UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE COMPUTAÇÃO
CURSO DE CIÊNCIA DA COMPUTAÇÃO

FELIPE DIVENSI ECKL

COMPRESSÃO DE IMAGENS COM PERDA UTILIZANDO REDES NEURAIS ARTIFICIAIS

TRABALHO DE CONCLUSÃO DE CURSO

MEDIANEIRA

2019

FELIPE DIVENSI ECKL

COMPRESSÃO DE IMAGENS COM PERDA UTILIZANDO REDES NEURAIS ARTIFICIAIS

Trabalho de Conclusão de Curso apresentado ao Departamento Acadêmico de Computação da Universidade Tecnológica Federal do Paraná como requisito parcial para obtenção do título de "Bacharel em Computação".

Orientador: Prof. Dr. Pedro Luiz de Paula Filho

Co-orientador: Prof. Dr. Arnaldo Candido Junior

MEDIANEIRA

2019

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Diretoria de Graduação e Educação Profissional
Coordenação do Curso de Ciência da Computação

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

TERMO DE APROVAÇÃO

COMPRESSÃO DE IMAGENS COM PERDA UTILIZANDO REDES NEURAIS ARTIFICIAIS

Por

FELIPE DIVENSI ECKL

Este Trabalho de Conclusão de Curso foi apresentado às 09:00h do dia 24 de Junho de 2019 como requisito parcial para a obtenção do título de Bacharel no Curso de Ciência da Computação, da Universidade Tecnológica Federal do Paraná, Câmpus Medianeira. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Prof. Dr. Pedro Luiz de Paula Filho
UTFPR - Câmpus Medianeira

Prof. Dr. Paulo Lopes de Meneses
UTFPR - Câmpus Medianeira

Prof. Msc. Jorge Aikes Junior
UTFPR - Câmpus Medianeira

A folha de aprovação assinada encontra-se na Coordenação do Curso.

RESUMO

DIVENSI ECKL, Felipe. COMPRESSÃO DE IMAGENS COM PERDA UTILIZANDO REDES NEURAIS ARTIFICIAIS. 47 f. Trabalho de Conclusão de Curso – Curso de Ciência da Computação, Universidade Tecnológica Federal do Paraná. Medianeira, 2019.

Com o rápido crescimento de mídia que atualmente é transmitida pela Internet, algoritmos de compressão de imagens com perda são indispensáveis. Por isso existem diversos estudos que buscam melhorar os algoritmos de compressão, com o objetivo de obter maior qualidade nas imagens e maiores taxas de compressão. O objetivo deste trabalho é apresentar uma abordagem de compressão de imagens utilizando Redes Neurais Artificiais baseada na arquitetura de rede autoencoder. Os modelos criados, ao serem comparados ao algoritmo JPEG e aplicados em um conjunto de imagens binárias, alcançaram taxas de compressão até 7 vezes maior com índice de similaridade estrutural (SSIM) 19% maior. Ao serem aplicados a um conjunto de imagens em escala de cinza, alcançaram taxa de compressão até 21 vezes maior, e com SSIM 6% abaixo do JPEG. E ao serem aplicados em um conjunto de imagens coloridas alcançaram taxa de compressão até 2 vezes maior, com um SSIM 6% acima do JPEG.

Palavras-chave: Redes neurais (Computação), Compressão de dados (Computação), Processamento de imagens

ABSTRACT

DIVENSI ECKL, Felipe. LOSSY IMAGE COMPRESSION USING ARTIFICIAL NEURAL NETWORKS. 47 f. Trabalho de Conclusão de Curso – Curso de Ciência da Computação, Universidade Tecnológica Federal do Paraná. Medianeira, 2019.

With the rapid growth of media currently being transmitted over the Internet, lossy image compression algorithms are indispensable. Therefore, there are several studies that seek to improve the compression algorithms, in order to obtain higher image quality and higher compression rates. The objective of this work is to present an image compression algorithm using Artificial Neural Networks based on the autoencoder network architecture. The models created, when compared to the JPEG algorithm and applied in a set of binary images, reached compression rates up to 7 times higher with a structural similarity index (SSIM) 19 % higher. When applied to a set of grayscale images, achieved compression rates up to 21 times higher, and SSIM 6 % below JPEG. And when applied to a set of colored images achieved a compression rate up to 2 times higher, with an SSIM 6 % above JPEG.

Keywords: Neural Networks, Data Compression, Image Processing

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

AM	Aprendizado de Máquina
A/D	Conversão Analógica para Digital
BMP	Bitmap
CAE	Compressive Autoencoder
ConvAE	Convolutional Autoencoder
DAE	Denoising Autoencoder
DeepAE	Deep Autoencoder
D/A	Conversão Digital para Analógica
GB	Giga Bytes
GIF	Graphics Interchange Format
GPU	Graphics Processing Unit (Placa de vídeo)
PSNR	Peak Signal-to-Noise Ratio
IA	Inteligência Artificial
JPEG	Joint Photographic Experts Group
KNN	K-Nearest Neighbours
MBR	Máquina de Boltsmann Restritas
MLP	Multi-Layer Perceptron
PBM	Portable BitMap
PGM	Portable GreyMap
PNG	Portable Network Graphics
PPM	Portable PixMap
RNA	Redes Neurais Artificiais
ReLU	Rectified Linear Unit
SSIM	Structural Similarity Index
TIFF	Tagged Image File Format

SUMÁRIO

1 INTRODUÇÃO

Com o aumento na resolução de displays e poder de processamento dos dispositivos móveis como smartphones, houve um grande crescimento no consumo de mídia através dos mesmos. O tráfego de dados móveis na Internet cresceu de $4,4$ exabytes (1 exabyte é igual a 1 milhão de terabytes) por mês no final de 2015 para $7,2$ exabytes no final de 2016, e há uma estimativa de crescimento para 49 exabytes por mês em 2021. No qual mídias como fotos e vídeos compuseram mais da metade do tráfego (CISCO, 2017).

Com o rápido crescimento da demanda de banda de dados e a inconsistência de disponibilidade das redes móveis, uma rápida transmissão de mídias de maneira confiável se prova um grande desafio na área de computação, e é extremamente necessária para uma boa experiência de usuário.

Na segunda parte da década de 2010, uma boa parte de smartphones possuem câmeras que fotografam em resolução maior do que 12 MP (12 milhões de píxeis) e gravam vídeos com uma resolução maior do que 1080p (nome popular da resolução 1920×1080). Por exemplo, um vídeo de apenas 1 minuto de duração com resolução de 1920×1080 profundidade de cor de 24 bits e framerate de 24 fps teria $8.957.952.000$ bytes, ou aproximadamente $8,96$ GB (Giga Bytes), e um filme de 120 minutos ocuparia mais de 800 GB.

Logo se vê que uma maneira de reduzir este tamanho é necessária. Com este objetivo algoritmos de compressão foram criados. Algoritmos de compressão criam uma representação dos dados que pode ou não ser retornada ao estado original. As que retornam são chamadas de compressão lossless, muito utilizadas em casos em que a perda de dados não é aceitável, como compressão de textos, aplicações, entre outros. Estes reduzem o tamanho do arquivo de maneira inversamente proporcional a entropia dos dados do mesmo, ou seja, imagens com muitas cores diferentes, e mudanças bruscas de iluminação não seriam tão facilmente comprimidas quanto uma imagem com menos variação. Algoritmos de compressão lossy, que geram perda, geralmente são somente aplicados em imagens, áudio e vídeos, já que estes são dados que podem perder detalhes e ainda serem reconhecidos por pessoas sem maiores problemas. Os algoritmos que aplicam

este tipo de compressão geralmente buscam conteúdos com detalhes semelhantes e os juntam como sendo um só, sem ter conhecimento do contexto ou conteúdo do mesmo.

Alternativamente pode ser aplicado um algoritmo de compressão que consiga reconhecer detalhes como animais, objetos ou pessoas, e crie uma representação sobre eles para reconstruí-los posteriormente. Para tal propósito, um agente inteligente que consiga aprender o conteúdo do dado é necessário. Para tanto podem ser usadas redes neurais artificiais (RNA).

As RNAs são sistemas que buscam simular o aprendizado feito por redes neurais biológicas. Elas melhoram seu desempenho recebendo um grande número de exemplos de dados de entrada associados a saídas esperadas, criando um modelo que possa estimar uma saída em relação a dados reais. RNAs já foram propostas em diversos estudos com o objetivo de aumentar a qualidade de imagens e vídeo artificialmente, sendo aplicadas por exemplo em técnicas como super-resolução de imagem (LI; WANG, 2017), que busca aumentar a resolução e definição de imagens e interpolação de vídeo (NIKLAUS et al., 2017), que estimam o movimento entre frames de entrada para sintetizar um frame intermediário a partir deles, criando um vídeo que represente movimento de maneira mais suave.

1.1 OBJETIVOS GERAL E ESPECÍFICOS

O objetivo geral deste trabalho é utilizar RNAs para fazer a super-resolução de imagens com o objetivo de recuperar a qualidade original das mesmas, após uma grande compressão para diminuir a quantidade de dados que precisam ser transmitidos em rede. Este objetivo pode ser dividido entre os seguintes objetivos específicos:

- Obter imagens para fazer o treinamento de uma RNA;
- Fazer o pré-processamento das imagens, buscando adequá-las para ao treinamento;
- Selecionar os algoritmos e parâmetros mais adequados para serem aplicados;
- Testar o desempenho dos algoritmos para recuperar as imagens;
- Analisar o resultado do desempenho dos algoritmos.

1.2 JUSTIFICATIVA

As capacidades de armazenamento e transmissão de dados se expandem com inovações tecnológicas constantes, de acordo com a Lei de Parkinson aplicada à computação: "Os dados se expandem para preencher o espaço disponível para armazenamento" (ARGE et al., 1993). Ou seja, usuários e novas aplicações sempre exigem ao máximo os recursos disponíveis. Portanto, algoritmos de compressão de imagens e vídeo mais eficientes trariam benefícios para vários grupos de pessoas, por exemplo, provedores de serviços de Internet, reduzindo a carga de dados transmitidos e usuários e criadores de conteúdo, pois incentivaria o consumo de mídias por um menor custo.

2 REFERENCIAL TEÓRICO

Neste capítulo serão mostrados conceitos básicos sobre compressão, com ênfase nos pontos mais relevantes à aplicação na área de compressão de imagem. Também serão apresentadas técnicas de Inteligência Artificial, Aprendizado de Máquina e Redes Neurais Artificiais que serão utilizadas neste trabalho, assim como outros trabalhos relevantes na área.

2.1 COMPRESSÃO DE DADOS

Compressão de dados é um processo de codificação que busca reduzir o número de bits necessários para representar a uma dada informação, com o objetivo de facilitar seu transporte ou armazenamento (ARGE et al., 1993).

Uma das primeiras aplicações de compressão de dados conhecidas é o código Morse, desenvolvido por Samuel Morse no século XIX. Nele, textos são codificados como pontos, traços e pausas. Letras mais comuns do alfabeto inglês, como “e” e “a” são representadas por códigos mais curtos do que letras menos comuns como “q” e “j” (ARGE et al., 1993).

