Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Compressão de imagens com perda usando Redes Neurais

Raphael Soares Ramos

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientador
Prof. Dr. Teófilo Emidio de Campos

Brasília
2019

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Compressão de imagens com perda usando Redes Neurais

Raphael Soares Ramos

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Prof. Dr. Teófilo Emidio de Campos (Orientador)
CIC/UnB

Dr. Edson Mintsu Hung
FT/UnB

Prof. Dr. Bruno L. Macchiavello Espinoza
CIC/UnB

Prof. Dr. Edison Ishikawa
Coordenador do Bacharelado em Ciência da Computação

Brasília, 8 de julho de 2019

Dedicatória

Dedico esse trabalho a todas pessoas que sempre me apoiaram e acreditaram em mim.

Agradecimentos

Agradeço a todos que me ajudaram a chegar até aqui.

Resumo

Os recursos necessários para armazenar e transmitir imagens são imensos, o que torna a sua compressão necessária. Todos os esforços feitos em algoritmos de compressão de imagens clássicos abordam o problema de compressão de um ponto de vista empírico: humanos desenvolvem várias heurísticas para reduzir a quantidade de informação necessária para representar a imagem explorando imperfeições no sistema visual humano, de modo que seja possível reconstruí-la sem muita perda de informação. Compressão de imagens usando redes neurais tem sido uma área ativa de pesquisa em tempos recentes com vários desafios a serem enfrentados para que essas técnicas sejam competitivas com os codificadores clássicos.

O objetivo do presente trabalho é estudar e explorar soluções para o desafio de comprimir imagens. Para isso, foi analisado o método de compressão clássico *JPEG* e métodos usando *autoencoders* convolucionais. Esses métodos foram testados em bases de dados comumente usadas para este tipo de problema.

Palavras-chave: codificação de imagens, redes neurais, aprendizado profundo, processamento de imagens, aprendizado de máquina, processamento de sinais

Abstract

The resources required to store and transmit images are huge, making their compression essential. All the efforts made in classical image compression algorithms address the problem from an empiric point of view: experts develop several heuristics to reduce the amount of information needed to represent the image by exploiting imperfections in the human visual system. This way, it is possible to reconstruct it without much perceptible information loss. The success of deep convolutional neural networks (CNNs) in computer vision application has been inspiring researchers from the image compression community to try to develop algorithms that learn from data, rather than relying on expert knowledge. So far, these algorithms have not lead to an unquestionable improvement over classical codecs. The objective of the present work is to study and explore CNN solutions to the challenge of compressing images, aiming to propose a method that outperforms classical codecs. To achieve this goal, the classical compression method JPEG is evaluated in databases commonly used for this type of problem. Next, some preliminary experiments on encoder-decoder CNNs have been performed.

Keywords: image coding, neural networks, deep learning, image processing, machine learning, signal processing

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

AE Autoencoder.

CLIC Challenge on Learned Image Compression.

CNN Convolutional Neural Network.

dB Decibéis.

DC Direct Current Coefficient.

DCT Discrete cosine transform.

DIV2K Diverse 2k high resolutional quality images.

EYE UDH and HD images Eye tracking dataset.

FDCT Forward Discrete cosine transform.

IDCT Inverse Discrete cosine transform.

JFIF JPEG File Interchange Format.

MS-SSIM Multi-Scale Structural Similarity Index.

MSE Mean squared error.

PNG Portable Network Graphics.

PSNR Peak Signal-to-Noise Ratio.

ReLU Rectified Linear Unit.

SGD Stochastic Gradient Descent.

SSIM Structural Similarity Index.

XOR Exlusive OR.

Capítulo 1

Introdução

Codificação e compressão de imagens é um grande desafio no campo de processamento de imagens. Para entender melhor a complexidade e a necessidade de se resolver esse desafio, este capítulo apresenta uma motivação do tema na seção 1.1; na seção 1.2 é apresentada a hipótese; os objetivos gerais e específicos são propostos na seção 1.3 e resultados esperados na seção 1.4.

1.1 Motivação

A codificação de dados é a transformação feita nos dados para atingir um certo objetivo, como compressão ou criptografia. O principal objetivo dos algoritmos de compressão é a redução do comprimento da mensagem (codificação da fonte), enquanto a criptografia tem como foco transformar os dados para proteger sigilo ou integridade daquilo que eles significam, e/ou acesso a tais dados, durante a sua transmissão através de um canal vulnerável.

Compressão de dados é o processo de codificar uma determinada informação utilizando uma menor representação. Os dois principais benefícios trazidos pela compressão de dados são o aumento significativo na capacidade de armazenamento de um sistema e menor largura de banda necessária para transmití-los. De forma sucinta, compressão de dados é arte ou ciência de representar informação de forma compacta [13]. Nós criamos essas representações compactas identificando e usando estruturas que existem nos dados para que seja possível extrair redundância dos dados e descrevê-la em forma de um modelo que será usado como base para a codificação [13].

Existem dois tipos de compressão: com perdas e sem perdas. A compressão com perdas (*lossy*) potencializa uma melhor taxa de compressão em troca de perda de informação enquanto na compressão sem perdas (*lossless*) não há perda de informação. Esta última é requerida em algumas aplicações, como sinais biomédicos. No contexto de imagens

digitais, a compressão sem perdas permite que, após a codificação da imagem, a imagem decodificada seja idêntica à original, enquanto na compressão com perdas a imagem decodificada não é idêntica à original e há perda de qualidade visual.

O motivo pelo qual precisamos de usar compressão de dados é porque estamos gerando e usando cada vez mais dados digitais. O número de *bytes* necessários para representados dados multimídia pode ser enorme. Por exemplo, para representar digitalmente 1 segundo de vídeo sem compressão usando o formato CCIR 601 [13], é necessário mais do que 20 *megabytes* para armazenar ou 160 megabits para transmitir [13]. Considerando o número de segundos em um filme, é fácil ver porque compressão é necessárias à determinadas aplicações. Para serviços de streaming de mídia como Netflix, não usar compressão não é uma opção.

Os recursos necessários para armazenar e transmitir imagens são imensos, o que torna a sua compressão necessária. O objetivo em codificar uma imagem é representá-la com o menor número possível de bits, preservando a qualidade e a inteligibilidade necessárias à sua aplicação de modo a facilitar sua transmissão e armazenamento. São utilizadas medidas de desempenho para a codificação sem perdas e com perdas que diz respeito a taxa de compressão e distorção. Uma das formas de medir distorção comumente utilizada em processamento de imagens é o MSE[1]:

$$\frac{1}{n}\sum_{n=1}^{N}\left(x(n)-\hat{x}(n)\right)^2, \text{ onde } x \text{ representa a imagem original e } \hat{x} \text{ a imagem decodificada} \tag{1.1}$$

Algoritmos de compressão de imagens aproveitam da percepção visual e propriedades estatísticas de dados da imagem para fornecer resultados superiores quando comparados com métodos de compressão de dados genéricos, que são usados para outros dados digitais. A tarefa de compressão de imagens foi cuidadosamente examinada durante anos por pesquisadores e times como o *Joint Pictures Experts Group* que desenvolveram os métodos de compressão de imagens JPEG [1] e JPEG2000 [14]. Mais recentemente, o algoritmo WebP [15] foi proposto para melhorar as taxas de compressão em imagens de alta resolução, que vem sendo cada vez mais utilizadas. O codec (codificador e decodificador) do estado da arte atual é o BPG [16].

Assim como os outros codecs, o *JPEG* explora as características imperfeitas da nossa percepção. Ele foi o primeiro padrão internacional de compressão para imagens monocromáticas e coloridas. Até hoje é um padrão bastante utilizado e possui métodos para compressão com perdas (método baseado em transformada discreta de cosseno) e sem perdas (método preditivo). Para criar um arquivo JPEG, primeiro a imagem é conver-

[1]MSE também é bastante utilizada como função de loss para modelos de aprendizado profundo baseados em redes neurais

tida para outro espaço de cor: o *YCbCr*. Este espaço, que é usado em vários vídeos de alta definição, codifica a cor de uma forma diferente do RGB, apesar de cobrir as mesmas cores. Os componentes Cb e Cr (crominância azul e vermelha, respectivamente) são altamente compressíveis, enquanto o componente de luminância indica quão brilhante o pixel é. Na superfície da retina existem dois tipos de células que contêm pigmentos: os cones e os bastonetes. Os bastonetes existem em maior quantidade na periferia da retina e são estimulados com luz de baixa intensidade. Eles servem para dar um quadro geral do campo de visão e não estão envolvidos com a visão colorida (em baixos níveis de luz, nós praticamente não vemos cor, pois a iluminação é muito baixa para estimular os cones da nossa retina). Os cones, por sua vez, são muito sensíveis às cores e ocorrem principalmente na região central da retina. Seu estímulo depende de altas intensidades luminosas. É nessa região que a imagem é formada com maior nitidez, pois são estimulados pela luz mais intensa. Os cones são especializados na acuidade da visão diurna e em reconhecer a cor. O cérebro interpreta os sinais recebidos por esses cones, o que permite processar a diferenciação das cores. O olho humano é capaz de discriminar o brilho de uma imagem muito mais que sua informação de cor, visto que existem cerca de 120 milhões de bastonetes distribuídos sobre a superfície da retina contra apenas 6 à 7 milhões de cones[2]. Isto significa que os valores dos componentes de luminância precisam de muito mais fidelidade que os componentes de crominância (o mesmo vale para o componente de cor verde no espaço RGB, por exemplo).

Os algoritmos de compressão existentes atualmente podem estar longe de serem os ideais para os novos formatos de mídia como vídeos em 360 graus ou conteúdos de realidade virtual. Enquanto um desenvolvimento de um novo codec pode levar anos, um *framework* de compressão de imagens mais geral baseado em redes neurais pode ser capaz de se adaptar muito mais rápido à estas diferentes tarefas e ambientes.

Algoritmos padrão de compressão de imagens tendem a fazer suposições sobre a escala da imagem. Por exemplo, usualmente assume-se que um *patch* (pedaço retangular da imagem) de uma imagem natural de alta resolução irá conter muita informação redundante. De fato, quanto maior a resolução da imagem, mais provável que a maior parte dos *patches* que a compõem conterão informação de baixa frequência onde não há muita variação nos valores dos pixels. Esse fato é explorado pela maior parte dos codecs de imagens, de modo que eles tendem a ser muito eficientes em comprimir imagens de alta resolução. Entretanto, tais suposições são invalidadas ao criar miniaturas de imagens naturais de alta resolução, visto que um *patch* obtido de uma miniatura provavelmente conterá informação de alta frequência que é mais difícil de ser comprimida.

[2]Esta diferença no número de bastonetes se dá por motivos evolutivos pois era mais importante identificar possíveis predadores ou presas durante a noite do que identificar cor

Compressão em larga escala de miniaturas (imagens 32x32, por exemplo) é uma aplicação importante, tanto em termos de redução de armazenamento em disco quanto em fazer melhor uso da banda limitada da internet. Várias miniaturas são atualmente transmitidas ao longo da *web* para prévias de *sites*, galerias de fotos, ferramentas de busca e várias outras aplicações. Uma grande fração do tráfego da internet é agora dirigido por requisições de dispositivos móveis com telas relativamente pequenas e requerimentos de largura de banda rigorosos. Portanto, aumentar a compressão de miniaturas além das capacidades dos codificadores existentes é de grande relevância, visto que economizar *bytes* reduz custos, melhoram a experiência dos usuários e possibilitam diversas aplicações.

1.2 Hipótese

Nos últimos anos, redes neurais profundas se tornaram a base dos resultados do estado da arte para reconhecimento de imagens [17], detecção de objetos [18], reconstrução tridimensional de objetos [19], reconhecimento de faces [20], reconhecimento de discurso [21], *machine translation* [22], geração de legendas de imagens [23], tecnologia de carros autônomos [24], entre outros.

É natural buscar usar essa poderosa classe de métodos para melhorar a tarefa de compressão de imagens, especialmente para tamanhos de imagens que não são cuidadosamente desenvolvidas para codecs otimizados por heurísticas, como miniaturas. Considerando um codificador de imagem como um problema de análise/síntese com uma camada de gargalo no meio, é possível encontrar uma grande área de pesquisa que usa redes neurais para encontrar representações comprimidas. Muito desse trabalho se concentra em uma classe de redes neurais conhecida como *autoencoders* [25]. Alguns resultados já existentes para compressão com perdas usando autoencoders se mostraram promissores: [26, 27, 28], e redes neurais já atingiram o estado da arte para compressão sem perdas [29, 30].

Compressão de imagens tem sido uma área ativa de pesquisa em tempos recentes com vários desafios a serem enfrentados para que essas técnicas sejam competitivas com os codificadores clássicos, portanto deseja-se verificar se modelos baseados em *autoencoders* convolucionais são competitivos com o clássico codec *JPEG*.

1.3 Objetivos

Deseja-se propor um *autoencoder* convolucional que estenda a estrutura básica de um autoencoder, gerando uma representação binária para a imagem ao quantizar a camada de gargalo ou as variáveis latentes correspondentes.

Para isto, serão estudadas propostas de compressão de imagens na literatura, de modo a obter estatísticas e informações para guiar a escolha e implementação de codificadores baseados em redes neurais. Serão construídas bases de dados próprias, a partir de bases de dados comumente utilizadas para esse problema, para que seja avaliado o desempenho em vários cenários de *autoencoders* convolucionais e do codec *JPEG* em imagens de baixa resolução espacial, mais especificamente *patches* com 32 pixels de largura e altura.

1.4 Resultados Esperados

Visto que já existem trabalhos semelhantes na literatura, acredita-se que seja possível obter desempenho similar ao método de compressão *JPEG* em imagens com 32 pixels de largura e 32 de altura.

Capítulo 2

Trabalhos similares e conceitos fundamentais

Este capítulo descreve trabalhos similares e conceitos fundamentais de codecs clássicos e arquiteturas baseadas em redes neurais utilizadas neste trabalho.

2.1 JPEG

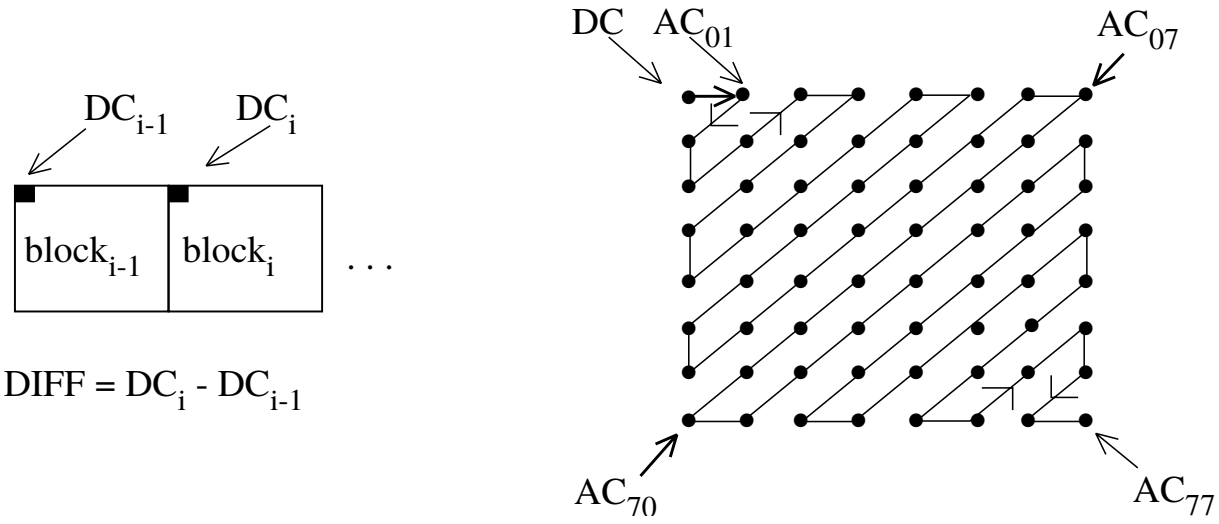

Figura 2.1: Diagram do método de compressão JPEG. Fonte: [1].

