



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 Mitsu Hung Prof. Dr. Luís Paulo Faina Garcia  
FT/UnB CIC/UnB

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

Brasília, 2 de Novembro 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 requeridos 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.

O sucesso das redes neurais convolucionais profundas (CNNs) em visão computacional tem inspirado pesquisadores da comunidade de compressão de imagens a tentar desenvolver algoritmos que aprendem com os dados, em vez de confiar no conhecimento de especialistas. As redes do tipo autoencoder são utilizadas nesses algoritmos, visto que compressão faz parte do funcionamento delas. Até agora esses algoritmos não levaram a uma melhoria significativa em relação aos codecs clássicos.

O objetivo do presente trabalho é estudar e explorar soluções usando *autoencoders* convolucionais recorrentes para o desafio de comprimir imagens, buscando propor um método que possa ser competitivo em relação aos codecs clássicos. Para isso, foram testados e avaliados os métodos clássicos *JPEG* e *JPEG2000* que foram os primeiros codecs vastamente utilizados. Esses métodos foram comparados, em bases de dados comumente usadas para este tipo de problema, com o framework de compressão de imagens usando *autoencoders* convolucionais desenvolvido.

**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 images by exploiting imperfections in the human visual system. This way, it is possible to reconstruct them 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. Autoencoder networks are used in these algorithms, since compression is part of their operation. So far, these algorithms have not lead to a significant improvement over classical codecs.

The objective of the present work is to study and explore neural network solutions to the challenge of compressing images, aiming to propose a method that can be competitive to classical codecs in some scenarios. To achieve this goal, the classical compression methods JPEG and JPEG2000 are evaluated in databases commonly used for this type of problem, as they were the first two methods widely used. Next, some experiments on encoder-decoder image compression framework are presented.

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

# Sumário

<b>1 Introdução</b>	<b>1</b>
1.1 Motivação . . . . .	1
1.2 Hipóteses . . . . .	4
1.3 Objetivos . . . . .	5
1.4 Resultados Esperados . . . . .	5
<b>2 Trabalhos similares e conceitos fundamentais</b>	<b>6</b>
2.1 Métodos clássicos de compressão . . . . .	6
2.1.1 Conversão do espaço de cor . . . . .	6
2.1.2 Aplicação da transformada direta de cossenos . . . . .	7
2.1.3 Quantização . . . . .	9
2.1.4 Codificação . . . . .	12
2.2 Redes Neurais . . . . .	12
2.2.1 Taxa de aprendizagem ( <i>learning rate</i> ) . . . . .	13
2.2.2 Redes Neurais Convolucionais . . . . .	15
2.2.3 Autoencoders . . . . .	16
<b>3 Metodologia</b>	<b>23</b>
3.1 Bases de Dados . . . . .	23
3.2 Modelos desenvolvidos . . . . .	25
3.2.1 Formando o Bitstream . . . . .	27
3.2.2 Modelo 1 . . . . .	30
3.2.3 Modelo 2 . . . . .	31
<b>4 Experimentos e Resultados</b>	<b>32</b>
4.1 JPEG . . . . .	32
4.2 Modelo 1 . . . . .	33
4.3 Modelo 2 . . . . .	34
4.4 Modelo 3 . . . . .	34
4.5 Modelo 4 . . . . .	35

4.6 Modelos 5 e 6 . . . . .	49
4.7 Ganhos obtidos pelo GZIP . . . . .	51
4.7.1 Modelo 4 . . . . .	51
4.7.2 Modelos 5 e 6 . . . . .	51
<b>5 Conclusão</b>	<b>55</b>
5.1 Limitações do Trabalho . . . . .	55
5.2 Análise Crítica e Perspectivas Futuras . . . . .	55
<b>Referências</b>	<b>57</b>

# Listas de Figuras

2.1	Diagram do método de compressão JPEG. Fonte: [1]. . . . .	7
2.2	Imagen 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 ao codificar 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 exibi-la). Fonte: [2]. . . . .	8
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 (direita) cuja codificação ( <i>bitstream</i> ) possui 64 vezes menos pixels de crominância (Cb e Cr) menos cor da imagem original. . . . .	8
2.4	64 (8 por 8) ondas base de cossenos com frequências variadas. Fonte: [3]. .	10
2.5	Comparação imagem de texto com e sem compressão. Fonte: [4]. . . . .	11
2.6	Sequência zig-zague usada para melhorar codificação. Fonte: [1]. . . . .	11
2.7	Um modelo linear aplicado diretamente à entrada original não pode implementar a função XOR, visto que o espaço da função XOR não é linearmente separável. Para isso é necessário transformar o espaço original usando uma função de ativação. Fonte: [5]. . . . .	13
2.8	Efeitos de várias taxas de aprendizagem no treinamento. Fonte: [6]. . . . .	14
2.9	Política <i>exp_range</i> de <i>learning rate</i> cíclica. Fonte: [7]. . . . .	15
2.10	Ilustração de um autoencoder. . . . .	17

2.11 Um <i>autoencoder</i> residual <i>fully-connected</i> . Esta figura mostra uma arquitetura com dois níveis (dois <i>autoencoders</i> empilhados). O primeiro nível codifica a imagem original. O resíduo da reconstrução é passado para o segundo nível. Cada nível produz um latente de 4 bits por pixel, que é representado nos retângulos. Os retângulos marcados com 512 são camadas <i>fully-connected</i> com 512 unidades e não-linearidades de tangente hiperbólica. Fonte: [?]. . . . .	20
2.12 O autoencoder residual convolucional. Os retângulos afiados representam as camadas convolucionais, enquanto os arredondados representam as camadas das convoluções transpostas. A dimensão do kernel é apresentado na primeira linha, enquanto a quantidade de canais de saída e o tamanho do <i>stride</i> são apresentados nas linhas seguintes. A loss é aplicada nos resíduos. Fonte: [?]. . . . .	20
3.1 Histograma da base de dados completa formada por 6,231,440 de patches. . . . .	25
3.2 Histograma da <b>BD0</b> . . . . .	26
3.3 Histograma da <b>BD1</b> . . . . .	27
3.4 Histograma da <b>BD2</b> . . . . .	28
3.5 Histograma da <b>BD3</b> . . . . .	29
3.6 Histograma da <b>BD4</b> . . . . .	30
3.7 Ilustração do <i>autoencoder</i> mais básico desenvolvido. . . . .	31
3.8 Ilustração do segundo modelo desenvolvido. . . . .	31
4.1 Imagem original (esquerda) e <i>patch</i> reconstruído pelo Modelo 2 (direita). . . . .	35
4.2 Comparação do Modelo 3 com o JPEG na métrica PSNR em diferentes taxas. . . . .	36
4.3 Comparação do Modelo 3 com o JPEG na métrica SSIM em diferentes taxas. . . . .	36
4.4 Comparação do Modelo 4 com o JPEG e JPEG2000 na métrica PSNR em diferentes taxas para a base Kodak [2]. . . . .	38
4.5 Comparação do Modelo 4 com o JPEG e JPEG2000 na métrica SSIM em diferentes taxas para a base Kodak [2]. . . . .	38
4.6 Comparação do Modelo 4 com o JPEG e JPEG2000 na métrica MS-SSIM em diferentes taxas para a base Kodak [2]. . . . .	39
4.7 Imagem Kodim05 [2]. . . . .	39
4.8 Imagem jason-leem [8]. . . . .	40
4.9 Comparação do Modelo 4 com o JPEG e JPEG2000 na métrica PSNR em diferentes taxas para a imagem Figura 4.7. . . . .	40

4.10	Comparação do Modelo 4 com o JPEG e JPEG2000 na métrica SSIM em diferentes taxas para a imagem Figura 4.7. . . . .	41
4.11	Comparação do Modelo 4 com o JPEG e JPEG2000 na métrica MS-SSIM em diferentes taxas para a imagem Figura 4.7. . . . .	41
4.12	Comparação do Modelo 4 com o JPEG e JPEG2000 na métrica PSNR em diferentes taxas para a imagem de [8]. . . . .	42
4.13	Comparação do Modelo 4 com o JPEG e JPEG2000 na métrica SSIM em diferentes taxas para a imagem de [8]. . . . .	42
4.14	Comparação do Modelo 4 com o JPEG e JPEG2000 na métrica MS-SSIM em diferentes taxas para a imagem de [8]. . . . .	43
4.15	Comparação do Modelo 4 com o JPEG e JPEG2000 na métrica PSNR em diferentes taxas para 47 imagens com muito conteúdo de alta frequência retiradas das bases [8] e [2]. . . . .	43
4.16	Comparação do Modelo 4 com o JPEG e JPEG2000 na métrica SSIM em diferentes taxas para 47 imagens com muito conteúdo de alta frequência retiradas das bases [8] e [2]. . . . .	44
4.17	Comparação do Modelo 4 com o JPEG e JPEG2000 na métrica MS-SSIM em diferentes taxas para 47 imagens com muito conteúdo de alta frequência retiradas das bases [8] e [2]. . . . .	44
4.18	Imagen Figura 4.7 reconstruída pelo Modelo 4 à uma taxa de 0.47 bits por pixel.. . . . .	46
4.19	Imagen Figura 4.7 reconstruída pelo <b>JPEG</b> à uma taxa de 0.50 bits por pixel.. . . . .	46
4.20	Imagen Figura 4.7 reconstruída pelo <b>JPEG2000</b> à uma taxa de 0.50 bits por pixel. . . . .	47
4.21	Imagen Figura 4.7 reconstruída pelo Modelo 4 à uma taxa de 0.12 bits por pixel. . . . .	47
4.22	Imagen Figura 4.7 reconstruída pelo <b>JPEG</b> à uma taxa de 0.21 bits por pixel.. . . . .	48
4.23	Imagen Figura 4.7 reconstruída pelo <b>JPEG2000</b> à uma taxa de 0.12 bits por pixel. . . . .	48
4.24	Comparação do Modelo 5 e 6 para a métrica MSE. . . . .	49
4.25	Comparação do Modelo 5 e 6 para a métrica SSIM. . . . .	50
4.26	Comparação do Modelo 5 e 6 para a métrica MS-SSIM. . . . .	50
4.27	Ganho percentual médio na taxa por nível ao usar o codificador de entropia <i>gzip</i> nos <i>bitstreams</i> de cada nível para a base Kodak [2]. . . . .	52
4.28	Imagen Kodim20 [2]. . . . .	52

4.29	Ganho percentual na taxa por nível ao usar o codificador de entropia <i>gzip</i> nos <i>bitstreams</i> de cada nível para a Figura 4.7. . . . .	53
4.30	Ganho percentual na taxa por nível ao usar o codificador de entropia <i>gzip</i> nos <i>bitstreams</i> de cada nível para a Figura 4.28. . . . .	53
4.31	Ganho percentual médio na taxa por nível ao usar o codificador de entropia <i>gzip</i> nos <i>bitstreams</i> de cada nível para a base Kodak [2]. . . . .	54

# Listas de Tabelas

4.1	Tabela contendo médias obtidas pelo <b>JPEG</b> em cada uma das bases de teste utilizadas. . . . .	32
4.2	Tabela contendo o valor da PSNR, em decíbeis, dos testes do Modelo 1 com o uso do otimizador <i>Adam</i> e <i>learning rate</i> 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 <b>BD</b> para treino. . . . .	33
4.3	Tabela contendo o valor da PSNR, em decíbeis, dos testes do Modelo 1 com o uso do otimizador <i>Adam</i> e <i>learning rate</i> 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 <b>BD</b> para treino. O uso de $x + y$ denota o uso de todas as imagens do conjunto $x$ e do conjunto $y$ para treinamento. . . . .	33
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. . . . .	34
4.5	Tabela contendo os resultados do Modelo 3. . . . .	35
4.6	Tabela contendo os valores da taxa (BPP) e PSNR (decíbeis) do Modelo 4, JPEG e JPEG2000 para a base [2]. . . . .	37
4.7	Tabela contendo os valores da taxa (BPP) e SSIM do Modelo 4, JPEG e JPEG2000 para a base [2]. . . . .	37
4.8	Tabela contendo os valores da taxa (BPP) e MS-SSIM do Modelo 4, JPEG e JPEG2000 para a base [2]. . . . .	37
4.9	Tabela contendo os valores da taxa (BPP) e PSNR do Modelo 4, JPEG e JPEG2000 para 47 imagens da base [2] e [8] com muito conteúdo de alta frequência. . . . .	45
4.10	Tabela contendo os valores da taxa (BPP) e SSIM do Modelo 4, JPEG e JPEG2000 para 47 imagens da base [2] e [8] com muito conteúdo de alta frequência. . . . .	45

4.11 Tabela contendo os valores da taxa (BPP) e MS-SSIM do Modelo 4, JPEG e JPEG2000 para 47 imagens da base [2] e [8] com muito conteúdo de alta frequência. . . . .	45
---	----

# **Lista de Abreviaturas e Siglas**

**AE** Autoencoder.

**BPP** Bits por Pixel.

**CLIC** Challenge on Learned Image Compression.

**CNN** Convolutional Neural Network.

**dB** Decibéis.

**DC** Direct Current Coefficient.

**DCT** Discrete cosine transform.

**DFT** Discrete Fourier Transform.

**DIV2K** Diverse 2k high resolutonal 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.

**LSTM** Long Short-Term Memory.

**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.

**RNN** Recurrent Neural Network.

**SGD** Stochastic Gradient Descent.

**SSIM** Structural Similarity Index.

**XOR** Exclusive 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 a arte ou ciência de representar informação de forma compacta [9]. 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 [9].

O desenvolvimento de algoritmos de compressão de dados podem ser divididos em duas fases [9]. A primeira fase é geralmente chamada de modelagem. Nessa fase, tentamos extrair informações sobre qualquer redundância existente nos dados e descrevemos a redundância na forma de um modelo. A segunda fase é chamada de codificação: uma

descrição do modelo e uma “descrição” de como os dados diferem do modelo são codificados, geralmente usando um alfabeto binário. A diferença entre os dados e o modelo é frequentemente referida como resíduo.

Existem dois tipos de compressão [9]: 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.

Durante um processo de compressão devemos balancear dois pontos: a capacidade de compressão (taxa), isto é, o tamanho final em bits da imagem comprimida, e a distorção, que é a diferença entre a imagem original e a imagem reconstruída. Essa otimização é representada pela equação:

$$J = D(B) + \lambda R(B),$$

onde  $D(B)$  representa a distorção entre a imagem original e a reconstruída e  $R(B)$  a quantidade de bits usada para representar a imagem,  $\lambda$  é um parâmetro adimensional e  $B$  é o *bitstream* gerado pela imagem.

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. Ou seja, busca-se minimizar  $J$ . 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 erro médio quadrático (MSE)<sup>1</sup>:

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

A princípio, uma maior quantidade de bits implicaria numa distorção  $D(B)$  menor, mas resultaria numa taxa  $R(B)$  maior. Contudo, esse é na verdade um problema intratável, como mostrado por [?]. Para uma dada taxa, existem diversas representações com distorções melhores e piores.

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 [9], é necessário mais do que 20

---

<sup>1</sup>MSE também é bastante utilizada como função de custo para modelos de regressão

MB para armazenar ou 160 Mb para transmitir [9]. 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.

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 [10]. Mais recentemente, o algoritmo WebP [11] 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 [12].

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 (baseado em transformada discreta de cosseno) e sem perdas (método preditivo). Para criar um arquivo JPEG, primeiro a imagem é convertida para outro espaço de cor: o *YCbCr*. Esse 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, visto que o sistema visual é capaz de discriminar o brilho de uma imagem com muito mais sensibilidade do que sua informação de cor<sup>2</sup>. Isto significa que os valores dos componentes de luminância precisam de muito mais fidelidade que os componentes de crominância.

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 mais rápido a essas 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* (pequeno 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

---

<sup>2</sup>Existem cerca de 120 milhões de bastonetes (células estimuladas pelo brilho) distribuídos sobre a superfície da retina [?] contra apenas 6 à 7 milhões de cones (células estimuladas pela cor). Essa 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

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 pode conter informação de alta frequência que é mais difícil de ser comprimida por estes algoritmos. Nos últimos anos, redes neurais profundas se tornaram a base dos resultados do estado da arte para categorização de imagens [13], detecção de objetos [14], reconstrução tridimensional de objetos [15], reconhecimento de faces [16], reconhecimento de discurso [17], tradução automática [18], geração de legendas de imagens [19], tecnologia de carros autônomos [20], entre outros.

É natural buscar usar essa poderosa classe de métodos para melhorar a tarefa de compressão de imagens, especialmente para 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* [21]. Alguns resultados já existentes para compressão com perdas usando autoencoders se mostraram promissores: [22, ?, ?], e redes neurais já atingiram o estado da arte para compressão sem perdas [23, 24].

## 1.2 Hipóteses

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. Serão verificadas as seguintes hipóteses:

- Se modelos baseados em *autoencoders* convolucionais recorrentes são competitivos com os clássicos codecs *JPEG* e *JPEG2000* para imagens com muito conteúdo de alta frequência;
- Se o codificador de entropia utilizado no latente será capaz de comprimir em proporções semelhantes para todos as redes de todos os níveis;
- Se ao usar imagens, no treinamento, que codificadores clássicos têm dificuldade para comprimir é benéfico para os resultados dele;
- Se há um grande impacto nos resultados ao usar funções de custo variadas;

Estas verificações serão dadas usando-se métricas visuais objetivas e as imagens serão trabalhadas a baixas taxas de bits por pixel.