Um processo de compressão é composto por dois algoritmos: o primeiro, chamado de encoder, recebe como entrada uma sequência de dados e gera uma representação reduzida a partir dela. O segundo, conhecido como decoder, recebe a representação e gera uma reconstituição do arquivo original (WADE, 1994; MAHONEY, 2010).

Os métodos de compressão podem ser divididos entre duas classes principais: lossless e lossy. Na compressão lossless, o decoder gera uma reconstrução da imagem exatamente igual á entrada recebida pelo encoder, ou seja, o processo de compressão é reversível, pois não haverá perdas. A compressão lossy recebe este nome pois no processo de compressão, o encoder descarta partes da informação em troca de criar uma

representação muito menor do que uma gerada por um algoritmo lossless (GUPTA et al., 2016).

Algoritmos de compressão podem ter sua eficiência medida com a taxa de compressão, que pode ser expressa pela Equação 1, na qual t_i é tamanho do dado antes do processo e t_f é o tamanho final do mesmo após ser reduzido pelo encoder.

$$t_c = \frac{t_f}{t_i} \tag{1}$$

2.1.1 Algoritmos de compressão lossless

Algoritmos de compressão lossless são aqueles que podem reduzir o tamanho de um dado e retorná-lo a um estado exatamente igual ao original. Estes algoritmos são utilizados em aplicações nas quais a perda de dados não é tolerável, como compressão de textos, aplicações entre outros (SHANNON, 1948).

Estes algoritmos se aproveitam da redundância estatística que ocorre naturalmente em dados para fazer a compressão. Existem diferentes tipos de redundância que podem ocorrer em um dado, por exemplo:

- Redundância alfabética: A repetição de uma dada sequência em uma string, como a grande quantidade de vezes que a letra “A” aparece em palavras na língua portuguesa;
- Redundância contextual: Um dado que ao aparecer em uma string, acompanham algum outro, por exemplo a letra “Q” que quase sempre é acompanhada pela letra “U” na língua portuguesa;
- Redundância de imagem: Em figuras, pixeis adjacentes aos outros têm uma grande chance de terem cores iguais ou semelhantes.

Um exemplo de algoritmo de compressão sem perda é a codificação de Huffman. Neste algoritmo uma sequência de dados é mapeada para uma string com códigos de tamanho variável. Os códigos são atribuídos de acordo com a frequência em que eles aparecem na sequência original. Os dados que ocorrem mais, recebem os códigos mais curtos, e os mais raros, recebem os mais longos. Este tipo de compressão faz a utilização de um código de prefixo, ou seja, um caractere que não ocorre no texto original. Este caractere é utilizado para indicar o começo de uma sequência que foi compressa.

Isto garante que o código possa ser retornado para seu estado original sem que exista ambiguidade (SHANNON, 1948).

Para fazer a compressão de um texto utilizando a codificação de Huffman, é necessário primeiramente criar uma árvore de Huffman, esta é uma árvore binária do tipo min-heap organizados pela sua frequência, ou seja, as sequências de caracteres mais frequentes ficam em nós folha mais próximos à raiz, e os códigos menos frequentes ficam mais distantes. Além da frequência, os nós folha armazenam o valor original da sequência. Ao finalizar a criação da árvore, para se obter as representações reduzidas do texto original é necessário navegar pela árvore, atribuindo valores para cada transição de nó. Ao final do processo se obtém a árvore e um vetor de sequências reduzidas. Para retornar o texto original, as sequências do vetor são lidas e utilizadas para navegar pela árvore, buscando os valores dos nós folha (SHANNON, 1948).

Nem todo dado pode ser comprimido sem perda isso pode ser exemplificado com o princípio da casa dos pombos: se existem l pombos que devem entrar em m casas, e se o número de pombos for maior do que o número de casas, pelo menos uma casa conterá mais de um pombo. Fazendo uma analogia com compressão, supondo que exista um algoritmo que faça com que todos os dados binários possam ter seu tamanho reduzido por pelo menos 1 byte, e depois ser retornado para seu tamanho original, e considerando que todas as sequências de bytes são um dado diferente, ao reduzir seu tamanho de 2^n bytes para 2^{n-1} bytes é impossível retornar todas as sequências de informação para seu tamanho original sem que existam colisões entre as saídas obtidas (MAHONEY, 2010).

O tamanho final de uma compressão sem perdas está diretamente relacionado com o quão variado os dados são. Esta variação pode ser obtida calculando a entropia da origem da informação a ser tratada de acordo com a Equação 2. A entropia (H) é dada pela soma das probabilidades (p) de ocorrência de cada símbolo no alfabeto (X) (SHANNON, 1948; MAHONEY, 2010). De acordo com Shannon (1948), a máxima compressão possível ao codificar um conjunto de dados é dada pela razão da entropia destes dados sobre o valor máximo.

$$\begin{aligned} H &= \sum_{i=1}^{n} P_i \cdot H_i \\ P_i &= -\,p(X_i) \\ H_i &= log_2 p(X_i) \end{aligned} \tag{2}$$

2.1.2 Algoritmos de compressão lossy

Algoritmos lossy fazem a utilização de técnicas que reduzem significantemente o tamanho para se representar dados, mas impossibilitam a recuperação dos dados em seu estado original. Estes algoritmos são utilizados em problemas nos quais esta diferença entre a entrada e saída é tolerável, como áudio, vídeo e imagens (SHANNON, 1948).

Grande parte dos algoritmos de compressão com perda, se baseiam em representar um grande conjunto de entrada com um pequeno conjunto de códigos. Esta redução de um conjunto de entrada para um conjunto menor é chamada de quantização. Dois exemplos de quantização que podem ser aplicados em imagem são a quantização escalar e a quantização vetorial (ARGE et al., 1993).

2.1.2.1 Quantização Escalar

Uma abordagem para a quantização em um conjunto de entradas é a quantização escalar, que utiliza uma escala de precisão para aproximar os valores. Na quantização escalar, o encoder faz um processo de Conversão Analógica para Digital (A/D), na qual valores reais são recebidos na entrada e o valor discreto mais próximo é retornado em formato binário. O decoder faz o processo contrário, recebendo o valor binário e o transformando em um valor real novamente, processo que é chamado de Conversão Digital para Analógica (D/A). Um exemplo deste processo pode ser visto na Tabela 1 e na Figura 1. Neste exemplo as saídas da função $f(t) = 4cos(2\pi t)$ foram quantizadas em quatro bits com precisão de $0,5$ (ARGE et al., 1993; SAYOOD, 2000).

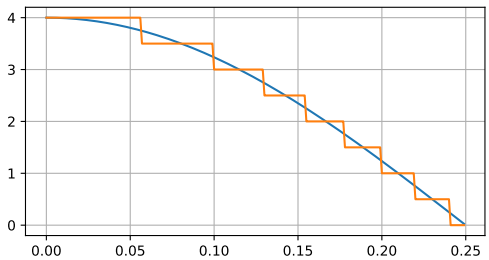


Figura 1 – Quantização de dados escalar

Fonte: (Autoria Própria)

Tabela 1 – Tabela de quantização de dados em 4 bits.

t	$4cos(2\pi t)$	Entrada A/D	Saída D/A	Erro
0	4	1000	4	0
0.05	≈ 3.804	1000	4	≈ 0.196
0.1	≈ 3.236	0110	3	≈ 0.236
0.15	≈ 2.351	0101	2.5	≈ 0.149
0.2	≈ 1.236	0010	1	≈ 0.236

Fonte: (ARGE et al., 1993)

2.1.2.2 Quantização Vetorial

Um dos principais conceitos da quantização vetorial é o codebook. O codebook é uma tabela chave-valor no qual os valores são vetores de tamanho fixo com padrões comuns ao conjunto de entrada. Na compressão, o encoder, ao receber uma entrada, as separa em vetores e as compara com as entradas do codebook. O índice do vetor do codebook que mais se assemelhar ao vetor original é adicionado a um conjunto de índices. O resultado da compressão é o codebook e a lista de índices. O decoder então só precisa buscar os índices no codebook para obter a saída (ARGE et al., 1993; SAYOOD, 2000).

A parte mais complexa do processo de quantização vetorial é o algoritmo de criação do codebook. Existem diversas implementações para criar codebooks, geralmente sendo implementados utilizando algoritmos de agrupamento, por exemplo, o algoritmo de aprendizado de máquina K-Nearest Neighbours (KNN) (ARGE et al., 1993).

2.1.3 Compressão de Imagem

O olho humano possui dois tipos de receptores de luz: cones e bastonetes. Bastonetes são muito sensíveis a mudanças de luminosidade, e são responsáveis pela visão em baixa luminosidade. Cones são mais sensíveis a diferenças na frequência da luz. Existem 3 tipos de cones, cada um com um diferente pico de sensibilidade no espectro de luz. Um cone é mais sensível à cores próximas do vermelho, a cores semelhantes a azul e outro as cores que se aproximam de verde. O olho possui muito mais bastonetes do que cones, o que faz com que haja muito mais sensibilidade para mudanças na luminância (diferença entre brilho em pontos de uma imagem) do que às mudanças de crominância (diferença entre as cores) (SAYOOD, 2000).

Muitos formatos que utilizam compressão de imagem com perda se aproveitam desta imprecisão do olho humano para reduzir a banda de cores de imagens sem muita diferença perceptível (ARGE et al., 1993; AGUILERA, 2007).

Alguns formatos de imagem muito utilizados são:

- Bitmap (BMP): formato de imagem lossless e sem compressão. Pode conter 1, 4, 8 ou 24 bits de profundidade de cores e suporta 1 ou 3 canais de cores (preto e branco, escala de cinza e vermelho, verde e azul). Desenvolvido pela Microsoft (AGUILERA, 2007; MURRAY; VANRYPER, 1996);
- Graphics Interchange Format (GIF): Formato de imagem lossless que suporta compressão. Desenvolvido pela Compuserve Information Service. Utiliza o algoritmo de compressão LZW (HU; CHANG, 2000). Apesar de ser um formato "sem perdas", tem a limitação de somente suportar 8 bits de cores (256 cores). É um formato muito utilizado para gráficos na Internet (SAYOOD, 2000);
- Portable Network Graphics (PNG): Formato de imagem lossless que suporta compressão. Aceita imagens em escala de cinza ou 3 canais de cores e somente suporta 8 ou 24 bits de profundidade de cores. Utiliza o algoritmo de compressão Deflate (AGUILERA, 2007; SAYOOD, 2000);
- Tagged Image File Format (TIFF): Formato lossless que suporta os algoritmos de compressão PackBits, LZW, CCITT e Jpeg. Suporta imagens de 1 até 24 bits de cores. Criado pela Aldus em conjunto com a Microsoft para utilização com o formato PostScript. (AGUILERA, 2007; MURRAY; VANRYPER, 1996);
- Joint Photographic Experts Group (JPEG): Formato de imagem com compressão lossy. Suporta imagens em escala de cinza (8 bits) ou imagens com 3 canais de cores (24 bits). O algoritmo de compressão JPEG tem a vantagem de poder ter seu nível de compressão ajustado. Quanto mais alto o nível de compressão menos fiel será a imagem em relação à sua origem, mas seu tamanho diminui significantemente. Subdivide a imagem em blocos e faz a compressão para homogeneizá-los. Criado pelo comitê Joint Photographic Experts Group, é o formato mais utilizado para armazenar e distribuir imagens na Internet (ARGE et al., 1993);
- Joint Photographic Experts Group 2000 (JPEG 2000): Também criado pelo comitê Joint Photographic Experts Group, teve o objetivo de substituir o original JPEG. Possui melhores taxas de compressão e qualidade perceptível do que o JPEG original (ARGE et al., 1993);
- Portable BitMap (PBM): Formato de imagens lossless sem compressão, para imagens binárias (1 bit por pixel) (BOURKE, 1997);

- Portable GreyMap (PGM): Formato de imagens lossless sem compressão, para imagens em escala de cinza (8 bits por pixel) (BOURKE, 1997);
- Portable PixMap (PPM): Formato de imagens lossless sem compressão, para imagens coloridas (24 bits por pixel) (BOURKE, 1997).

