de gerenciamento.

Uma outra característica dos dynos é que eles podem executar sob três configurações distintas: *web*, *worker* e *one-off*. A configuração do tipo *web* permite ao dyno receber requisições HTTP. O dyno do tipo *worker* é utilizado para tarefas que executam em segundo plano, geralmente pré-agendadas. Por fim, o dyno *one-off* é um container temporário, utilizados para tarefas administrativas como migração de banco de dados.

4 ESPECIFICAÇÃO DOS REQUISITOS DO SISTEMA

Para que o projeto cumpra com os seus propósitos, é necessário que o sistema gere automaticamente a restauração da fotografia digital fornecida pelo usuário.

Para isso, é imprescindível que o processo de geração das máscaras binárias consiga identificar as áreas danificadas com precisão. Ou seja, as regiões identificadas como danos mas que, na realidade, são áreas íntegras da fotografia, devem ser as mínimas possíveis. Assim, minimizando os falsos positivos, as informações originais da imagem são preservadas. Similarmente, as regiões identificadas como áreas íntegras, mas que, na realidade, são áreas danificadas, também devem ser as menores possíveis. Dessa forma, minimizando os falsos negativos, maximiza-se a cobertura das regiões que devem passar pelo processo de restauração.

Finalmente, é necessário que o sistema consiga distinguir corretamente o rosto do fundo da imagem, para que ele faça uso das implementações do processo de *inpainting* corretas para cada caso.

5 PROJETO E IMPLEMENTAÇÃO

A arquitetura do sistema é composta de dois componentes principais. Um dos componentes é a aplicação *web* desenvolvida para que o processo de restauração possa ser solicitado por meio de uma interface intuitiva e de fácil acesso.

O outro componente realiza o processo de restauração da fotografia digital. Para isso, ele realiza a varredura da imagem original e gera automaticamente a máscara binária. Essa máscara informa quais regiões foram identificadas como deteriorações e, portanto, devem ser reconstruídas. Em seguida, realiza a identificação de rostos e olhos e os separa do fundo da imagem. Feito isso, o rosto e o fundo da imagem são alimentados aos algoritmos de *inpainting* adequados. Na Figura 21 ilustra-se mais detalhadamente o fluxo da implementação realizada.

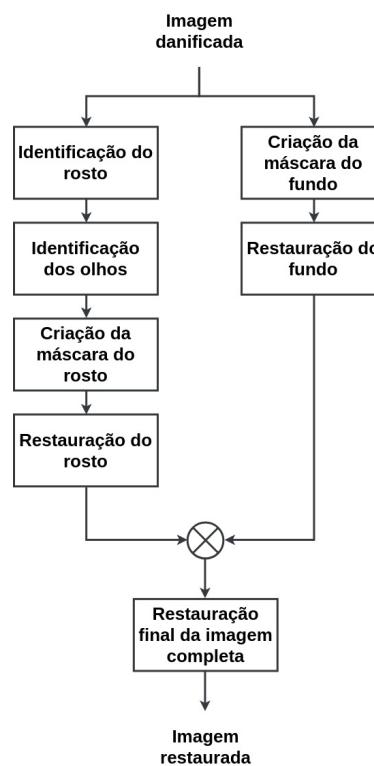


Figura 21: Diagrama do processo de restauração automática de retratos proposto neste trabalho.
Fonte: Elaborado pelos autores (2018)

Cada etapa do processo apresentado no diagrama da Figura 21 é detalhado nas seções 5.1 a 5.4. Por fim, a aplicação *web* é apresentada em detalhes na seção 5.5 Para exemplificar o funcionamento do processo, cada seção apresentará os resultados da execução de sua respectiva etapa exemplificando com base no retrato danificado da Figura 22.



Figura 22: Exemplo de retrato danificado por vincos. Fonte: *Old photo restoration and repair* (CORBIN, 2012)

5.1 Identificação do rosto

A identificação do rosto é realizada pelo método de reconhecimento facial disponibilizado pelo OpenCV. Esse método é um classificador Haar Cascade já treinado com imagens positivas e negativas de rostos. Para utilizá-lo, é necessário importar o arquivo XML que contém as informações do treino e então instanciar o classificador com a imagem a ser analisada. O resultado da identificação de um rosto com um Haar Cascade é apresentado na Figura 23.