## 1.3 Objetivos

O objetivo deste trabalho é construir e avaliar o desempenho de um *framework* de compressão de imagens ponta a ponta (modelagem e codificação) usando *autoencoders* empilhados convolucionais recorrentes com o auxílio de um codificador de entropia para comprimir o latente binarizado gerado pelo *encoder*.

Para isto foram 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 *autoencoders* que estendam a estrutura básica de um *autoencoder*, gerando uma representação binária para a imagem ao quantizar a camada de gargalo ou as variáveis latentes correspondentes. Esta representação binária será ainda comprimida usando um codificador de entropia.

Além disso serão construídas bases de dados próprias, a partir de bases de dados comumente utilizadas para esse problema, para que seja avaliado o desempenho, a diferentes taxas, de *autoencoders* convolucionais e dos codecs *JPEG* e *JPEG2000* nas mais variadas imagens.

## 1.4 Resultados Esperados

Espera-se obter desempenho superior aos métodos de compressão *JPEG* e *JPEG2000* para imagens com muito conteúdo de alta frequência, visto que estes métodos assumem que baixas frequências são mais visualmente relevantes. Além disso, visto que existem trabalhos semelhantes na literatura acredita-se que será obtido melhor desempenho com o uso de imagens mais difíceis de comprimir para treinamento e com o uso de modelos recorrentes.

# Capítulo 2

## Trabalhos similares e conceitos fundamentais

Este capítulo descreve conceitos fundamentais de codecs clássicos e trabalhos similares utilizadas como base para este trabalho.

### 2.1 Métodos clássicos de compressão

Arquivos comprimidos pelo clássico codec JPEG normalmente são descritos no formato JFIF, que é uma limitação,(um subconjunto) 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<sup>1</sup>, 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 de modo que para cada 4 pixels Y da imagem só terá 1 pixel Cb e 1 Cr no *bitstream* correspondente da imagem codificada. Assim, quando o *decodificador* do *codec* lê o *bitstream* que representa a imagem comprimida, os pixels restantes serão inferidos e não necessariamente serão

---

<sup>1</sup>O método completo JPEG permite que qualquer espaço de cor (organização específica de cores que permite representações de cores) seja usado, mas a maior parte das pessoas seguiram com o espaço de cor definido no padrão JFIF por praticidade.

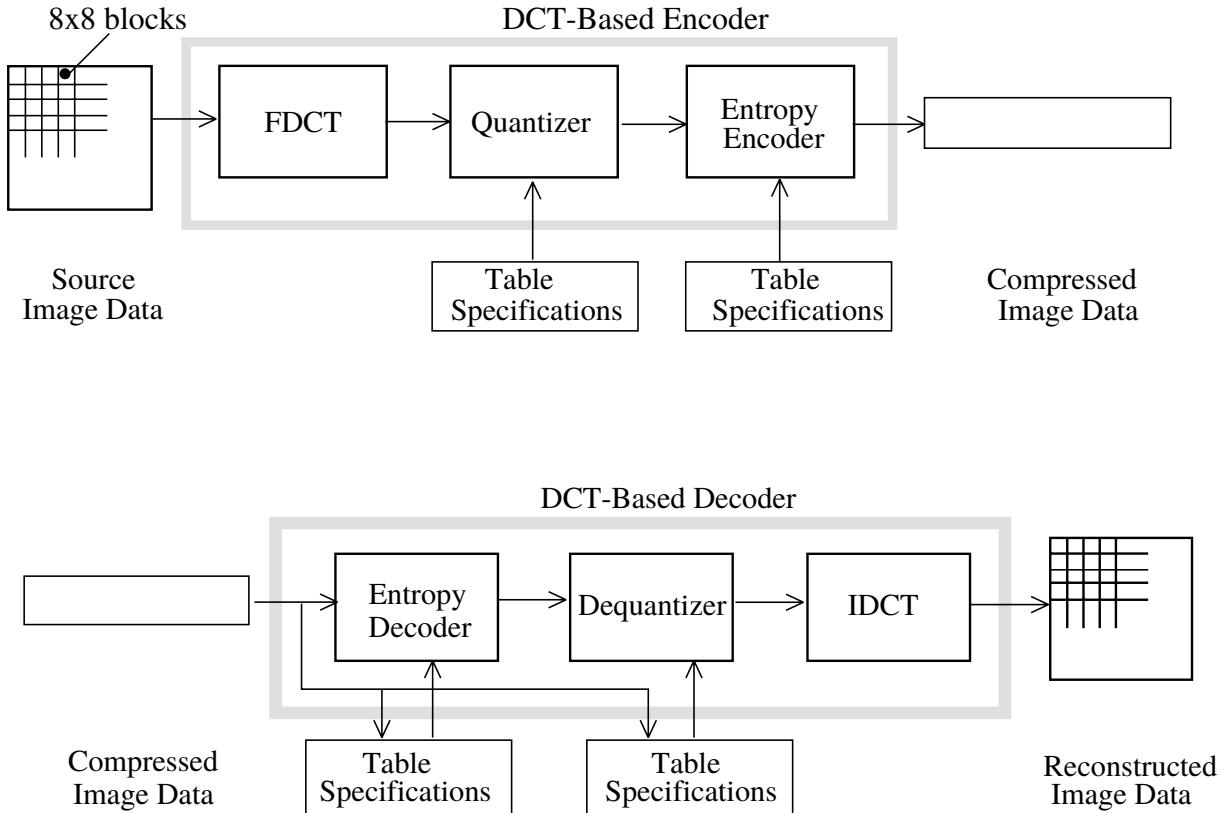


Figura 2.1: Diagram do método de compressão JPEG. Fonte: [1].

iguais aos originais. 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, mesmo a codificação da imagem da direita (*bitstream*) possuindo 64 vezes menos pixels de crominância (Cb e Cr) do que a imagem da esquerda. Entretanto, olhando a Figura 2.3 é possível notar certa discrepância nas bordas da arara vermelha.

### 2.1.2 Aplicação da transformada direta de cossenos

Nesse passo 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 DCT [25]. 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. Cada um dos blocos serão transformados separadamente e podem ser exatamente repli-



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 ao codificar 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 exibi-la). Fonte: [2].



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 (direita) cuja codificação (*bitstream*) possui 64 vezes menos pixels de crominância (Cb e Cr) menos cor da imagem original.

cados por ondas de cossenos 8 por 8, onde varia-se frequências e amplitudes de cada uma delas.

A representação de um sinal em DCT tende a ter maior parte da sua energia (definimos energia de um sinal como sendo:  $\sum_{k=-\infty}^{\infty} |x_k|^2$ ) concentrada em um número menor de coeficientes quando comparado com outras transformações como a transformação discreta de *Fourier* (DFT), o que é mais desejável para algoritmos de compressão. Além disso, a DFT tem coeficientes complexos, o que dobra a quantidade de bits necessários para a representação do sinal.

Basicamente, na transformada discreta de cossenos são calculados 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. 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.

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 denominado 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.

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 perto 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.

### 2.1.3 Quantização

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.

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

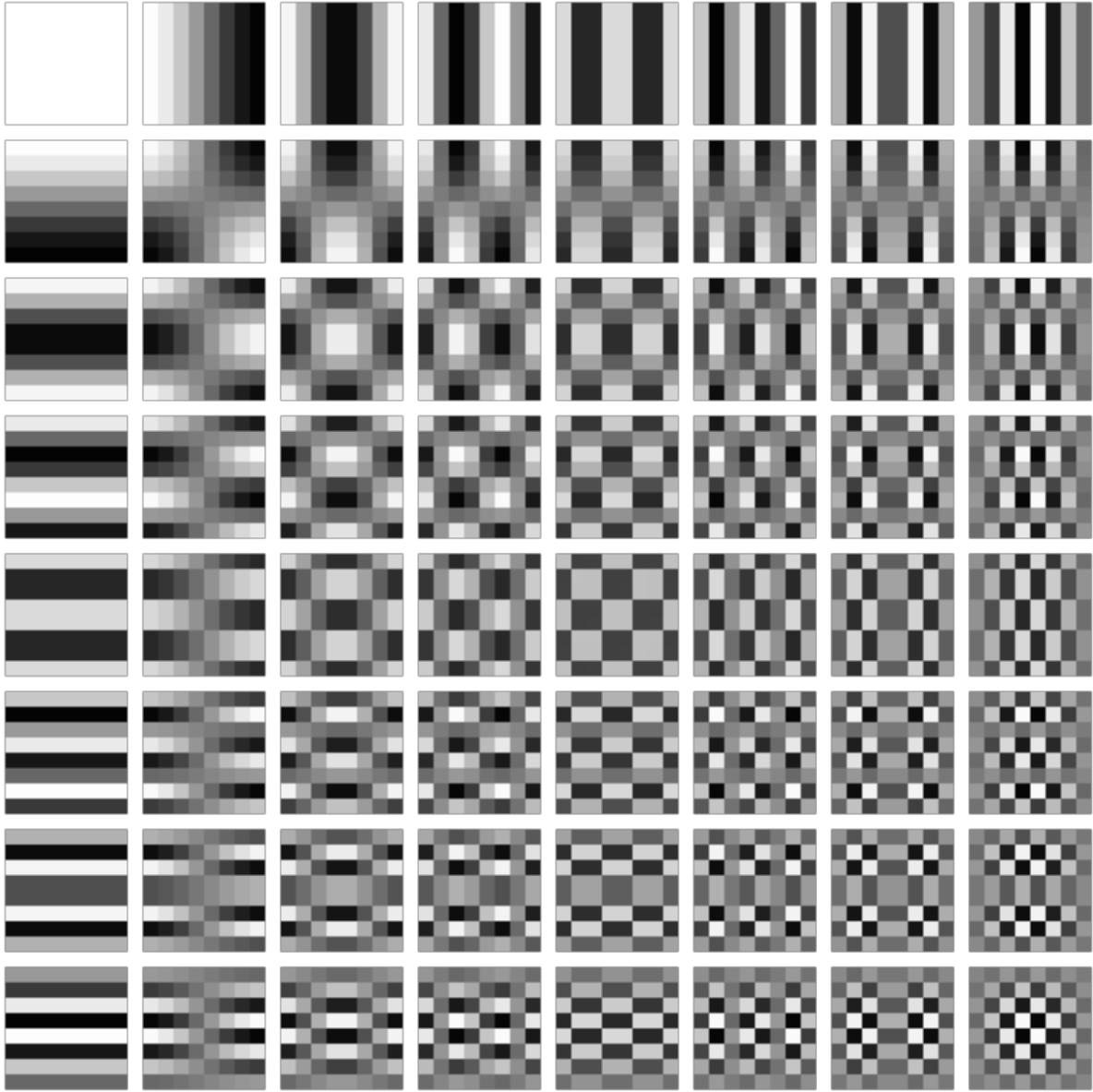


Figura 2.4: 64 (8 por 8) ondas base de cossenos com frequências variadas. Fonte: [3].

em que há a maior perda de informação na imagem. É dada pela seguinte equação:

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

onde  $F^Q(u, v)$  será o novo valor do coeficiente,  $F(u, v)$  é o valor atual e  $Q(u, v)$  é o passo de quantização que controla a qualidade da imagem definido para a 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



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

elementos, que é definida pelo nível de qualidade escolhido para a aplicação.

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.

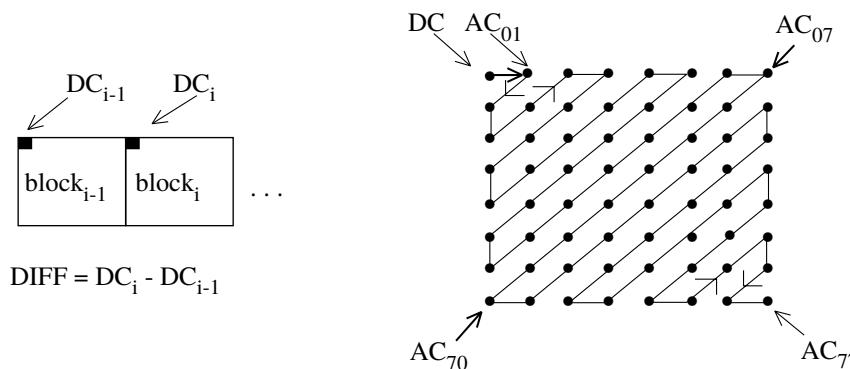


Figura 2.6: Sequência zig-zague usada para melhorar codificação. Fonte: [1].

Vale ressaltar que existe apenas um controle de qualidade pelas matrizes de quantização e não uma otimização da taxa-distorção  $J$ .

#### 2.1.4 Codificação

O passo final do codificador é a codificação de entropia responsável por gerar o *bitstream* comprimido que representará a imagem. 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 [26] e Codificador Aritmético [27].

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. Posteriormente, essa matriz é transformada usando a IDCT que resultará na matriz no espaço de cor  $YCbCr$ .

O *JPEG2000* tem um funcionamento geral similar ao *JPEG*, entretanto é utilizado uma transformada discreta *wavelet* no lugar da DCT. Esta transformada é aplicada na imagem inteira, o que elimina o efeito de *blocking* causado pelo *JPEG* mas causa o efeito de *ring*.

## 2.2 Redes Neurais

Algumas tarefas em inteligência artificial podem ser resolvidas desenvolvendo características a serem extraídas dos dados. Entretanto, para muitas tarefas é difícil saber quais *features* (informações incluída nas representações dos dados) 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 algoritmos comumente usados em *deep learning* capazes de fazer previsões aprendendo uma função que relaciona as características dos dados a respostas observadas/desejadas. Elas aprendem essa função ao tentar minimizar a função de custo/erro (*loss*) definida para a aplicação. 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) [28]. 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] [5].

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 facilmente otimizável com métodos comumente usados como descida de gradiente [29]. Métodos de descida de gradiente são métodos que atualizam 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.

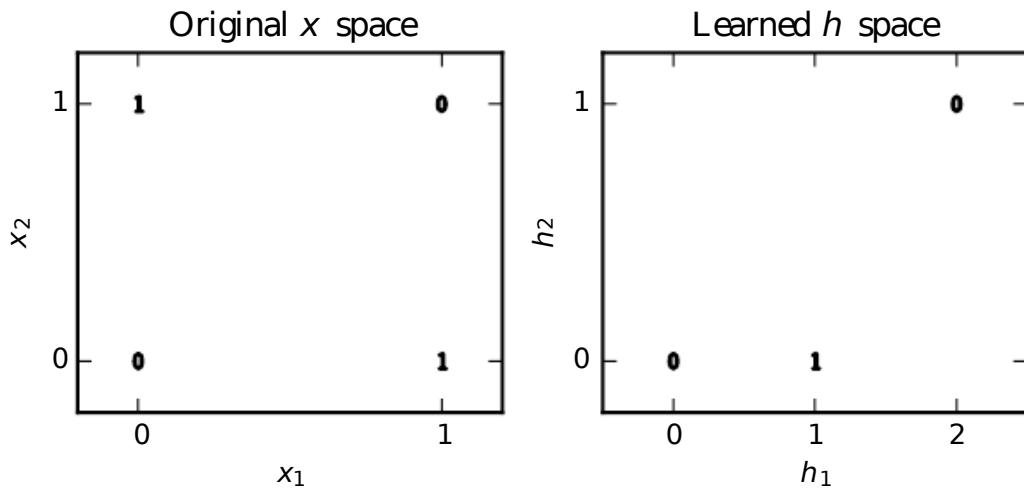


Figura 2.7: Um modelo linear aplicado diretamente à entrada original não pode implementar a função XOR, visto que o espaço da função XOR não é linearmente separável. Para isso é necessário transformar o espaço original usando uma função de ativação. Fonte: [5].

### 2.2.1 Taxa de aprendizagem (*learning rate*)

A *learning rate* ( $l$ ) é um hiperparâmetro que controla o quanto nós ajustamos os parâmetros aprendíveis da nossa rede com respeito ao gradiente  $g$ . Quanto menor o seu valor, menor será a influência do gradiente obtido em direção à menor 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, visto que os novos pesos

$n$  serão calculados através da seguinte equação:

$$n = o - lg \quad (2.2)$$

, onde  $o$  representa os pesos antigos.