2.2 INTELIGÊNCIA ARTIFICIAL E APRENDIZADO DE MÁQUINA

Com o rápido crescimento tanto do volume de dados que necessitam ser processados quanto a complexidade dos problemas a serem resolvidos, ferramentas computacionais que possam agir de maneira autônoma vêm se tornando cada vez mais necessárias. Por isso, a Inteligência Artificial (IA), uma área anteriormente vista como um campo puramente teórico, vem ganhando destaque na computação (LORENA et al., 2000).

Atualmente, a IA é aplicada em vários campos, por exemplo: veículos autônomos, smart speakers, serviços de diagnóstico médico, reconhecimento de fala, processamento de linguagem natural, entre outros(SAKAI et al., 2018).

Na IA, o maior desafio é o aprendizado de forma automatizada. Por isso a área de Aprendizado de Máquina (AM) recebe muita atenção. Em AM se busca criar modelos que aprendam de forma indutiva. Na área de AM, os algoritmos de aprendizado recebem um conjunto de instâncias, chamado dataset, relacionado ao problema e através deste, aprende como se aproximar de uma resultado satisfatório. Existem diversas arquiteturas diferentes para efetuar o aprendizado, que buscam inspiração para resolver o problema do aprendizado de diversas áreas como Probabilidade e Estatística, Geometria Analítica, Teoria da Informação, Teoria da Computação e Neurociência, entre outras (LORENA et al., 2000).

2.2.1 Redes Neurais Artificiais

Na era em que estava sendo procurada uma arquitetura genérica para que todos

os computadores pudessem seguir, John Von Neumann propôs um modelo computacional chamado Autômato Celular, no qual os dados eram processados simultaneamente de maneira paralela em células de processamento. Um grande desafio para este modelo era a maneira com que as células poderiam se comunicar e se coordenar (DIETTERICH, 2009).

Uma das possíveis soluções para o problema do processamento de dados de maneira paralela são as redes neurais, como o sistema nervoso de animais (FELDMAN; ROJAS, 2013). Sistemas nervosos de animais são estruturas complexas compostas por milhões de células. Estes sistemas possuem um alto nível de redundância. As células, chamadas neurônios, se comunicam entre si de maneira hierárquica e são capazes de se auto organizar e processar dados de várias maneiras diferentes (FELDMAN; ROJAS, 2013; DIETTERICH, 2009).

O funcionamento de um neurônio pode ser simplificado em quatro elementos básicos: sinapses, dendritos, soma e axônio. As sinapses são os impulsos elétricos que o neurônio recebe através dos seus dendritos. Este sinal é passado para a soma (corpo da célula) que o processa através de reações químicas e físicas. O sinal resultante deste processo é enviado para o axônio que é responsável por transmitir a sinapse ao próximo neurônio, como mostrado na Figura 2 (DIETTERICH, 2009; RUSSELL; NORVIG, 2010).

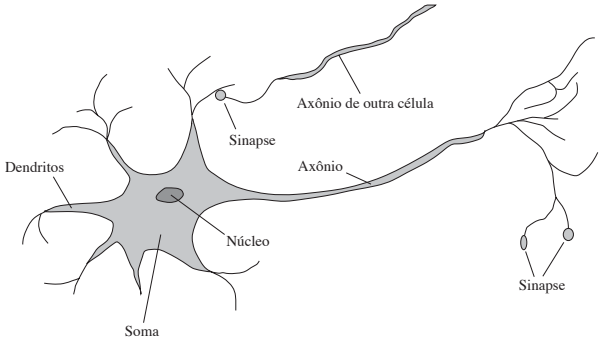


Figura 2 – Representação Gráfica de um neurônio Biológico
Fonte: (RUSSELL; NORVIG, 2010)

Rede neural artificial (RNA) é um modelo computacional criado com o objetivo de simular o funcionamento de redes neurais biológicas. Este é um método de fazer a aproximação do resultado de funções e consegue trabalhar com valores discretos, reais e vetoriais. Este modelo é composto por neurônios artificiais, como mostrado na Figura 3 (HAYKIN, 2008; FELDMAN; ROJAS, 2013).

O funcionamento de um neurônio k se dá por: m sinapses $[x_1, x_2, \ldots, x_m]$ são recebidas e em seguida multiplicadas pelos pesos sinápticos correspondentes

$[w_{k1}, w_{k2}, \ldots, w_{km}]$. O resultado dos produtos é somado em Σ e este valor é somado a um bias b_k. O resultado desta operação é chamado potencial de ativação (u_k) que é aplicado na função de ativação $\varphi(\cdot)$. O resultado se torna a saída no neurônio (y_k). Este neurônio também pode ser representado em notação matemática como a Equação 3 (HAYKIN, 2008; FELDMAN; ROJAS, 2013).

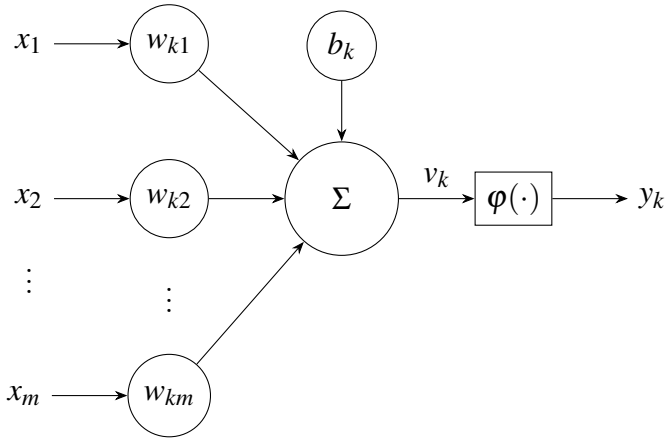


Figura 3 – Representação de um neurônio artificial.
Fonte: (HAYKIN, 2008)

$$\begin{aligned} u_k &= \sum_{j=1}^{m} w_{kj} x_j \\ y_k &= \varphi(u_k + b_k) \end{aligned} \tag{3}$$

RNAs podem possuir múltiplas camadas com vários neurônios, que ficam conectadas entre si. As camadas são divididas em 3 grupos:

- A camada de entrada, que é responsável por receber os dados na rede;
- Uma ou mais camadas ocultas, que são representadas por camadas que não possuem acesso externo;
- A camada de saída que tem como saída a predição da rede (FELDMAN; ROJAS, 2013).

2.2.1.1 Funções de Ativação

As funções de ativação ($\varphi(\cdot)$) têm como objetivo padronizar a amplitude de um sinal para um valor finito, geralmente entre 0 e 1, -1 a 1 ou -3 a 3 (HAYKIN, 2008). Alguns exemplos de função de ativação são:

- Função Limiar, representada pela Equação 4, é uma função não contínua que é

definida por um limiar v, ela tem baixo custo de processamento, e é usada quando se deseja obter uma saída binária (neste caso 0 ou 1) (HAYKIN, 2008);

$$\varphi_1(v) = \begin{cases} 0 & \text{se } v \geq 1 \\ 1 & \text{se } v < 1 \end{cases} \tag{4}$$

- Função Sigmoide Logística, representada pela Equação 5, varia de 0 a 1, na qual entradas negativas que se aproximam de $-\infty$ tendem a 0 e valores que se aproximam de $+\infty$ tendem a 1. A função também pode receber uma constante a que altera o declive da função, quanto mais alto o valor de a, mais brusca é a curva. Se o valor de a se aproximar de ∞ o funcionamento da função se assemelha a uma Função Limiar (4) (HAYKIN, 2008);

$$\varphi_2(v) = \frac{1}{1+e^{-av}} \tag{5}$$

- A Função da Tangente Hiperbólica, representada pela Equação 6, tem funcionamento similar à Função Sigmoide Logística, mas varia de -1 a 1 (HAYKIN, 2008);

$$\varphi_3(v) = tanh(v) \tag{6}$$

- Rectified Liner Unit (ReLU) (Equação 7): É uma função que preserva as entradas v positivas e descarta valores negativos, os substituindo por 0. Esta função tem as vantagens de ter um baixo custo de execução e não sofrer de problemas, caso os valores de entrada sejam muito pequenos ou grandes. Varia de 0 a $+\infty$ (XU et al., 2015);

$$\varphi_4(v) = \begin{cases} 0 & \text{se } v < 0 \\ v & \text{se } v \geq 0 \end{cases} \tag{7}$$

- Leaky ReLU (Equação 8): Semelhante a função ReLU, mas ao invés de descartar v se for negativa, preserva uma fração dela, dividida por uma constante arbitrária a. Varia de $-\infty$ a $+\infty$ (XU et al., 2015).

$$\varphi_5(v) = \begin{cases} \frac{v}{a} & \text{se } v < 0 \\ v & \text{se } v \geq 0 \end{cases} \tag{8}$$

Na Figura 4 pode-se ver graficamente o comportamento das funções previamente mencionadas. A Escolha da função de ativação apropriada mais adequada depende do

tamanho do conjunto de dados, e como eles estão dispostos no espaço (XU et al., 2015).