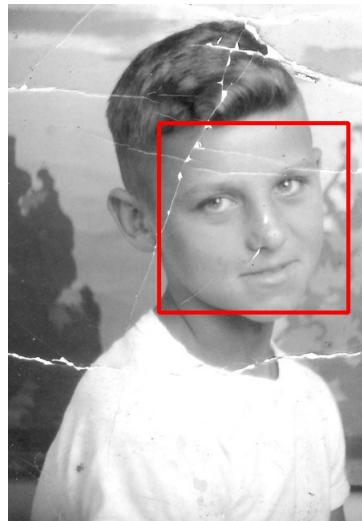


Figura 23: Exemplo de identificação do rosto com o algoritmo de Viola-Jones. Adaptado de: *Old photo restoration and repair* (CORBIN, 2012)

5.2 Identificação dos olhos

Utiliza-se o mesmo classificador da seção 5.1, agora treinado com imagens positivas e negativas de olhos, para reconhecer os olhos presentes na imagem. Como sabe-se que os eles estarão sempre dentro da região identificada como o rosto, define-se o rosto como a região de interesse (do inglês, *Region Of Interest* ROI) e procura-se os olhos apenas dentro da ROI, poupando recursos computacionais e mitigando a possibilidade de falsos positivos. O resultado proveniente da identificação dos olhos é apresentado na Figura 24.

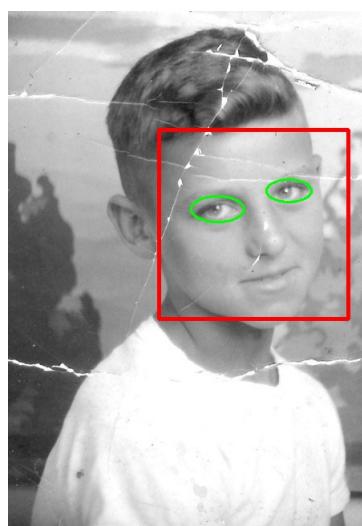


Figura 24: Exemplo de identificação dos olhos com o algoritmo de Viola-Jones. Adaptado de: *Old photo restoration and repair* (CORBIN, 2012)

5.3 Criação da máscara binária

A máscara binária tem a função de definir quais dados sofrerão algum tipo de tratamento. No caso dos retratos danificados, a máscara define quais regiões da imagem passam pelo processo de restauração. Para tornar automática a geração da máscara, procura-se entender, dentro do escopo, quais regiões seriam incluídas nela. Conclui-se que as regiões com alto valor de brilho e delimitadas por bordas são fortes candidatas a serem regiões de dano.

Assim, o processo se inicia com a geração de duas máscaras. Uma delas utiliza um limite dinâmico de valor brilho no pixel e a outra utiliza um detector de bordas. Na Figura 25 são exibidos os resultados dessas máscaras.



Figura 25: Imagem original danificada (esquerda), máscara gerada utilizando um limite dinâmico (centro) e máscara gerada usando um detector de bordas (direita). Adaptado de: *Old photo restoration and repair* (CORBIN, 2012)

Percebe-se que nenhuma das máscaras exibidas funcionaria para a restauração da imagem original. Ambas conseguem identificar as regiões danificadas, porém elas também englobam muitas outras regiões que contém informações intactas.

A solução encontrada foi uma combinação dessas duas máscaras utilizando uma operação lógica *and* bit a bit. Sendo assim, a máscara final contém os pixels presentes tanto na primeira quanto na segunda máscara. Seu resultado é exibido na Figura 26.



Figura 26: Combinação das máscaras exibidas na Figura 25. Adaptado de: *Old photo restoration and repair* (CORBIN, 2012)

A ideia por trás dessa combinação foi identificar as características presentes nos danos e como separá-los do resto da imagem. Os rasgos, de maneira geral, apresentam uma cor tendendo ao branco. No momento da criação da máscara, a imagem original estava mapeada no espaço de cores BGR (*Blue Green Red*), que é o mapa padrão para leitura de imagens da biblioteca OpenCV. Utilizando um método de conversão, converte-se a imagem para o mapa de cores HSV (*Hue Saturation Value*) e assim, cria-se a máscara com um filtro de *value*, valor que representa o brilho presente no pixel.

Além da cor destacada, os rasgos também têm como característica serem traços finos que percorrem a imagem. Considerando essa propriedade, é possível aplicar na imagem uma operação chamada *Canny Edge detection*, que utiliza o gradiente da imagem para identificar os traços mencionados. Essa operação é aquela que identifica as bordas e retorna uma imagem binária onde os pixels de valor 1 são regiões de traço. Assim, a combinação dessas duas imagens resultou na máscara desejada.

5.3.1 Criação da máscara do rosto

Como já mencionado na seção 5.1, utiliza-se o classificador Haar Cascade para identificar na imagem a região onde o rosto se encontra. Em seguida, como explicado na seção 5.3, gera-se a máscara do rosto. Devido a limitações da rede, as regiões a serem restauradas pela rede gerativa devem ser retangulares. No entanto, a máscara codificada até o momento gera regiões de formato livre. Assim, aplica-se sobre a máscara do rosto uma operação que analisa todas as suas regiões e as transforma em retângulos. A Figura 27 exemplifica o resultado obtido.