O gráfico Figura 2.8 mostra os diferentes cenários de *learning rate* e como ela afeta o treinamento. Leslie N. Smith argumenta em [7] que é benéfico para a aprendizagem variar

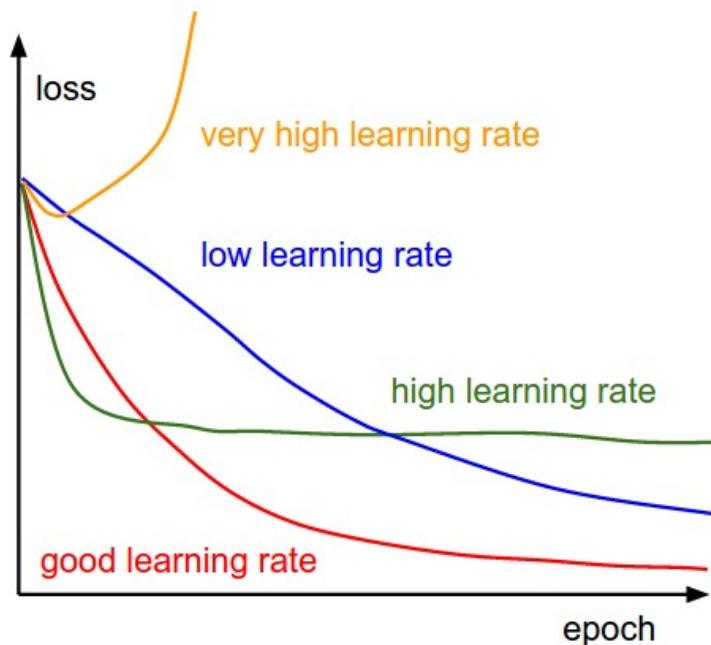


Figura 2.8: Efeitos de várias taxas de aprendizagem no treinamento. Fonte: [6].

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 fenômeno ficou conhecido como “superconvergência”).

Smith 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  $\gamma$  que representam, respectivamente, *learning rate* inicial, *learning rate* máxima, número de iterações correspondente a 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:

$$lr = baselr + (maxlr - baselr)\max \max(0, 1 - |itr/step - 2cycle + 1|)\gamma^{itr}. \quad (2.3)$$

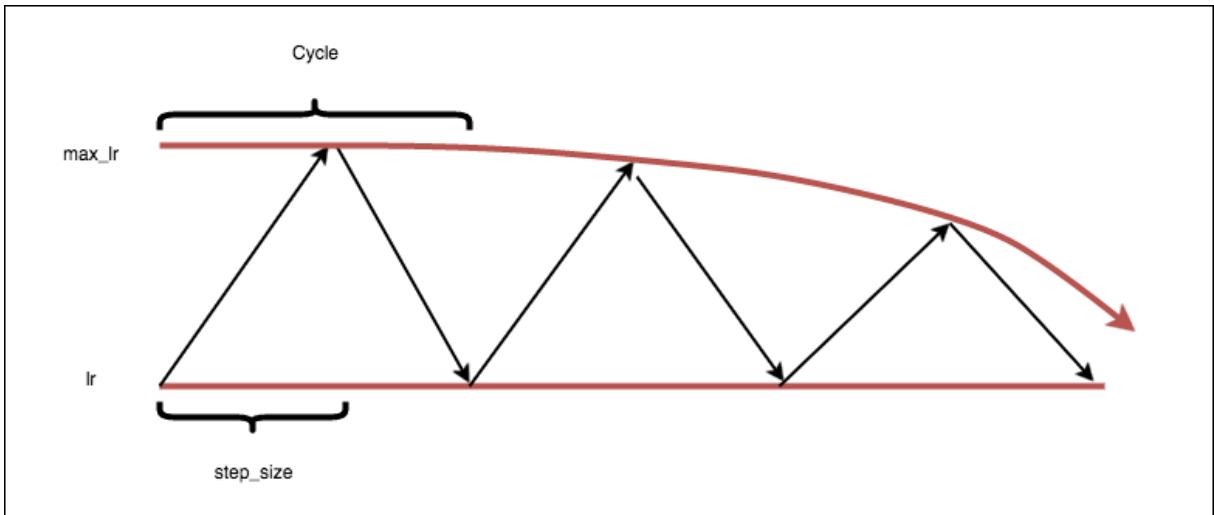


Figura 2.9: Política *exp\_range* de *learning rate* cíclica. Fonte: [7].

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* [30] 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. Ele é um método mais caro, portanto não há necessidade de usar políticas de *learning rate* adaptativas com ele.

## 2.2.2 Redes Neurais Convolucionais

Em *deep learning* (aprendizado profundo), uma rede neural convolucional (CNN) [31] 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. 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 a estímulos apenas em uma região restrita do campo visual que é conhecida como campo receptivo. Uma coleção de sobreposição desses campos cobre toda a área visual [32].

Uma CNN é capaz de capturar dependências 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 a 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. Essa extração de características é feita por meio da aplicação de filtros<sup>2</sup> no domínio do espaço denominados convoluções.

### 2.2.3 Autoencoders

Um Autoencoder (AE) é uma técnica de aprendizado não-supervisionado na qual pode-se utilizar redes neurais para a tarefa de aprendizado de representações. Um *autoencoder* é formado 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  $\mathbf{x}$ ,  $f(\mathbf{x}) = h$ , onde  $\mathbf{x}$  é a imagem original.
- 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  $h$ ,  $g(h) = \hat{\mathbf{x}}$

Uma rede do tipo AE pode ser descrita como  $g(f(\mathbf{x})) = \hat{\mathbf{x}}$ . Normalmente, o objetivo é apenas diminuir a diferença entre  $\mathbf{x}$  e  $\hat{\mathbf{x}}$  (nesse caso, a função de custo a ser minimizada normalmente é a  $MSE(\mathbf{x}, \hat{\mathbf{x}})$  [Equação 1.1]). A camada entre o *encoder* e o *decoder*, onde os dados de entrada são representados em uma dimensionalidade menor [Figura 2.10] e que força o *encoder* a comprimir informação da representação original gerando o espaço latente, é denominada *bottleneck* (camada de gargalo). Considera-se que esta camada faz parte do *encoder*.

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,

---

<sup>2</sup>Filtro se refere a aceitar ou rejeitar certos componentes de frequência no domínio da frequência []

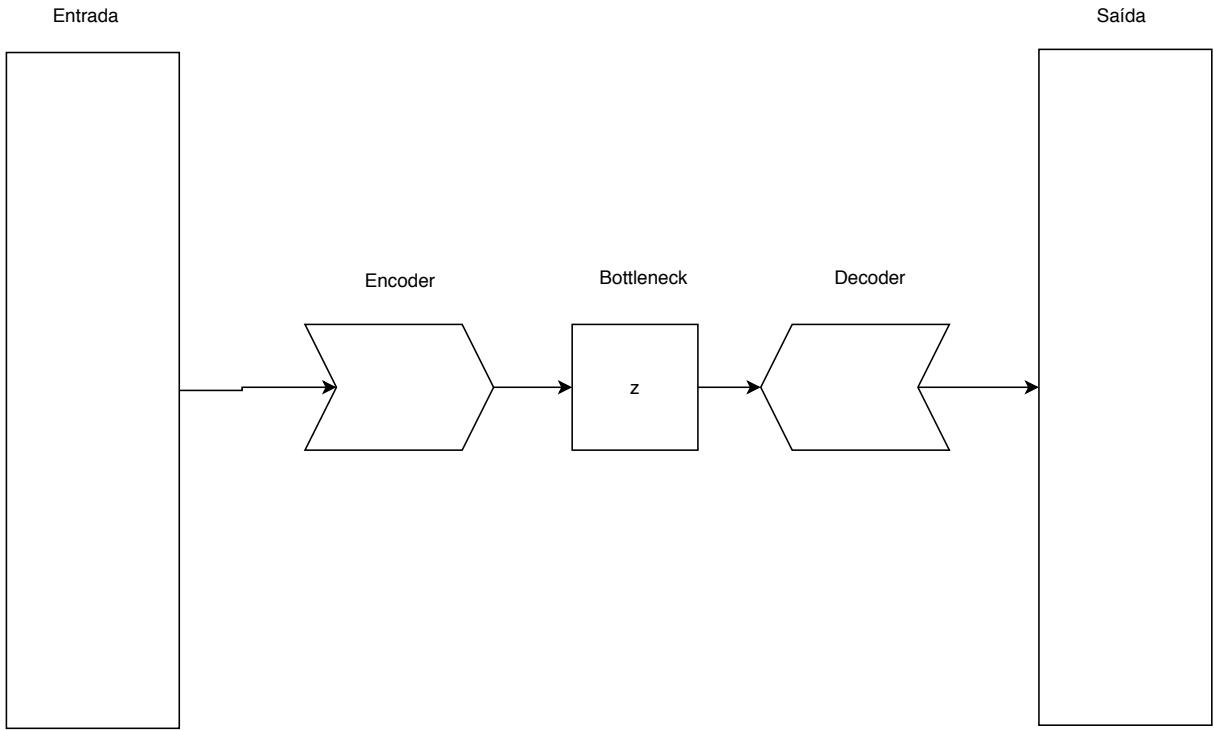


Figura 2.10: Ilustração de um autoencoder.

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. Como a entropia é dada pela fórmula  $H = -\sum_{i=1}^N P(i)\log_2 P(i)$ , onde  $N$  é o comprimento da sequência gerada por um alfabeto e  $P(i)$  é a probabilidade que a sequência  $i$  ocorra, o binarizador é necessário pois os códigos práticos precisam ter entropia finita ( $P(X = X_i)$  para um variável aleatória  $X$  contínua é 0, o que faz com que o logaritmo seja “infinito negativo”). Logo valores contínuos precisam ser quantizados para um conjunto finito de valores discretos, o que introduz erro. Com a binarização também é possível 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:

1. *PSNR* (Peak Signal-to-Noise Ratio) definida por

$$20 \cdot \left( \frac{\text{MAX}_I}{\sqrt{\text{MSE}}} \right), \quad (2.4)$$

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* [33]. Seja  $\mathbf{x} = \{x_i | i = 1, 2, \dots, N\}$  e  $\mathbf{y} = \{y_i | i = 1, 2, \dots, N\}$  dois sinais discretos não negativos e  $\mu_x, \sigma_x^2$  e  $\sigma_{xy}$  serem a média de  $\mathbf{x}$ , a variância de  $\mathbf{x}$  e a covariância de  $\mathbf{x}$  e  $\mathbf{y}$ , respectivamente. A SSIM é dada por:

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}, \quad (2.5)$$

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* [34] é uma forma avançada da SSIM. Ela é conduzida sobre múltiplas escalas através de um processo de múltiplos estágios de processamento em imagens com dimensionalidade reduzida.

### Compressão de Imagens com Taxa Variável usando Redes Neurais Recorrentes

Nesse trabalho [?] é feita uma abordagem de ponta a ponta (modelagem e codificação) para compressão de imagens. Esta abordagem é baseada em várias redes neurais empilhadas, especificamente, uma pilha de *autoencoders*, o que possibilita a transmissão de informação incremental. Cada autoencoder comprime sua entrada e tenta reconstruí-la. O erro residual é propagado para o próximo *autoencoder* que, novamente, tenta reconstruir seu resíduo de entrada. Considerando  $E$  como o *encoder*,  $D$  como o decoder,  $B$  como a função de binarização, e  $x$  como a entrada, um *autoencoder* pode ser representado como

$$\hat{x} = D(B(E(x))). \quad (2.6)$$

Esse equação pode ser usada para compor uma pilha de *autoencoders* residuais pelo seguinte conjunto de equações:

$$\begin{aligned} b_t &= B(E_t(r_{t-1})), \hat{\mathbf{x}}_t = D_t(b_t) + \gamma \hat{\mathbf{x}}_{t-1}, \\ r_t &= x - \hat{\mathbf{x}}_t, r_0 = x, \hat{\mathbf{x}}_0 = 0 \end{aligned} \quad (2.7)$$

onde  $t \in \{1, \dots, n\}$ , para um modelo com  $n$  níveis. O par  $(E_t, D_t)$  é o *autoencoder* para o  $t$ -ésimo nível com  $r_{t-1}$  como entrada. A entrada inicial é a imagem original. O binarizador  $B$ , nessa formulação, é o mesmo para todos os níveis de iteração. Usa-se  $\gamma = 1$  para a reconstrução aditiva, de modo que a reconstrução final para um modelo com  $t$  níveis será igual a  $\hat{\mathbf{x}}_t$ , que será a soma das saídas de todos os níveis. Para a reconstrução “one-shot” usada em modelos com memória, usa-se  $\gamma = 0$ .

O trabalho [?] propõe uma função de binarização com quantização em dois passos. O primeiro passo é a geração de valores no intervalo  $[-1, 1]$ . A segunda parte envolve converter esses valores para o conjunto  $\{-1, 1\}$ . Para esse propósito, uma camada completamente convolucional com ativações de tangente hiperbólica é usada para produzir as saídas no intervalo desejado. Em seguida, uma abordagem estocástica é aplicada. Esses passos podem ser representados pelo seguinte conjunto de equações:

$$b(x) = x + \epsilon \in \{-1, 1\}, \quad (2.8)$$

$$\epsilon \sim \begin{cases} 1 - x & \text{com probabilidade } \frac{1+x}{2}, \\ -1 - x & \text{com probabilidade } \frac{1-x}{2}, \end{cases} \quad (2.9)$$

onde  $\epsilon$ , corresponde ao ruído de quantização.

O encoder pode ser summarizado pela seguinte equação:

$$B(x) = b(\tanh(Wx + b)). \quad (2.10)$$

$W$  e  $b$  são os pesos e bias padrões da rede. A quantização é realizada somente na *forward pass*, visto que substituir o arredondamento por uma aproximação completamente pode levar o *decoder* a aprender a inverter essa aproximação, removendo a informação da *bottleneck* (camada de gargalo) que força a rede a comprimir informação. Para a *backward pass* da *back-propagation*, é usada a derivada da esperança de modo que os gradientes serão passados por  $b$  sem mudanças pois a esperança de  $b(x)$  é igual a  $x$ ,  $\forall x \in [-1, 1]$ .

Para ter uma representação fixa para uma entrada, uma vez que a rede esteja treinada  $b$  é substituída por

$$b^{inf}(x) = \begin{cases} -1, & \text{se } x < 0, \\ 1, & \text{caso contrário.} \end{cases} \quad (2.11)$$

## Redes Neurais não-recorrentes

Toderici et al. [?] primeiramente propôs um esquema de compressão com esta estratégia usando diferentes arquiteturas de redes neurais. Esse esquema é baseado em um framework de codificação aditivo que restringe o número de bits de codificação. A Figura 2.11 mostra um exemplo deste empilhamento de *autoencoders* completamente convolucionais. Para essa arquitetura, cada rede é composta de um *encoder*, que produz uma representação latente de 8 bits por pixel e um *decoder* que tenta reconstruir a entrada a partir do latente. Cada camada possui 512 *kernels* com tangente hiperbólica como função de ativação. Os resíduos são as entradas para as próximas redes. Considerando 16

níveis de resíduos, um total de 128 bits é usado, o que dá uma taxa de bits por pixel de  $\frac{2 \cdot 2 \cdot 32}{32 \cdot 32} = 0.125$  para representar a imagem de entrada de tamanho 32x32.

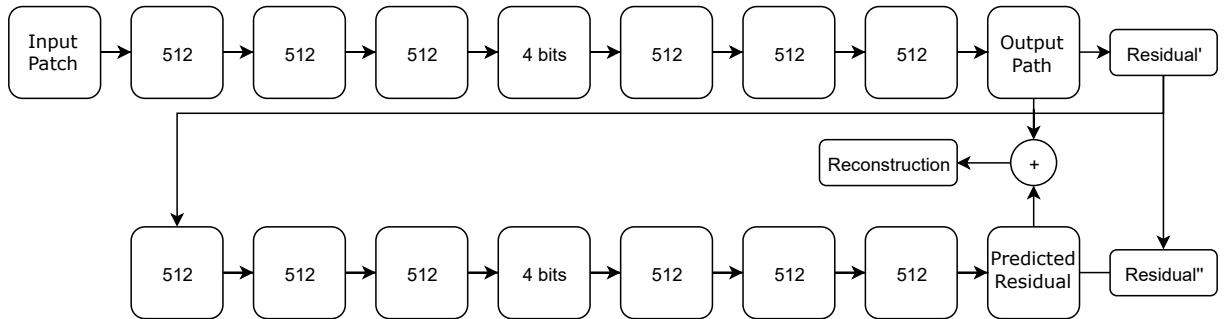


Figura 2.11: Um *autoencoder* residual *fully-connected*. Esta figura mostra uma arquitetura com dois níveis (dois *autoencoders* empilhados). O primeiro nível codifica a imagem original. O resíduo da reconstrução é passado para o segundo nível. Cada nível produz um latente de 4 bits por pixel, que é representado nos retângulos. Os retângulos marcados com 512 são camadas *fully-connected* com 512 unidades e não-linearidades de tangente hiperbólica. Fonte: [?].

Os autores propõe uma versão iterativa dessa rede usando convoluções em vez de camadas *fully-connected*. A Figura 2.12 ilustra essa arquitetura. Nesse caso, são utilizados 2 bits por pixel na *bottleneck* binarizada.