Arquivos JPEG normalmente são descritos no formato JFIF [31], que é uma limitação do padrão JPEG completo. O método de compressão JPEG descrito na Figura 2.1, assim como os outros métodos, exploram a imperfeição do sistema visual humano. Os 5 passos usados para codificação no padrão JFIF são descritos nas subseções seguintes.

2.1.1 Conversão do espaço de cor

Para o padrão JFIF[1], primeiro, é feita a conversão do espaço RGB da imagem de entrada para o espaço de cor *YCbCr*. Os valores dos pixels estarão no intervalo de 0 à 255 (mesmo do RGB). Uma vez que o espaço de cor é transformado para *YCbCr*, é necessário decidir qual será o fator usado para reduzir a quantidade de pixels nos componentes de crominância, visto que o olho humano é muito mais sensível ao brilho do que a cor. Normalmente é usado um fator de 2 nas duas direções, o que dá 4 vezes menos cor (para cada 4 pixels Y só terá 1 pixel Cb e 1 Cr). Esse fator é determinado pelo argumento *quality* passado como parâmetro para codificação da imagem (com *quality* máxima, não haverá redução e a imagem possuirá a mesma resolução de cor).

Nota-se que na Figura 2.2 praticamente não há diferença visual olhando a um nível normal de zoom, mesmo a codificação da imagem da direita possuindo 64 vezes menos cor do que a imagem da esquerda. Entretanto, olhando a Figura 2.3 é possível notar certa discrepância nas bordas da arara vermelha.

Figura 2.2: Imagem original sem compressão retirada de [2] e imagem (direita), gerada a partir da imagem original, com dimensionalidade nos canais de crominância reduzida por um fator de 8 nas duas direções. Pode-se jogar informação da imagem original fora e então usar a imagem da direita para armazenamento ou transmissão, que terá novos valores em seus canais de cores (aumenta-se a dimensionalidade apenas quando for necessário exibí-la). Fonte: [2].

2.1.2 Aplicação da transformada direta de cossenos

Aqui é feita a divisão da imagem em blocos com 8 pixels de largura e 8 de altura que serão convertidos em uma nova matriz com o auxílio de uma transformada discreta de cossenos

[1]JPEG permite você usar qualquer espaço de cor que queira, mas a maior parte das pessoas seguiram com o padrão JFIF por praticidade

Figura 2.3: Versão com zoom das imagens mostradas na Figura 2.2. A região mostrada é exatamente a mesma (espacialmente) para as duas imagens, com a mesma quantidade de pixels. Imagem original sem compressão (esquerda) e imagem com 64 vezes menos cor (direita).

DCT [32] (*JPEG* sempre usa *DCT-II*). Essa transformação, que é similar a transformada de *Fourier*, analisa as frequências dos valores originais dos pixels da imagem ao longo de cada linha e coluna usando um conjunto de ondas cossenos oscilando em diferentes frequências e amplitudes e tentamos representar a imagem usando essas ondas. Cada um dos blocos serão codificados separadamente com sua própria transformada discreta de cossenos e podem ser exatamente replicados por ondas de cossenos 8 por 8, onde varia-se frequências e amplitudes de cada uma delas.

A Figura 2.4 mostra as 64 funções de cosseno que podem ser combinadas para formar qualquer imagem 8 por 8. Nota-se que a partir do bloco superior esquerdo, as frequências das ondas de cosseno crescem tanto na direção horizontal quanto na vertical. Além disso, o bloco inferior direito constituído de um padrão de xadrez, é o que possui maior frequência. Para criar qualquer imagem 8 por 8, basta combinar todos esses blocos ao mesmo tempo. Cada um será ponderado baseado em um número denonimado coeficiente que representará a contribuição de cada um desses blocos individuais para o todo. Assim, se a contribuição de um bloco for zero não haverá nenhuma parte desta função de cossenos na imagem 8 por 8 buscada. Basicamente, na transformada discreta de cossenos é calculado os coeficientes das ondas de cossenos. Os coeficientes podem ser considerados como a quantidade relativa das frequências espaciais 2D contidas no sinal de entrada. O coeficiente com frequência zero nas duas dimensões é chamado de coeficiente corrente direto (DC) e os 63 restantes são chamados de coeficientes correntes alternados (*AC*). O decodificador reverte este passo usando a função inversa da DCT (IDCT) que pega os 64 coeficientes *Forward DCT* (FDCT) do codificador já quantizados e reconstrói o sinal da imagem de 64 pontos somando os sinais base. Se a FDCT e a IDCT pudessem ser computadas com acurácia perfeita e se os coeficientes da DCT não fossem quantizados no codificador, o sinal original

de 64 pontos poderia ser exatamente recuperado.

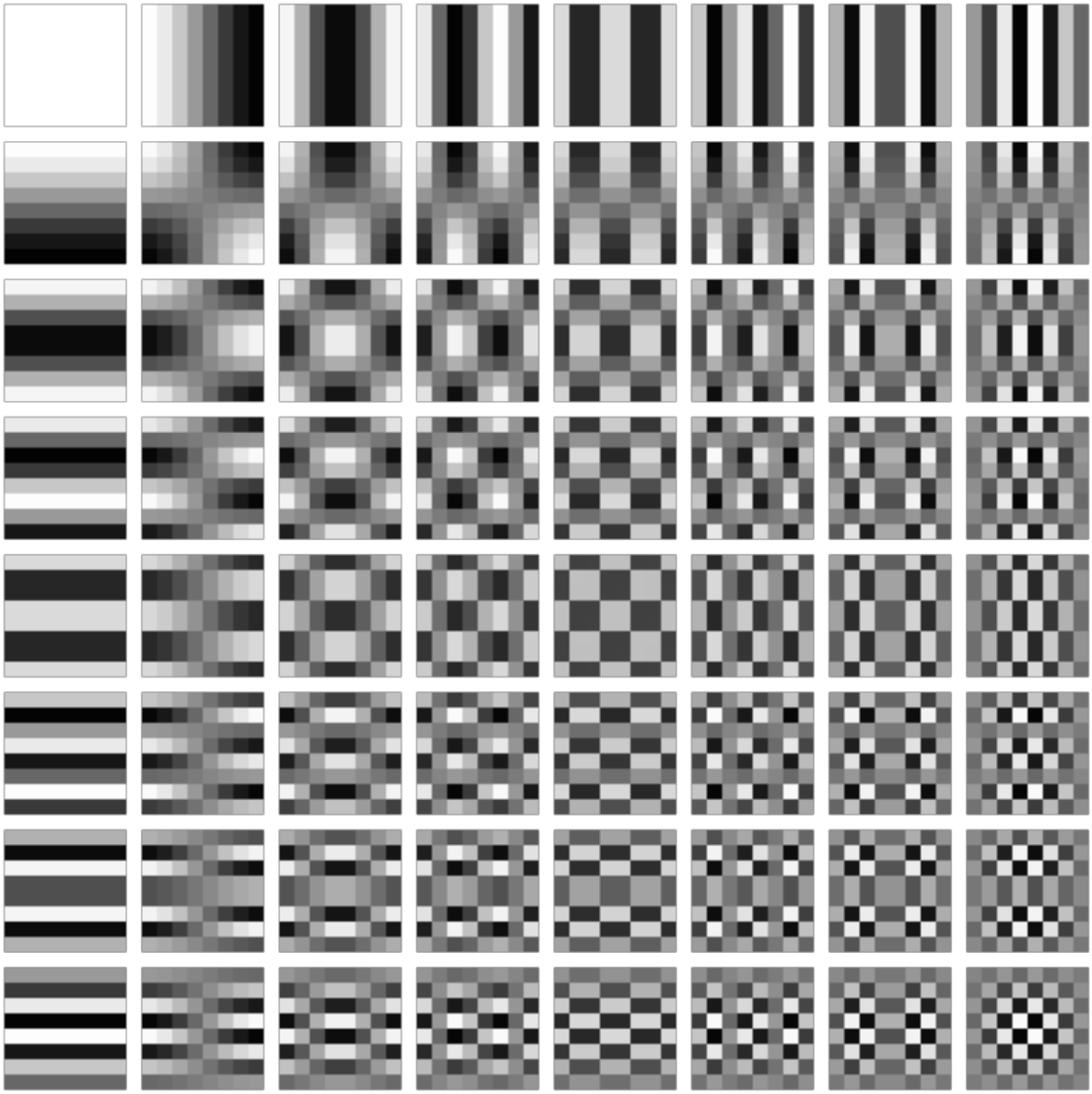

Figura 2.4: 64 (8 por 8) ondas base de cossenos com frequências variadas. Fonte: [3].

Mudanças de altas frequências podem ser minimizadas ou zeradas, visto que nós não percebemos suas mudanças na imagem tão bem quanto componentes de baixas frequência. Ou seja, blocos de imagens cujos valores de pixels mudam de intensidade muito rápido (normalmente pertos das bordas da imagem) podem ser borrados sem perda significativas de qualidade visual, o que economiza uma quantidade enorme de espaço. Por isso, os coeficientes das ondas de cossenos de alta frequência não contribuem muito para a imagem final. Entretanto, isso não é verdade para textos, o que faz com que o JPEG não seja

uma boa escolha quando o objetivo é comprimir imagens de texto, conforme mostra a Figura 2.5.

Figura 2.5: Comparação imagem de texto com e sem compressão. Fonte: [4].

2.1.3 Quantização

Quantização é definida dividindo cada coeficiente pelo passo do quantizador especificado na tabela de quantização, seguido por um arredondamento para o inteiro mais próximo. Na dequantização é feito o processo inverso multiplicando pelo passo do quantizador, com o auxílio da mesma tabela usada para quantização. Quantização é o passo em que há a maior perda de informação na imagem. É dada pela seguinte equação:

$$\left[F^Q(u,v) = \frac{F(u,v)}{Q(u,v)}\right] \tag{2.1}$$

$F^Q(u,v)$ será o novo valor do coeficiente, $F(u,v)$ é o valor atual e $Q(u,v)$ é o passo de quantização definido pela aplicação (altas frequências são removidas usando um valor maior para $Q(u,v)$. Cada um dos 64 coeficientes DCT são uniformemente quantizados em conjunto com uma tabela de quantização de 64 elementos, que é definida pelo nível de qualidade escolhido para a aplicação. O propósito da quantização é alcançar mais compressão ao representar os coeficientes DCT com a menor precisão possível para alcançar a qualidade da imagem especificada. Isso é feito descartando informação que não é visualmente significante.

Após a quantização, os coeficientes DC são tratados separadamente devido à alta correlação destes coeficientes em blocos 8 por 8 adjacentes da imagem, considerando que eles geralmente possuem maior valor e muito impacto na imagem. Assim, eles são codificados como a diferença do coeficiente DC do bloco anterior na ordem de codificação. Por fim, todos os coeficientes quantizados são ordenados em uma sequência zig-zag conforme mostra a Figura 2.6 com o objetivo de facilitar a codificação (que será usada no próximo passo) ao ordenar os coeficientes de frequência similares próximos uns dos outros.

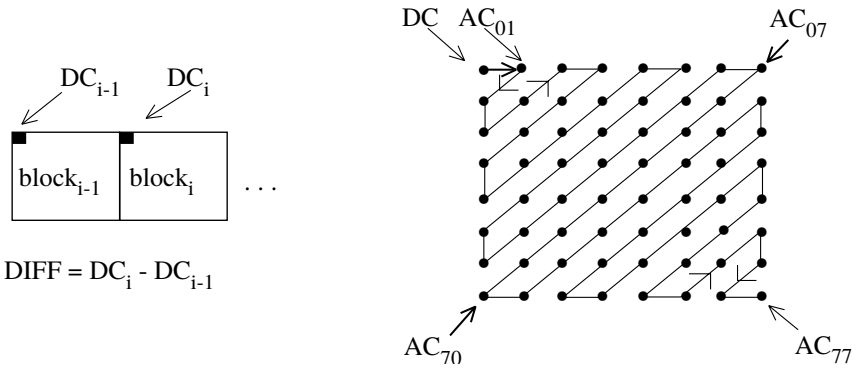

Figura 2.6: Sequência zig-zague usada para melhorar codificação. Fonte: [1].

2.1.4 Codificação

O passo final do codificador é a codificação de entropia. Este passo permite compressão sem perdas adicional ao codificar os coeficientes DCT quantizados de forma mais compacta baseando-se em suas características estatísticas. O *JPEG* propõe o uso de dois métodos de codificação de entropia: Codificador de Huffman [33] e Codificador Aritmético [34]. No final, para cada bloco 8 por 8 da imagem original, teremos três matrizes 8 por 8 quantizadas, onde as matrizes correspondentes aos canais *Cb* e *Cr* serão as mais comprimidas.

Terminada a codificação, o papel do decodificador será de reverter os passos do codificador para reconstruir a imagem, conforme mostra a Figura 2.1: primeiro, a matriz quantizada será obtida decodificando os blocos comprimidos. Depois, é possível obter a matriz DCT multiplicando a matriz quantizada pela matriz de quantização utilizada pelo codificador. Depois, essa matriz é transformada usando a IDCT que resultará na matriz no espaço de cor *YCbCr*.

2.2 Redes Neurais

Um algoritmo simples de aprendizado de máquina (*machine learning*) chamado regressão logística pode determinar quando recomendar cesariana para uma paciente [35]. O desempenho desses algoritmos dependem muito da representação dos dados fornecidos. Cada pedaço de informação incluída na representação dos dados é conhecida como *feature* (característica) [5]. Algumas tarefas em inteligência artificial podem ser resolvidas desenvolvendo características a serem extraídas dos dados. Entretanto, para muitas tarefas é difícil saber quais *features* devem ser extraídas. Uma solução para esse problema é descobrir não apenas a função que mapeará a entrada para a saída mas também a própria representação. Essa abordagem é conhecida como *representation learning* (aprendizado de representações) [5].

Deep learning (aprendizado profundo) resolve o problema de *representation learning* introduzindo representações que são expressadas em termos de outras representações mais simples [5]. Por exemplo, usando um modelo de *deep learning* é possível representar o conceito de uma imagem de um carro combinando conceitos mais simples, como bordas e contornos.