Figura 27: Rosto detectado pelo algoritmo de Viola-Jones (esquerda), máscara gerada (centro) e máscara retangularizada (direita). Adaptado de: *Old photo restoration and repair* (CORBIN, 2012)

Como é possível observar na Figura 27, a máscara retangularizada não tem todas as regiões presentes na máscara original. Isso acontece pois regiões com uma área grande poderiam gerar retângulos igualmente grandes, que resultariam em uma máscara imprecisa. Por conta disso, as regiões a serem retangularizadas foram limitadas a regiões que representam no máximo 0,5% da área da máscara. Portanto se a máscara tem um tamanho de 256 x 256, como é o caso da imagem de exemplo, os retângulos terão no máximo uma área de 327 pixels.

Conforme foram feitas as implementações e testes, foi identificado que os olhos quase sempre são marcados na máscara. Por serem pequenas regiões de cor clara e com diversas linhas definindo-os, a máscara acaba englobando eles, mesmo quando eles estão intactos. Assim, foi decidido removê-los da máscara utilizando a técnica de detecção de olhos apresentada na seção 5.1. Caso o classificador não identifique os olhos, então existe algum dano sobre ele e a melhor solução é realizar o *inpainting* normalmente.

Para lidar com as regiões que possuem uma área maior que a definida na condição, foi decidido realizar o *inpainting* através do OpenCV de maneira separada. Portanto a máscara do rosto utilizada no *inpainting* com a rede é gerada através da retangularização das regiões geradas na máscara e a máscara usada no *inpainting* do OpenCV corresponde as regiões da máscara original que não foram retangularizadas.

5.3.2 Criação da máscara do fundo

A criação da máscara do fundo da imagem é feita pelo mesmo processo que a do rosto, mas sem a etapa de retangularização. Para agilizar o processo de restauração do fundo, remove-se a região do rosto da máscara gerada, uma vez que essa região é restaurada a parte, com outros métodos. Na figura 28 são apresentadas a imagem original e a sua máscara correspondente, já eliminada a região do rosto.



Figura 28: Imagem original (esquerda) e máscara do fundo (direita). Adaptado de: *Old photo restoration and repair* (CORBIN, 2012)

5.3.3 Criação da máscara final

O processo de criação da máscara final utiliza o método descrito na seção 2.1.3.3, que analisa a imagem em busca de regiões da imagem que sejam o máximo local. O algoritmo utilizado então retorna essas regiões como um mapa em escala de cinza, onde as regiões com maiores valores de máximo são diferenciadas das demais regiões.

Para tornar esse mapa binário, utiliza-se o método de limiarização mencionado na seção 2.1.3.1. Assim, o resultado obtido é uma máscara binária que marca regiões de valores máximos na imagem. A região dos olhos é removida da mesma forma que nas máscaras anteriores por não ser pertinente à restauração. A Figura 29 mostra a máscara gerada com esse método, antes e depois do processo de limiarização.



Figura 29: Imagem restaurada (esquerda), máscara final sem limiarização (centro) e máscara final limiarizada (direita). Adaptado de: *Old photo restoration and repair* (CORBIN, 2012)

5.4 Restauração do retrato

5.4.1 Restauração do rosto

A restauração do rosto é feita em duas etapas: a primeira consiste em utilizar uma rede gerativa treinada com um imagens de rostos para preencher as regiões retangulares da máscara. A rede apresentada em 2.3.5 realiza o preenchimento das regiões do rosto indicadas por uma máscara binária. A rede foi treinada para receber imagens de 256 x 256 pixels acompanhadas de uma máscara binária com regiões retangulares, de tamanho arbitrário. Os autores da arquitetura que utiliza-se neste trabalho disponibilizaram os parâmetros resultantes do treino da rede com uma base de rostos. As imagens utilizadas pertencem ao Large-scale CelebFaces Attributes (LIU et al., 2015), que contém mais de 200 mil fotos de rostos de celebridades, com diferentes poses e fundos. Esse banco de imagens é adequado ao nosso projeto e, portanto, foi feito o uso da rede treinada com ele.

Conforme mencionado no parágrafo anterior, a rede recebeu em seu treino imagens de 256 x 256 pixels. Assim, os melhores resultados são obtidos quando as imagens a serem restauradas possuem as mesmas dimensões daquelas que foram utilizadas no treino. Por esse motivo, tem-se que redimensionar o rosto a ser restaurado. Como esse redimensionamento é para baixo, perde-se algumas informações da imagem original, mas não perde-se de qualidade.