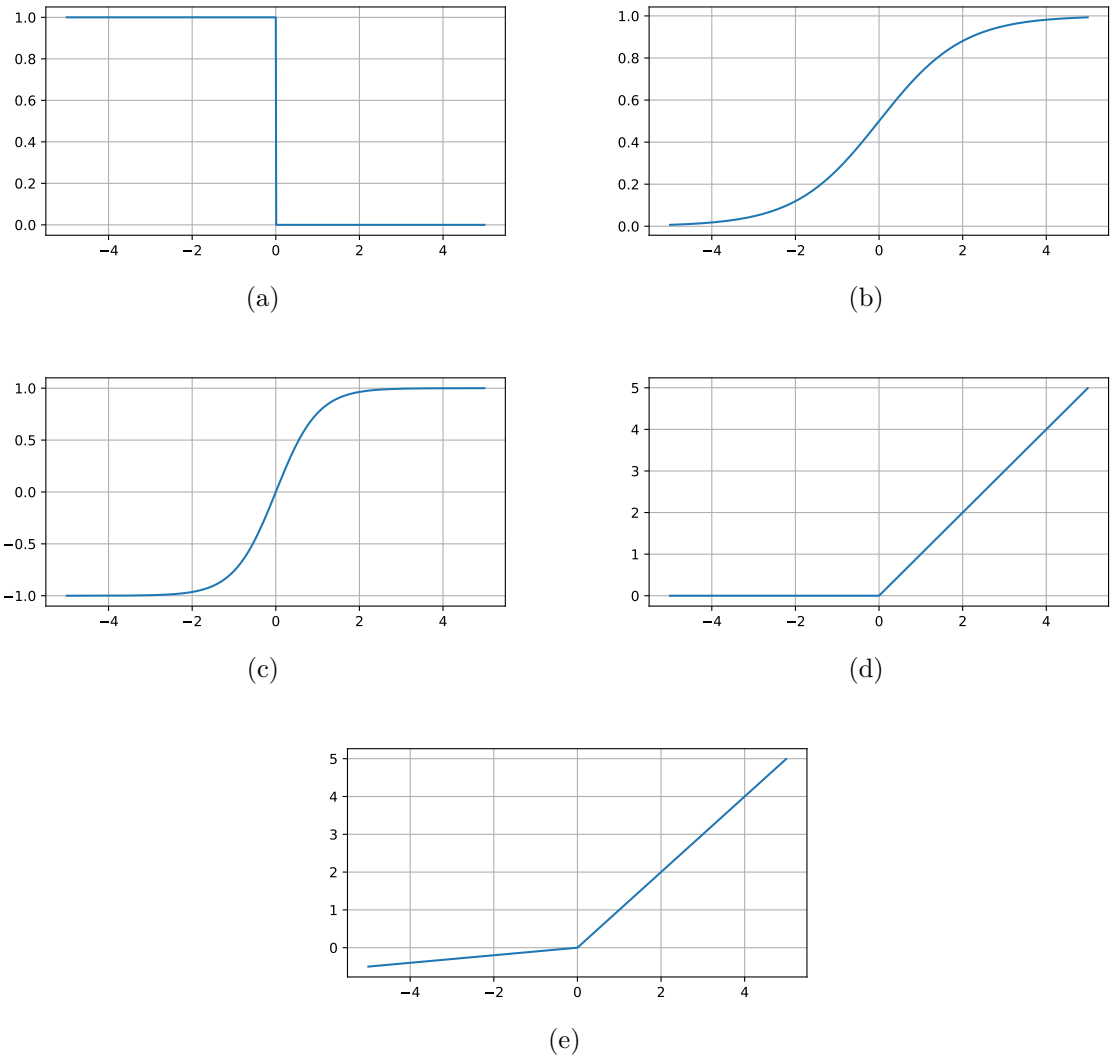


Figura 4 – Funções de Ativação aqui representadas: (a) Limiar, (b) Sigmoide, (c) Tangente Hiperbólica, (d) ReLU e (e) Leaky ReLU.

Fonte: (Autoria Própria)

2.2.1.2 Perceptron

A forma mais simples de uma RNA é um sistema Feed-Forward, como a rede exibida na Figura 5. Nestes sistemas, os neurônios são separados em camadas, no qual cada uma recebe a entrada da camada anterior, e suas saídas são repassadas a uma próxima, sem nenhum tipo de retorno às camadas anteriores. Nestes sistemas a menor unidade (neurônio) é chamada de perceptron (Equação 9) que recebe valores reais $[x_1, x_2, \cdots, x_n]$, os multiplica pelos pesos $[w_1, w_2, \cdots, w_n]$ e calcula uma combinação linear

entre eles. Se o resultado dos somatório destes valores somado a um bias b for maior do que 1 a função retorna 1, caso contrário retorna -1 (DIETTERICH, 2009).

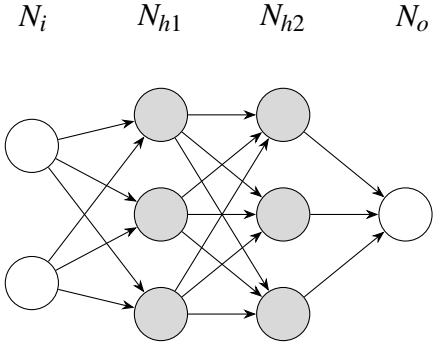


Figura 5 – Representação de uma RNA.
Fonte: (FELDMAN; ROJAS, 2013)

$$\varphi(x_1, x_2, \cdots, x_n) = \begin{cases} 1 & \text{se } b + \sum_{i=1}^{n} w_n x_n > 0 \\ -1 & \text{se } b + \sum_{i=1}^{n} w_n x_n \leq 0 \end{cases} \tag{9}$$

Um único perceptron consegue resolver diversas operações booleanas como E (AND), OU (OR), ¬ E (NAND) e ¬ OU (NOR), pois as mesmas são linearmente separáveis (podem ser classificadas com apenas uma linha reta). Operações booleanas não linearmente separáveis como OU-Exclusivo (XOR) podem ser resolvidas com uma rede de perceptrons com pelos menos duas camadas. Estas redes são conhecidas como Multi-Layer Perceptron (MLP) (DIETTERICH, 2009).

Para que seja possível que estas redes multi-camadas tenham um desempenho satisfatório, é necessário que os pesos das sinapses entre os neurônios tenham valores que façam com que a rede aproxime seu funcionamento a uma função ideal. Encontrar estes valores é um processo trabalhoso, portanto são utilizados algoritmos que fazem esta aproximação gradualmente, num processo chamado treinamento. Algoritmos como Gradiente Descendente e Backpropagation podem ser utilizados para fazer treinamento de MLPs (DIETTERICH, 2009).

2.2.1.3 Gradiente Descendente

Antes de aplicar este algoritmo, os pesos iniciais são decididos de maneira randômica. Em seguida se aplicam os exemplos de treinamento na rede iterativamente. Toda vez que a rede classificar um exemplo de maneira errada é necessário modificar os pesos. Com a regra de treinamento perceptron é necessário recalcular os pesos w_i

a cada passo na rede de acordo com a regra, dada pela Equação 10. A entrada do perceptron é representada por x_i, a saída obtida é o e saída desejada do exemplo é t. A constante η representa a taxa de aprendizado da rede, sendo geralmente definida como um valor decimal pequeno (por exemplo $0,1$), limitando a mudança que os pesos sofrem a cada passo. Este método pode aproximar funções linearmente separáveis (FELDMAN; ROJAS, 2013; DIETTERICH, 2009).

$$\begin{aligned} w_i &\Leftarrow w_i + \Delta w_i \\ \Delta w_i &= \eta (t - o) x_i \end{aligned} \tag{10}$$

Se a função a ser aproximada não for linearmente separável é necessário algum outro método para fazer o treinamento da rede. Um exemplo de método para fazer isto é a Regra Delta, que faz a utilização do Gradiente Descendente para buscar no espaço de hipóteses quais são os pesos que melhor se encaixam com os exemplos dados (DIETTERICH, 2009).

Gradiente Descendente é um algoritmo que aplica gradientes no espaço da Hipóteses. Gradientes são os cálculos do maior aclive possível a partir de um ponto em uma função no espaço. Os gradientes são calculados a cada passo do treinamento de uma rede e multiplicados por -1, eventualmente convergindo para um mínimo local da função de erro. Este ponto representa o conjunto de pesos com menor erro que foi encontrado pela Gradiente Descendente (YUAN, 2008).

Para buscar uma regra de aprendizado para a atualização dos pesos é necessária uma medida do erro para o vetor de pesos atuais. Uma maneira de calcular este erro é a Equação 11. O Erro do vetor de pesos ($E(\vec{w})$) é equivalente ao somatório das diferenças entre a saída desejada t_d da instância atual e o_d a saída do perceptron para a instância atual ao quadrado, no qual D é o conjunto de treinamento (FELDMAN; ROJAS, 2013; DIETTERICH, 2009).

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \tag{11}$$

O Gradiente Descendente segue a descida mais brusca possível a partir do ponto atual no espaço de hipóteses. Esta direção pode ser obtida calculando o gradiente da Função do erro E a respeito de $\vec{w}$, como mostrado na Equação 12. Logo, a Regra Delta (Equação 13) é definida pelo vetor de pesos $\vec{w}$ somado ao vetor $\delta\vec{w}$. Este vetor é definido pelo gradiente $E(\vec{w})$, mas como gradientes encontram a maior taxa de crescimento e o que se busca é a minimização do erro, o gradiente é multiplicado por -1 para inverter a direção

do vetor, e o resultante é multiplicado pela constante de taxa de aprendizado η que limita o tamanho do passo a ser dado pelo Gradiente Descendente. Diferenciando a Função de erro E e substituindo em Δw_i obtém-se a Equação 14, que é a regra de atualização de pesos do Gradiente Descendente 14 (FELDMAN; ROJAS, 2013; DIETTERICH, 2009).

$$\nabla E(\vec{w}) \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots, \frac{\partial E}{\partial w_n}\right] \tag{12}$$

$$\begin{aligned} \vec{w} &\Leftarrow \vec{w} + \Delta\vec{w} \\ \Delta\vec{w} &= -\nabla E(\vec{w})\eta \end{aligned} \tag{13}$$

$$\Delta\vec{w} = \eta \sum_{d \in D} (t_d - o_d) x_{id} \tag{14}$$

2.2.1.4 Backpropagation

O algoritmo backpropagation faz a utilização do método Gradiente Descendente para buscar o mínimo da função de erro no espaço de Hipóteses. Os pesos encontrados por esta busca são uma solução ao problema de aprendizado. Como o gradiente da função de erro deve ser recalculado a cada iteração da rede, o mesmo deve ser contínuo e diferenciável. Portanto uma função de ativação limiar não pode ser aplicada neste caso, pois uma função de ativação não contínua causaria a não continuidade dos erros. Algumas funções de ativação muito utilizadas para redes baseadas em backpropagation são as previamente mencionadas funções Sigmoide (Equação 5), Tangente Hiperbólica (Equação 6), ReLU (Equação 7) e Leaky ReLU (Equação 8) (FELDMAN; ROJAS, 2013; DIETTERICH, 2009; YUAN, 2008).

O algoritmo de backpropagation é composto por quatro etapas:

- O Feed-Forward;
- O backpropagation da camada de saída;
- O backpropagation das camadas ocultas;
- A atualização dos pesos.

Na primeira etapa, o Feed-Forward, um conjunto de treinamento $\vec{x}$ é introduzido à rede, e a saída de o_n de cada unidade n é computada (FELDMAN; ROJAS, 2013; DIETTERICH, 2009).