Redes neurais são modelos de *deep learning* capazes de fazer predições aprendendo uma função que relaciona as características dos dados à respostas observadas/desejadas. Redes neurais são consideradas aproximadores universais de funções, o que significa que elas podem computar e são capazes de aproximar qualquer função (não só lineares) [36]. Para isso, é necessário que elas sejam profundas o suficiente e possuam funções de ativações (funções não-lineares), visto que a saída de uma rede sem funções de ativação seria apenas uma função linear (polinômio de grau um) que não é capaz de representar algumas funções como a função XOR [Figura 2.7]. As ativações permitem que o modelo aprenda funções mais complexas, ao introduzir transformações não-lineares nas saídas das camadas. A *Rectified Linear Unit* (ReLU), definida como $f(x) = max(0, x)$, é uma das funções de ativações não-lineares mais comuns e recomendadas pois ela é quase linear, o que faz com que o modelo seja fácil de otimizar com métodos comumente usados como descida de gradiente [37] (método que atualiza os pesos com base no gradiente, de modo que a função de erro será minimizada dando passos proporcional ao negativo do gradiente em direção ao ponto mínimo) e preserva várias propriedades existentes em modelos lineares que permitem que eles generalizem bem [5].

2.2.1 Taxa de aprendizagem (*learning rate*)

A *learning rate* é um hiperparâmetro que controla o quanto nós ajustamos os parâmetros aprendíveis da nossa rede com respeito ao gradiente. Quanto menor o valor, menor o

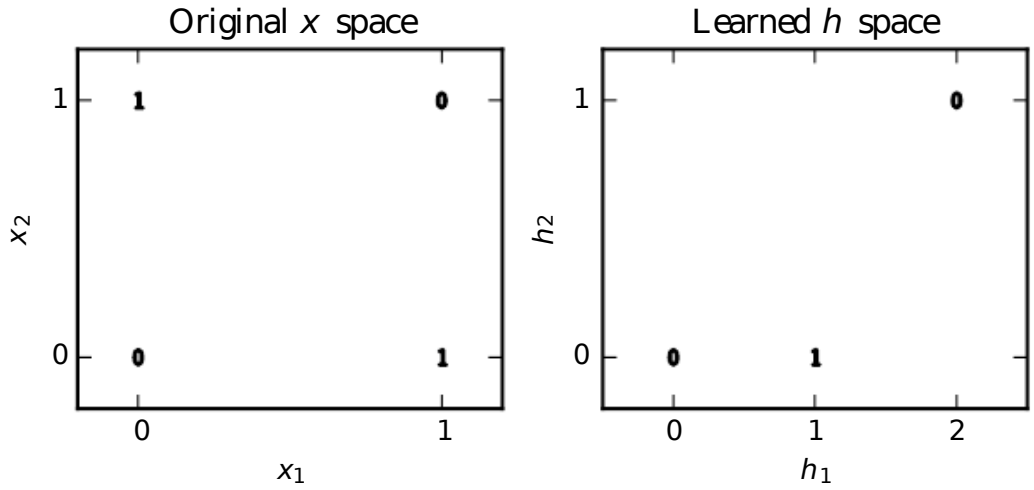

Figura 2.7: Um modelo linear aplicado diretamente à entrada original não pode implementar a função XOR. Para isso é necessário transformar o espaço original usando uma função de ativação. Fonte: [5].

passo dado ao longo do declive em direção ao mínimo da função de custo. A *learning rate* é um dos hiperparâmetros que devem ser escolhidos com cuidado, pois ela pode ter uma grande influência na convergência do seu modelo. O gráfico Figura 2.2 mostra os diferentes cenários de *learning rate* e como ela afeta o treinamento. Leslie N. Smith

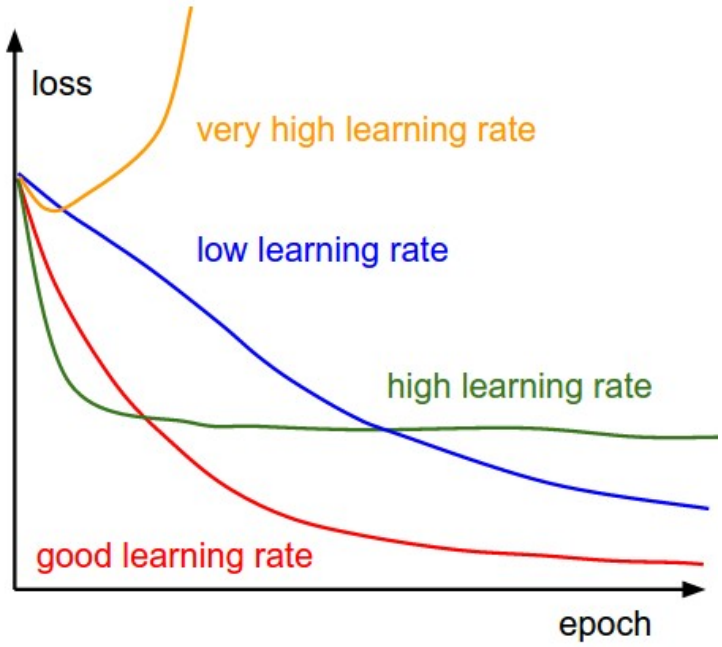

Figura 2.8: Efeitos de várias taxas de aprendizagem no treinamento. Fonte: [6]).

argumenta em [7] que é benéfico para a aprendizagem variar a *learning rate* de forma cíclica durante o treinamento para evitar cair em mínimos locais não ótimos (pontos de sela). Também é mostrado que é possível atingir resultados iguais ou superiores com menos iterações de treinamento usando este método quando comparado com uma rede que foi treinada com *learning rate* fixa ou usando outro método padrão de variação (este fênomeno ficou conhecido como "superconvergência").

Leslie propõe ciclos para variar a *learning rate*. Um ciclo é definido como o número de iterações necessárias para *learning rate* ir do valor mínimo até o máximo definido no ciclo e voltar ao mínimo. Dadas as constantes $baselr$, $maxlr$, $step$ e γ que representam, respectivamente, *learning rate* inicial, *learning rate* máxima, número de iterações correspondente à metade de um ciclo e constante responsável por diminuir limite superior do ciclo; e as variáveis itr que representa a iteração atual no treinamento e $cycle = \left\lfloor 1 + \frac{itr}{2 \cdot step} \right\rfloor$ o ciclo atual, a *learning rate* lr para uma itr qualquer na política *exp_range* [Figura 2.9], é calculada pela seguinte equação:

$$lr = baselr + (maxlr - baselr)max(0,\, 1 - |itr/step - 2cycle + 1|)\gamma^{itr}. \tag{2.2}$$

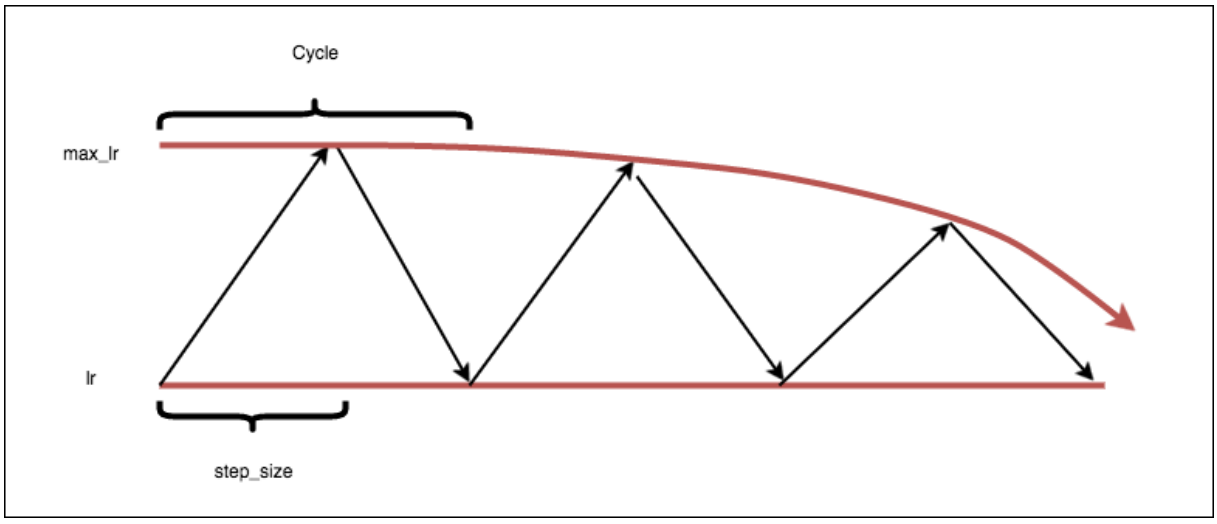

Figura 2.9: Política *exp_range* de *learning rate* cíclica. Fonte: [7].

O objetivo do treinamento das redes neurais é minimizar a função de custo/erro (*loss*) definida para a aplicação. Pode-se usar um conjunto de validação para salvar e usar os pesos do modelo que obter a melhor métrica no conjunto de validação, pois o modelo que obtiver a menor função de erro no treinamento não necessariamente será aquele que obterá a melhor métrica no conjunto de teste usado para avaliação do modelo. Para que

este objetivo seja atingido, normalmente são usados otimizadores. O método clássico de descida do gradiente estocástico (SGD) consiste basicamente em usar amostras aleatórias (*mini-batch* para aproximar o verdadeiro gradiente que levará à minimização da função de custo/erro escolhida. O SGD mantém uma única *learning rate* (não muda durante o treino) para todas as atualizações de peso. O *Adam* [38] é um otimizador que adapta a *learning rate* baseando-se na média do primeiro momento e do segundo momento dos gradientes e na média móvel exponencial do gradiente e da raiz quadrada dele. Os parâmetros utilizados neste otimizador controlam as taxas de decaimento destas médias móveis.

2.2.2 Autoencoders

Um Autoencoder (AE) é uma classe de redes neurais que são formados por duas redes conectadas: um **encoder** e um **decoder**.

- O **encoder** tem como função converter a informação da entrada em uma representação menor e mais densa chamada de espaço latente. Pode ser representado como uma função de x, $f(x) = h$.
- O **decoder**, por sua vez, tenta reconstruir a informação original, passando do espaço latente criado pelo encoder para o espaço original da informação. Pode ser representado como uma função de h, $g(h) = \hat{x}$

Uma rede do tipo AE é treinada de forma não supervisionada e pode ser descrita como $g(f(x)) = \hat{x}$. Normalmente, o objetivo é apenas diminuir a diferença entre x e $\hat{x}$ (nesse caso, a função de custo a ser minimizada normalmente é a $MSE(x, \hat{x})$ [Equação 1.1]). A camada entre o *encoder* e o *decoder* que contém menos neurônios [Figura 2.10] e força o *encoder* a comprimir informação da representação original da entrada original gerando o espaço latente, denominado de *bottleneck* (camada de gargalo). O problema fundamental com autoencoders como modelo gerativo é que o espaço latente para qual as entradas são convertidas pode não ser contínuo ou permitem fácil interpolação [Figura 2.11].

Para o problema de compressão de imagens, normalmente usa-se um binarizador na camada de gargalo com o objetivo de binarizar o latente gerado pelo *encoder*. Assim, o *encoder* é forçado a comprimir informação e o *decoder* a diminuir a distorção usando menos informação. O binarizador transforma os valores em ponto flutuante (representação limitada dos números reais no computador) em inteiros que serão binarizados. Ele é utilizado para reduzir o espaço consumido pela imagem codificada, visto que números em pontos flutuante com precisão simples ocupam 32 bits o que levaria a uma alta taxa de bits por pixel. A avaliação dos modelos usados para este tipo de problema é dada considerando não só a taxa, mas também métricas visuais que calculam o nível de distorção da imagem reconstruída. Existem três tipos de métricas visuais comumente utilizadas:

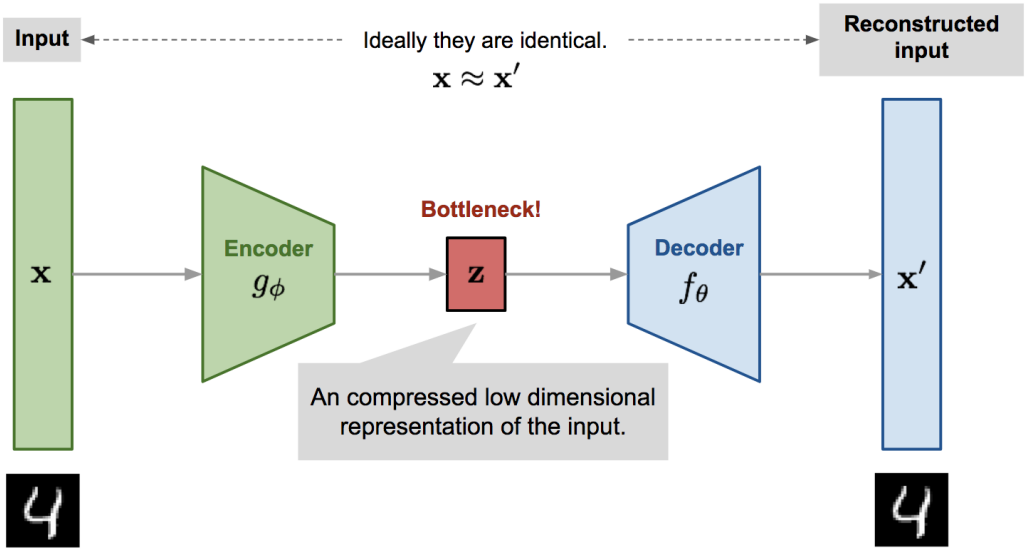

Figura 2.10: Ilustração de um *autoencoder* retirado de [8].

1. *PSNR* definida por

$$20 \cdot \left(\frac{MAX_I}{\sqrt{MSE}} \right), \tag{2.3}$$

onde MAX indica o maior valor possível para o pixel em uma imagem. Quando estes são representados em bits, usa-se $MAX_I = 2^B - 1$;

2. *SSIM* [39]. Seja $x = \{x_i | i = 1, 2, \ldots, N\}$ e $y = \{y_i | i = 1, 2, \ldots, N\}$ dois sinais discretos não negativos e μ_x, σ_x^2 e σ_{xy} serem a média de x, a variância de x e a covariância de x e y, respectivamente. A SSIM é dada por:

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\mu_x\mu_y + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\mu_x^2 + \mu_y^2 + C_2)}, \tag{2.4}$$

onde $C_1 = (K_1 L)^2$, $C_2 = (K_2 L)^2$ e $C_3 = C_2/2$. L é o intervalo dinâmico dos valores dos pixels ($L = 255$ para 8 bits por pixel) e K_1, K_2 são constantes.

3. *MS-SSIM* [40] é baseada na SSIM:

$$MSSSIM(x, y) = [l_M(x, y)]^{\alpha M} \cdot \prod_{j=1}^{M} [c_j(x, y)^{\beta_j} [s_j(x, y)]^{\gamma_j}, \tag{2.5}$$

onde os expoentes são usados para ajustar a importância relativa de cada componente e l, c e s são componentes de luminância, contraste e estrutura, respectivamente.