A segunda etapa consiste em utilizar o método de restauração da biblioteca OpenCV para realizar o *inpainting* do rosto com a máscara de traços não retangularizados. Essa biblioteca disponibiliza a implementação do algoritmo baseado nas heurísticas de reconstrução de ima-

gens, apresentado na seção 2.2.2.

Os resultados dessas duas restaurações são combinados e assim é produzido a restauração final do rosto. Para unir as imagens restauradas, analisa-se quais regiões da máscara de traços finos não foi contemplada pela máscara retangularizada. Para cada uma dessas regiões, transfere-se a restauração feita pelo OpenCV com a máscara de traços finos para a imagem restaurada pela rede generativa (feita com a máscara retangularizada). A restauração final do rosto é apresentada na Figura 30.



Figura 30: Rosto original (esquerda) e sua restauração final (direita). Adaptado de: *Old photo restoration and repair* (CORBIN, 2012)

5.4.2 Restauração do fundo

A restauração do fundo da imagem é feita exclusivamente com base na máscara de traços finos utilizando o método de *inpainting* disponibilizado pelo OpenCV. A Figura 31 compara a imagem original com a restauração obtida.

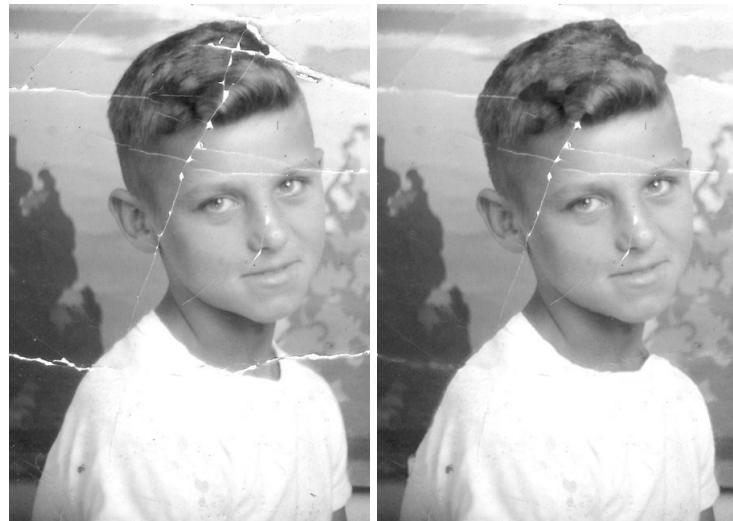


Figura 31: Imagem original (esquerda) e restauração do fundo com algoritmo baseado em heurísticas de *inpainting* disponível na biblioteca OpenCV (direita). Adaptado de: *Old photo restoration and repair* (CORBIN, 2012)

5.4.3 Restauração da imagem final

Finalmente, para obter a restauração final, combina-se as restaurações do rosto com a do fundo da imagem. Para isso, utiliza-se como base a imagem restaurada do fundo e sobre ela insere-se a restauração do rosto. Como o rosto restaurado tem dimensões 256 x 256 pixels, é necessário redimensioná-lo para seu tamanho original. Para evitar com que haja muita perda de informação devido ao redimensionamento, insere-se na restauração final apenas os trechos do rosto que sofreram restauração. Para isso, é feito o redimensionamento da máscara para o tamanho original do rosto. Com essa máscara em mãos, consegue-se saber quais as regiões que sofreram modificações e insere-se apenas elas na restauração final, preservando as demais áreas do rosto original. O resultado é apresentado na Figura 32.

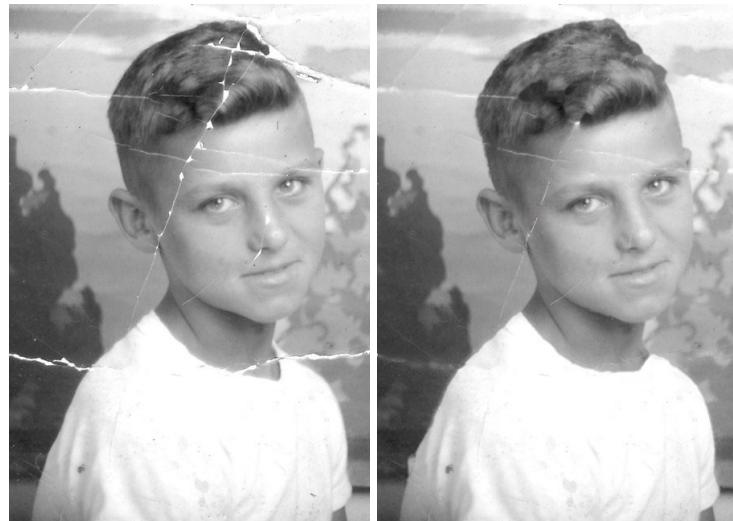


Figura 32: Imagem original (esquerda) e imagem obtida pela união do rosto e fundo restaurados (direita). Adaptado de: *Old photo restoration and repair* (CORBIN, 2012)