Na segunda etapa, o backpropagation da camada de saída, em cada unidade j na camada de saída, o erro propagado δ_j é calculado através da Equação 15, na qual x_j é a entrada do nó, o_j é a saída do nó (FELDMAN; ROJAS, 2013; DIETTERICH, 2009).

$$\delta_j = x_j(1-x_j)(x_j-o_j) \tag{15}$$

Na terceira etapa, o backpropagation das camadas ocultas, cada unidade l em uma camada oculta está conectada a cada unidade m na camada anterior, com um peso w_{lm} entre eles. O erro propagado δ_l deve ser calculado considerando todos os caminhos de retorno possíveis para a camada anterior de acordo com a Equação 16 (FELDMAN; ROJAS, 2013; DIETTERICH, 2009).

$$\delta_l = x_l(1-x_l)\sum_{m=1} w_{lm}\delta_m \tag{16}$$

Na quarta etapa, os pesos são atualizados de acordo com a Equação 17. Cada peso w_{lm} entre as unidades l e m recebem a entrada x_l multiplicada pelo erro propagado δ_m e a constante de aprendizado η negativa, para inverter a direção da gradiente com o objetivo de reduzir o erro (FELDMAN; ROJAS, 2013; DIETTERICH, 2009).

$$\Delta w_{lm} = -\eta x_l \delta_m \tag{17}$$

O algoritmo de backpropagation é muito eficiente para buscar pesos e iterativamente reduzir o erro da rede em funções. Porém não há garantia que ele vá encontrar a melhor solução possível para o problema. No espaço de hipóteses, ou seja o espaço dos possíveis pesos da rede, o gradiente encontra o primeiro mínimo local e não o mínimo global da função de erro. Em casos em que o espaço possui muitos mínimos locais, o resultado encontrado pode não ser muito adequado (DIETTERICH, 2009).

2.2.2 Máquinas de Boltsmann Restritas

Máquinas de Boltsmann Restritas (MBR) são um modelo de RNA não supervisionado utilizado para reduzir dimensionalidade de problemas. MBRs são compostas por apenas dois tipos de camadas: o primeiro é chamada de visível (que age como uma camada de entrada inicialmente), e o segundo são as camadas ocultas da rede (LAROCHELLE et al., 2012).

O treinamento destas redes se dá em dois passos: No primeiro, os nós fazem estimativas estocásticas (randômicas) para decidir que a entrada que eles receberam vai

ser transmitida ou não, da camada visível até a última camada oculta da rede. No segundo passo é feita a reconstrução dos dados, que seguem o caminho contrário na rede, recebendo os dados que foram perdidos e fazendo outras estimativas para tentar fazer uma aproximação dos dados originais, até retornar á camada visivel novamente. No final o erro da função é calculado verificando a diferença entre o dado original e o gerado pela rede. As atualizações dos pesos da rede podem ser feitos utilizando backpropagation (LAROCHELLE et al., 2012).

2.2.3 Autoencoder

Autoencoder é um tipo de RNA que recebe um conjunto de treinamento não rotulado e busca recriar este mesmo conjunto na saída da rede, mas com uma quantidade de nós reduzida nas camadas ocultas. A menor dimensionalidade das camadas ocultas faz com que o autoencoder precise encontrar uma representação reduzida dos dados, logo, se o dado original possuir características correlacionadas entre partes da entrada, a rede tentará encontrá-las. Esta arquitetura de RNA faz o uso de backpropagation para recalcular os pesos da rede em cada iteração (KUCHAIEV; GINSBURG, 2017).

O autoencoder, como o da Figura 6, pode ser dividido em duas partes: um encoder, que transforma uma entrada X_1^n de n dimensões em uma representação R^d com d dimensões, e um decoder, que rebebe R^d de entrada que retorna um X_2^n, de tal forma que o erro entre X_1^n e X_2^n seja minimizado (KUCHAIEV; GINSBURG, 2017).

A capacidade de recriação dos dados em um autoencoder faz com que existam diversos modelos derivados, por exemplo:

- Denoising Autoencoder (DAE): Durante a fase de treinamento, ruídos randômicos ou corrupções são somados aos dados sendo inseridos na representação reduzida dos dados, antes deles serem inseridos no decoder. Logo, para o decoder conseguir recriar um dado semelhante ao original, ele também deve aprender a remover o ruído dos dados antes de recuperá-los (GONDARA; WANG, 2018);
- Compressive Autoencoder (CAE): Semelhante ao DAE, mas faz a utilização de um modelo probabilístico para gerar os ruídos que são adicionados aos dados;
- Deep Autoencoder (DeepAE): Autoencoders que utilizam mais de uma camada no encoder e decoder, e portanto conseguem aproximar transformações não lineares mais complexas;

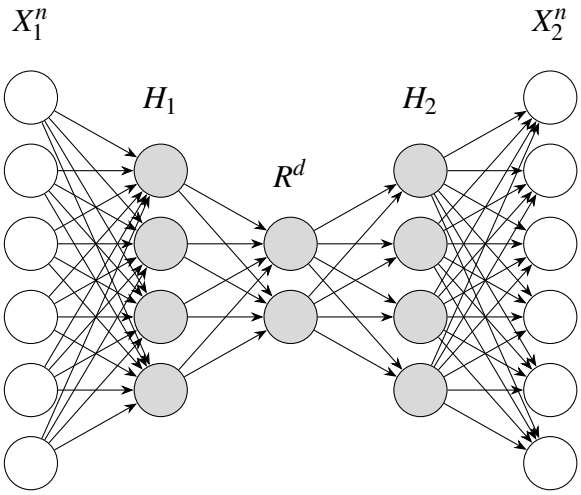


Figura 6 – Um exemplo de Autoencoder.
Fonte: (KUCHAIEV; GINSBURG, 2017)

- Convolutional autoencoder (ConvAE): Um autoencoder que possui camadas convolucionais. Camadas convolucionais possuem múltiplas dimensões, o que melhora a capacidade da rede em aprender em imagens.

2.3 CONSIDERAÇÕES FINAIS

Neste capítulo foram vistos os conceitos sobre compressão, como eles se aplicam a imagens, e também apresentados algoritmos de RNA e arquiteturas de rede que podem ser usadas para compressão. No próximo capítulo serão abordados os materiais e métodos que serão utilizados neste trabalho.

3 MATERIAIS E MÉTODOS

Após o estudo sobre os conceitos de compressão de imagem com perdas e sobre algoritmos de RNA, serão definidos neste capítulo os materiais e métodos necessários para fazer o desenvolvimento do algoritmo de compressão, assim como métodos de análise e avaliação do desempenho obtido com o mesmo. As atividades deste trabalho serão executadas como no fluxograma de atividades, disposto na Figura 7.

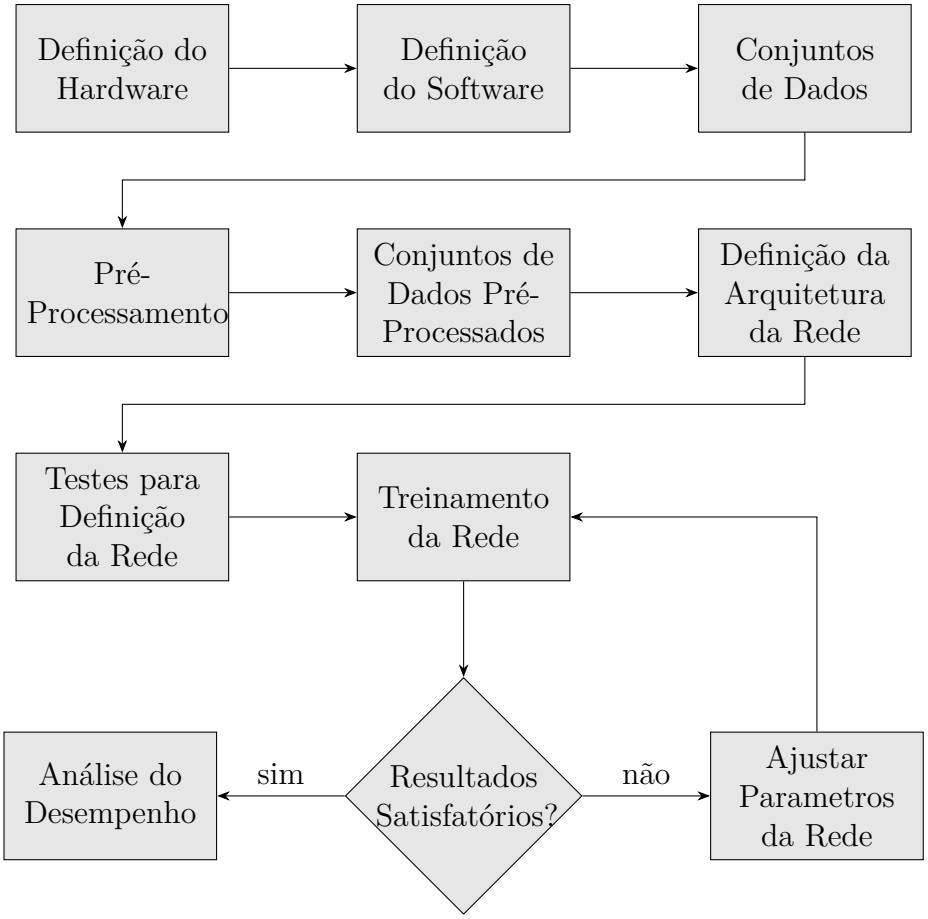


Figura 7 – Fluxograma de atividades

Fonte: (Autoria Própria)

3.1 HARDWARE

O computador que será utilizado primariamente é um notebook Lenovo Thinkpad T420 equipado com um processador Intel Core i5-2540M com dois núcleos de processamento e quatro threads com clock 2.60 GHz, 12 GB de memória RAM e uma placa de vídeo (GPU) NVIDIA Quadro NVS 4200M com 48 CUDA Cores. Caso o processamento a ser efetuado exceda a viabilidade neste dispositivo, poderá ser utilizado um computador Dell Optiplex 780 equipado com um processador Intel Core 2 Duo E8400, 8GB de RAM e uma GPU NVIDIA 750Ti com 640 CUDA Cores.

Como mencionado anteriormente, na Seção 2.2, o treinamento de RNAs é altamente paralelizável. Portanto, um dispositivo que consiga executar vários processos simultaneamente é necessário. Consequentemente uma GPU, que possui múltiplos núcleos pois são desenvolvidas para efetuar operações matriciais com propósito gráfico, tem uma grande vantagem sobre o treinamento em processadores convencionais. Uma GPU utilizada em processamento de tarefas que não envolvam renderização e processamento recebe o nome de GPU de propósito geral (GPGPU).

3.2 SOFTWARE

Os dois computadores utilizados possuem o Sistema Operacional Arch Linux instalado. Este sistema é uma distribuição GNU/Linux executando o kernel versão 4.16. Todos os softwares mencionados estarão na mesma versão nos dois dispositivos.

As linguagens de programação e scripts que serão utilizadas neste trabalho serão o GNU bash versão 4.4 e Python versão 3.6. O GNU bash é uma interface de login de linha de comando que também executa arquivos de script. Python é uma linguagem de programação de alto nível que é amplamente utilizada na área de Aprendizado de Máquina.

O ambiente de desenvolvimento a ser utilizado será o Spyder[1] (Scientific Python Development Environment) versão 3.2 (Figura 8). O Spyder é uma ferramenta Open

[1] https://www.spyder-ide.org/

Source que possui integração com o console IPython, um explorador de variáveis que visualiza arrays e capacidade de executar scripts parcialmente.