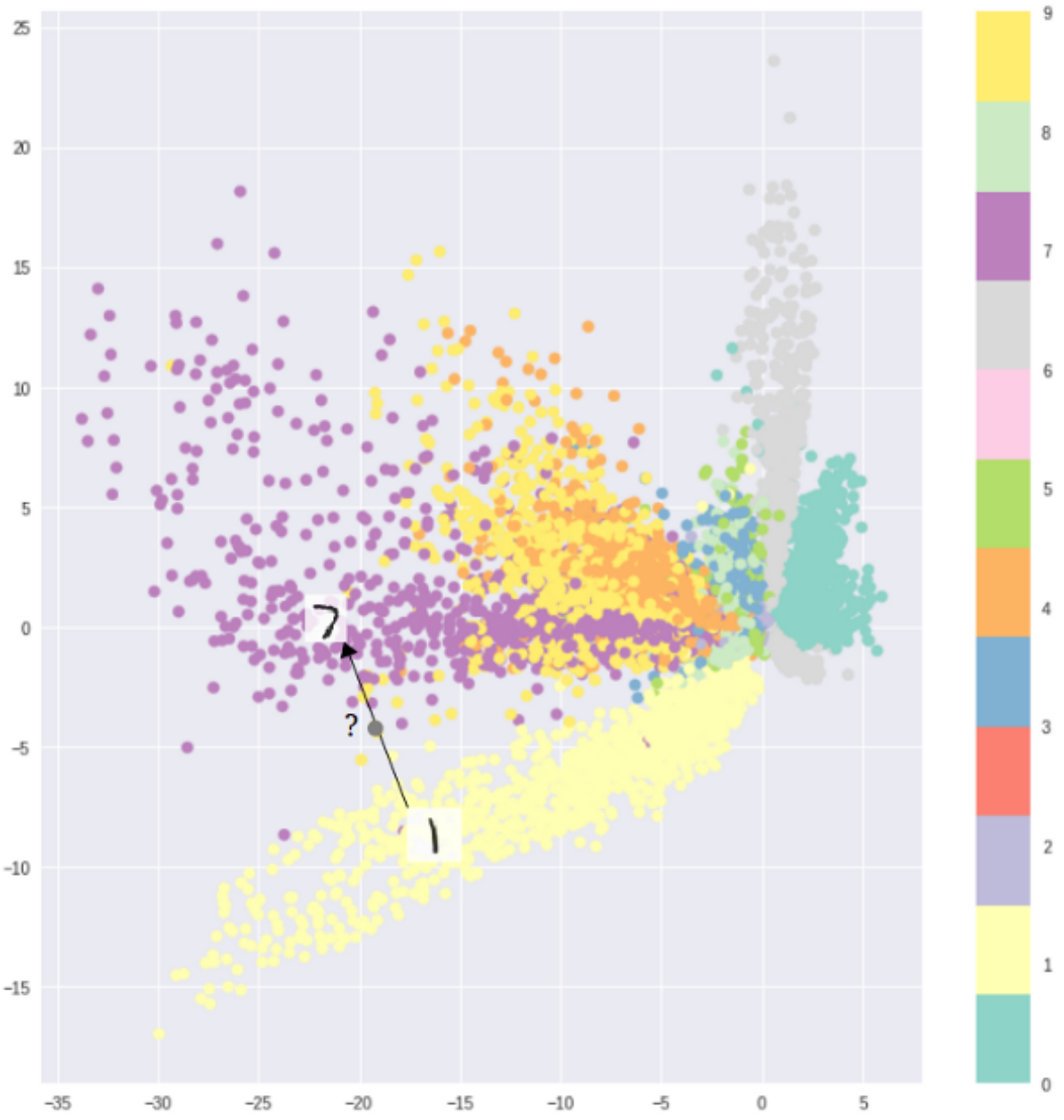

Figura 2.11: Exemplo de espaço latente de um dataset de dígitos de numeros. Nota-se que, devido à descontinuidades, otimizar um AE apenas pelo erro de reconstrução não é bom quando é necessário fazer mais do que apenas replicar a mesma imagem, como gerar novas imagens ou alterações delas. Fonte: [9].

Compressão de Imagens com Taxa Variável usando Redes Neurais Recorrentes

Nesse trabalho [28] é descrito um autoencoder geral baseado em redes neurais para compressão de imagens. São apresentadas três instanciações desta arquitetura: *Long Short*

Term Memory (LSTM) [41], *Fully-Connected* e Convolucional. Para cada arquitetura, existe uma função E que pega o *patch* da imagem como entrada e produz uma representação codificada. Esta representação é passada para uma função de binarização B, que é a mesma para todas redes. B usa tangente hiperbólica (*tanh*) como função de ativação antes de realizar a binarização, de forma a gerar um x tal que $x \in [-1, 1]$. A seguir, é calculado $b(x)$:

$$b(x) = x + \epsilon \in \{-1, 1\}, \tag{2.6}$$

$$\epsilon \sim \begin{cases} 1 - x \text{ com probabilidade } \frac{1+x}{2}, \\ -1 - x \text{ com probabilidade } \frac{1-x}{2}, \end{cases} \tag{2.7}$$

onde ϵ, corresponde ao ruído de quantização. A função completa de binarização usada na camada de gargalo é:

$$B(x) = b(tanh(W^{bin}x + b^{bin})). \tag{2.8}$$

W^{bin} e b^{bin} são os pesos e bias padrões das camadas anteriores da rede. Esta formulação para a binarização é usada em todos os modelos para a *forward pass* da rede. Para a *backward pass* da *back-propagation*, é usada a derivada da esperança. Visto que a esperança de $b(x)$ será igual a x para todo x, os gradientes serão passados por b sem mudanças. Essa binarização é aplicada somente em tempo de treino. Em tempo de teste, b é substituída por

$$b^{inf}(x) = \begin{cases} -1, & \text{se } x < 0, \\ 1, & \text{caso contrário.} \end{cases} \tag{2.9}$$

Finalmente, para cada arquitetura é considerada um função decodificadora D, que pega a representação binária produzida por B e gera a reconstrução de um *patch*. Esses três componentes formam um *autoencoder*: $\hat{x} = D(B(E(x)))$, que é a base de todos os modelos de compressão apresentados.

No trabalho posterior, Toderici [27] mostrou em seus resultados que o efeito de usar um conjunto de treinamento de alta entropia[2] é benéfico para a rede, dada a importância de treinar modelos de *machine learning* com exemplos difíceis. No seu trabalho, ele definiu o conjunto de alta entropia como sendo o conjunto formado por imagens que difíceis de comprimir pelo PNG (método de compressão sem perdas de imagens), ou seja, os arquivos no formato PNG com o maior número de bytes.

[2]Em processamento de imagens entropia diz respeito à quantidade de informação na imagem. Alta entropia, pode ser visto como maior variância nos valores dos pixels

Compressão de Imagens com Perdas usando Autoencoders Compressivos

Este trabalho [10] alcança performance similar ou superior ao *JPEG2000* quando avaliado na qualidade visual. Entretanto, ao contrário do *JPEG2000* o framework proposto pode ser otimizado para classes de imagens específicas como miniaturas ou imagens não-naturais, métricas arbitrárias e é facilmente generalizável para outras formas de mídia. Essa performance é alcançada usando arquiteturas de redes neurais eficientes que permitem uma decodificação quase em tempo real de grandes imagens mesmo em dispositivos de baixa potência. Para isso, inspirado no trabalho de [42] a maior parte das convoluções são realizadas em um espaço reduzido para aumentar a velocidade da computação. O *upsample* é realizado usando convoluções seguidas por uma remodelação dos coeficientes (subpix na Figura 2.12).

Figura 2.12: Ilustração do autoencoder compressivo usado no paper. A notação $C \times K \times K$ significa convoluções $K \times K$ com C filtros. Fonte: [10].

O autoencoder compressivo possui três componentes: um codificador f, um decodificador g e um modelo probabilístico Q:

$$f : \mathbb{R}^N \to \mathbb{R}^M, \;\; g : \mathbb{R}^M \to \mathbb{R}^N, \;\; Q : \mathbb{Z}^M \to [0, 1]$$

A distribuição de probabilidade discreta, definida por Q é usada para mapear o número de bits à suas representações baseado nas suas frequências, isto é, codificação de entropia. O objetivo é otimizar o balanceamento (*trade-off*) entre usar um número pequeno de bits e ter pouca distorção. A principal fonte de perda de informação é a quantização. A saída quantizada do codificador usada para representar uma imagem é salva sem perdas. Informação adicional pode ser descartada pelo codificador, e o decodificador pode não ser capaz de decodificar a informação disponível de forma perfeita, aumentando a distorção.

Os autores discutem duas funções principais para realizar a quantização: arredondamento estocástico e arredondamento para o inteiro mais próximo.

Para a realização de arredondamento estocástico é usada a seguinte equação:

$$\{y\} \approx \lfloor y \rfloor + \epsilon, \;\; \epsilon \in \{0, 1\}, \;\; P(\epsilon = 1) \; = \; y - \lfloor y \rfloor, \tag{2.10}$$

A derivada da função chão é zero em todos os lugares, exceto nos inteiros, onde ela é indefinida. Por isso, a derivada na passagem de volta (*backward pass*) usada no algoritmo

de backpropagation [37] é substituída com a derivada da esperança:

$$\frac{d}{dy}\{y\} := \frac{d}{dy}E[\{y\}] = \frac{d}{dy} = 1. \tag{2.11}$$

A função de arredondamento também é não-diferenciável, por isso na *backward pass* os gradientes são passados sem modificação do *decoder* para o encoder *encoder*. A quantização é realizada normalmente na *forward pass*, visto que substituir o arredondamento por uma aproximação completamente pode levar o *decoder* a aprender à inverter essa aproximação, removendo a informação da *bottleneck* (camada de gargalo) que força a rede a comprimir informação.

2.2.3 Redes Neurais Convolucionais

Em *deep learning* (aprendizado profundo), uma rede neural convolucional (CNN) [43] é uma classe de redes neurais profundas que usa convoluções para detectarem características e padrões presentes nas imagens. As primeiras camadas detectam características que podem ser reconhecidas e interpretadas de maneira relativamente fácil. Camadas posteriores detectam características mais abstratas e usualmente presentes em muitas das características detectadas por camadas anteriores, conforme mostra a Figura 2.13. A arquitetura de uma CNN é análoga ao padrão de conectividade dos neurônios no cérebro humano e foi inspirada pela organização do córtex visual. Neurônios individuais respondem à estímulos apenas em uma região restrita do campo visual que é conhecida como campo receptivo. Uma coleção de sobreposição desses campos cobrem toda a área visual [44].

Uma CNN é capaz de capturar depêndencias espaciais e temporais na imagem de forma bem-sucedida através da aplicação de filtros relevantes e dispensa a necessidade de engenharia de características. A arquitetura realiza um melhor ajuste ao conjunto de imagens devido à redução do número de parâmetros envolvidos e a reusabilidade dos pesos. Em outras palavras, a rede pode ser treinada para entender melhor a complexidade da imagem e suas características relevantes. Esta extração de características é feita por meio da aplicação de filtros[3] no domínio do espaço denominados convoluções. Para uma máscara (filtro) de tamanho $m \times n$, $m = 2a+1$ e $n = 2b+1$, onde a e b são inteiros positivos. Para qualquer ponto (x, y) na imagem f, a resposta do filtro é a soma dos produtos dos coeficientes do filtro e dos pixels da imagem englobados pelo filtro (Equação 2.12).

$$g(x, y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t) \tag{2.12}$$

[3]Filtro é um termo que vem de processamento no domínio da frequência e se refere a aceitar ou rejeitar certos componentes de frequência

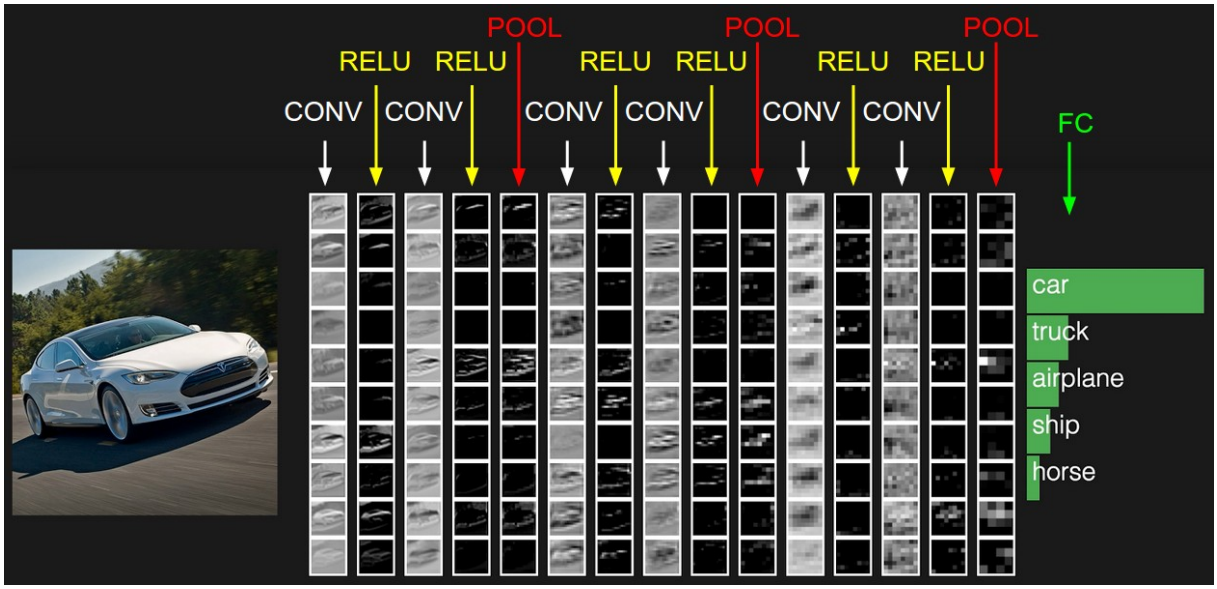

Figura 2.13: Exemplo mostrando a saída de cada camada de uma rede neural convolucional. Fonte: [11].

Conforme mostra a Figura 2.14, x e y variam de modo que cada pixel em w visite todos os pixels em f.

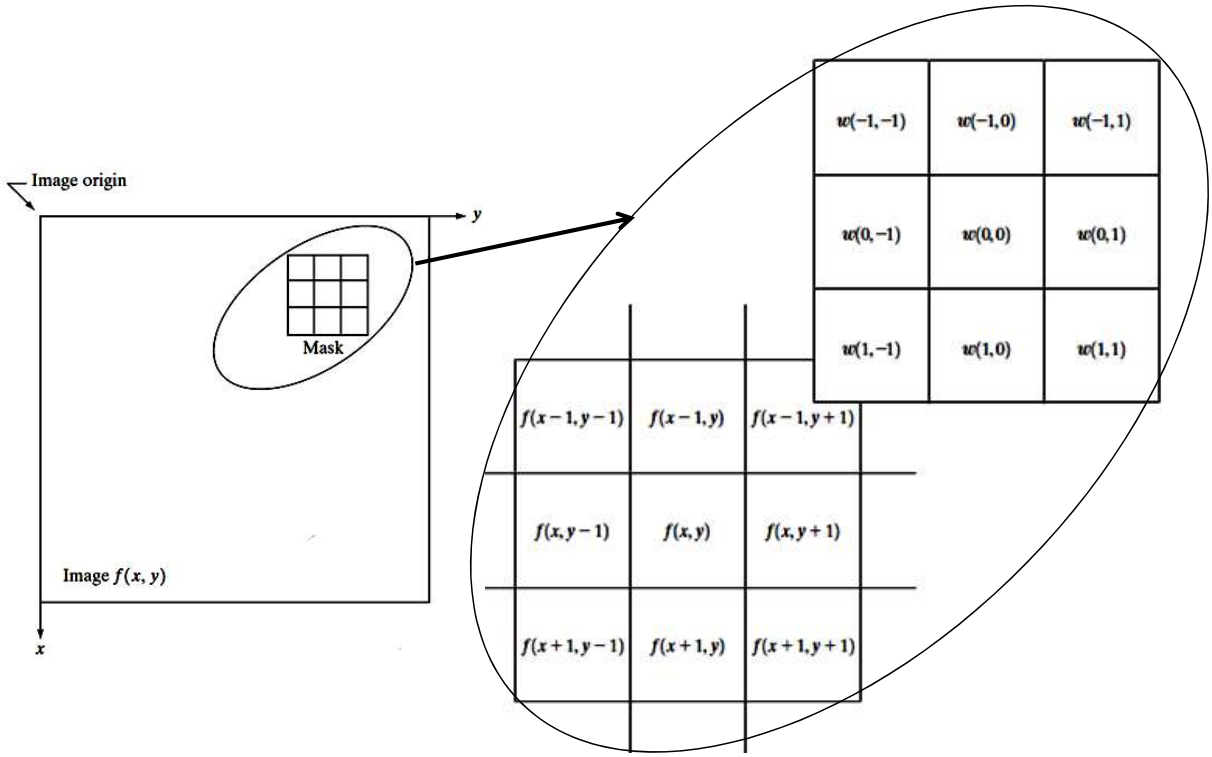

Figura 2.14: Ilustração da operação de filtragem no domínio do espaço (convolução). w é o kernel do filtro e f é a área da imagem coberta pelo filtro. Fonte: [12].