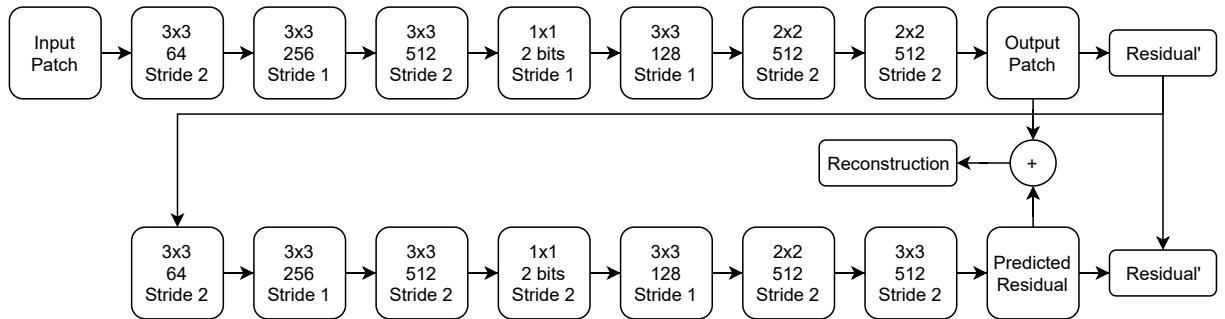


Figura 2.12: O autoencoder residual convolucional. Os retângulos afiados representam as camadas convolucionais, enquanto os arredondados representam as camadas das convoluções transpostas. A dimensão do kernel é apresentado na primeira linha, enquanto a quantidade de canais de saída e o tamanho do *stride* são apresentados nas linhas seguintes. A loss é aplicada nos resíduos. Fonte: [?].

O conjunto de dados é composto de 216 milhões de imagens aleatórias coletadas da internet. Então, 90% destas imagens são usadas para treinamento, enquanto 10% são usadas para teste (não usa-se conjunto de validação devido ao tamanho enorme de imagens usadas para treinamento). Esse conjunto de dados simula o cenário de compressão de miniaturas de imagens.

O treinamento foi realizado com o otimizador Adam usando uma função de custo normalizada e variando com as seguintes *learning rates* 0.1, 0.3, 0.5, 0.8 e 1. O número de níveis varia de 8 à 16. A SSIM é usada para avaliar o desempenho em tempo de teste.

## Redes Neurais recorrentes

Além das operações usadas pelas redes não recorrentes, a abordagem recorrente usa camadas recorrentes. Uma rede neural recorrente (RNN) é um tipo de rede neural com memória para salvar informação sobre entradas passadas. Para isto, estas unidades de memória possuem conexões para elas mesmas. Esta informação temporal muda o comportamento da camada para a entrada atual [?].

Para manter informação temporal, muitas implementações das camadas foram propostas. Uma das primeiras abordagens efetivas se espelham na *Long Short-Term Memory (LSTM)*.

A camada LSTM possui a seguinte formulação, onde  $\mathbf{x}_t$ ,  $\mathbf{c}_t$  e  $\mathbf{h}_t$  denotam a *input*, *cell* e *hidden states* na iteração  $t$  [?]:

$$\begin{aligned} [f, i, o, j]^\top &= [\sigma, \sigma, \sigma, \tanh]^\top ((Wx_t + Uh_{t-1}) + b) \\ c_t &= f \odot c_{t-1} + i \odot j \\ h_t &= o \odot \tanh(c_t), \end{aligned} \tag{2.12}$$

onde  $\odot$  é a multiplicação elemento a elemento,  $b$  é o bias,  $\sigma$  é a função sigmoide,  $i$  é a *input gate*,  $o$  é a *output gate*,  $f$  é a *forget gate* e  $j$  é a *input modification gate*. A saída da camada é  $h_t$ .  $W$  e  $U$  são transformações lineares convolucionais.

Considerando as camadas de memória descritas por essas equações, a abordagem recursiva pode ser summarizada usando a seguinte formulação [?]:

$$\begin{aligned} b_t &= B(E_t(r_{t-1})), \hat{x}_t = D_t(b_t) \\ r_t &= x - \hat{x}_t, r_0 = x, \hat{x}_0 = 0, \end{aligned} \tag{2.13}$$

onde  $D_t$  e  $E_t$  são o *encoder* e o *decoder* com os estados na iteração  $t$ , e  $b_t$  é a representação binária progressiva.  $\hat{x}_t$  é reconstrução “one-shot” e  $r_t$  é o resíduo entre  $\mathbf{x}$  e a construção  $\hat{x}_t$ . Neste caso,  $B$  irá produzir uma representação binarizada de tamanho fixo.

No trabalho posterior, Toderici et al. [?] mostrou em seus resultados que o efeito de usar um conjunto de treinamento de alta entropia é benéfico para a rede, dada a importância de treinar modelos de *machine learning* com exemplos difíceis. Em processamento de imagens entropia diz respeito à quantidade de informação na imagem (medida da quantidade de informação gerada pela fonte, do ponto de vista de processamento de sinais). Alta entropia pode ser visto como maior variância nos valores dos pixels. No seu trabalho, ele definiu

o conjunto de alta entropia como sendo o conjunto formado por imagens que o PNG tem mais dificuldade de comprimir, ou seja, os arquivos no formato PNG com o maior número de bytes.

# Capítulo 3

## Metodologia

Este capítulo apresenta a metodologia seguida para a verificação das hipóteses, incluindo o modelo construído e os *datasets* utilizados. Os modelos foram construídos usando o *framework* PyTorch [35] e avaliados usando métricas visuais: PSNR, SSIM e MS-SSIM. Também foi avaliada a quantidade de bits por pixel da imagem decodificada.

### 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* [8]. 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* [36]. Deste *dataset* foram pegos 2 conjuntos com os seguintes nomes e tamanhos:
  - (a) Train: 800 imagens;
  - (b) Valid: 100 imagens;
3. *EYE* [37]. 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*.

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  $BD_i$ ,  $i \in \{0, \dots, 4\}$ , 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 de acordo com o *patch* com maior entropia);
- **BD2:** formada por 1248725 de *patches* que pertencem ao grupo dos 20% com maior entropia;
- **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% dos *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.

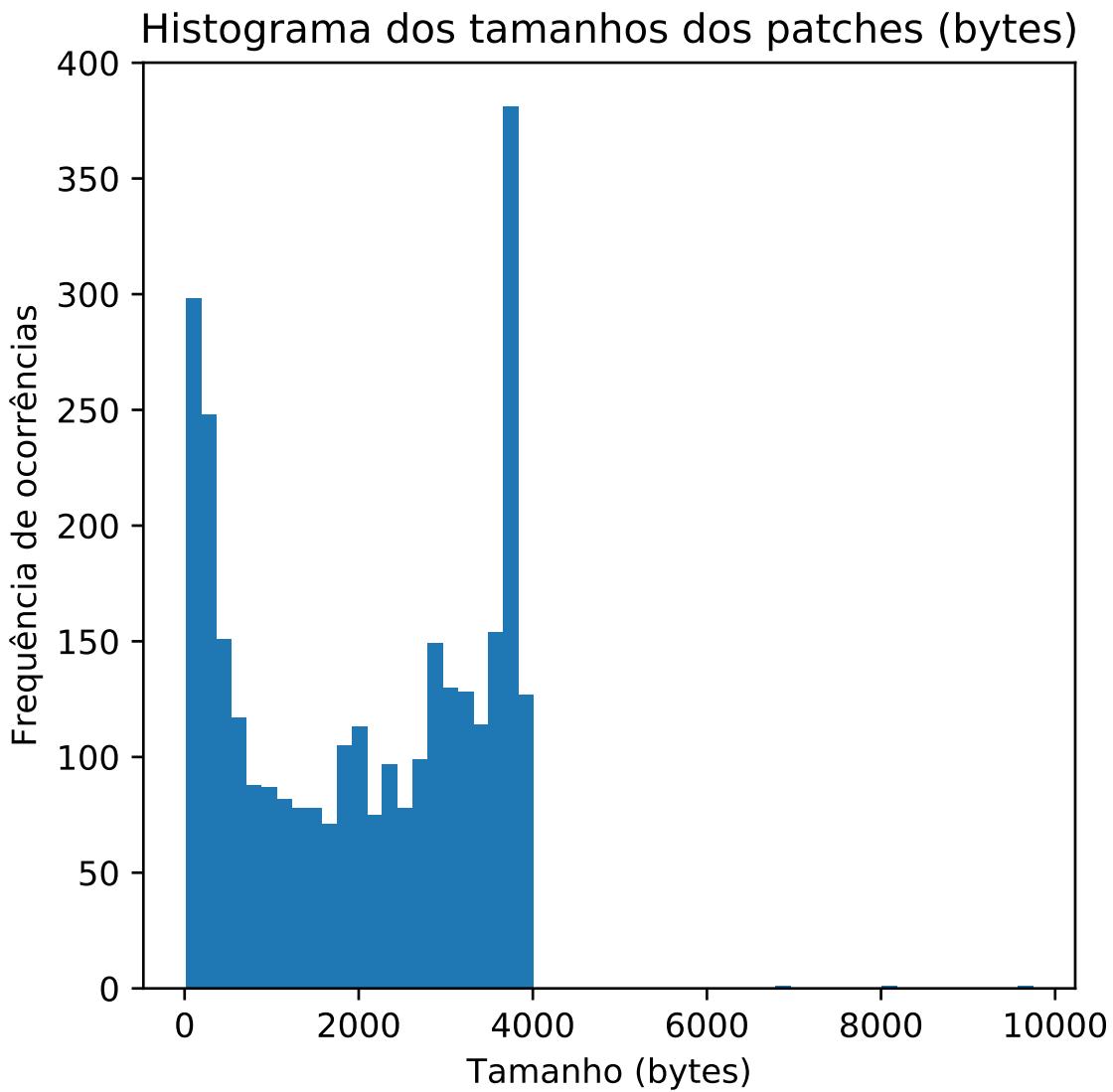


Figura 3.1: Histograma da base de dados completa formada por 6,231,440 de patches.

## 3.2 Modelos desenvolvidos

Foram testados *autoencoders* convolucionais com o objetivo de avaliar o potencial de cada um nas bases de dados propostas. O primeiro não possui binarização/quantização 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 Modelo 3 corresponde ao modelo não recursivo convolucional usando apenas um nível de resíduo. Os últimos são os modelos convolucionais recursivos propostos por *Toderici* em [?] com a utilização de  $\hat{b}(x)$  e de um codificador de entropia, de modo que se alguma redundância ainda for encontrada

### Histograma dos tamanhos dos patches (bytes)

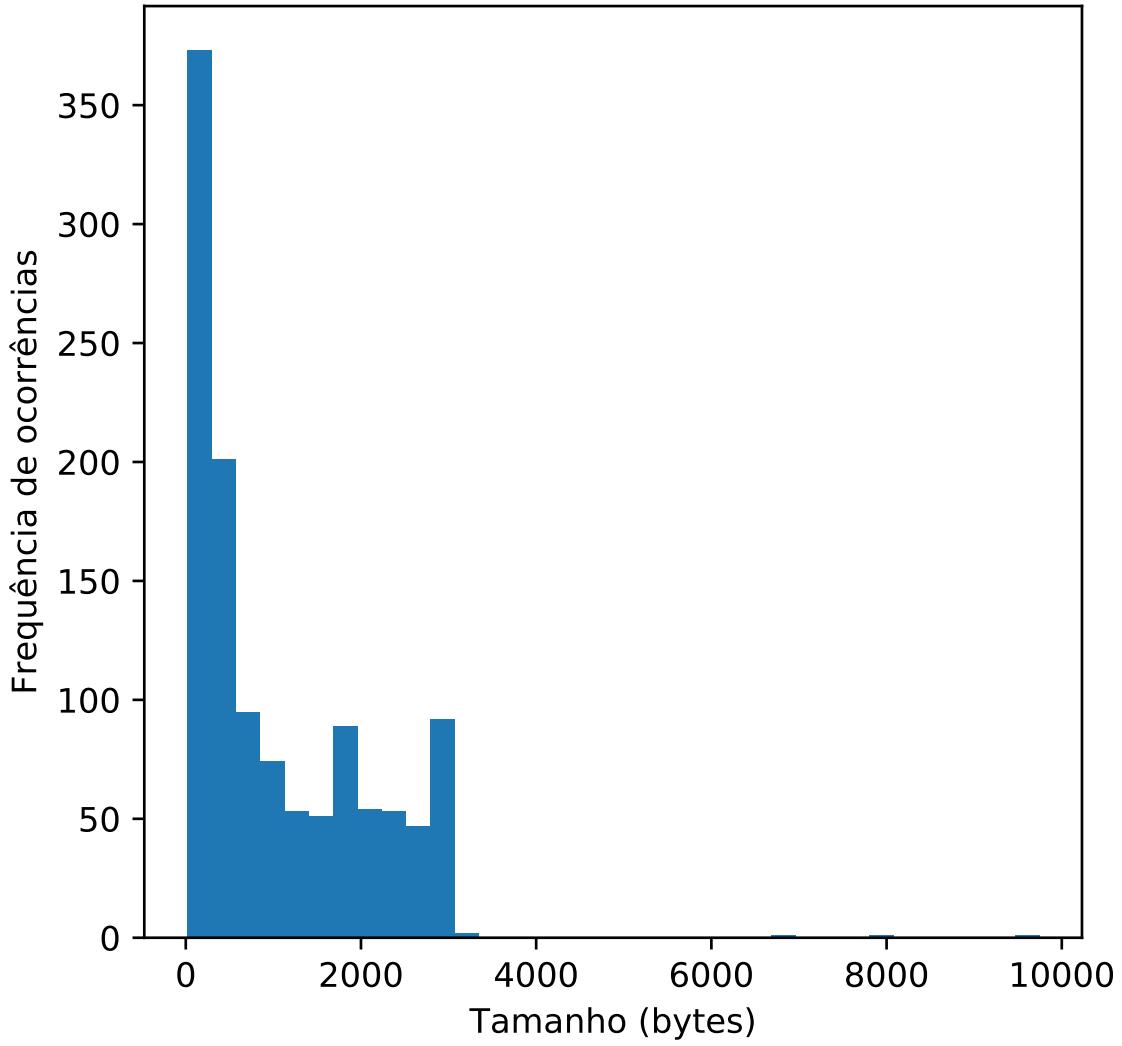


Figura 3.2: Histograma da BD0.

no latente a lógica é que o codificador irá explorá-la de alguma forma, reduzindo o tamanho do *bitstream* comprimido. Os Modelos 4 em diante são os recursivos convolucionais usando LSTM. Os Modelos 5, 6 e 7 são comparados utilizando *K-fold Cross-Validation* que é uma técnica onde o *dataset* é dividido em  $k$  grupos e cada grupo único  $x_i, i \in 1 \dots k$  é usado como conjunto de teste para os outros  $k - 1$  grupos usados como treino nesta rodada.

Pode-se pensar que estes últimos modelos realizam o trabalho de transformação e quantização, empacotando e descartando informação, e que o codificador de entropia irá finalizar o trabalho, explorando alguma redundância ainda não explorada.

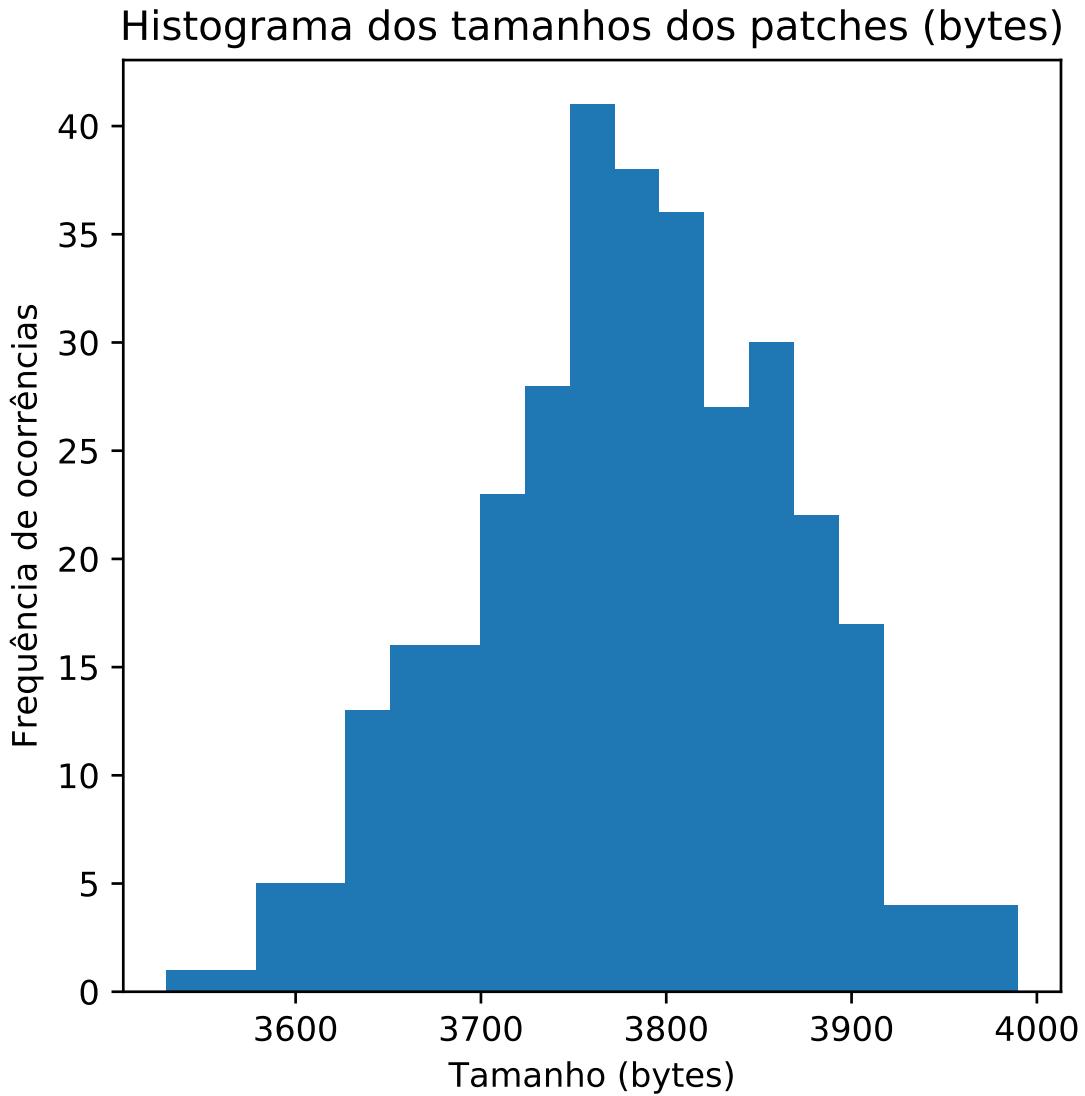


Figura 3.3: Histograma da BD1.