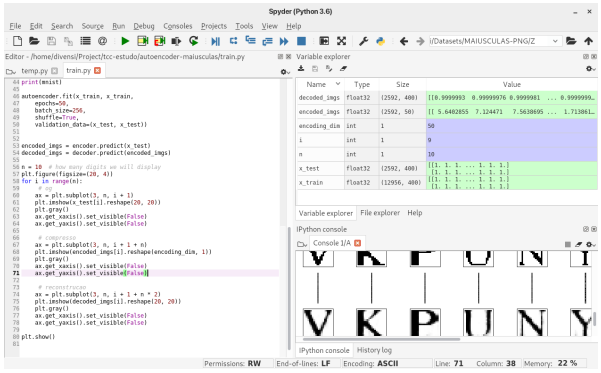


Figura 8 – Editor Spyder

Fonte: (Autoria Própria)

Para construir os modelos de RNA será utilizada a biblioteca de deep learning Keras (2018)[2]. Keras é uma API escrita em Python criada para facilitar e acelerar a prototipação e desenvolvimento de algoritmos de AM. O Keras pode funcionar em cima de três bibliotecas de computação numérica: TensorFlow, CNTK ou Theano.

A biblioteca de computação escolhida para este trabalho é o Tensorflow (2018)[3], por possuir suporte a processamento em GPGPU com CUDA, e por possuir uma grande comunidade de desenvolvedores já utilizando-a.

Neste sistema a ferramenta de programação em GPU proprietária NVIDIA CUDA[4] versão 9.1 foi instalada em conjunto com o NVIDIA cuDNN[5] versão 7.1. CUDA é uma plataforma de programação criada para facilitar a utilização de GPGPUs em diversas aplicações. cuDNN é uma biblioteca que faz parte do NVIDIA Deep Learning SDK, um kit de desenvolvimento criado para otimizar a execução de tarefas relacionadas com Aprendizado de Máquina que utiliza CUDA.

Também serão utilizadas as bibliotecas matplotlib[6], para criar gráficos e visualizar imagens, e OpenCV[7], para converter imagens e fazer o pré-processamento dos mesmos.

[2]https://keras.io/
[3]https://www.tensorflow.org/
[4]https://developer.nvidia.com/cuda-zone
[5]https://developer.nvidia.com/cudnn
[6]https://matplotlib.org/
[7]https://opencv.org/

(a) (b)

Figura 9 – Amostra dos datasets mencionados: (a) Conjunto de Letras Maiúsculas (PAULA FILHO, 2018), (b) Conjunt de faces LFWcrop (SANDERSON; LOVELL, 2009)

Fonte: (Autoria Própria)

3.3 CONJUNTOS DE DADOS

Este trabalho utilizará três conjuntos de dados de diferentes níveis de complexidade para avaliar o desempenho e a escalabilidade dos algoritmos aplicados:

- O primeiro conjunto de dados é composto por imagens de resolução 32×32 com 1 bit de cores, composto por letras maiúsculas e minúsculas em imagens binárias. O conjunto vem em imagens individuais no formato PGM (PAULA FILHO, 2018);
- O segundo conjunto de dados é o LFWcrop. Este é um dataset de faces baseado no dataset Labeled Faces in the Wild, com pré-processamento já feito para exibir apenas as faces, processo feito por Sanderson e Lovell (2009). O conjunto contém mais de 13000 instâncias, e cada imagem do conjunto tem resolução de 64×64 pixeis com 24 bits de cores. O conjunto é disponibilizado em um arquivo zip com arquivos no formato PPM;

Na Figura 9 pode ser visualizada uma amostra dos conjuntos de dados que serão utilizados.

3.4 PRÉ-PROCESSAMENTO DOS DADOS

Os conjuntos de dados que serão utilizados têm diferentes formatos, portanto é necessário fazer o pré-processamento individual de cada conjunto. O conjunto de letras maiúsculas vêm no formato PGM, que é um formato de imagens em escala de cinza, na resolução 20×20. Como o Keras não suporta este formato nativamente, foi escrito um script em Python utilizando OpenCV para convertê-los para o formato PNG (formato escolhido por não causar perda de qualidade).

O conjunto LFWcrop vem com imagens no formato PPM, um formato semelhante ao PGM, mas que suporta cores. Estas imagens também serão convertidas para o formato PNG com um script em Python com OpenCV. Para testar as redes criadas com uma crescente complexidade de dados, um novo conjunto será derivado do LFWcrop, aplicando um filtro de escala de cinza utilizando Python e OpenCV. Estas imagens vão ser importadas e armazenadas em um único arquivo, utilizando um script em Python utilizando a biblioteca Pickle, para facilitar a leitura das redes.

Após o pré-processamento dos três conjuntos, em ordem de complexidade crescente, serão obtidos:

- O conjunto binário de letras maiúsculas;
- O conjunto de faces LFWcrop em escala de cinza;
- O conjunto de faces LFWcrop colorido.

3.5 ARQUITETURAS DE REDE NEURAL

A arquitetura de RNA que será utilizada para criar o modelo computacional será o autoencoder, pois o mesmo funciona de maneira análoga a um algoritmo de compressão. Para decidir a derivação que será utilizada, alguns testes devem ser efetuados para decidir o mais adequado. Nestes testes uma rede será criada no formato de cada derivação e nesta será aplicado um dataset de menor complexidade. A melhor rede será definida com relação ao tempo requerido para o aprendizado e a qualidade das imagens obtidas a partir da rede.

Para a criação de uma rede do tipo autoencoder, alguns pontos precisam ser

definidos previamente:

- O número de camadas do encoder;
- O número de camadas do decoder;
- O tamanho da versão reduzida da imagem;
- O otimizador a ser utilizado;
- A função de ativação.

  E a cada camada de neurônios os seguintes parâmetros são definidos:

- Tipo da camada;
- A quantidade de neurônios;
- A função de ativação da camada.

3.6 ANÁLISE DO DESEMPENHO

Nesta etapa será calculado o desempenho das redes, e elas serão comparadas com algoritmos de compressão tradicionais. O algoritmo criado será treinado com três conjuntos de dados em ordem de complexidade. O primeiro será o conjunto de letras maiúsculas, por ser composto de imagens binárias. O segundo será o conjunto de faces pré-processado para escala de cinza. E por último será o conjunto de faces original com cores.

As métricas de comparação entre os algoritmos serão três, que já foram utilizadas por outros autores em trabalhos de compressão de imagem como o de Santurkar et al. (2017):

- bits/pixel: Uma simples métrica que mede com quantos bits a compressão representa cada pixel da imagem original;
- Peak Signal-to-Noise Ratio (PSNR): Baseada no Erro Quadrático Médio (ENRIQUE et al., 2003), é uma das medidas mais simples e utilizadas para comparar imagens, porém não possui uma boa correlação com a qualidade percebida por humanos (BOSSE et al., 2017);
- Structural Similarity Index (SSIM): De acordo com Reddy (2017), o SSIM se tornou a métrica padrão para qualidade de imagens. Ela calcula a degradação da estrutura de uma imagem em relação a uma imagem original.

3.7 CONSIDERAÇÕES FINAIS

Neste capítulo foram apresentados os materiais e métodos que serão utilizados para desenvolver este trabalho, assim como os métodos de avaliação do algoritmo que será criado.

4 RESULTADOS E DISCUSSÃO

Utilizando os materiais e métodos previamente apresentados, neste capítulo serão amostrados e discutidos os resultados obtidos a partir das redes criadas, assim como comparações a outros métodos de compressão utilizados comercialmente. Os resultados serão apresentados na ordem de complexidade das imagens apresentados na Seção 3.3. As métricas apresentadas foram calculadas se fazendo uma média de uma amostra de 10 imagens para cada teste. As redes criadas foram testadas a cada 1000 épocas e o treinamento foi encerrado quando três testes consecutivos não obtiveram melhorias significativas nos índices utilizados.

4.1 EXPERIMENTO COM IMAGENS BINÁRIAS DE LETRAS MAIÚSCULAS

Para a compressão do conjunto de imagens binárias de letras, foi criada uma RNA com duas camadas ocultas. A camada de entrada desta rede é composta por 1024 entradas, correspondentes à resolução de 32x32 pixeis das imagens, a primeira camada oculta é uma camada completamente conectada com função de ativação ReLU. Esta é responsável por fazer o trabalho de encoder na rede portanto o seu número de nós foi variado para verificar os resultados considerando a qualidade das imagens obtidas na camada de saída, considerando as métricas de MSE e SSIM discutidas na Seção 3.6. A última camada oculta é a camada de saída com 1024 nós, que são redimensionados novamente para 32x32 pixeis obtendo-se a aproximação da imagem original. A rede utilizou o otimizador adadelta e a função de perda Binary Cross-Entropy.

Foram criadas redes variando a primeira camada oculta entre 10, 20 e 30 nós, assim tendo uma variação entre a taxa de compressão que foi obtida e as métricas de qualidade de imagem.

Na Tabela 2 são exibidos os resultados das redes criadas, para comparação,

também são exibidos exemplos de compressão utilizando o formato JPEG com o parâmetro de qualidade 0%, 50% e 100%. Neste se pode ver que de acordo com o MSE, aonde menores índices apontam menor erro entre a imagem após a compressão e a original, todas as redes criadas tiveram resultado melhor ao comparado com JPEG de qualidades 0%, 50% e 100%. Porém, de acordo com o SSIM, aonde maiores índices indicam maior similaridade estrutural entre as imagens, o melhor resultado foi com o autoencoder de 30 nós, seguido pelo autoencoder de 20 nós, e os JPEG de qualidade 100% e 50%, com o autoencoder de 10 nós e o JPEG de qualidade 0%. Todas as imagens obtidas pelas redes criadas obtiveram maior índice de compressão em relação ao JPEG. Amostras dos conjuntos de imagens podem ser vistos na Figura 10.

Tabela 2 – Resultados do conjunto de imagens binárias.

Imagem	Tamanho	Taxa de compressão	MSE	SSIM
Original	1.024 bytes	1,000	0,00	1,00
JPEG Qualidade 0%	200 bytes	0,195	0,18	0,59
JPEG Qualidade 50%	213 bytes	0,208	0,20	0,60
JPEG Qualidade 100%	233 bytes	0,227	0,17	0,63
Autoencoder 10 nós	10 bytes	0,001	9,60	0,57
Autoencoder 20 nós	20 bytes	0,019	6,58	0,79
Autoencoder 30 nós	30 bytes	0.029	4,10	0,82

Fonte: (Autoria Própria)

Na Figura 11 há ver uma imagem ampliada para uma análise mais precisa dos resultados e do comportamento dos algoritmos. Nesta é possível ver a imagem original de uma letra “E” maiúscula que possui uma leve falha no seu lado esquerdo. A imagem processada pelo codec JPEG de qualidade 100% possui diversos artefatos de compressão, mas manteve a falha. A imagem processada pela rede autoencoder de 30 nós não possui artefatos semelhantes, porém perdeu o detalhe do lado esquerdo e na serifa inferior direita, assim como causou um efeito de borramento em algumas partes da letra. O índice MSE ficou muito abaixo do JPEG, mas o índice de SSIM da rede de 30 nós foi 19% maior e sua taxa de compressão foi mais de 7 vezes maior.