Capítulo 3

Metodologia

Este capítulo apresenta a metologia seguida para a verificação das hipóteses, incluindo o modelo construído e os *datasets* utilizados. Os modelos foram construídos usando o *framework* PyTorch [45] e avaliados usando métricas visuais: PSNR, SSIM e MS-SSIM. Também foi avaliada a quantidade de bits por pixel da imagem decodificada. O objetivo é balancear a taxa e distorção das imagens reconstruídas pelo modelo.

3.1 Bases de Dados

Foram utilizadas, principalmente, cinco bases de dados para o treinamento do modelo proposto, construídas usando as imagens das seguintes bases de dados:

1. *CLIC* [46]. Deste *dataset* foram pegos 4 conjuntos com os seguintes nomes e tamanhos:
   (a) Professional valid: 41 imagens;
   (b) Professional train: 585 imagens;
   (c) Mobile valid: 61 imagens;
   (d) Mobile train: 1048 imagens;
2. *DIV2K* [47]. Deste *dataset* foram pegos 2 conjuntos com os seguintes nomes e tamanhos:
   (a) Train: 800 imagens;
   (b) Valid: 100 imagens;
3. *EYE* [48]. Deste *dataset* foram pegos 2 conjuntos com os seguintes nomes e tamanhos:

(a) HD: 38 imagens;

(b) UHD: 40 imagens;

Também foi utilizada a base [2] e o conjunto *Mobile test* da base CLIC para teste. Todas as imagens usadas são de alta qualidade (sem ruído, boa iluminação, alta nitidez), sendo que as imagens *professional* possuem qualidade maior que as mobile. A base EYE consiste de imagens naturais de alta qualidade e resolução adquiridas usando várias câmeras. As imagens cobrem uma quantidade variada de cenas, incluindo cenas ao ar livre e interiores, imagens da natureza, pessoas, animais e cenas históricas retratadas em pinturas. As imagens das bases DIV2K e EYE têm resolução maior que as do CLIC, entretanto isso não faz muita diferença visto que são usados patches com 32 pixels de largura e altura na rede.

Primeiramente, todas as imagens foram separadas em *patches* com 32 pixels de largura e altura, resultando em 6,231,440 *patches*. Cada imagem foi codificada sem perdas no formato PNG, e o tamanho de cada arquivo é usado como critério para a entropia do *patch* (*patches* com tamanhos menores são considerados como sendo de "baixa entropia"). O histograma da base de dados completa é mostrado na Figura 3.1. Foram geradas cinco base de dados com cerca de 1.25 milhões de *patches* em cada uma. Para cada base é pego um subconjunto do total de *patches*. Dado $b(i)$, onde $i \in \{1, \ldots, n\}, b(i) =$ número de ocorrências de patches com esse tamanho e $n \in \mathbb{Z}$ é a quantidade total de *bins* de modo que $\sum_{i=1}^{n} b(i) = 6231440$, (cada *bin* é formado pela quantidade de ocorrências de *patches* com aquele tamanho em *bytes*), os *bins* são ordenados de forma crescente usando o tamanho do arquivo (entropia) como critério. Em seguida, para cada base, é escolhido o menor inteiro x de *bins* tal que $\sum_{i=1}^{x} b(i) \geq 0.2 \sum_{i=1}^{n} b(i)$. Por exemplo, a base de dados 0 é constituída dos primeiros k *bins* de *patches*, pegos em ordem crescente, tal que $\sum_{i=1}^{k} b(i) \geq 6231440/5$, conforme mostra a Figura 3.2.

Cada base de dados tem características específicas e entendê-las é um fator essencial para avaliar o modelo proposto e o impacto de métodos e hiperparâmetros diferentes nos resultados. As bases, nomeadas BDi, $i \in \{0, \cdots, 4\}$, possuem as seguintes características:

- **BD0**: formada por 1248978 de *patches* que pertencem ao grupo dos 20% com menor entropia;
- **BD1**: formada por 1251421 de *patches* que pertencem ao grupo dos que estão na faixa 40% à 60% (porcentagem dada pelo *patch* com maior entropia);
- **BD2**: formada por 1248725 de *patches* que pertencem ao grupo dos 20% com maior entropia;

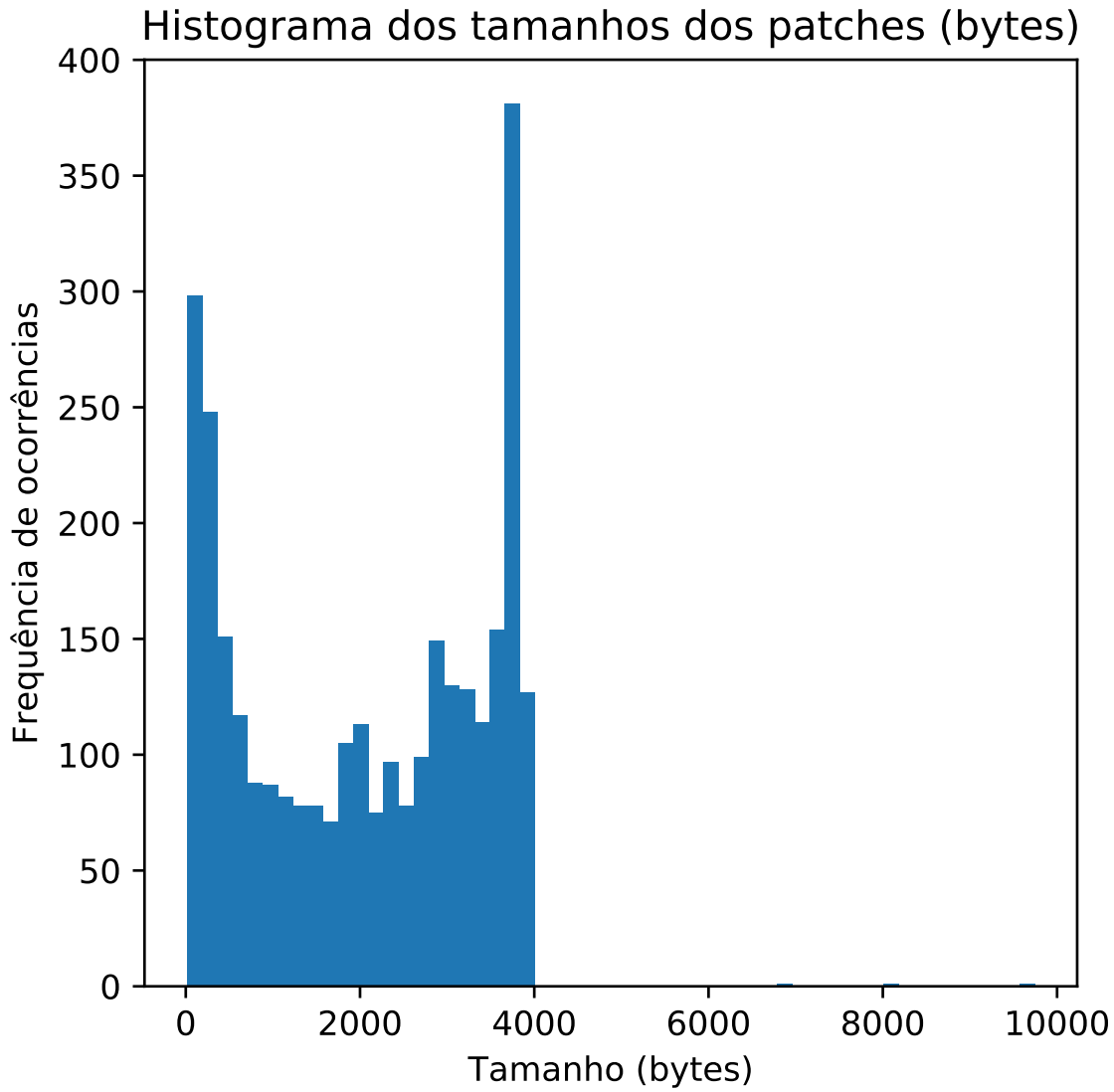

Figura 3.1: Histograma da base de dados completa formada por 6,231,440 de patches.

- **BD3**: formada por 1247033 de *patches* pegos de forma aleatória. Correspondem à 20% do total.
- **BD4**: formada por 1246698 de *patches*. 20% do total retirados aleatoriamente dos 50% de *patches* com maior entropia.

Por construção, não há sobreposição entre as bases de dado 0, 1 e 2, mas existe sobreposição destas bases com as bases 3 e 4. Um histograma de cada base [Figuras 3.2 a 3.6] é dado.

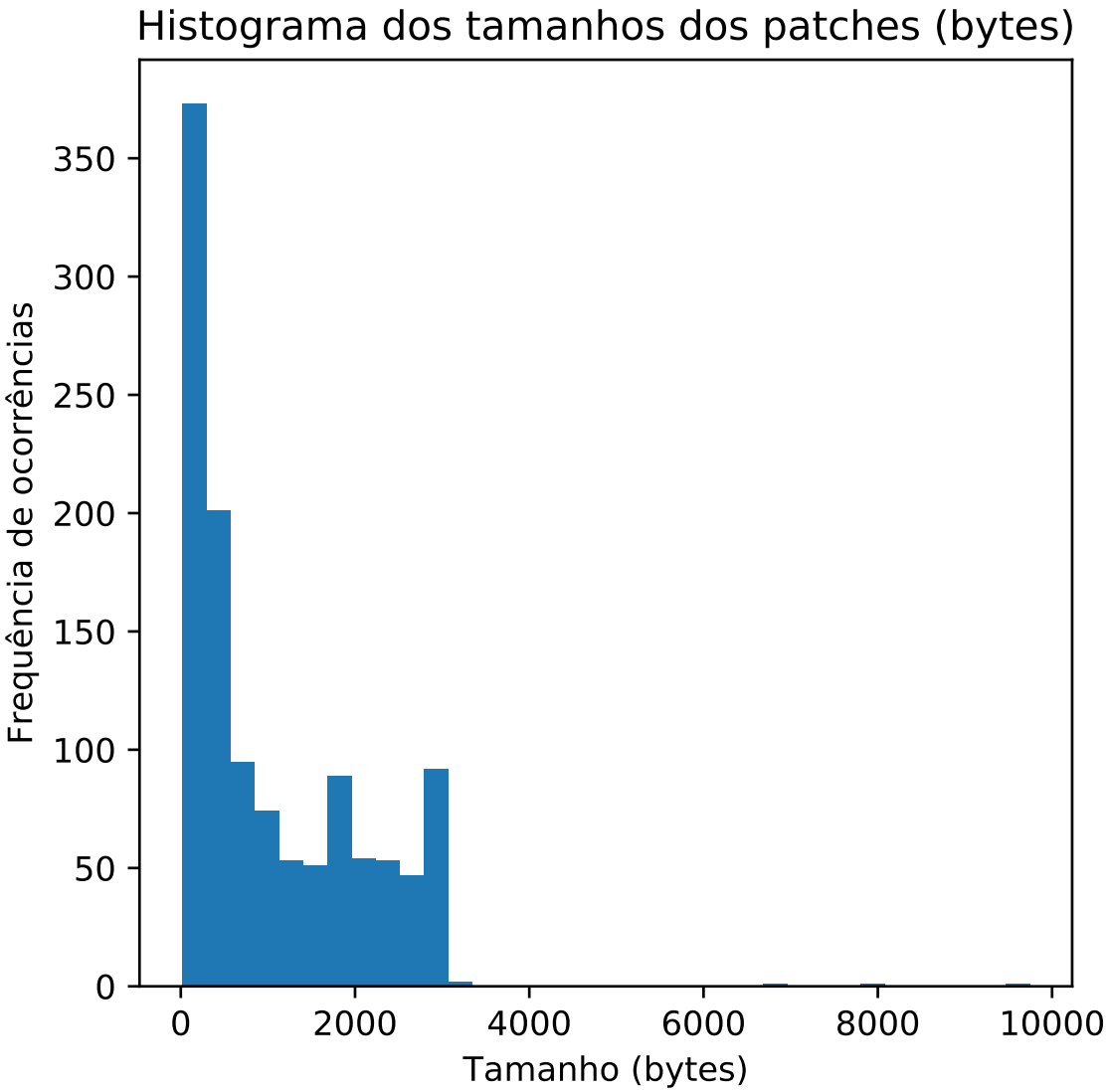

Figura 3.2: Histograma da **BD0**.