Apesar da melhora na imagem, ainda é possível identificar pequenas regiões que apresentam danos. Esses danos não foram identificados pelas máscaras ou não foram restaurados adequadamente na última etapa. Como uma etapa final no processo de restauração, utiliza-se um método de geração de máscara mais sensível a esse tipo de dano, já mencionada na seção 5.3.3. Esse método busca essas regiões mais finas, varrendo toda a imagem. Como foi feito anteriormente, a região dos olhos é removida dessa máscara pois ela identifica o brilho presente na pupila causado pelo reflexo do olho como um dano e o remove, causando um aspecto indesejado na imagem. O *inpainting* novamente é feito pelo método presente no OpenCV e o resultado final do processo de restauração, bem como a imagem original para uma última comparação, são apresentadas na Figura 33.

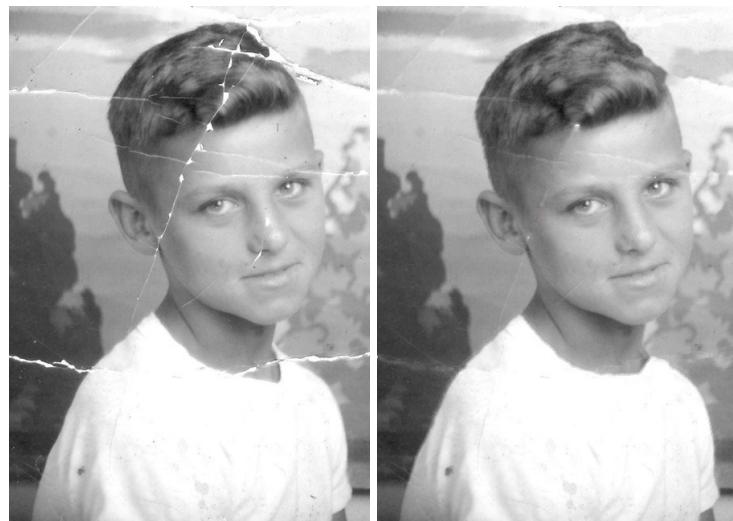


Figura 33: Imagem original (esquerda) e restauração final obtida após *inpainting* nas regiões mapeadas pelo detector de pico utilizando o algoritmo disponível no OpenCV (direita). Adaptado de: *Old photo restoration and repair* (CORBIN, 2012)

5.5 Aplicação web

Com o programa desenvolvido em mãos, foi possível transferir sua execução para a nuvem. A plataforma escolhida foi o Heroku, apresentado na seção 3.4, como servidor da nossa aplicação. O Heroku foi escolhido devido às configurações de seus containers, que servem bem os propósitos deste projeto.

Como o interesse é apenas em demonstrar a aplicação executando remotamente, foi feita a opção do container mais simples disponível. O dyno escolhido possui 512MB de RAM, sua configuração é do tipo *web* e, como todos os demais containers fornecidos pelo Heroku, executa Linux e possui fácil integração com o sistema de versionamento de código Github.

Um servidor em Flask que executa no dyno do Heroku foi desenvolvido. Esse servidor recebe a requisição do usuário por HTTP, junto com a imagem a ser restaurada, e executa a aplicação. Por fim, apresenta ao usuário tanto a imagem original quanto a restaurada.

6 TESTES E AVALIAÇÃO

Neste capítulo são discutidas as métricas de avaliação utilizadas na seção 6.1. Em seguida, são apresentados os testes realizados na seção 6.2 e, por fim, os resultados obtidos são apresentados na seção 6.3.

6.1 Métricas de avaliação

Métricas de qualidade medem diversos tipos de ruídos que podem prejudicar a legibilidade de uma imagem. Esses ruídos podem ser provenientes da captura da imagem, compressão, processamento, reprodução, entre outros.

De maneira geral, existem duas técnicas para se avaliar a qualidade de uma imagem: a subjetiva e a objetiva. A técnica subjetiva consiste em utilizar a percepção humana para julgar a qualidade de uma ou várias imagens. Já a objetiva utiliza técnicas computadorizadas para fazer a avaliação automaticamente através de algoritmos e métricas predefinidas.

Neste trabalho foi utilizada a técnica subjetiva, pois não existem métricas predefinidas disponíveis para avaliação da qualidade de uma imagem que passou pelo processo de *inpainting*.

6.2 Testes

A qualidade da restauração obtida é avaliada comparando o resultado proveniente da execução automática do processo desenvolvido neste projeto (Figura 35) com o resultado obtido a partir da geração manual de uma máscara (Figura 36).