(a) Original, Taxa de compressão 1,000, MSE: 0,00, SSIM: 1,00

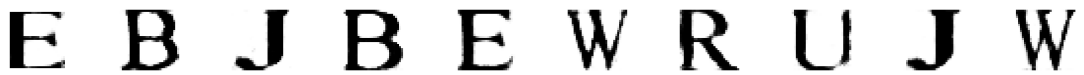
(b) JPEG Qualidade 0%, Taxa de compressão 0,195, MSE: 0,18, SSIM: 0,59

(c) JPEG Qualidade 50%, Taxa de compressão 0,208, MSE: 0,20, SSIM: 0,60

(d) JPEG Qualidade 100%, Taxa de compressão 0,227, MSE: 0,17, SSIM: 0,63

(e) Autoencoder 10 nós, Taxa de compressão 0,001, MSE: 9,60, SSIM: 0,57

(f) Autoencoder 20 nós, Taxa de compressão 0,019, MSE: 6,58, SSIM: 0,79



(g) Autoencoder 30 nós, Taxa de compressão 0,029, MSE: 4,10, SSIM: 0,82

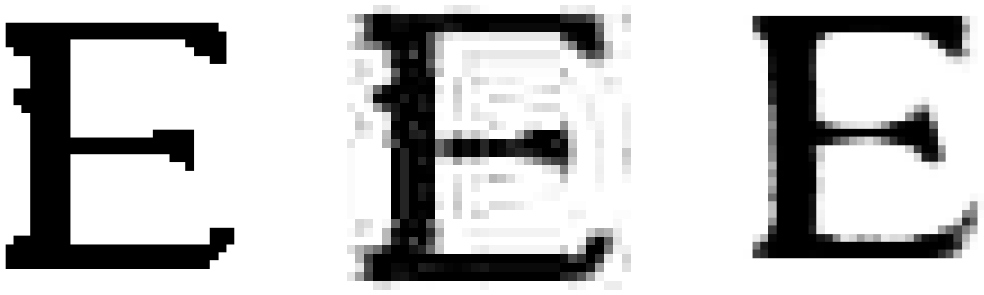
Figura 10 – Amostra de resultados de compressão com conjunto binário

Fonte: (Autoria Própria)

4.2 EXPERIMENTO COM IMAGENS EM ESCALA DE CINZA DO CONJUNTO LFWCROP

Com base nas redes criadas no teste anterior, foram criadas 3 novas redes para este experimento. Desta vez com camadas ocultas de 64 e 128 nós adaptadas para a entrada de 4096 nós (resultante da resolução de 64x64 pixeis), e uma rede convolucional com camada oculta de 4x4x8 nós (totalizando 128 nós).

Na Tabela 3 são exibidos os resultados das redes criadas, para comparação, também são exibidos exemplos de compressão utilizando o formato JPEG com os mesmos



(a) Original (b) JPEG Qualidade 100% (c) Autoencoder 30 nós

Figura 11 – Amostra detalhada de uma imagem binária.
Fonte: (Autoria Própria)

parâmetros do conjunto anterior. Uma amostra das imagens resultantes pode ser vista na Figura 12.

Neste experimento, pelo índice MSE as três redes possuem resultados melhores quando comparados ao JPEG de qualidades 50% e 0%, mas perdem em comparação ao JPEG com qualidade ajustada para 100%. Pelo SSIM o JPEG com qualidade 100% novamente recebe o maior índice de similaridade, com a rede convolucional 4x4x8 ficando em segundo lugar seguido pelo JPEG de qualidade 50%, autoencoder de 128 nós, autoencoder de 64 nós e em último fica o JPEG de qualidade 0 %.

Ao se analisar as taxas de compressão obtidas, todas as redes obtiveram melhores colocações em relação a todas as qualidades de JPEG analisadas. Com destaque á rede autoencoder convolucional, que obteve melhores resultados nos índices MSE e SSIM á rede autoencoder não convolucional utilizando a mesma quantidade de dados.

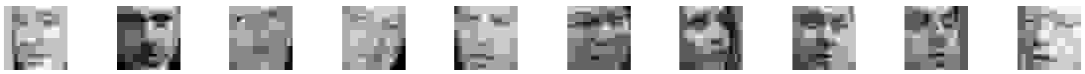
Tabela 3 – Resultados do conjunto de imagens em escala de cinza.

Imagem	Tamanho	Taxa de compressão	MSE	SSIM
Original	4.096 bytes	1,000	0,00	1,00
JPEG Qualidade 0%	882 bytes	0,200	255,91	0,55
JPEG Qualidade 50%	1.229 bytes	0.300	7,05	0,69
JPEG Qualidade 100%	2.704 bytes	0.660	0,08	0,81
Autoencoder 64 nós	64 bytes	0,015	4,21	0,60
Autoencoder 128 nós	128 bytes	0,031	3,67	0,67
Autoencoder Conv. 4x4x8	128 bytes	0,031	3,32	0,75

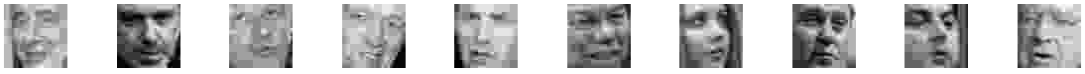
Fonte: (Autoria Própria)

Na Figura 13, uma imagem do conjunto foi ampliada para análise do comportamento dos algoritmos. Na imagem original, se percebe que a face da pessoa

(a) Original, Taxa de compressão 1,000, MSE: 0,00, SSIM: 1,00



(b) JPEG Qualidade 0%, Taxa de compressão 0,200, MSE: 255,91, SSIM: 0,55



(c) JPEG Qualidade 50%, Taxa de compressão 0,300, MSE: 7,05, SSIM: 0,81

(d) JPEG Qualidade 100%, Taxa de compressão 0,660, MSE: 0,08, SSIM: 0,81

(e) Autoencoder 64 nós, Taxa de compressão 0,015, MSE: 4,21, SSIM: 0,60

(f) Autoencoder 128 nós, Taxa de compressão 0,031, MSE: 3,67, SSIM: 0,67

(g) Autoencoder Convolucional 4x4x8, Taxa de compressão 0,031, MSE: 3,32, SSIM: 0,75

Figura 12 – Amostra de resultados de compressão com conjunto em escala de cinza
Fonte: (Autoria Própria)

possui óculos e é possível ver a direção que este está olhando. Na imagem processada pelo codec JPEG de qualidade 100%, é difícil identificar a direção que a pessoa está olhando, e novamente se vêem diversos artefatos na imagem, na qual é possível perceber que o algoritmo segmenta a imagem em diversas regiões de interesse para poder fazer a compressão. Na imagem processada pelo autoencoder convolucional 4x4x8, pode-se perceber que a imagem tem menos detalhes perceptíveis do que as outras duas, tendo uma aparência borrada, na qual é difícil se identificar atributos como os óculos da pessoa e a direção que o mesmo está olhando. Nesta imagem, o índice de SSIM da rede convolucional 4x4x8 foi 6% menor em comparação ao JPEG e o índice MSE também foi muito abaixo, mas a taxa de compressão foi mais de 21 vezes maior.

(a) Original (b) JPEG Qualidade 100% (c) Autoencoder Conv. 4x4x8

Figura 13 – Amostra detalhada de uma imagem em escala de cinza.
Fonte: (Autoria Própria)

4.3 EXPERIMENTO COM IMAGENS COLORIDAS DO CONJUNTO LFWCROP

Tendo em vista os resultados vantajosos do conjunto anterior com a rede convolucional, para o experimento no conjunto de imagens LFWcrop coloridas foram criadas três redes autoencoder convolucionais, com base nas redes anteriormente utilizadas.

As alterações feitas nas redes com relação ao conjunto anterior foi na quantidade de neurônios de suas camadas, que possuem entrada e saída de 64x64x3 (representando a resolução de 64x64 com 3 cores), a camada de menor tamanho das redes difere para que sua taxa de compressão possa variar. Foram criadas redes com camadas ocultas de tamanho 4x4x8 (128 nós), 8x8x9 (576 nós) e 16x16x8 (2.048 nós).

Uma amostra das imagens resultantes pode ser vista na Figura 14. Na Tabela 4 é apresentado o resultado das redes criadas em comparação com o formato JPEG. Neste experimento, considerando o resultado do índice MSE, as três redes conseguiram melhores resultados do que o codec JPEG em qualidade 50% e 0%, mas novamente ficaram atrás do JPEG de qualidade 100%.

Já no SSIM, as imagens mais bem colocadas foram obtidas com o autoencoder convolucional 16x16x8, seguido pelo JPEG com qualidade 100%, autoencoder convolucional 8x8x9, JPEG com qualidade 50%, e por último empatados a rede convolucional 4x4x8 e JPEG com qualidade 0%.

As melhores taxas de compressão obtidas foram pelas redes 4x4x8 e 8x8x9, seguidas pelas imagens do algoritmo JPEG de qualidades 0% e 50%, do autoencoder

convolucional 16x16x8 e pelo JPEG com qualidade 100%.

Tabela 4 – Resultados do conjunto de imagens coloridas.

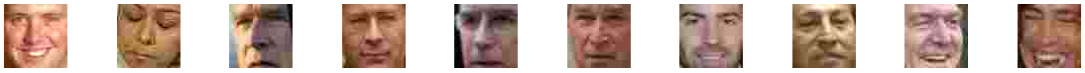
Imagem	Tamanho	Taxa de compressão	MSE	SSIM
Original	12.288 bytes	1,000	0,00	1,00
JPEG Qualidade 0%	1.289 bytes	0,104	1340,66	0,51
JPEG Qualidade 50%	1.730 bytes	0,140	32,19	0,73
JPEG Qualidade 100%	4.874 bytes	0,396	1,44	0,85
Autoencoder Conv. 4x4x8	128 bytes	0,010	9,60	0,51
Autoencoder Conv. 8x8x9	576 bytes	0,047	6,58	0,75
Autoencoder Conv. 16x16x8	2.048 bytes	0,166	4,10	0,91

Fonte: (Autoria Própria)

(a) Original, Taxa de compressão 1,000, MSE: 0,00, SSIM: 1,00

(b) JPEG Qualidade 0%, Taxa de compressão 0,104, MSE: 1340,66, SSIM: 0,51

(c) JPEG Qualidade 50%, Taxa de compressão 0,140, MSE: 32,19, SSIM: 0,73



(d) JPEG Qualidade 100%, Taxa de compressão 0,396, MSE: 9,60, SSIM: 0,85

(e) Autoencoder Convolucional 4x4x8, Taxa de compressão 0,010, MSE: 9,60, SSIM: 0,51

(f) Autoencoder Convolucional 8x8x9, Taxa de compressão 0,047, MSE: 6,58, SSIM: 0,75

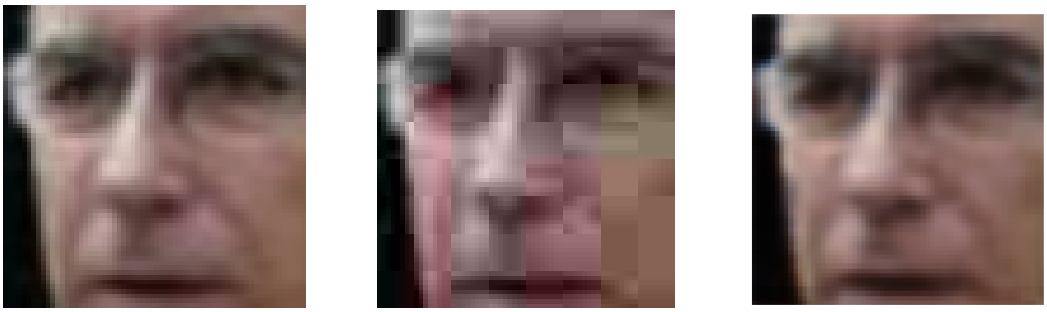
(g) Autoencoder Convolucional 16x16x8, Taxa de compressão 0,166, MSE: 4,10, SSIM: 0,91

Figura 14 – Amostra compressão com imagens coloridas

Fonte: (Autoria Própria)

Na Figura 15, uma imagem do conjunto foi ampliada para análise. Na imagem

original é possível ver que a pessoa da imagem possui óculos e a direção que esta está olhando. Na imagem processada por JPEG com qualidade definida em 100% é possível ver que há artefatos visuais como os quadrados mencionados no conjunto de imagens anterior, assim como aberrações cromáticas, que é a falha ao recuperar as cores da imagem ao seu ponto original. Na imagem processada pelo autoencoder convolucional, não existem tais artefatos na imagem, e ainda é possível identificar os óculos e a direção para que a pessoa está olhando, porém ainda é possível ver que a imagem não é 100% fiel à original, tendo sombras levemente mais escuras. O índice de MSE deste novamente foi muito inferior ao resultado do JPEG, porém o índice SSIM foi 6% maior e a taxa de compressão foi mais de 2 vezes melhor.