3.2 Modelos desenvolvidos

Foi utilizada uma função de binarização $\hat{b}(x) \in \{-1, 1\}$. Aqui as probabilidades, denotadas por p, foram obtidas a partir de uma distribuição uniforme no intervalo $[0, 1]$. $\hat{b}(x)$ é definida como:

$$\hat{b}(x) = \begin{cases} 1, & \text{se } \frac{1-x}{2} \leq p, \\ -1, & \text{caso contrário.} \end{cases} \tag{3.1}$$

Foram desenvolvidos três *autoencoders* convolucionais com o objetivo de avaliar o potencial de cada um nas bases de dados utilizadas. O primeiro não possui binarização/quantização

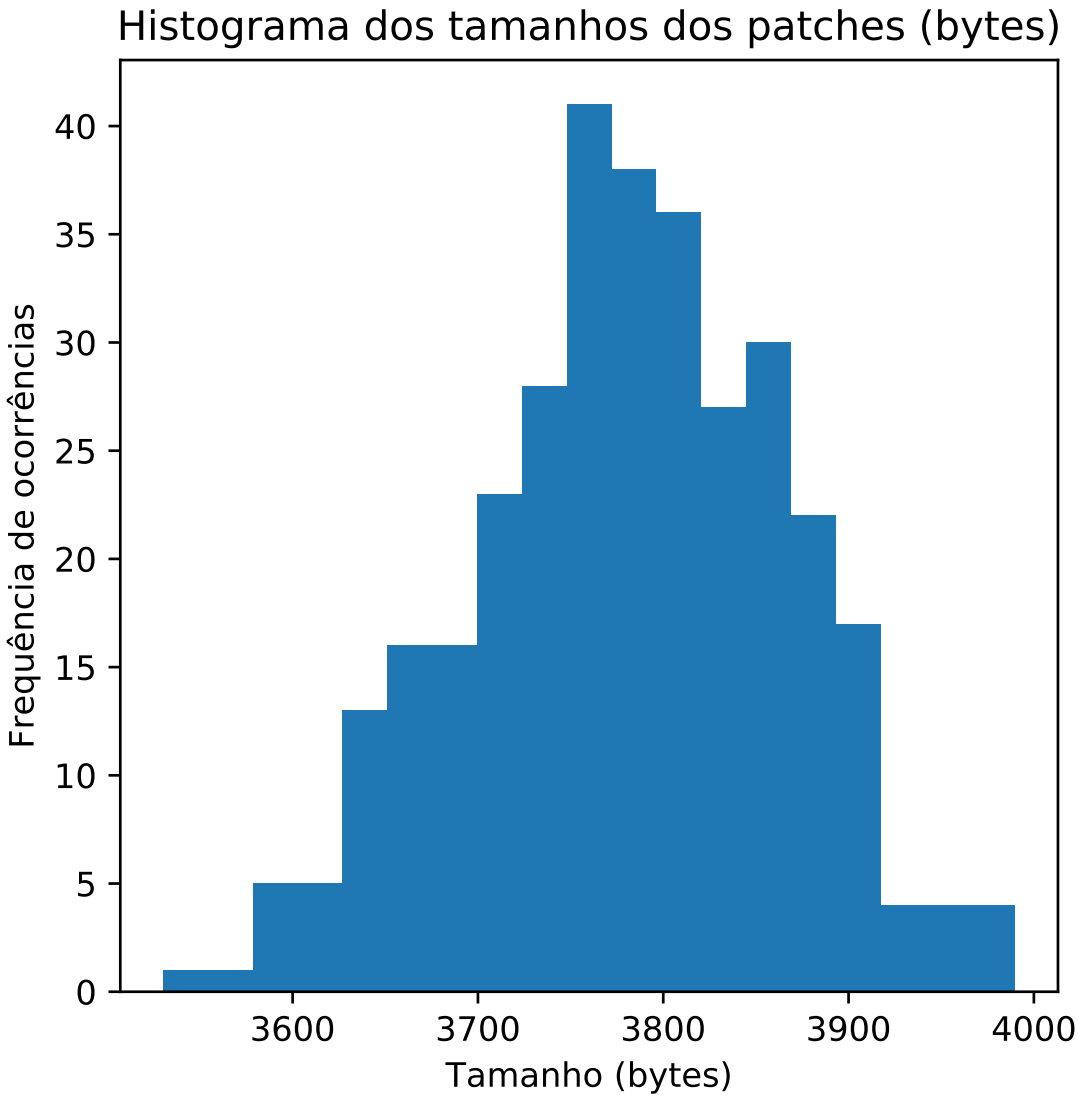

Figura 3.3: Histograma da **BD1**.

no latente gerado pelo *encoder*. O segundo incorpora o binarizador (representado com um trapézio nas figuras) baseado no do *Toderici* usando $\hat{b}(x)$. O último é a arquitetura proposta por *Toderici* em [28] com a utilização de $\hat{b}(x)$. Nas seguintes subseções são apresentadas as ilustrações das arquiteturas utilizadas. Os retângulos indicam as convoluções e os retângulos arredondados indicam as convoluções transpostas. O tamanho do *kernel* w é indicado na primeira linha do retângulo. A segunda linha informa o número de filtros (canais de saídas). A última linha indica o tamanho (igual nas duas direções) do *stride* utilizado e a função de ativação. A última camada do *decoder* (convolução transposta) de cada um dos modelos é usada para recuperar a informação de cor.

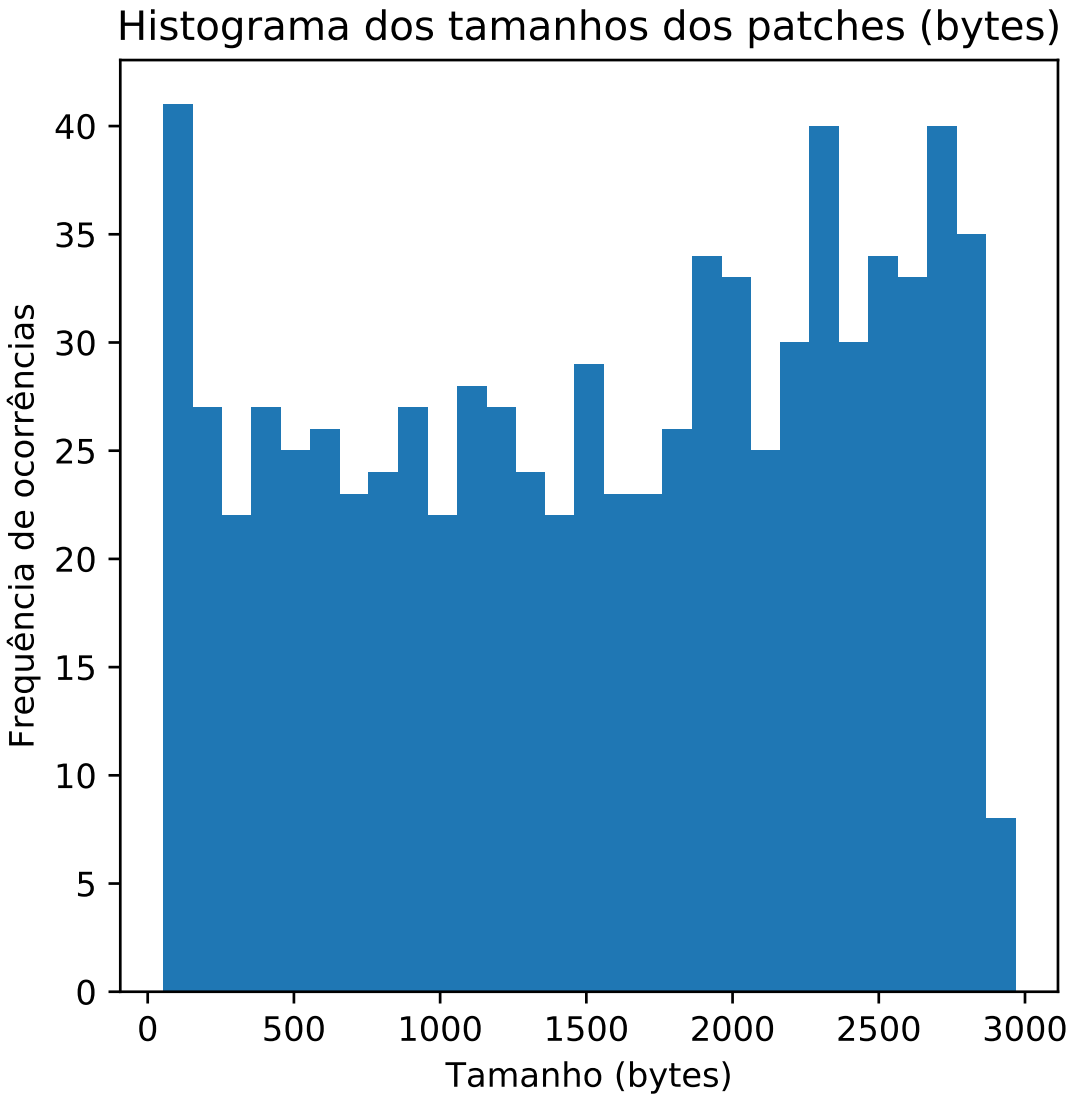

Figura 3.4: Histograma da **BD2**.

3.2.1 Modelo 1

Este primeiro modelo foi desenvolvido com o objetivo de avaliar o potencial de um *autoencoder* convolucional simples sem camada de gargalo com binarizador/quantizador. Aqui, o armazenamento da imagem codificada pelo *encoder* seria custoso, visto que não há binarização. Portanto, o objetivo é apenas avaliar a distorção das imagens reconstruídas. A arquitetura é ilustrada na Figura 3.7.

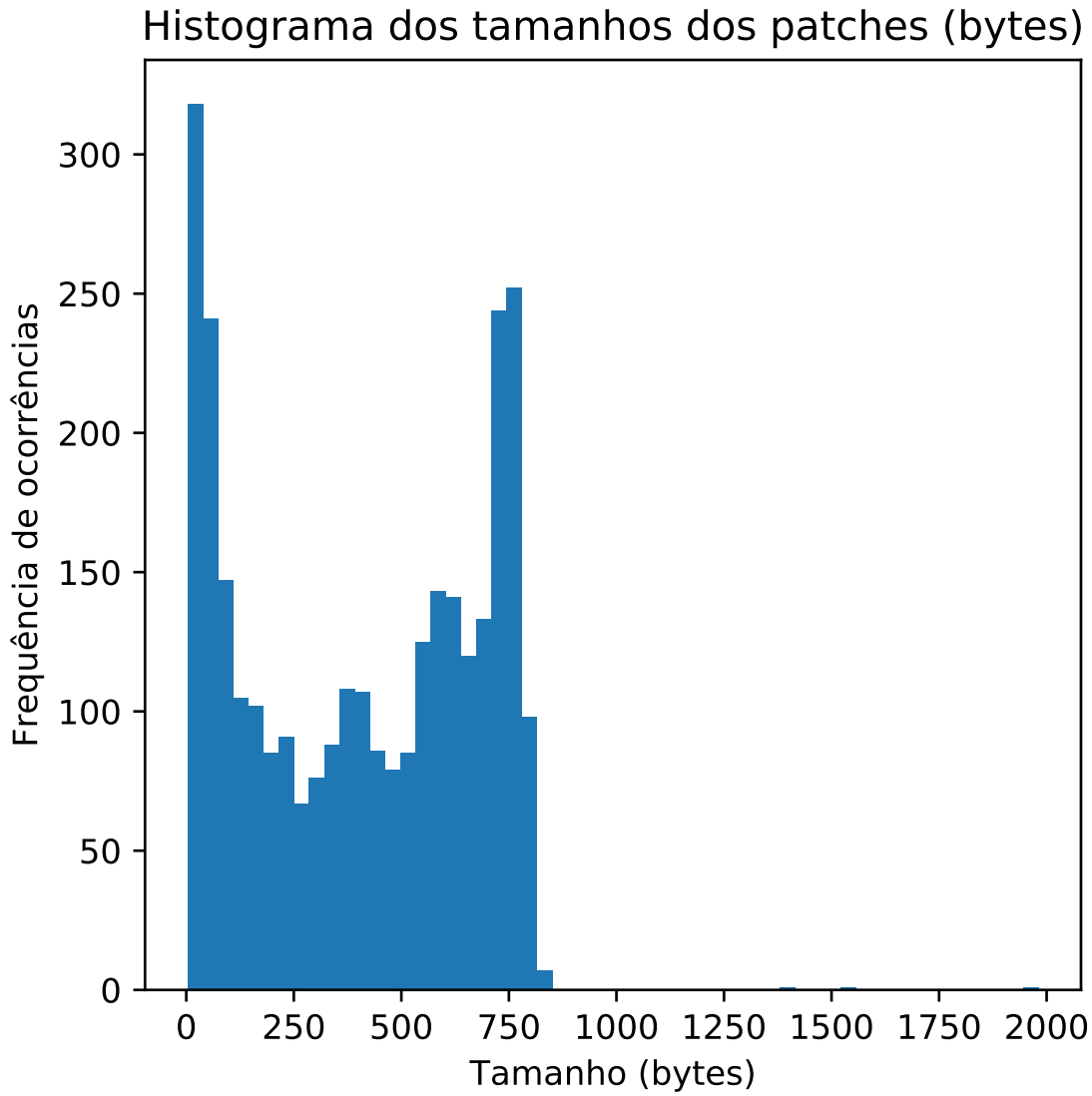

Figura 3.5: Histograma da **BD3**.

3.2.2 Modelo 2

O segundo modelo adiciona o binarizador na camada de gargalo, o que força o *encoder* à comprimir informação visto que será gerada uma saída inteira discreta no conjunto $\{-1, 1\}$ a partir da entrada real contínua. Há uma grande perda de informação em troca de ganho em espaço, pois cada valor pode ser salvo usando apenas um bit agora. Nesta arquitetura [Figura 3.8] a taxa *nominal* de bits por pixel é $\frac{8 \cdot 8 \cdot 128}{32 \cdot 32} = 8$.

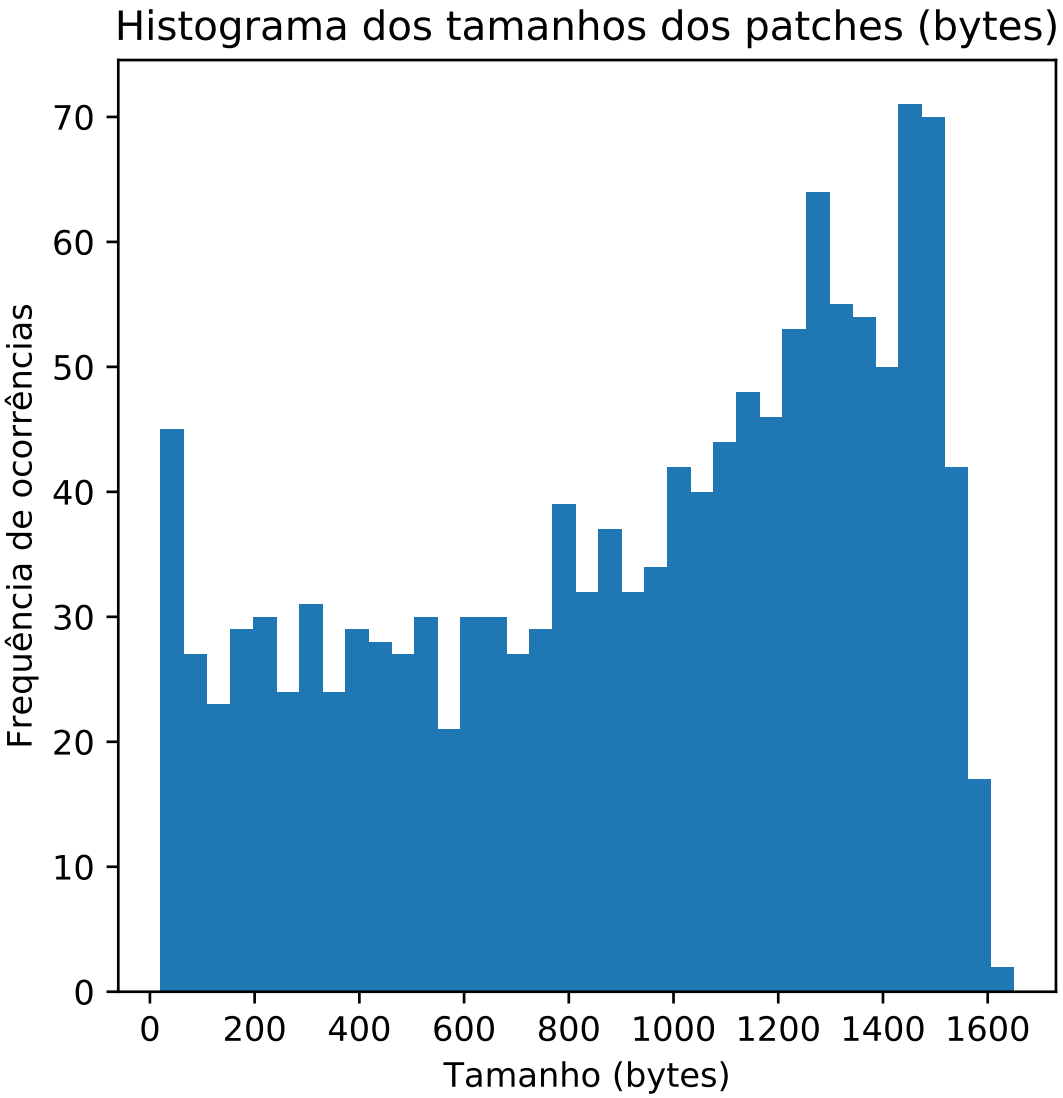

Figura 3.6: Histograma da **BD4**.