Por fim, os resultados obtidos pelos algoritmos de *inpainting* (Figuras 35 e 36) são comparados com as restaurações feitas por artistas profissionais (Figura 37).

Todas as restaurações apresentadas foram feitas com base nas imagens danificadas apresentadas na Figura 34.

6.3 Resultados

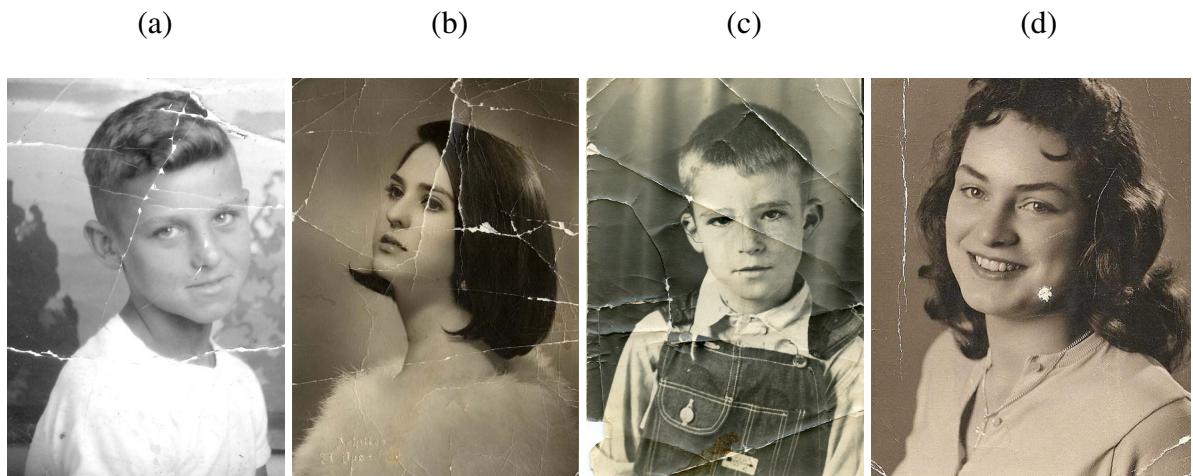


Figura 34: Imagens originais danificadas. Fonte: (a) *Old photo restoration and repair* (CORBIN, 2012) (b) *Photo Restoration Services* (CROSSWAYS, 2017) (c) *Copy & Restoration of Old Photos* (NEW SOMES STUDIO, 2018) (d) *Photo Restoration Service* (BIG PIXEL GRAPHICS, 2018)



Figura 35: Restaurações obtidas automaticamente usando o processo desenvolvido neste projeto. Adaptado de: (a) *Old photo restoration and repair* (CORBIN, 2012) (b) *Photo Restoration Services* (CROSSWAYS, 2017) (c) *Copy & Restoration of Old Photos* (NEW SOMES STUDIO, 2018) (d) *Photo Restoration Service* (BIG PIXEL GRAPHICS, 2018)

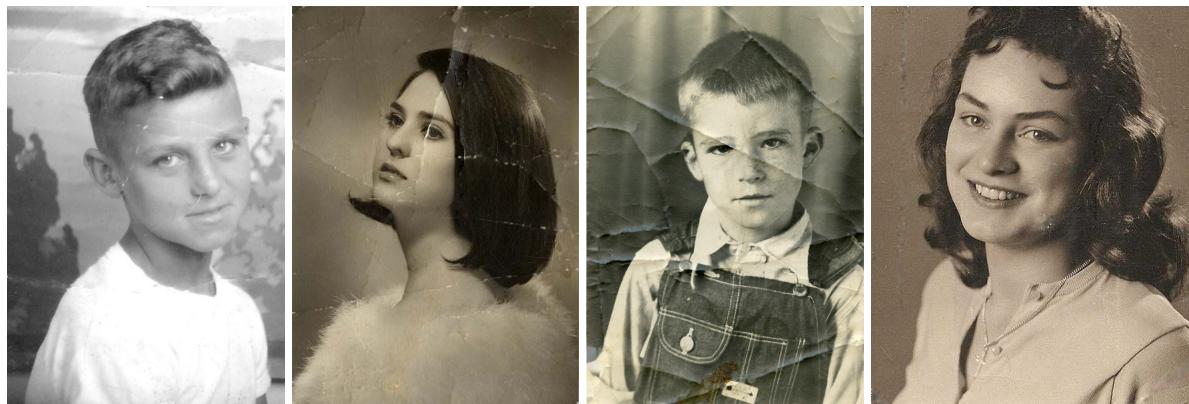


Figura 36: Restaurações obtidas pelos mesmos algoritmos de *inpainting* utilizados neste projeto, mas com máscaras criadas manualmente. Adaptado de: (a) *Old photo restoration and repair* (CORBIN, 2012) (b) *Photo Restoration Services* (CROSSWAYS, 2017) (c) *Copy & Restoration of Old Photos* (NEW SOMES STUDIO, 2018) (d) *Photo Restoration Service* (BIG PIXEL GRAPHICS, 2018)