(a) Original (b) JPEG Qualidade 100% (c) Autoencoder Conv. 16x16x8

Figura 15 – Amostra detalhada de uma imagem colorida.

Fonte: (Autoria Própria)

4.4 DISCUSSÃO

Neste capítulo foram apresentados os resultados obtidos com as redes criadas. Os resultados aqui apresentados, se aplicados em grande escala, podem implicar em uma redução significativa no tamanho de imagens, reduzindo também a quantidade de pacotes de rede necessários para carregamento destas imagens. Isto acarreta em um aumento na velocidade de carregamento de páginas online, redução no tráfego de dados por página e no custo de acesso por página navegada, facilitando o acesso a internet em lugares remotos com baixa recepção.

5 CONSIDERAÇÕES FINAIS

Neste capítulo serão apresentadas as considerações finais deste trabalho. Neste é discutido se os objetivos do trabalho foram cumpridos, aplicações para o trabalho criado, assim como recomendações de trabalhos futuros com base neste.

5.1 CONCLUSÕES

O objetivo proposto a este trabalho foi utilizar RNAs para fazer a super-resolução de imagens com o objetivo de recuperar a qualidade original das mesmas após um processo de compressão.

Para atingir este objetivo foram definidos 3 conjuntos de imagens, um com imagens binárias, outro com imagens em escala de cinza e o último com imagens coloridas, para apresentar uma crescente complexidade na compressão das imagens. Estas foram pré processadas para ficarem padronizadas de maneira que facilitasse o treinamento.

Foi selecionada a arquitetura de redes neurais autoencoder para poder se fazer o treinamento dos modelos. Após alguns testes empíricos foi definida a utilização de uma rede autoencoder convolucional, que obteve resultados consideravelmente melhores ao autoencoder simples. Principalmente nos conjuntos de imagens coloridas.

Após treinada, os resultados das redes criadas foram analisados em comparação ao codec JPEG com o parâmetro de qualidade definido para 0%, 50% e 100%, utilizando as métricas de SSIM, MSE e taxa de compressão.

No conjunto de imagens binárias, imagens processadas pelas redes criadas obtiveram com índices SSIM superiores a imagens processadas por JPEG, alcançaram taxas de compressão mais de 7 vezes maiores. No conjunto de imagens em escala de cinza, imagens com índices SSIM 6% abaixo de imagens processadas por JPEG alcançaram taxas de compressão de mais 21 vezes maiores. No conjunto de imagens coloridas, imagens com

índices SSIM 6% maiores em relação a imagens processadas por JPEG alcançaram taxas de compressão de mais de 2 vezes maiores.

Com estes resultados obtidos, em grande maioria obtendo taxas de compressão diversas vezes maior a imagens em JPEG com índices SSIM próximos, é possível dizer que o trabalho cumpriu com o proposto, obtendo resultados em grande parte maiores a algoritmos clássicos de compressão de imagem com perda.

5.2 TRABALHOS FUTUROS

Para possíveis trabalhos futuros baseados neste, se indica:

- Utilizar um algoritmo semelhante para fazer a ampliação de imagens para uma resolução maior do que o original;
- Aplicar algoritmos em imagens com maior resolução, possivelmente subdividindo-as em imagens menores;
- Aplicar outras arquiteturas de Redes Neurais Artificiais para possíveis melhores resultados;
- Aplicar algoritmos semelhantes em vídeos, aplicando a técnica em cada frame, ou em frames agrupados;
- Generalizar o algoritmo para trabalhar com diferentes categorias de imagem.

REFERÊNCIAS

AGUILERA, P. Comparison of different image compression formats ECE 533 Project Report. p. 5–9, 2007.

ARGE, L. et al. Introduction to Data Compression. [S.l.: s.n.], 1993. 83–94 p. ISSN 0302-9743. ISBN 978-3-540-73948-7.

BOSSE, S. et al. SHEARLET-BASED REDUCED REFERENCE IMAGE QUALITY ASSESSMENT Fraunhofer Institute for Telecommunications , Heinrich Hertz Institute , Berlin , Germany Department of Electrical Engineering , Technical University of Berlin , Germany . p. 315–319, 2017.

BOURKE, P. B. PPM / PGM / PBM image files. Jul 1997. Disponível em: <http://paulbourke.net/dataformats/ppm/>.

CISCO. Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2011–2016 [Visual Networking Index (VNI)]. Cisco, p. 2016–2021, 2017. ISSN 1553-877X. Disponível em: <https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.pdf\%0Ahttps://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-5>.

DIETTERICH, T. G. Machine learning in ecosystem informatics and sustainability. [S.l.: s.n.], 2009. 8–13 p. ISSN 10450823. ISBN 9781577354260.

ENRIQUE, G.; PRADO, D. A.; BATISTA, A. Pré-processamento de Dados em Aprendizado de Máquina Supervisionado. American Journal of Distance Education, p. 1 – 204, 2003.

FELDMAN, J.; ROJAS, R. Neural Networks: A Systematic Introduction. Springer Berlin Heidelberg, 2013. ISBN 9783642610684. Disponível em: <https://books.google.com.br/books?id=4rESBwAAQBAJ>.

GONDARA, L.; WANG, K. Recovering Loss to Followup Information Using Denoising Autoencoders. p. 1936–1945, 2018.

GUPTA, R.; KUMAR, M.; BATHLA, R. Data Compression - Lossless and Lossy Techniques. v. 5, n. 7, p. 120–125, 2016. Disponível em: <https://pdfs.semanticscholar.org/d5ef/f500c2502c9ca3292418f4c4715ed1cb8618.pdf>.

HAYKIN, S. Neural Networks and Learning Machines. [S.l.: s.n.], 2008. 906 p. ISSN 14337851. ISBN 9780131471399.

HU, Y.-C.; CHANG, C.-C. A new lossless compression scheme based on Huffman coding scheme for image compression. Signal Processing: Image Communication, v. 16, n. 4, p. 367–372, 2000. ISSN 09235965. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0923596599000648>.

KERAS. Documentação Keras. 2018. `https://keras.io/`. [Online; acessado em 21/05/2018].

KUCHAIEV, O.; GINSBURG, B. Training Deep AutoEncoders for Collaborative Filtering. 2017. Disponível em: <http://arxiv.org/abs/1708.01715>.

LAROCHELLE, H. et al. Learning Algorithms for the Classification Restricted Boltzmann Machine. The Journal of Machine Learning Research, v. 13, p. 643–669, 2012. ISSN 15324435.

LI, D.; WANG, Z. Video Super-Resolution via Motion Compensation and Deep Residual Learning. IEEE Transactions on Computational Imaging, v. 3, n. 4, p. 1–1, 2017. ISSN 2333-9403. Disponível em: <http://ieeexplore.ieee.org/document/7858640/>.

LORENA, A. C.; GAMA, J.; FACELI, K. Inteligência Artificial: Uma abordagem de aprendizado de máquina. [S.l.]: Grupo Gen-LTC, 2000.

MAHONEY, M. Data Compression Explained. v. 77, n. C, p. 1–81, 2010. Disponível em: <papers3://publication/uuid/56CF053B-A0E0-4978-ADA6-017E007F0953>.

MURRAY, J.; VANRYPER, W. Encyclopedia of graphics file formats. [S.l.]: O'Reilly & Associates, 1996. (O'Reilly Series). ISBN 9781565921610.

NIKLAUS, S.; MAI, L.; LIU, F. Video Frame Interpolation via Adaptive Separable Convolution. Proceedings of the IEEE International Conference on Computer Vision, v. 2017-Octob, p. 261–270, 2017. ISSN 15505499. Disponível em: <https://arxiv.org/pdf/1708.01692.pdf>.

PAULA FILHO, P. L. de. Material de aula da Disciplina de Processamento de Imagens e Reconhecimento de Padrões. [S.l.]: Universidade Tecnológica Federal do Paraná, Março 2018.

REDDY, K. S. P. Comparative study of Structural Similarity Index (SSIM) by using different edge detection approaches on live video frames for different color models. p. 932–937, 2017.

RUSSELL, S. J.; NORVIG, P. Artificial Intelligence: A Modern Approach. [s.n.], 2010. 1132 p. ISSN 00206539. ISBN 0137903952. Disponível em: <http://amazon.de/o/ASIN/0130803022/>.

SAKAI, M.; HIGUCHI, H.; UESUGI, M. Importance of Including Practical Machine Learning Application in the Universities ' Curriculum. 2018 5th International Conference on Business and Industrial Research (ICBIR), IEEE, p. 608–612, 2018.

SANDERSON, C.; LOVELL, B. C. Multi-region probabilistic histograms for robust and scalable identity inference. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), v. 5558 LNCS, p. 199–208, 2009. ISSN 03029743.

SANTURKAR, S.; BUDDEN, D.; SHAVIT, N. Generative Compression. 2017. ISSN 1703.01467. Disponível em: <http://arxiv.org/abs/1703.01467>.

SAYOOD, K. Introduction to Data Compression. Managing, p. xx + 636, 2000. ISSN 14710528.

SHANNON, C. E. A Mathematical Theory of Communication. Bell System Technical Journal, v. 27, n. 3, p. 379–423, 1948. ISSN 15387305.

TENSORFLOW. Documentação Tensorflow. 2018. `https://www.tensorflow.org/`. [Online; acessado em 21/05/2018].

WADE, G. Signal Coding and Processing. 2nd. ed. Cambridge University Press, 1994. 460 p. ISBN 0521423368. Disponível em: <https://books.google.com.br/books?id=CJswCy7>.

XU, B. et al. Empirical Evaluation of Rectified Activations in Convolutional Network. 2015. Disponível em: <http://arxiv.org/abs/1505.00853>.

YUAN, Y.-x. Step-sizes for the gradient method. Ams Ip Studies in Advanced Mathematics, v. 42, p. 785–796, 2008. Disponível em: <ftp://159.226.92.9/pub/yyx/papers/p0504.pdf>.