Figura 3.7: Ilustração do *autoencoder* mais básico desenvolvido.

3.2.3 Modelo 3

O terceiro modelo [Figura 3.9] segue a arquitetura proposta pelo *Toderici* em [28]. Possui mais camadas em comparação com os outros dois modelos propostos aqui, e uma taxa

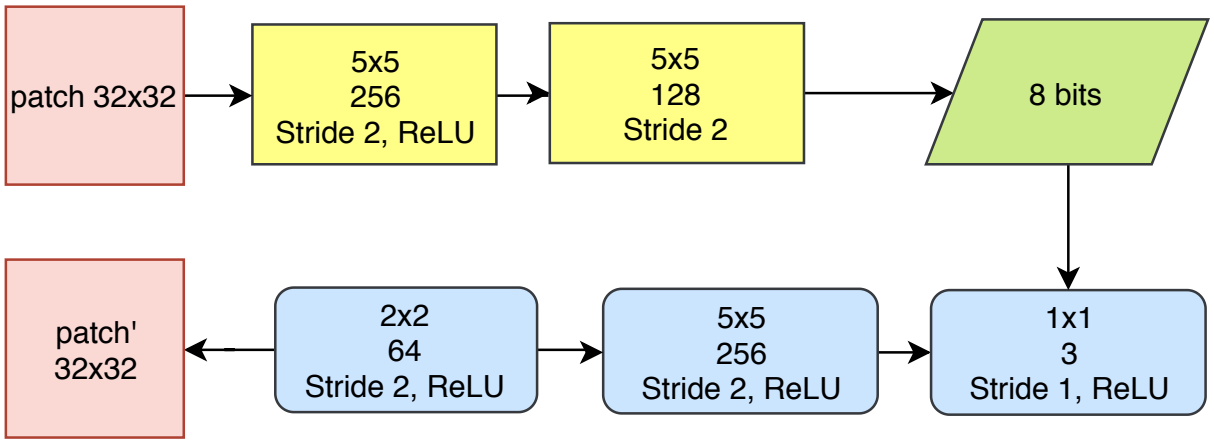

Figura 3.8: Ilustração do segundo modelo desenvolvido.

nominal de $\frac{8 \cdot 8 \cdot 32}{32 \cdot 32} = 2$ bits por pixel, visto que é aplicada uma convolução de tamanho 1 por 1 com 32 filtros antes da função de ativação tangente hiperbólica ser aplicada.

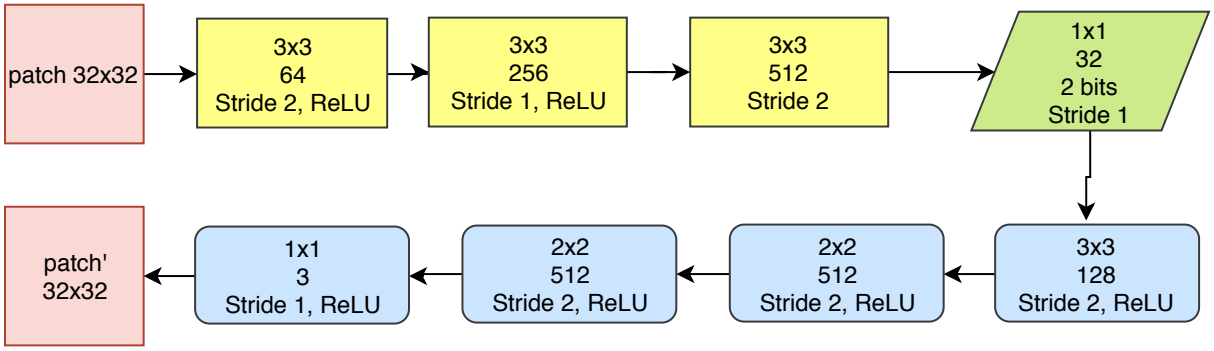

Figura 3.9: Ilustração do terceiro modelo desenvolvido.

Capítulo 4

Experimentos e Resultados

Foram realizados vários experimentos com os 3 modelos apresentados no capítulo anterior e com o *JPEG* com o objetivo de avaliar o potencial de cada um deles nas bases de dados propostas. Os *patches* usados em todos os experimentos realizados neste capítulo possuem 32 pixels de largura e altura. Para os testes em que o conjunto de treino e de teste são a mesma base, foram gerados dois subconjuntos disjuntos: um para treino e um para teste. Este último possui cerca de 10% do tamanho total e, evidentemente, não faz parte do conjunto de treino. Os resultados do **JPEG** nas bases de teste utilizadas são apresentados na seção 4.1 e os resultados dos modelos apresentados no capítulo anterior são apresentados nas seções 4.2, 4.3 e 4.4.

4.1 JPEG

Nesta seção é apresentado o desempenho obtido pelo **JPEG** nas bases de teste.

Tabela 4.1: Tabela contendo médias obtidas pelo **JPEG** em cada uma das bases de teste utilizadas.

Bases	**BPP**	**PSNR**	**SSIM**	**MSSIM**	**Quality**
CLIC Mobile test (patches 32)	8	44.23	0.98	0.99	94.06
CLIC Mobile test	2	39.58	0.96	0.99	88.69
Kodak	2	36.77	0.95	0.99	85.91
BD0	8	57.85	0.99	0.99	99.18
BD1	8	42.94	0.97	0.99	95.94
BD2	8	32.31	0.95	0.99	83.69
BD3	8	43.84	0.96	0.99	93.94
BD4	8	36.72	0.96	0.99	89.68

Tabela 4.2: Tabela contendo o valor da PSNR, em decíbeis, dos testes do Modelo 1 com o uso do otimizador *Adam* e *learning rate* fixa. As linhas denotam a base de treino utilizada. As colunas denotam as bases de teste usadas para avaliação do modelo. O índice "todas" se refere ao uso de todas as imagens de todas as bases **BD** para treino.

Treino (linhas) x Teste (colunas)	**BD0**	**BD1**	**BD2**	**BD3**	**BD4**
BD0	49.66	38.07	26.34	38.05	31.13
BD1	47.57	44.08	34.54	42.66	38.69
BD2	**53.69**	**51.16**	**44.07**	**50.06**	**47.14**
BD3	50.62	47.88	39.76	46.59	43.25
BD4	47.30	46.98	41.10	45.63	43.75
Todas	46.77	46.35	43.94	45.88	45.08

Tabela 4.3: Tabela contendo o valor da PSNR, em decíbeis, dos testes do Modelo 1 com o uso do otimizador *Adam* e *learning rate* fixa. As linhas denotam a base de treino utilizada. As colunas denotam as bases de teste usadas para avaliação do modelo. O índice "todas" se refere ao uso de todas as imagens de todas as bases **BD** para treino. O uso de $x + y$ denota o uso de todas as imagens do conjunto x e do conjunto y para treinamento.

Treino (linhas) x Teste (coluna)	**CLIC Mobile test**
BD0	34.77
BD1	44.11
BD2	48.97
BD3	45.34
BD4	42.42
Todas	51.27
Todas + CLIC Mobile train	**55.78**
Todas + Clic Mobile train + Clic Professional train	47.26
CLIC Mobile Train	46.63

4.2 Modelo 1

Primeiramente, o Modelo 1 [Figura 3.7] foi testado em todas as bases de dados usando *learning rate* fixa durante todo o treinamento e o otimizador *Adam*. Foi utilizada apenas uma época para treino em todos as avaliações realizadas. Os resultados são apresentados na Tabela 4.2. Com estes mesmos hiperparâmetros para treino, foram realizados alguns testes na base de teste (nomeada como *test*) do CLIC [46]. Alguns destes testes também usaram, em adição aos conjuntos de treino montados, os conjuntos de treino (*train*) e validação (*valid*) do CLIC. Os resultados são apresentados na Tabela 4.3.

É interessante notar que o **BD2** é a melhor base de dados para treino, o que reforça os resultados encontrados por *Toderici* em [27]. Outro resultado interessante obtido ocorre ao treinar na base de dados com menor entropia e testar na base com maior entropia.

Tabela 4.4: Tabela contendo os resultados do Modelo 2 para as métricas visuais PSNR, SSIM e MS-SSIM a uma taxa nominal de 8 bits por pixel.

Bases de Treino e Teste	**BPP**	**PSNR**	**SSIM**	**MS-SSIM**	**Épocas**
CLIC Mobile test	8	35.03	0.94	0.98	30
BD1	8	35.26	0.93	0.98	30
BD2	8	29.10	0.95	0.98	30

Pode-se notar que foi muito maléfico para a aprendizagem da rede treinar somente com exemplos fáceis para testar em exemplos difíceis. Posteriormente foram realizados alguns testes usando o método de atualização de *learning rate* explicado no capítulo anterior. A política utilizada foi a exp_range, pois foi a que obteve melhores resultados. Após a realização de vários testes, foi possível aumentar os dB obtidos anteriormente de 44.08 e 44.07 ao treinar e testar no BD0 e BD1 para 50.81 e 47.83, respectivamente. Considerando que todos esses treinamentos referentes aos resultados apresentados nas Tabelas 4.2 a 4.3 foram executados utilizando uma única época, esta melhora pode ser considerada como a "superconvergência" apontada em [7]. É interessante notar que, ao contrário do que foi observado por *Leslie* neste artigo, o uso do otimizador *Adam* com a exp_range apresentou melhoras significativas para o Modelo 1.

4.3 Modelo 2

Para o Modelo 2 [Figura 3.8], foram feitos testes nas bases **CLIC Mobile**, **BD1** e **BD2**. Os resultados são apresentados na Tabela 4.4. Conforme esperado, dentre as bases **BD1** e **BD2**, os piores resultados do *JPEG* e do *autoencoder* foram encontrados no **BD2** que é o com maior entropia (PNG teve mais dificuldade para comprimir). Nota-se que a menor diferença proporcional entre o resultado do modelo e do *JPEG* na métrica PSNR se dá no **BD2**, o que reforça a observação feita em [28] de que em baixas taxas e resoluções espaciais, os artefatos blocantes do *JPEG* (ruído causado pela perda de informação) se tornam mais comuns. Na Figura 4.1 é mostrado um *patch* reconstruído que obteve 36.69 dB de *PSNR*.

4.4 Modelo 3

Para o Modelo 3 [Figura 3.9], foram feitos testes nas bases **CLIC Mobile**, **BD0**, **BD1**, **BD2**, **BD3**, **BD4** e **Kodak**. Os resultados são apresentados na Tabela 4.5. Uma comparação com o *JPEG* para diferentes taxas e métricas de distorção é apresentada na Figuras 4.3 a 4.2.

Figura 4.1: Imagem original (esquerda) e *patch* reconstruído pelo Modelo 2 (direita).

Tabela 4.5: Tabela contendo os resultados do Modelo 3.

Bases	**BPP**	**PSNR**	**SSIM**	**MS-SSIM**
CLIC Mobile test (patches 32)	2	33.75	0.92	0.97
Kodak (patches 32)	2	31.46	0.88	0.96
BD0	2	40.24	0.97	0.99
BD1	2	35.00	0.91	0.98
BD2	2	27.53	0.91	0.97
BD3	2	33.27	0.91	0.97
BD4	2	30.16	0.90	0.97

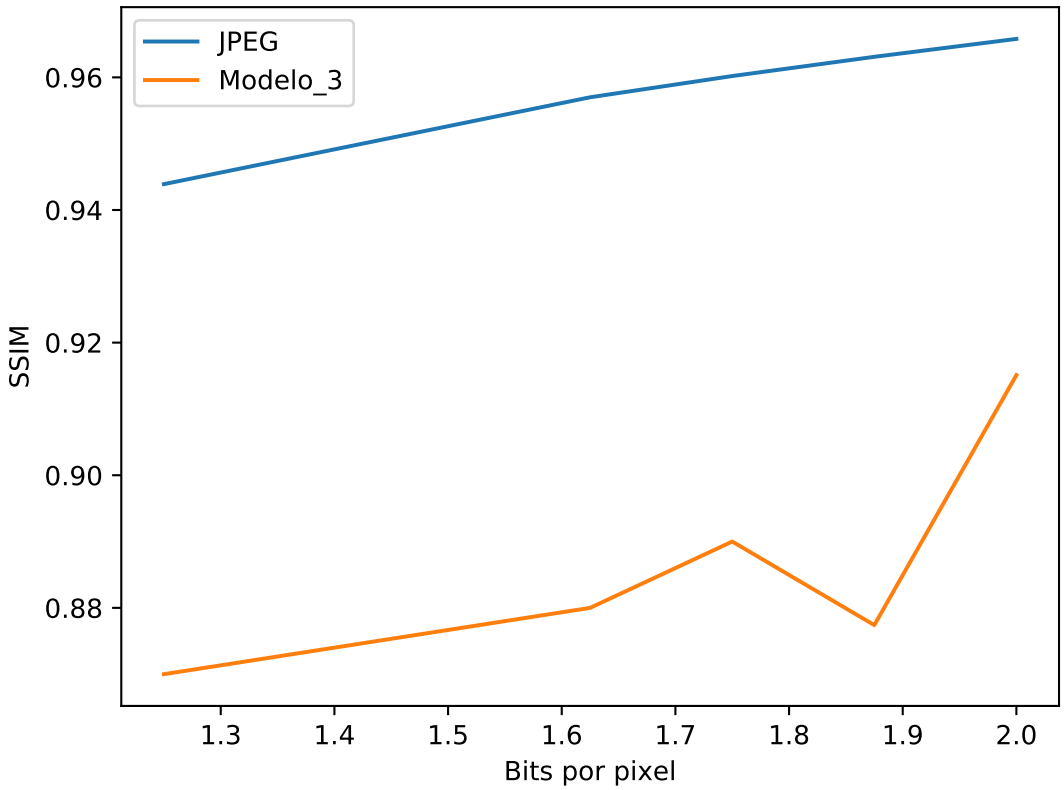

Figura 4.2: Comparação do Modelo 3 com o JPEG na métrica PSNR à diferentes taxas.