Figura 37: Restaurações realizadas por artistas profissionais. Fonte: (a) *Old photo restoration and repair* (CORBIN, 2012) (b) *Photo Restoration Services* (CROSSWAYS, 2017) (c) *Copy & Restoration of Old Photos* (NEW SOMES STUDIO, 2018) (d) *Photo Restoration Service* (BIG PIXEL GRAPHICS, 2018)

Comparando os retratos da Figura 35 com aqueles da Figura 36, nota-se que, no geral, ambos os métodos produziram *inpaintings* muito próximos, principalmente na Figura (a). Ainda sim, a restauração feita pelo processo automático desenvolvido neste projeto conseguiu obter um resultado melhor que aquela gerada pela máscara manual no caso da Figura (b), pois conseguiu eliminar melhor os vincos presentes no cabelo da moça. No caso da Figura (c), nota-se que o *inpainting* produzido automaticamente detectou e eliminou alguns detalhes da roupa do menino, como o botão e a etiqueta em seu macacão. Isso ocorreu devido ao fato de que esses elementos são discrepantes em relação aos elementos ao seu redor. Assim, o algoritmo os

identificou, erroneamente, como danos e passou-os pelo processo de preenchimento. Apesar disso, é interessante notar que a cor do botão foi alterada, mas sua textura está compatível com a de um botão arredondado sobre o qual incide a luz que ilumina o menino. Por fim, no caso da Figura (d), nosso algoritmo automatizado não conseguiu gerar um preenchimento sutil no rasgo presente na parede ao fundo da imagem. Possivelmente as fronteiras da máscara ficaram muito próximas à região dos cabelos da moça. Assim, quando o algoritmo de *inpainting* buscou informações para preencher a região da máscara, ele encontrou as cores escuras do cabelo e as propagou para preencher o rasgo. Note que as porções do rasgo que estão mais distantes do cabelo foram preenchidas com tons mais semelhantes aos da parede. Além disso, nosso algoritmo falhou em identificar o olho esquerdo da moça. Como o olho é um elemento do rosto, ele foi restaurado pela rede generativa. É interessante notar que o olho gerado, apesar de pouco nítido, possui características semelhantes a um olho humano: existem cílios superiores, um esboço de pálpebra, uma pupila circunscrita em uma íris e o globo ocular. A rede foi capaz de identificar que naquela região do rosto mapeada pela máscara deveria existir uma estrutura (que é um olho) e, com isso, preencheu essa região com um olho gerado a partir das informações que aprendeu no treino.

Finalmente, comparando os resultados dos *inpaintings* das Figuras 35 e 36 com os resultados obtidos após as restaurações profissionais apresentados nas Figuras 37, é possível observar as limitações dos algoritmos de *inpainting*. Por exemplo, na Figura (b), os vincos da fotografia geraram elevações que alteraram a intensidade de cor nos arredores no rasgo. Assim, quando o algoritmo de *inpainting* preenche o vinco, ele propaga essas cores alteradas. Consequentemente, o amassado que antes era branco e escuro em seus arredores, continua escuro. Seria necessário suavizar as cores ao redor dessa região crítica, como fez o artista, para que fosse possível obter, efetivamente, textura e cores uniformes. No mais, o artista é capaz de incluir detalhes com mais propriedade, uma vez que é dotado de uma visão global da imagem e entende o contexto em que a região que está sendo restaurada se encontra.

7 CONCLUSÕES

Este capítulo discute as conclusões do projeto e apresenta as principais contribuições deste trabalho nas seções 7.1 e 7.2, respectivamente. Por fim, possibilidades de melhorias e trabalhos futuros são listadas na seção 7.3.

7.1 Conclusões do projeto

O projeto desenvolvido conseguiu alcançar seu objetivo de realizar a restauração de retratos danificados por vincos brancos por meio do processo de *inpainting*. Apesar de algumas limitações, discutidas na análise dos resultados, o projeto desenvolvido consegue gerar automaticamente máscaras que capturam as regiões danificadas da imagem. Essas máscaras alimentam os dois algoritmos de *inpainting* empregados neste projeto. O primeiro, baseado em heurísticas de reconstrução de imagens, é empregado na restauração do fundo da imagem, que não contém estruturas peculiares. O segundo, uma rede generativa treinada para gerar preenchimentos com base nos conhecimentos aprendidos sobre rostos humanos, é utilizada para gerar os preenchimentos no caso de danos nas faces. Por fim, ambas geram os melhores preenchimentos possíveis dadas as informações disponíveis e então seus resultados são combinados para gerar o retrato final restaurado.