### 3.2.1 Formando o Bitstream

É importante descrever como o *bitstream* é formado quando se aplica o binarizador  $B$ . Em todos os modelos, a imagem é dividida em *patches* de tamanho 32x32 *pixels*. Assim, se uma imagem possuir 9 *patches*, ela terá um *bitstream* para cada latente de cada *patch* e o *bitstream* da imagem será dado pela concatenação dos *bitstreams* de cada *patch*.

Para cada um desses *patches*, o modelo é treinado de maneira residual conforme explicado no capítulo anterior. Para os modelos que usam mais de 1 nível de resíduo, a taxa nominal é acrescida de 0.125 bits por pixel para cada nível.

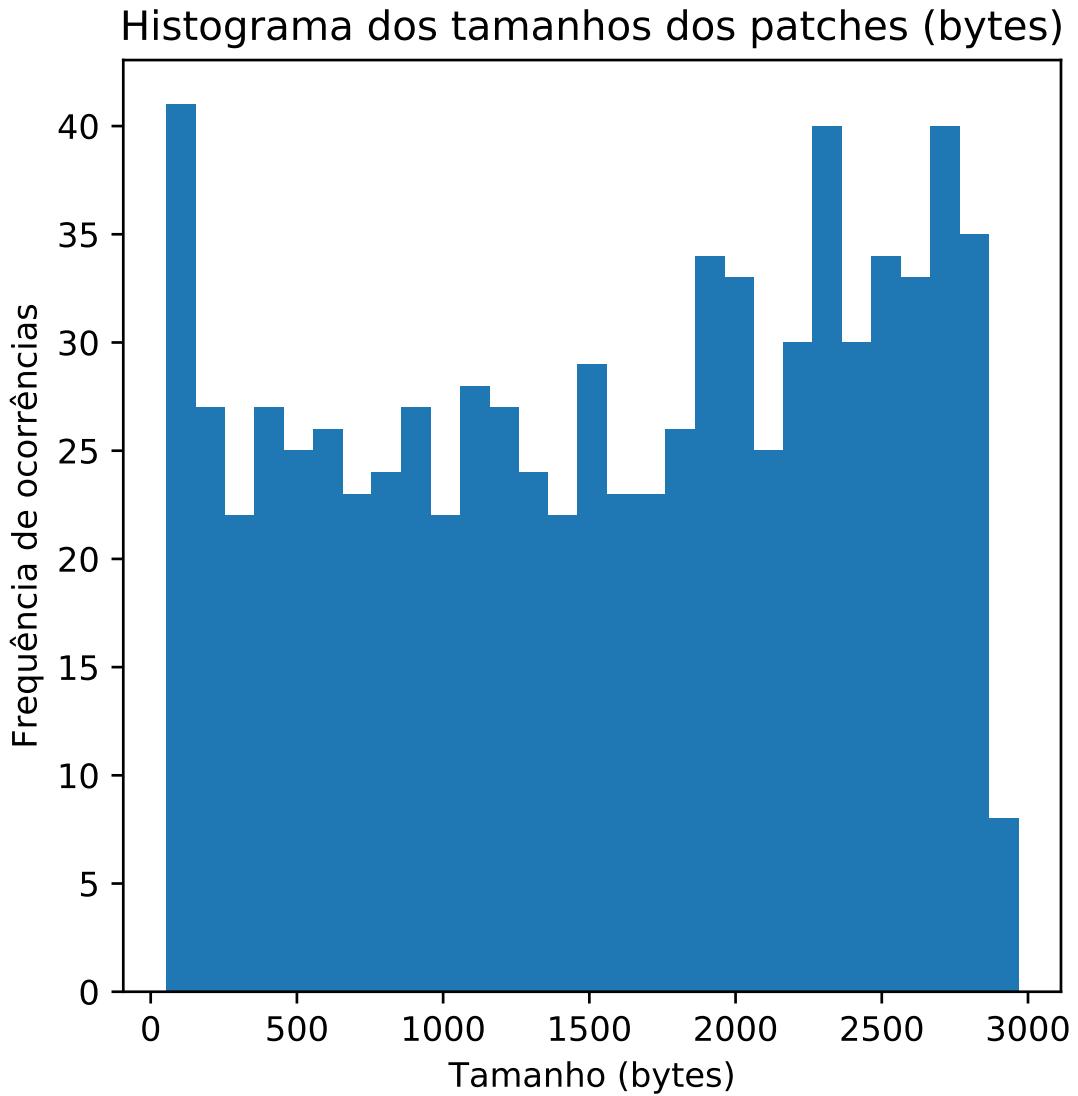


Figura 3.4: Histograma da **BD2**.

Os latentes são concatenados de modo que para o nível  $n$  tem-se  $n$  latentes binarizados concatenados. Assim, para o nível 1 tem-se o *bitstream*  $b_1$  correspondente ao latente binarizado do nível 1, para o nível 2 tem-se o *bitstream*  $b_1 + b_2$ , e assim em diante. De maneira geral, para o nível  $n$  tem-se  $b_1 + b_2 + \dots + b_n$ , onde  $+$  denota a concatenação de *bitstreams*.

Do modelo 4 em diante foi usado o *gzip* (codificador de entropia) no latente com o objetivo de reduzir a taxa de bits por pixel.

Nas seguintes subseções são apresentadas as ilustrações de algumas arquiteturas utilizadas. Os retângulos indicam as convoluções e os retângulos arredondados indicam as

### Histograma dos tamanhos dos patches (bytes)

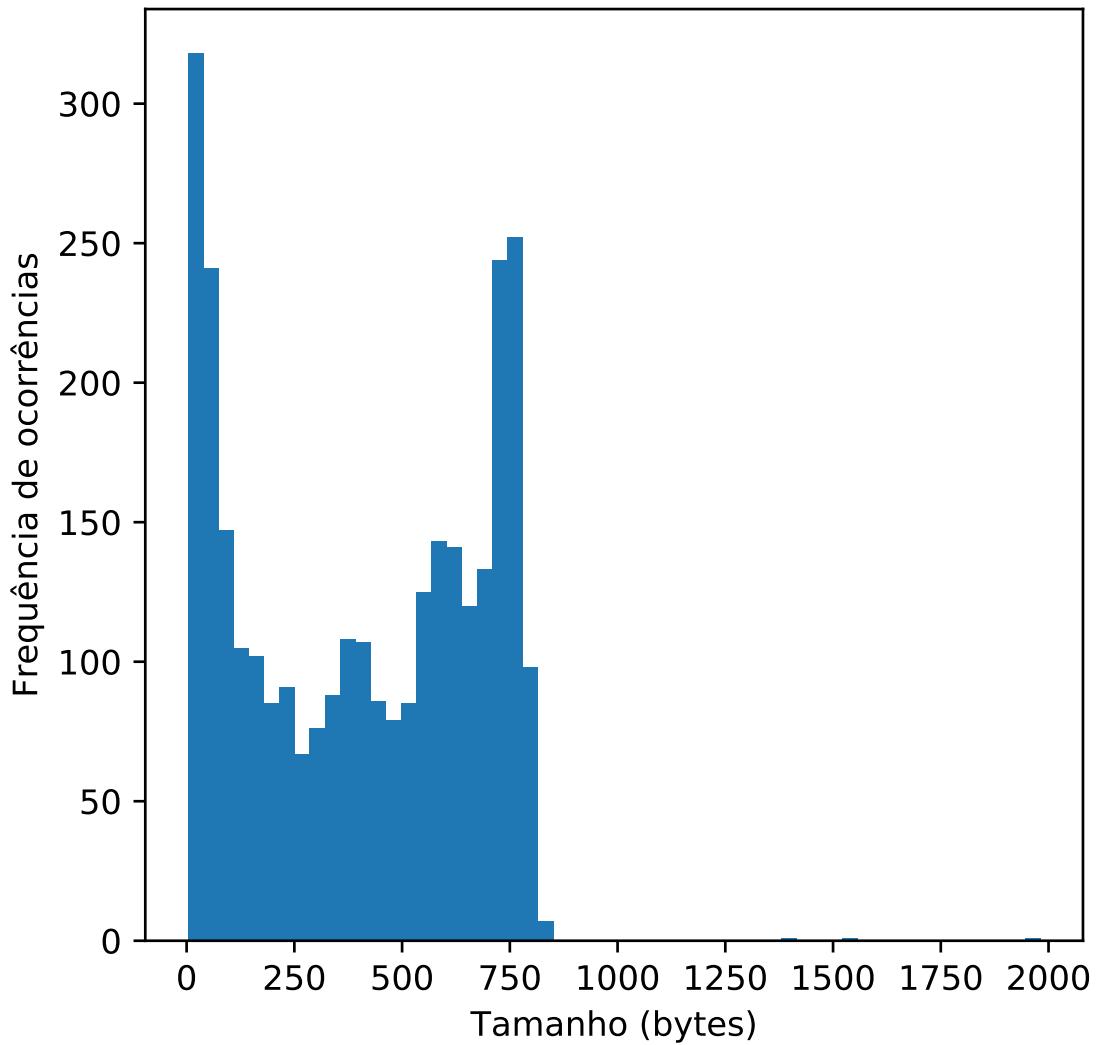


Figura 3.5: Histograma da **BD3**.

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.

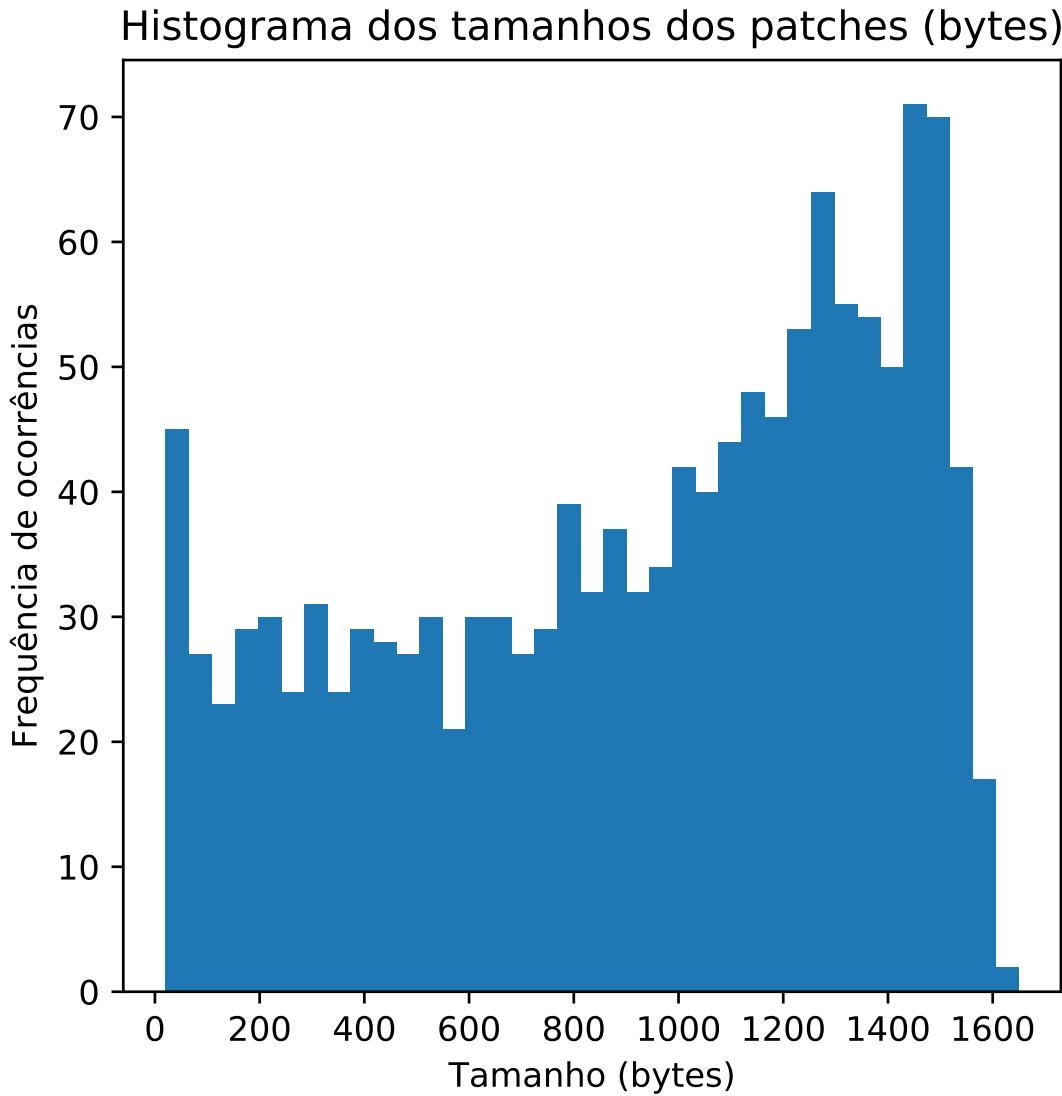


Figura 3.6: Histograma da BD4.

### 3.2.2 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.

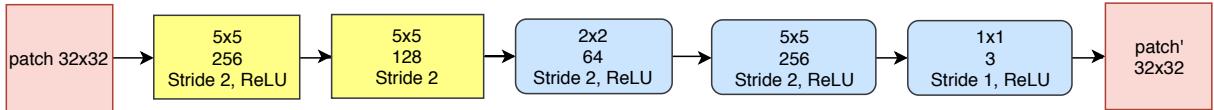


Figura 3.7: Ilustração do *autoencoder* mais básico desenvolvido.

### 3.2.3 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  $\frac{8 \cdot 8 \cdot 128}{32 \cdot 32} = 8$ .

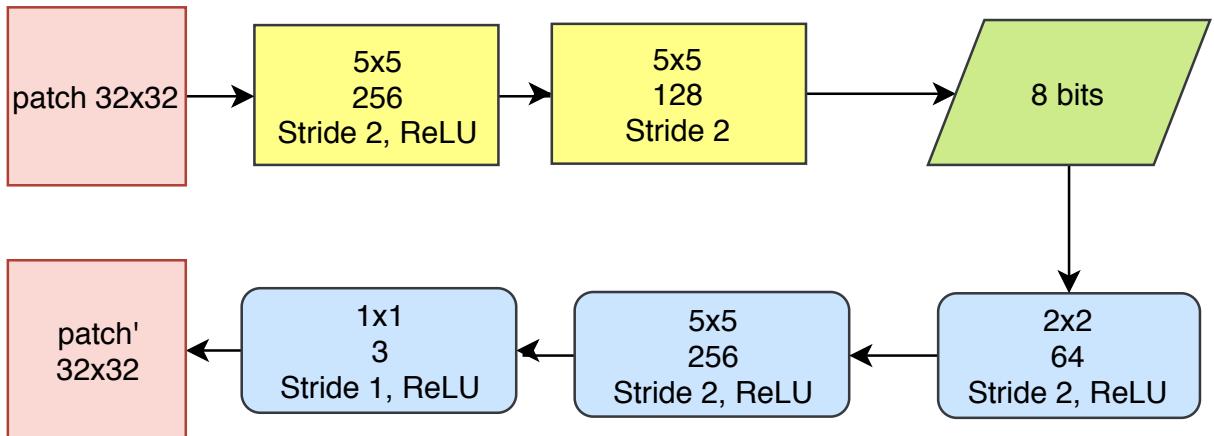


Figura 3.8: Ilustração do segundo modelo desenvolvido.

# Capítulo 4

## Experimentos e Resultados

Foram realizados vários experimentos com os modelos apresentados no capítulo anterior e com o **JPEG** e **JPEG2000** 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, 4.4, 4.5. Os três primeiros modelos apresentados possuem um único nível de resíduo. Todas as taxas estão em bits por pixel.