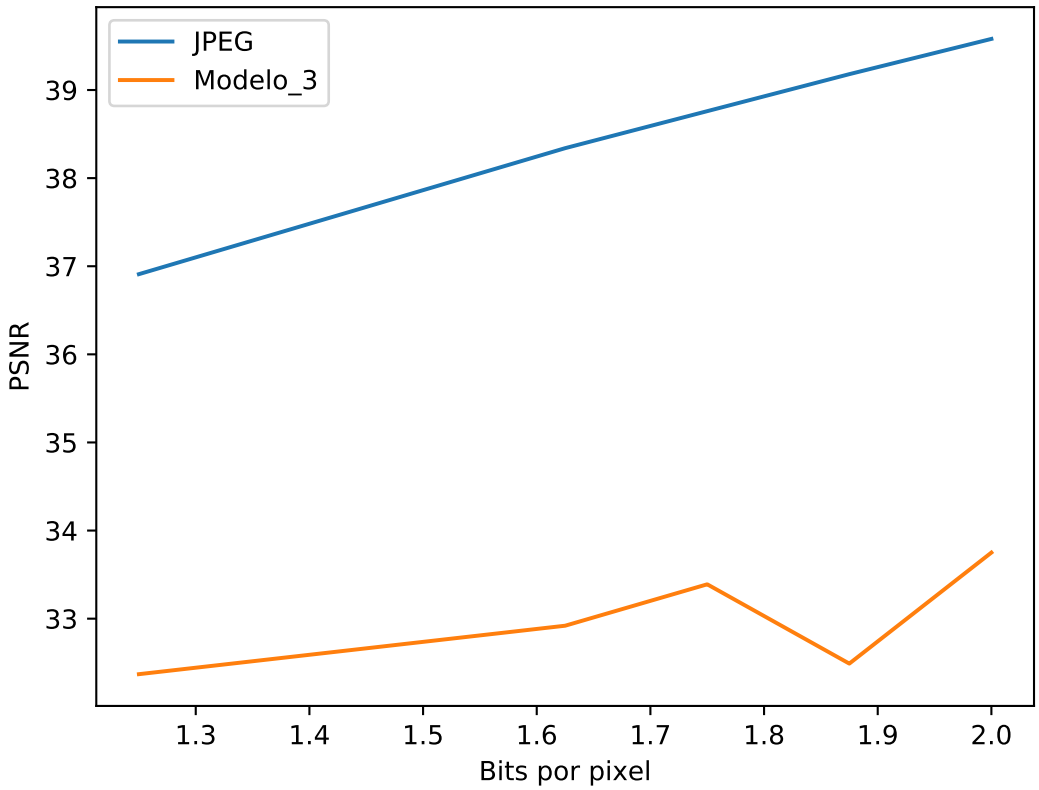

Figura 4.3: Comparação do Modelo 3 com o JPEG na métrica SSIM à diferentes taxas.

Capítulo 5

Conclusão

Este capítulo mostra o que se alcançou com os objetivos. É feita uma análise crítica na seção 5.2 para verificar a completude dos objetivos propostos pelo trabalho. A seção 5.3 indica as perspectivas futuras e os próximos passos a serem dados. A seção 5.1 aborda as limitações enfrentadas durante o desenvolvimento do trabalho.

5.1 Limitações do Trabalho

Modelos baseados em redes neurais para compressão de imagens precisam de muitas imagens para serem treinadas e possuem custo computacional superior aos codecs clássicos para codificar e decodificar imagens. Métodos baseados em redes neurais possuem o potencial de suprir uma necessidade crescente por algoritmos de compressão com perdas flexíveis. Entretanto, compressão com perdas é uma problema não diferenciável. Em particular, quantização é uma parte integral do *pipeline* de compressão mas não é diferenciável, o que dificulta o trabalho de treinar redes neurais para esta tarefa.

5.2 Análise Crítica

Neste trabalho, não foi proposto um novo método para comprimir imagens mas foram replicadas técnicas já existentes na literatura alcançando resultados razoáveis para um trabalho que tinha como principal objetivo se familiarizar com a literatura e propostas existentes. No entanto, foi detectado uma necessidade de usar outros tipos de modelos para que seja possível superar os *codecs* clássicos no âmbito do tema deste trabalho. Esta necessidade é abordada na seção 5.3.

5.3 Trabalhos Futuros

Ainda há muito espaço para novas soluções no âmbito do problema abordado, portanto foram definidas algumas atividades chaves para dar sequência ao trabalho. Será estudado formas de melhorar o desempenho dos modelos usando técnicas propostas nos trabalhos apresentados em 2.2.2 com técnicas comumente usadas em redes neurais convolucionais. Segue um cronograma em formato de Diagrama de Gantt que organiza as atividades do próximo semestre letivo.

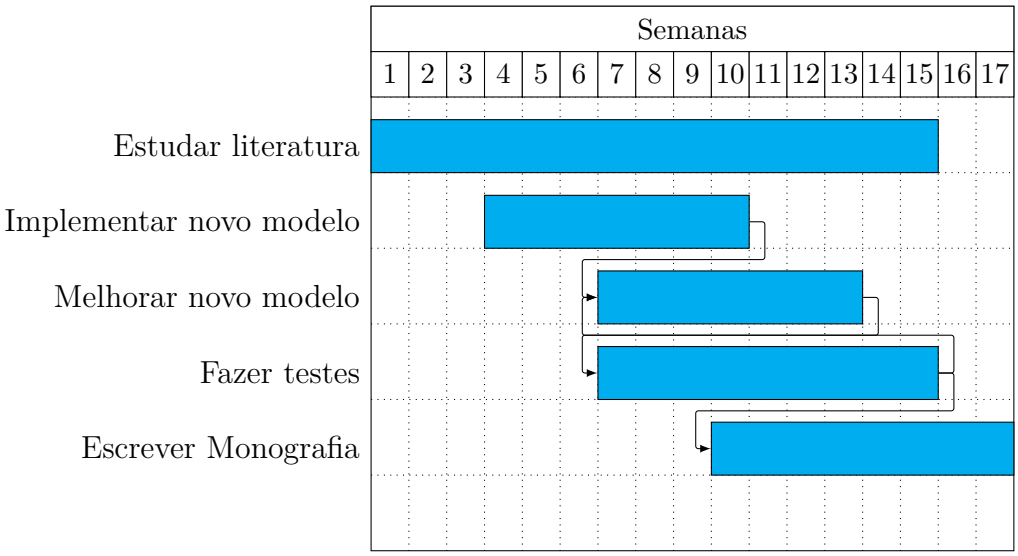

Diagrama de Gantt

Referências

[1] Wallace, G. K.: *The jpeg still picture compression standard*. IEEE Transactions on Consumer Electronics, 38(1):xviii–xxxiv, Feb 1992, ISSN 0098-3063. ix, 2, 6, 11

[2] Kodak., E.: *Kodak lossless true color image suite*. `http://r0k.us/graphics/kodak/`, 2014. [Online; accessed 25-June-2019]. ix, 7, 24

[3] Commons, Wikimedia: *File:dct-8x8.png — wikimedia commons, the free media repository*. `https://commons.wikimedia.org/w/index.php?curid=10414002`, 2015. [Online; accessed 30-June-2019]. ix, 9

[4] Computerphile: *The problem with jpeg - computerphile*. `https://youtu.be/yBX8GFqt6GA?t=48`, 2015. [Online; accessed 30-June-2019]. ix, 10

[5] Goodfellow, Ian, Yoshua Bengio e Aaron Courville: *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`. ix, 12, 13

[6] Materials, CS231n Course: *Cs231n convolutional neural networks for visual recognition*. `http://cs231n.github.io/neural-networks-3/`, 2019. [Online; accessed 30-June-2019]. ix, 13

[7] Smith, Leslie N: *Cyclical learning rates for training neural networks*. Em *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, páginas 464–472. IEEE, 2017. ix, 14, 34

[8] Jordan, Jeremy: *Introduction to autoencoders*. `https://www.jeremyjordan.me/autoencoders/`, 2018. [Online; accessed 30-June-2019]. ix, 16

[9] Shafkat, Irhum: *Towards data science — intuitively understanding variational autoencoders*. `https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf`, 2018. [Online; accessed 30-June-2019]. ix, 17

[10] Theis, Lucas, Wenzhe Shi, Andrew Cunningham e Ferenc Huszár: *Lossy image compression with compressive autoencoders*. Relatório Técnico, Cornell University Library, 2017. arXiv:1703.00395. ix, 19

[11] Materials, CS231n Course: *Cs231n convolutional neural networks for visual recognition*. `http://cs231n.github.io/convolutional-networks/`, 2019. [Online; accessed 30-June-2019]. x, 21

[12] Zaghetto, Alexandre: *Intensity transformation and spatial filtering*. `https://github.com/zaghetto/ImageProcessing`, 2018. [Online; accessed 30-June-2019]. x, 22

[13] Sayood, Khalid: *Introduction to data compression*. Morgan Kaufmann, 2017. 1, 2

[14] Christopoulos, C., A. Skodras e T. Ebrahimi: *The jpeg2000 still image coding system: an overview*. IEEE Transactions on Consumer Electronics, 46(4):1103–1127, Nov 2000, ISSN 0098-3063. 2

[15] Google, WebP: *Compression techniques*. `https://developers.google.com/speed/webp/docs/compression`. Acessed: 2019-06-16. 2

[16] Bellard, Fabrice: *BPG image format*. `https://bellard.org/bpg/`. Accessed: 2019-06-13. 2

[17] Simonyan, Karen e Andrew Zisserman: *Very deep convolutional networks for large-scale image recognition*. Relatório Técnico, Cornell University Library, 2014. arXiv:1409.1556. 4

[18] Girshick, Ross, Jeff Donahue, Trevor Darrell e Jitendra Malik: *Rich feature hierarchies for accurate object detection and semantic segmentation*. Em *Proceedings of the IEEE conference on computer vision and pattern recognition*, páginas 580–587, 2014. 4

[19] Choy, Christopher B, Danfei Xu, JunYoung Gwak, Kevin Chen e Silvio Savarese: *3d-r2n2: A unified approach for single and multi-view 3d object reconstruction*. Em *Proceedings of the European Conference on Computer Vision (ECCV)*, 2016. 4

[20] Taigman, Yaniv, Ming Yang, Marc'Aurelio Ranzato e Lior Wolf: *Deepface: Closing the gap to human-level performance in face verification*. Em *Proceedings of the IEEE conference on computer vision and pattern recognition*, páginas 1701–1708, 2014. 4

[21] Graves, Alex e Navdeep Jaitly: *Towards end-to-end speech recognition with recurrent neural networks*. Em *International Conference on Machine Learning*, páginas 1764–1772, 2014. 4

[22] Sutskever, Ilya, Oriol Vinyals e Quoc V Le: *Sequence to sequence learning with neural networks*. Em *Advances in neural information processing systems*, páginas 3104–3112, 2014. 4

[23] Vinyals, Oriol, Alexander Toshev, Samy Bengio e Dumitru Erhan: *Show and tell: A neural image caption generator*. Em *Proceedings of the IEEE conference on computer vision and pattern recognition*, páginas 3156–3164, 2015. 4

[24] Huval, Brody, Tao Wang, Sameep Tandon, Jeff Kiske, Will Song, Joel Pazhayampallil, Mykhaylo Andriluka, Pranav Rajpurkar, Toki Migimatsu, Royce Cheng-Yue *et al.*: *An empirical evaluation of deep learning on highway driving*. Relatório Técnico, Cornell University Library, 2015. arXiv:1504.01716. 4

[25] Krizhevsky, Alex e Geoffrey E Hinton: *Using very deep autoencoders for content-based image retrieval.* Em *ESANN*, 2011. 4

[26] Gregor, Karol, Frederic Besse, Danilo Jimenez Rezende, Ivo Danihelka e Daan Wierstra: *Towards conceptual compression.* Em *Advances In Neural Information Processing Systems*, páginas 3549–3557, 2016. 4

[27] Toderici, George, Damien Vincent, Nick Johnston, Sung Jin Hwang, David Minnen, Joel Shor e Michele Covell: *Full resolution image compression with recurrent neural networks.* Em *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, páginas 5306–5314, 2017. 4, 18, 33

[28] Toderici, George, Sean M O'Malley, Sung Jin Hwang, Damien Vincent, David Minnen, Shumeet Baluja, Michele Covell e Rahul Sukthankar: *Variable rate image compression with recurrent neural networks.* Relatório Técnico, Cornell University Library, 2016. arXiv:1511.06085. 4, 17, 27, 30, 34

[29] Mentzer, Fabian, Eirikur Agustsson, Michael Tschannen, Radu Timofte e Luc Van Gool: *Practical full resolution learned lossless image compression.* Em *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 4

[30] Theis, Lucas e Matthias Bethge: *Generative image modeling using spatial lstms.* Em *Advances in Neural Information Processing Systems*, páginas 1927–1935, 2015. 4

[31] Hamilton, Eric: *Jpeg file interchange format.* 2004. 6

[32] Ahmed, Nasir, T_ Natarajan e Kamisetty R Rao: *Discrete cosine transform.* IEEE transactions on Computers, 100(1):90–93, 1974. 8

[33] Huffman, David A: *A method for the construction of minimum-redundancy codes.* Proceedings of the IRE, 40(9):1098–1101, 1952. 11

[34] Pennebaker, WB, JL Mitchell *et al.*: *Arithmetic coding articles.* IBM J. Res. Dev, 32(6):717–774, 1988. 11

[35] Mor-Yosef, Shlomo, Arnon Samueloff, Baruch Modan, Daniel Navot e Joseph G Schenker: *Ranking the risk factors for cesarean: logistic regression analysis of a nationwide study.* Obstetrics and gynecology, 75(6):944–947, 1990. 12

[36] Nielsen, Michael A.: *Neural Networks and Deep Learning.* Determination Press, 2015. 12

[37] Rumelhart, David E, Geoffrey E Hinton, Ronald J Williams *et al.*: *Learning representations by back-propagating errors.* Cognitive modeling, 5(3):1, 1988. 12, 20

[38] Kingma, Diederik P e Jimmy Ba: *Adam: A method for stochastic optimization.* Relatório Técnico, Cornell University Library, 2014. arXiv:1412.6980. 15

[39] Wang, Zhou, Alan C Bovik, Hamid R Sheikh, Eero P Simoncelli *et al.*: *Image quality assessment: from error visibility to structural similarity.* IEEE transactions on image processing, 13(4):600–612, 2004. 16

[40] Wang, Zhou, Eero P Simoncelli e Alan C Bovik: *Multiscale structural similarity for image quality assessment*. Em *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, volume 2, páginas 1398–1402. Ieee, 2003. 16

[41] Gers, Felix A, Jürgen Schmidhuber e Fred Cummins: *Learning to forget: Continual prediction with lstm*. 1999. 18

[42] Shi, Wenzhe, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P Aitken, Rob Bishop, Daniel Rueckert e Zehan Wang: *Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network*. Em *Proceedings of the IEEE conference on computer vision and pattern recognition*, páginas 1874–1883, 2016. 19

[43] LeCun, Yann, Koray Kavukcuoglu e Clément Farabet: *Convolutional networks and applications in vision*. Em *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, páginas 253–256. IEEE, 2010. 20

[44] Academy, Data Science: *Deep learning book*. `http://www.deeplearningbook.com.br`, 2019. [Online; accessed 30-June-2019]. 20

[45] Paszke, Adam, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga e Adam Lerer: *Automatic differentiation in pytorch*. 2017. 23

[46] George Toderici, Michele Covell, Wenzhe Shi Radu Timofte Lucas Theis Johannes Ballé Eirikur Agustsson Nick Johnston Fabian Mentzer: *Challenge on learned image compression*. `http://www.compression.cc/challenge/`, 2018. [Online; accessed 25-June-2019]. 23, 33

[47] Agustsson, Eirikur e Radu Timofte: *Ntire 2017 challenge on single image super-resolution: Dataset and study*. Em *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, July 2017. 23

[48] Group, Multimedia Signal Processing: *Ultra-eye: Uhd and hd images eye tracking dataset*. `https://mmspg.epfl.ch/downloads/ultra-eye/`, 2014. [Online; accessed 25-June-2019]. 23