7.2 Contribuições

A principal contribuição deste projeto é a automatização do processo de restauração de imagens, viabilizado por meio do desenvolvimento de um processo automático de mapeamento de regiões danificadas na fotografia digital. Esse processo é baseado em características morfológicas de imagens, e identifica potenciais regiões que contém danos, mapeando-as em uma máscara binária. Essa máscara binária é então consumida pelos algoritmos de *inpainting*, que produzem um preenchimento plausível com as informações disponíveis, gerando a imagem final restaurada.

7.3 Perspectivas de continuidade

Os trabalhos futuros podem ser desenvolvidos em três frentes principais:

1. Melhoria da criação da máscara binária

- A criação da máscara binária pode utilizar métodos de inteligência artificial para identificar com mais propriedade as regiões que são danos. Dessa forma, evita-se com que haja falsos positivos, ou seja, identificação de estruturas que não são danos.

2. Melhoria do algoritmo de *inpainting*

- O algoritmo de *inpainting* poderia ser aprimorado para que não houvesse a necessidade de retangularização das máscaras, conforme proposto em trabalhos recentes (YU et al., 2018a).

3. Melhoria da aplicação *web*

- A aplicação *web* pode ser incrementada com um módulo de login, de forma com que o usuário tenha uma conta que o permita visualizar as restaurações solicitadas, que podem ser armazenadas na nuvem.
- Uma outra possibilidade seria a de incluir na aplicação módulos de restauração para outros danos na fotografia, tais como manchas. Seria ainda possível adicionar a funcionalidade de colorização, por exemplo.

REFERÊNCIAS

ARJOVSKY, M.; CHINTALA, S.; BOTTOU, L. Wasserstein GAN. *CoRR*, abs/1701.07875, 2017. Disponível em: <<http://arxiv.org/abs/1701.07875>>.

Autor desconhecido. *A Beginner's Guide to Neural Networks and Deep Learning*. 2018. Disponível em: <<https://skymind.ai/wiki/neural-network>>.

BERTALMIO, M.; BERTOZZI, A. L.; SAPIRO, G. Navier-stokes, fluid dynamics, and image and video inpainting. In: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*. [S.l.: s.n.], 2001. v. 1, p. 1–355. ISBN 1063-6919.

BERTALMIO, M. et al. Image Inpainting. In: *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000. (SIGGRAPH '00), p. 417–424. ISBN 1-58113-208-5. Disponível em: <<http://dx.doi.org/10.1145/344779.344972>>.

CORBIN, H. *Cinematic Visual Media Solutions*. 2012. Disponível em: <<http://www.cinematic.com/graphic-design-services/photoediting/photo-restoration-and-repair-video/>>.

GONZALEZ, R. C.; WOODS, R. E. *Digital Image Processing*. 2nd. ed. [S.l.]: Prentice Hall, 2002. (International Edition). ISBN 9780201180756.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016.

HUI, J. *GAN—Wasserstein GAN and WGAN-GP*. 2018. Disponível em: <https://medium.com/@jonathan_hui/gan-wasserstein-gan-wgan-gp-6a1a2aa1b490>.

KARPATHY, A. *Convolutional Neural Networks (CNNs / ConvNets)*. 2015. Disponível em: <<http://cs231n.github.io/convolutional-networks/>>.

LIU, Y.; PAN, J.; SU, Z. Deep Blind Image Inpainting. *CoRR*, abs/1712.09078, 2017.

LIU, Z. et al. Deep Learning Face Attributes in the Wild. In: *Proceedings of International Conference on Computer Vision (ICCV)*. [S.l.: s.n.], 2015.

RADFORD, A.; METZ, L.; CHINTALA, S. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *CoRR*, abs/1511.06434, 2015. Disponível em: <<http://arxiv.org/abs/1511.06434>>.

VIOLA, P.; JONES, M. Rapid object detection using a boosted cascade of simple features. In: . [S.l.: s.n.], 2001. p. 511–518.

YU, J. et al. Free-Form Image Inpainting with Gated Convolution. *arXiv preprint arXiv:1806.03589*, 2018.

YU, J. et al. Generative Image Inpainting with Contextual Attention. *arXiv preprint arXiv:1801.07892*, 2018.

YURIN, D.; LEVASHOV, A. *Ridge and tree feature detection on images*. 2013.

ZHANG, W. et al. A Precise-Mask-Based Method for Enhanced Image Inpainting. *Mathematical Problems in Engineering*, v. 2016, p. 1–5, 12 2016.