### 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	Taxa	PSNR	SSIM	MSSIM	Quality
<b>CLIC Mobile test (patches 32)</b>	8	44.23	0.98	0.99	94.06
<b>CLIC Mobile test</b>	2	39.58	0.96	0.99	88.69
<b>Kodak</b>	2	36.77	0.95	0.99	85.91
<b>BD0</b>	8	57.85	0.99	0.99	99.18
<b>BD1</b>	8	42.94	0.97	0.99	95.94
<b>BD2</b>	8	32.31	0.95	0.99	83.69
<b>BD3</b>	8	43.84	0.96	0.99	93.94
<b>BD4</b>	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)	<b>BD0</b>	<b>BD1</b>	<b>BD2</b>	<b>BD3</b>	<b>BD4</b>
<b>BD0</b>	49.66	38.07	26.34	38.05	31.13
<b>BD1</b>	47.57	44.08	34.54	42.66	38.69
<b>BD2</b>	<b>53.69</b>	<b>51.16</b>	<b>44.07</b>	<b>50.06</b>	<b>47.14</b>
<b>BD3</b>	50.62	47.88	39.76	46.59	43.25
<b>BD4</b>	47.30	46.98	41.10	45.63	43.75
<b>Todas</b>	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
<b>BD0</b>	34.77
<b>BD1</b>	44.11
<b>BD2</b>	48.97
<b>BD3</b>	45.34
<b>BD4</b>	42.42
<b>Todas</b>	51.27
<b>Todas + CLIC Mobile train</b>	<b>55.78</b>
<b>Todas + Clic Mobile train + Clic Professional train</b>	47.26
<b>CLIC Mobile Train</b>	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 e reforçam a afirmação feita em [?]. Com esses mesmos hiperparâmetros para treino, foram realizados alguns testes na base de teste (nomeada como *test*) do CLIC [8]. Alguns desses 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 [?]. Outro resultado interessante obtido ocorre

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
<b>CLIC Mobile test</b>	8	35.03	0.94	0.98	30
<b>BD1</b>	8	35.26	0.93	0.98	30
<b>BD2</b>	8	29.10	0.95	0.98	30

ao treinar na base de dados com menor entropia e testar na base com maior entropia. 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 melhorias 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 [?] 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, 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



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

Tabela 4.5: Tabela contendo os resultados do Modelo 3.

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

*JPEG* para diferentes taxas e métricas de distorção é apresentada na Figuras 4.3 a 4.2. Este modelo possui um único nível de resíduo.

## 4.5 Modelo 4

Para este modelo foram usadas as bases **BD2**, **BD3** e **BD4** como treino. O modelo foi testado em duas bases: Kodak [2] e [8]. O modelo possui 10 níveis de resíduos e foi treinado por 500 mil iterações. Os resultados referentes à base Kodak, são mostrados nas Figuras 4.4 a 4.6 e sumarizados nas Tabelas 4.6 a 4.7. Os resultados referentes às 47 imagens com muito conteúdo de alta frequência são mostrados nas Figuras 4.15 a 4.17 e sumarizados nas Tabelas 4.9 a 4.11.

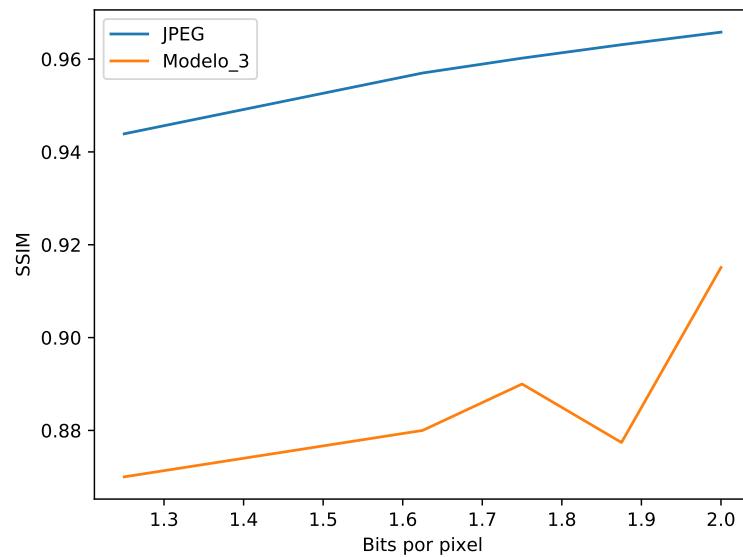


Figura 4.2: Comparação do Modelo 3 com o JPEG na métrica PSNR em diferentes taxas.

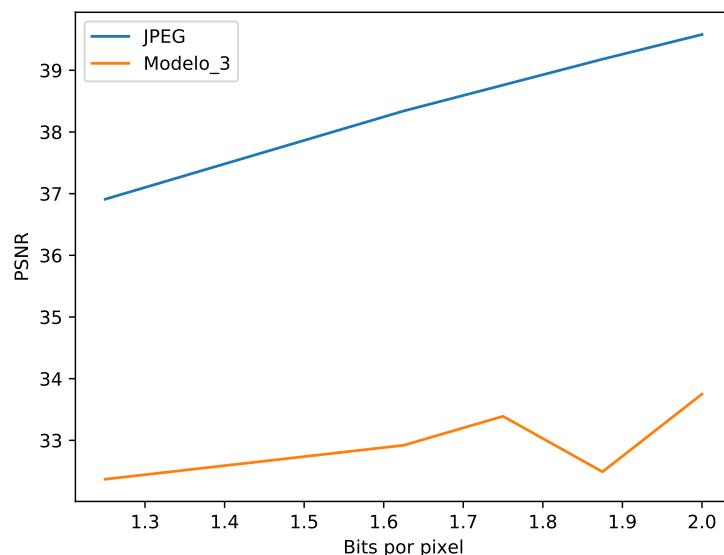


Figura 4.3: Comparação do Modelo 3 com o JPEG na métrica SSIM em diferentes taxas.

Tabela 4.6: Tabela contendo os valores da taxa (BPP) e PSNR (decíbeis) do Modelo 4, JPEG e JPEG2000 para a base [2].

Nível	Network		Network + GZIP		JPEG		JPEG 2000	
	Taxa	PSNR	Taxa	PSNR	Taxa	PSNR	Taxa	PSNR
<b>1</b>	0.125	24.44	0.10	24.44	0.17	21.52	0.10	26.23
<b>2</b>	0.250	26.81	0.23	26.81	0.23	24.51	0.23	28.29
<b>3</b>	0.375	28.32	0.35	28.32	0.36	27.53	0.35	29.54
<b>4</b>	0.500	29.46	0.48	29.46	0.48	29.08	0.47	30.66
<b>5</b>	0.625	30.44	0.60	30.44	0.60	30.23	0.60	31.68
<b>6</b>	0.750	31.26	0.73	31.26	0.73	31.14	0.72	32.55
<b>7</b>	0.875	31.98	0.85	31.98	0.85	31.92	0.85	33.43
<b>8</b>	1.000	32.63	0.98	32.63	0.98	32.60	0.97	34.10
<b>9</b>	1.125	33.19	1.10	33.19	1.10	33.21	1.10	34.81
<b>10</b>	1.250	33.65	1.23	33.65	1.22	33.72	1.21	<b>35.37</b>

Tabela 4.7: Tabela contendo os valores da taxa (BPP) e SSIM do Modelo 4, JPEG e JPEG2000 para a base [2].

Nível	Network		Network + GZIP		JPEG		JPEG 2000	
	Taxa	SSIM	Taxa	SSIM	Taxa	SSIM	Taxa	SSIM
<b>1</b>	0.125	0.65586	0.10	0.65586	0.17	0.56163	0.10	0.69056
<b>2</b>	0.250	0.74829	0.23	0.74829	0.23	0.66125	0.23	0.77412
<b>3</b>	0.375	0.80243	0.35	0.80243	0.36	0.76348	0.35	0.81706
<b>4</b>	0.500	0.83860	0.48	0.83860	0.48	0.81317	0.47	0.84603
<b>5</b>	0.625	0.86547	0.60	0.86547	0.60	0.84571	0.60	0.86805
<b>6</b>	0.750	0.88592	0.73	0.88592	0.73	0.86819	0.72	0.88374
<b>7</b>	0.875	0.90060	0.85	0.90060	0.85	0.88507	0.85	0.89826
<b>8</b>	1.000	0.91252	0.98	0.91252	0.98	0.89808	0.97	0.90900
<b>9</b>	1.125	0.92177	1.10	0.92177	1.10	0.90873	1.10	0.91981
<b>10</b>	1.250	0.92873	1.23	<b>0.92873</b>	1.22	0.91712	1.21	0.92769

Tabela 4.8: Tabela contendo os valores da taxa (BPP) e MS-SSIM do Modelo 4, JPEG e JPEG2000 para a base [2].

Nível	Network		Network + GZIP		JPEG		JPEG 2000	
	Taxa	MS	Taxa	MS	Taxa	MS	Taxa	MS
<b>1</b>	0.125	0.84270	0.10	0.84270	0.17	0.71702	0.10	0.88245
<b>2</b>	0.250	0.92192	0.23	0.92192	0.23	0.82584	0.23	0.93421
<b>3</b>	0.375	0.94893	0.35	0.94893	0.36	0.91058	0.35	0.95471
<b>4</b>	0.500	0.96244	0.48	0.96244	0.48	0.94160	0.47	0.96546
<b>5</b>	0.625	0.97054	0.60	0.97054	0.60	0.95843	0.60	0.97269
<b>6</b>	0.750	0.97581	0.73	0.97581	0.73	0.96788	0.72	0.97750
<b>7</b>	0.875	0.97993	0.85	0.97993	0.85	0.97420	0.85	0.98118
<b>8</b>	1.000	0.98274	0.98	0.98274	0.98	0.97840	0.97	0.98387
<b>9</b>	1.125	0.98502	1.10	0.98502	1.10	0.98158	1.10	0.98630
<b>10</b>	1.250	0.98665	1.23	0.98665	1.22	0.98381	1.21	<b>0.98807</b>

Apesar da função de perda utilizada ser a MSE, o modelo conseguiu obter melhores resultados comparados aos codecs na SSIM e MS-SSIM.

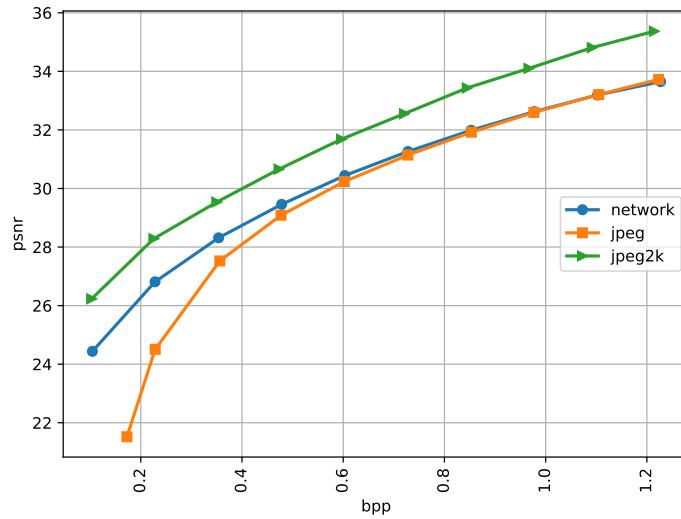


Figura 4.4: Comparação do Modelo 4 com o JPEG e JPEG2000 na métrica PSNR em diferentes taxas para a base Kodak [2].

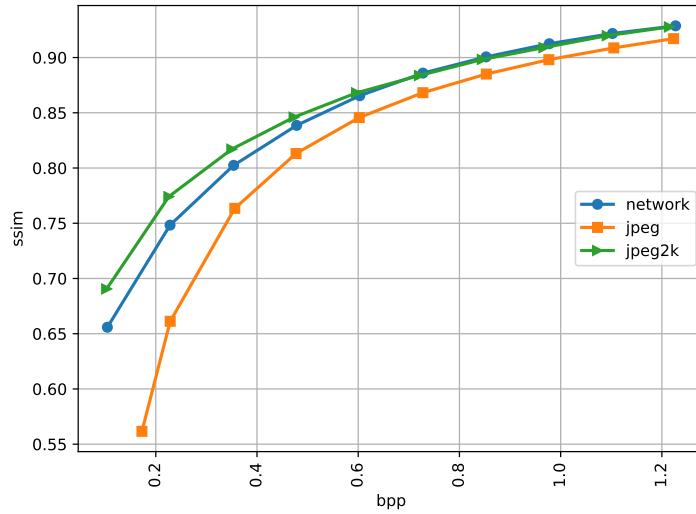


Figura 4.5: Comparação do Modelo 4 com o JPEG e JPEG2000 na métrica SSIM em diferentes taxas para a base Kodak [2].

Nota-se que para imagens (Figuras 4.7 a 4.8) com muito conteúdo de alta frequência (muitos detalhes), o modelo se sai melhor para todas as métricas visuais avaliadas. O que era esperado, visto que o JPEG e o JPEG2000 assumem que sinais de alta frequência não importam muito (assumem que maior parte energia da imagem estará contida em

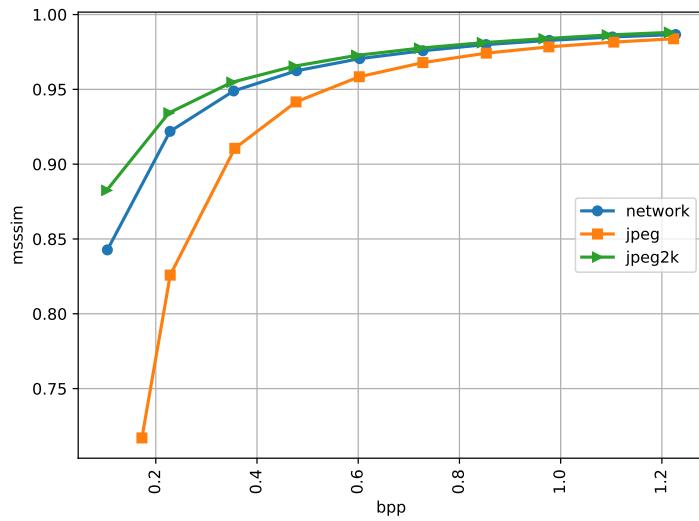


Figura 4.6: Comparação do Modelo 4 com o JPEG e JPEG2000 na métrica MS-SSIM em diferentes taxas para a base Kodak [2].

coeficientes de baixa frequência) e faz com que os coeficientes de baixa frequência tenham maior precisão ao quantizar.



Figura 4.7: Imagem Kodim05 [2].

Considerando os resultados anteriores obtidos para as imagens kodim05 e kodim09, o Modelo 4 também foi testado em um conjunto de 47 imagens com muito conteúdo de alta frequência retiradas da [2] e da [8]. Os resultados são mostrados nas Figuras 4.15 a 4.17 e sumarizados nas Tabelas 4.9 a 4.11.



Figura 4.8: Imagem jason-leem [8].

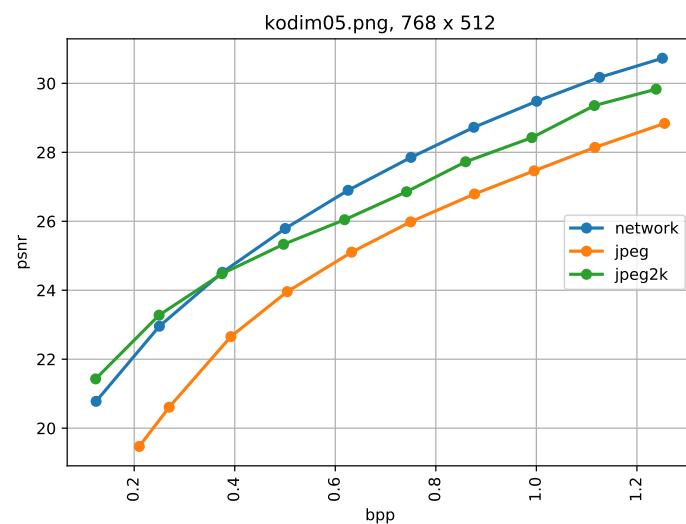


Figura 4.9: Comparaçāo do Modelo 4 com o JPEG e JPEG2000 na métrica PSNR em diferentes taxas para a imagem Figura 4.7.

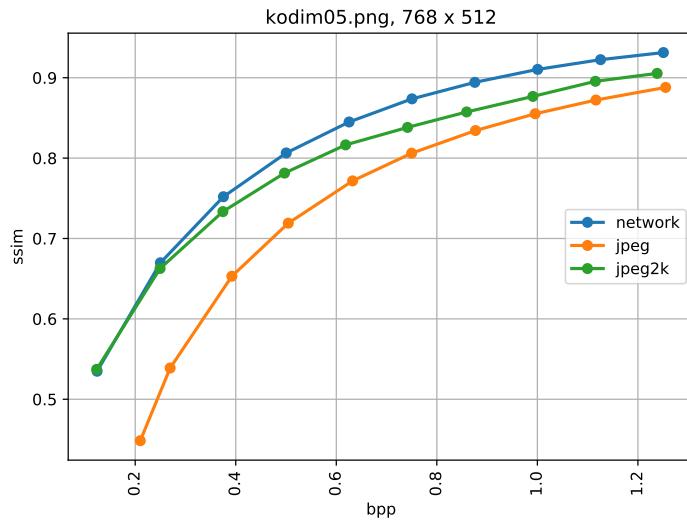


Figura 4.10: Comparaçāo do Modelo 4 com o JPEG e JPEG2000 na métrica SSIM em diferentes taxas para a imagem Figura 4.7.

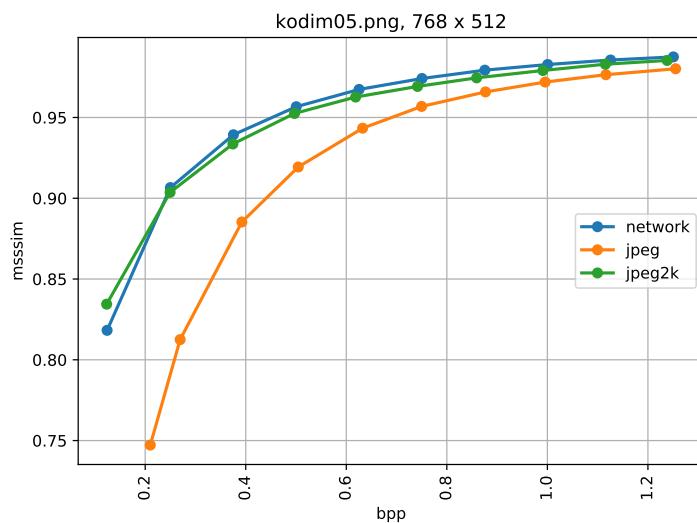


Figura 4.11: Comparaçāo do Modelo 4 com o JPEG e JPEG2000 na métrica MS-SSIM em diferentes taxas para a imagem Figura 4.7.

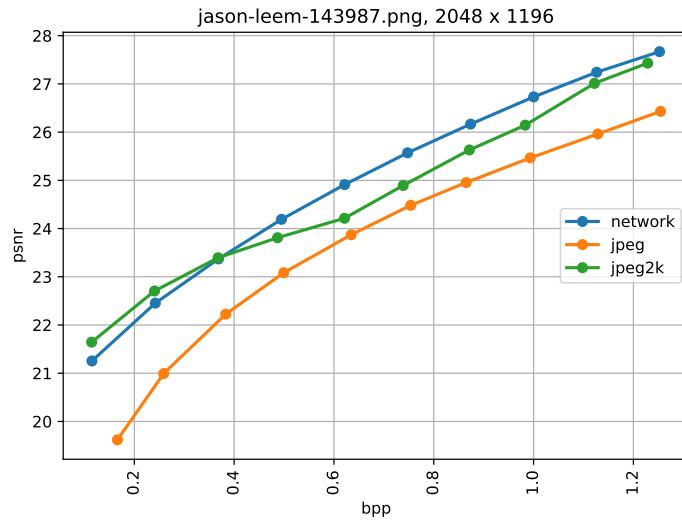


Figura 4.12: Comparação do Modelo 4 com o JPEG e JPEG2000 na métrica PSNR em diferentes taxas para a imagem de [8].

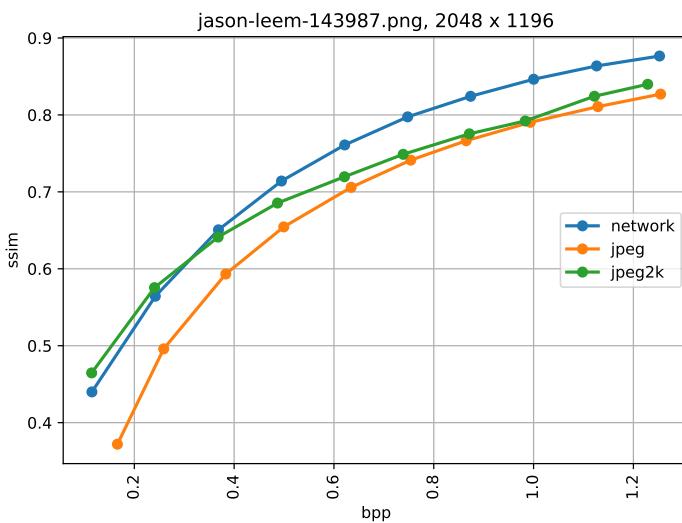


Figura 4.13: Comparação do Modelo 4 com o JPEG e JPEG2000 na métrica SSIM em diferentes taxas para a imagem de [8].

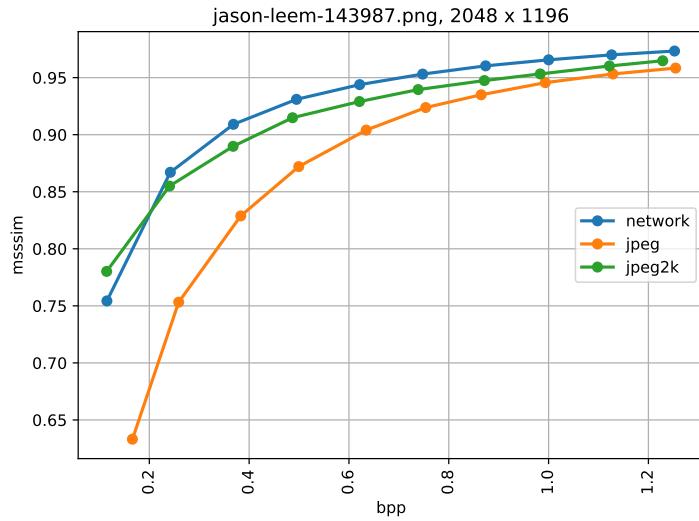


Figura 4.14: Comparaço do Modelo 4 com o JPEG e JPEG2000 na mtrica MS-SSIM em diferentes taxas para a imagem de [8].

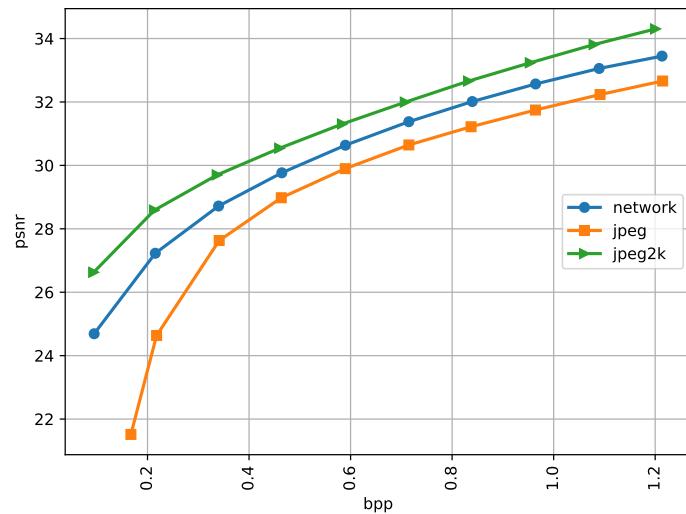


Figura 4.15: Comparaço do Modelo 4 com o JPEG e JPEG2000 na mtrica PSNR em diferentes taxas para 47 imagens com muito conteo de alta frequcia retiradas das bases [8] e [2].

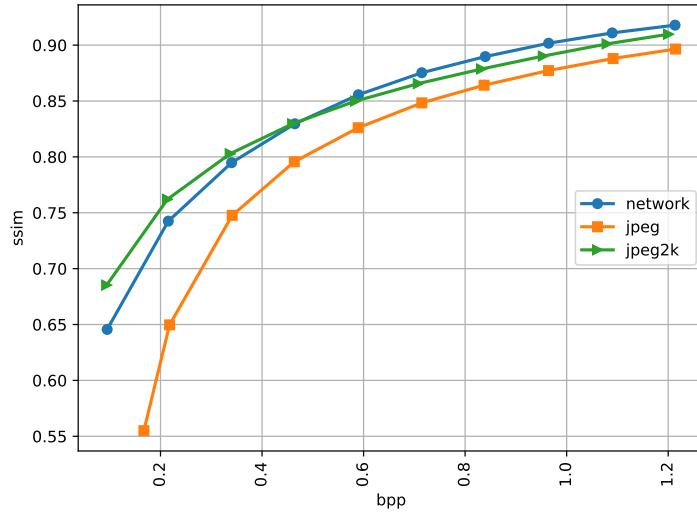


Figura 4.16: Comparação do Modelo 4 com o JPEG e JPEG2000 na métrica SSIM em diferentes taxas para 47 imagens com muito conteúdo de alta frequência retiradas das bases [8] e [2].

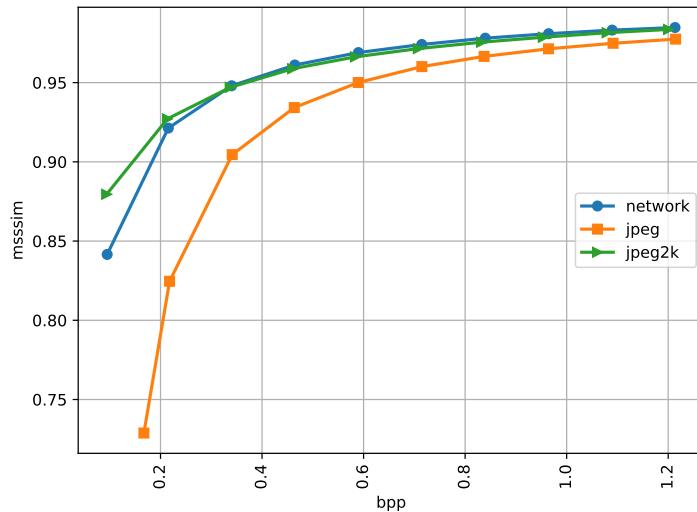


Figura 4.17: Comparação do Modelo 4 com o JPEG e JPEG2000 na métrica MS-SSIM em diferentes taxas para 47 imagens com muito conteúdo de alta frequência retiradas das bases [8] e [2].

Tabela 4.9: Tabela contendo os valores da taxa (BPP) e PSNR do Modelo 4, JPEG e JPEG2000 para 47 imagens da base [2] e [8] com muito conteúdo de alta frequência.

Nível	Network		Network + GZIP		JPEG		JPEG 2000	
	Taxa	PSNR	Taxa	PSNR	Taxa	PSNR	Taxa	PSNR
<b>1</b>	0.125	24.69	0.10	24.69	0.17	21.51	0.10	26.63
<b>2</b>	0.250	27.23	0.21	27.23	0.22	24.64	0.21	28.59
<b>3</b>	0.375	28.72	0.34	28.72	0.34	27.63	0.33	29.70
<b>4</b>	0.500	29.77	0.46	29.77	0.46	28.98	0.46	30.54
<b>5</b>	0.625	30.64	0.59	30.64	0.59	29.90	0.58	31.31
<b>6</b>	0.750	31.38	0.71	31.38	0.71	30.64	0.71	31.99
<b>7</b>	0.875	32.01	0.84	32.01	0.84	31.22	0.83	32.66
<b>8</b>	1.000	32.57	0.96	32.57	0.96	31.75	0.95	33.23
<b>9</b>	1.125	33.06	1.09	33.06	1.10	32.23	1.08	33.81
<b>10</b>	1.250	33.45	1.20	33.45	1.21	32.67	1.20	<b>34.30</b>

Tabela 4.10: Tabela contendo os valores da taxa (BPP) e SSIM do Modelo 4, JPEG e JPEG2000 para 47 imagens da base [2] e [8] com muito conteúdo de alta frequência.

Nível	Network		Network + GZIP		JPEG		JPEG 2000	
	Taxa	SSIM	Taxa	SSIM	Taxa	SSIM	Taxa	SSIM
<b>1</b>	0.125	0.64562	0.10	0.64562	0.17	0.55499	0.10	0.68526
<b>2</b>	0.250	0.74250	0.21	0.74250	0.22	0.64966	0.21	0.76202
<b>3</b>	0.375	0.79480	0.34	0.79480	0.34	0.74767	0.33	0.80283
<b>4</b>	0.500	0.82961	0.46	0.82961	0.46	0.79561	0.46	0.82961
<b>5</b>	0.625	0.85558	0.59	0.85558	0.59	0.82607	0.58	0.84989
<b>6</b>	0.750	0.87530	0.71	0.87530	0.71	0.84832	0.71	0.86553
<b>7</b>	0.875	0.88973	0.84	0.88973	0.84	0.86414	0.83	0.87862
<b>8</b>	1.000	0.90168	0.96	0.90168	0.96	0.87722	0.95	0.89020
<b>9</b>	1.125	0.91092	1.09	0.91092	1.10	0.88800	1.08	0.90109
<b>10</b>	1.250	0.91784	1.20	<b>0.91784</b>	1.21	0.89648	1.20	0.90973

Tabela 4.11: Tabela contendo os valores da taxa (BPP) e MS-SSIM do Modelo 4, JPEG e JPEG2000 para 47 imagens da base [2] e [8] com muito conteúdo de alta frequência.

Nível	Network		Network + GZIP		JPEG		JPEG 2000	
	Taxa	MS	Taxa	MS	Taxa	MS	Taxa	MS
<b>1</b>	0.10	0.84156	0.10	0.84156	0.17	0.72882	0.10	0.87953
<b>2</b>	0.21	0.92131	0.21	0.92131	0.22	0.82462	0.21	0.92713
<b>3</b>	0.34	0.94802	0.34	0.94802	0.34	0.90454	0.33	0.94699
<b>4</b>	0.46	0.96112	0.46	0.96112	0.46	0.93426	0.46	0.95877
<b>5</b>	0.59	0.96897	0.59	0.96897	0.59	0.95006	0.58	0.96628
<b>6</b>	0.71	0.97408	0.71	0.97408	0.71	0.96010	0.71	0.97148
<b>7</b>	0.84	0.97806	0.84	0.97806	0.84	0.96655	0.83	0.97552
<b>8</b>	0.96	0.98088	0.96	0.98088	0.96	0.97134	0.95	0.97863
<b>9</b>	1.09	0.98315	1.09	0.98315	1.10	0.97483	1.08	0.98151
<b>10</b>	1.20	0.98477	1.20	<b>0.98477</b>	1.21	0.97746	1.20	0.98356

Pelas Figuras 4.18 a 4.20 é possível notar o que as Figuras 4.9 a 4.11 já haviam mostrado: a reconstrução do Modelo 4 é visualmente superior para a Figura 4.7 quando comparada com o **JPEG** e **JPEG2000**, mesmo com uma taxa menor. Essa taxa corresponde ao quarto nível de resíduo para o Modelo 4.



Figura 4.18: Imagem Figura 4.7 reconstruída pelo Modelo 4 à uma taxa de 0.47 bits por pixel..



Figura 4.19: Imagem Figura 4.7 reconstruída pelo **JPEG** à uma taxa de 0.50 bits por pixel..



Figura 4.20: Imagem Figura 4.7 reconstruída pelo **JPEG2000** à uma taxa de 0.50 bits por pixel.

Para taxas muito baixas é possível notar uma grande dificuldade do **JPEG** em comprimir para a Figura 4.7 mesmo com uma taxa maior, conforme mostra as Figuras 4.21 a 4.23. Essa taxa corresponde ao primeiro nível de resíduo para o Modelo 4.



Figura 4.21: Imagem Figura 4.7 reconstruída pelo Modelo 4 à uma taxa de 0.12 bits por pixel.



Figura 4.22: Imagem Figura 4.7 reconstruída pelo **JPEG** à uma taxa de 0.21 bits por pixel..



Figura 4.23: Imagem Figura 4.7 reconstruída pelo **JPEG2000** à uma taxa de 0.12 bits por pixel.

## 4.6 Modelos 5 e 6

Esses dois modelos foram treinados com 150000 iterações nas bases **BD2** e **BD4**. Foram utilizados 6 níveis de resíduos.

A função de loss utilizada no Modelo 5 é a MSE.

A função de loss utilizada no Modelo 6 é

$$K e^{(-0.07r)} Q \quad (4.1)$$

onde  $K = 3.5 \times 10^{-7}$  e  $Q$  é a quantidade de bits 1 no latente binarizado. Com esta função de *loss* há uma penalização na ocorrência do bit 1 o que leva a uma diminuição da entropia do latente binarizado. Espera-se que o codificador de entropia seja mais eficaz dessa forma.

Os resultados são mostrados nas Figuras 4.24 a 4.26. Foram feitos alguns testes usando *10-fold cross validation*. A média obtida pelo modelo 5 foi 30.23 dB de PSNR e 0.86340 de SSIM à uma taxa de 0.71 bpp. A média obtida pelo modelo 6 obteve 30.31 dB de PSNR e 0.86498 de SSIM à uma taxa de 0.71 bpp.

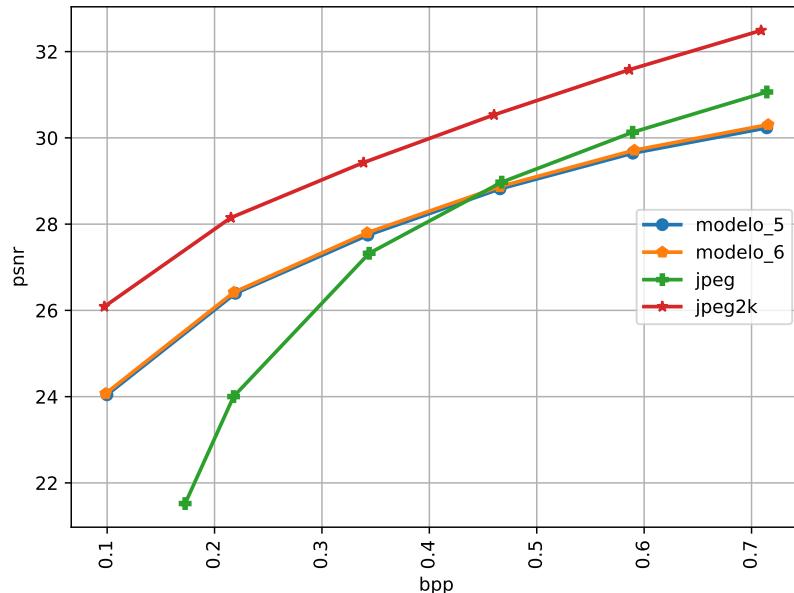


Figura 4.24: Comparação do Modelo 5 e 6 para a métrica MSE.

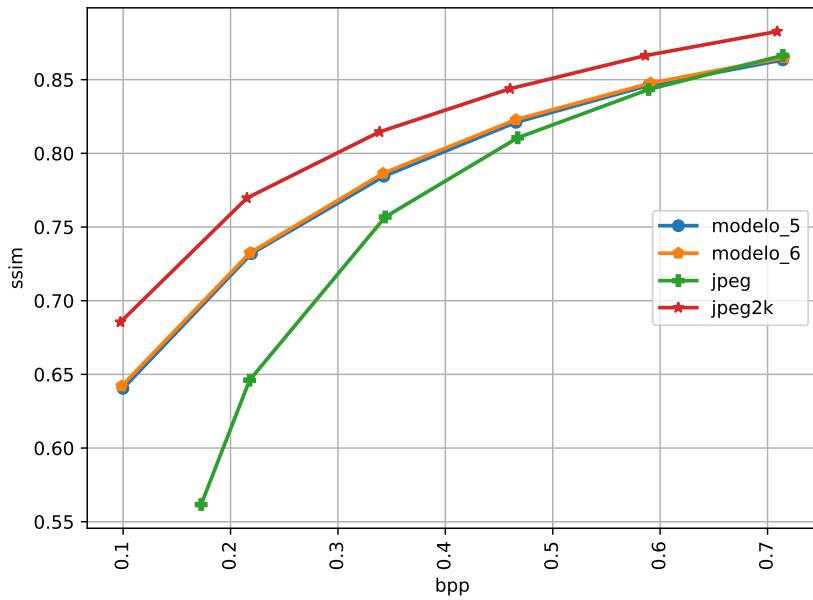


Figura 4.25: Comparaçao do Modelo 5 e 6 para a metrica SSIM.

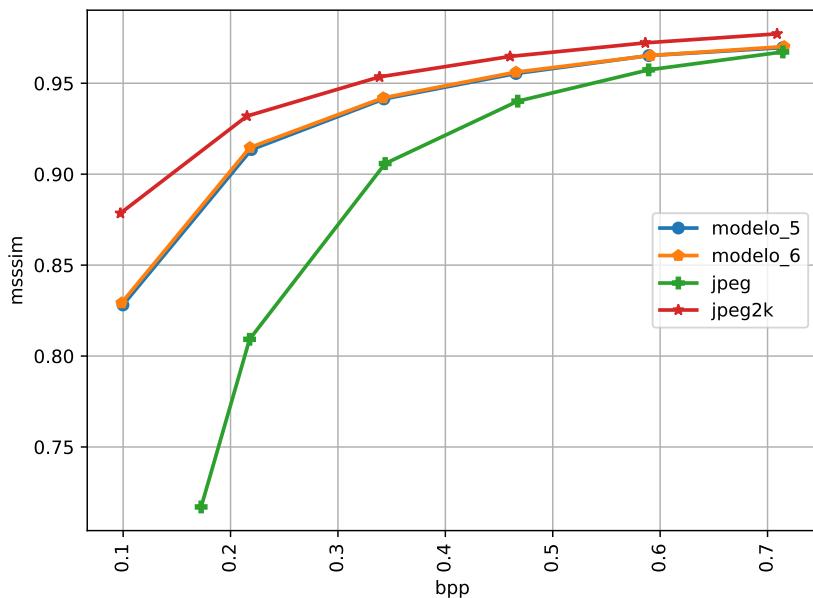


Figura 4.26: Comparaçao do Modelo 5 e 6 para a metrica MS-SSIM.

## 4.7 Ganhos obtidos pelo GZIP

Nas seguintes subseções será analisado o ganho na taxa obtido ao usar o codificador de entropia *gzip*.

### 4.7.1 Modelo 4

A Figura 4.27 mostra o ganho percentual médio por nível do Modelo 4 [4.5] para a base da Kodak [2]. Nota-se um decaimento exponencial no ganho. Considerando o funcionamento do modelo com resíduos, isto sugere uma especialização e robustez cada vez maior nos *encoders* dos níveis mais superiores ao explorar estruturas dos dados e comprimir informação nos latentes. Assim, os *bitstreams* gerados a partir dos latentes se tornam cada vez mais complexos (os resíduos possuirão muita variância nos valores) e com menos redundâncias fazendo com que o *gzip* não consiga explorar redundâncias bem ou que não haja muitas redundâncias.

Isto acontece na maior parte das imagens, principalmente nas que possui muito conteúdo de alta frequência e onde há muita informação que, consequentemente, geram latentes muito complexos, o que faz com que os *bitstreams* contenham muita informação já que os resíduos dos *patches* contém muita informação e geram muitas ativações no modelo.

Na Figura 4.29 é possível notar um ganho praticamente insignificante (e até negativo devido a informação de cabeçalho) ao usar o *gzip* para a Figura 4.7, enquanto na Figura 4.30 nota-se um ganho mais significativo com o *gzip* mesmo nos níveis mais superiores. Considerando que a Figura 4.7 possui muito conteúdo de alta frequência, os *bitstreams* gerados pelo modelo se tornam muito complexos. Enquanto, a Figura 4.28 possui um céu que faz parte da maior parte da imagem. É possível que os resíduos desses *patches* se tornem pequenos muito rápido de modo que o *gzip* é capaz de explorar bem esta grande quantidade de zeros.

### 4.7.2 Modelos 5 e 6

Na Figura ?? é possível notar um ganho superior ao ganho mostrado na Figura 4.27. Isso pode ter ocorrido devido ao menor número de níveis de resíduos existentes para esse modelo, o que leva à formação de um *bitstream* menos complexo para o modelo como um todo. Pode-se considerar que a diferença no ganho do **GZIP** para os Modelos 5 e 6 é inexistente.

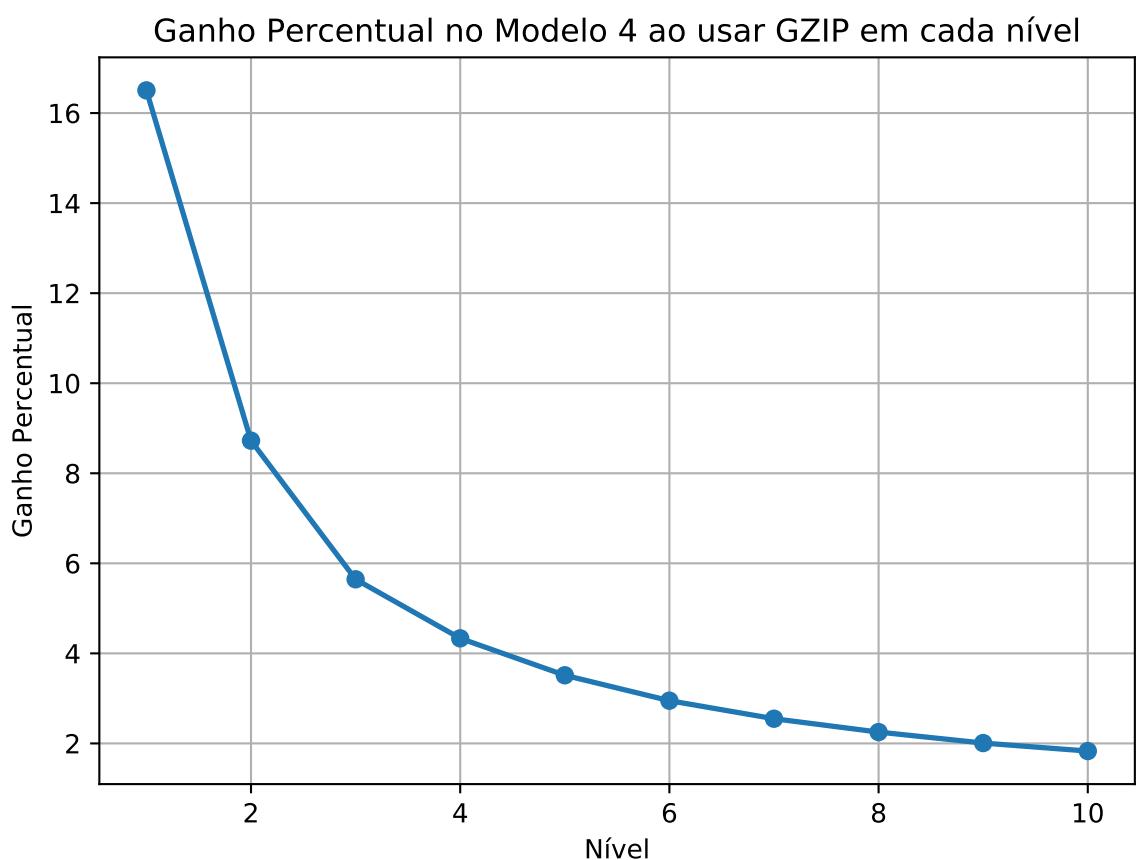


Figura 4.27: Ganho percentual médio na taxa por nível ao usar o codificador de entropia *gzip* nos *bitstreams* de cada nível para a base Kodak [2].



Figura 4.28: Imagem Kodim20 [2].

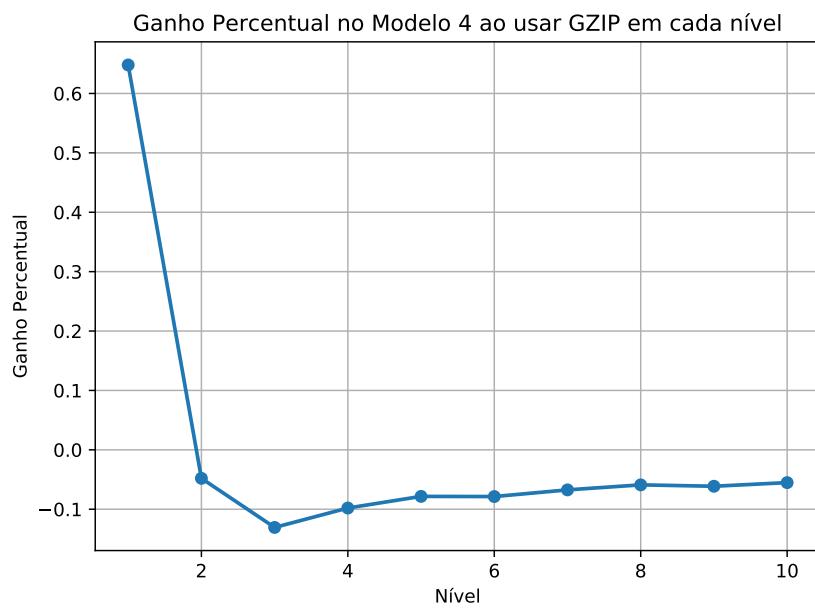


Figura 4.29: Ganho percentual na taxa por nível ao usar o codificador de entropia *gzip* nos *bitstreams* de cada nível para a Figura 4.7.

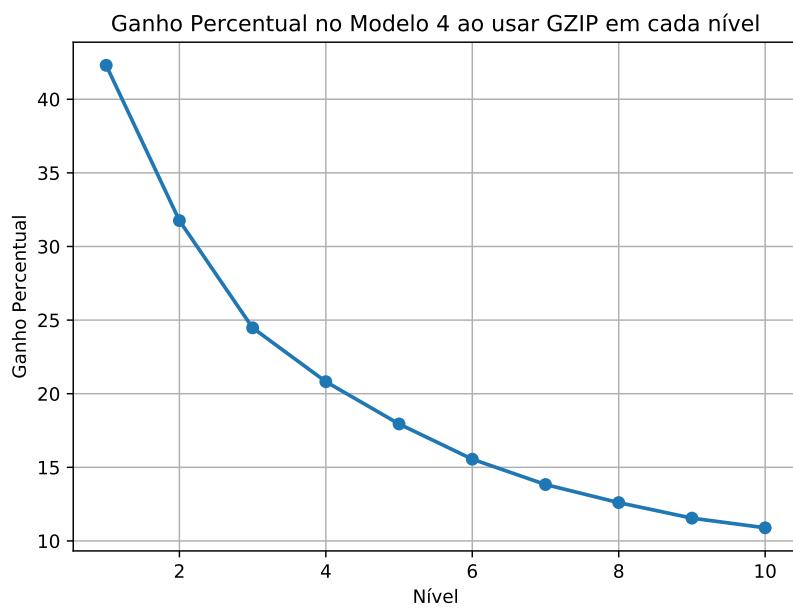


Figura 4.30: Ganho percentual na taxa por nível ao usar o codificador de entropia *gzip* nos *bitstreams* de cada nível para a Figura 4.28.

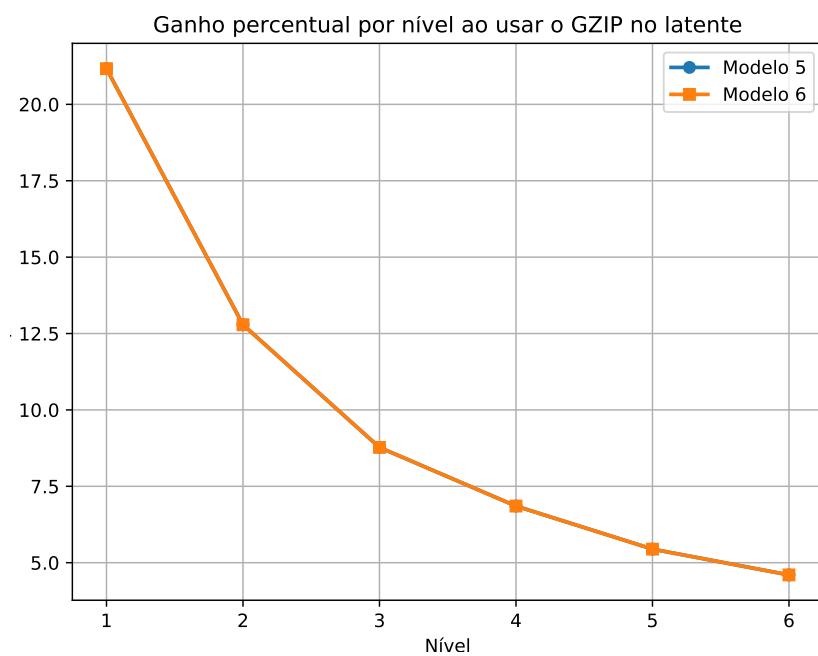


Figura 4.31: Ganho percentual médio na taxa por nível ao usar o codificador de entropia gzip nos bitstreams de cada nível para a base Kodak [2].

# 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 5.1 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. Esses modelos possuem o potencial de suprir uma necessidade crescente por algoritmos flexíveis de compressão com perdas. Entretanto, compressão com perdas é um 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 e Perspectivas Futuras

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 propor combinações de técnicas de *deep learning* e compressão de imagens já existentes. As seguintes conclusões podem ser tiradas:

- É possível concluir que o *GZIP* não está comprimindo muito após a primeira iteração. Portanto, seria necessário desenvolver um codificador de entropia específico para esta tarefa ou alterar a arquitetura do modelo para tentar fazer com que os latentes binarizados sejam mais adequados para o codificador.

- É possível notar que os modelos baseados em redes neurais são melhores do que os codecs clássicos para algumas imagens com alto conteúdo de alta frequência.
- É possível notar um impacto razoável em diferentes aspectos com o uso de diferentes funções de perdas, o que abre um espaço de exploração interessante para ser avaliado.
- Utiliza-se a mesma quantidade de bits para os mais diversos *patches*, o que não é uma boa abordagem do ponto de vista de compressão visto que seria possível utilizar menos bits para *patches* mais fáceis e mais bits para *patches* mais difíceis. Seria interessante fazer com que o modelo fosse capaz de realizar uma alocação dinâmica de bits.

Existe uma necessidade de desenvolver outros métodos para que seja possível superar os *codecs* clássicos no âmbito do tema deste trabalho. Portanto, ainda há muito espaço para novas soluções no problema abordado.

# 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, 3, 7, 11
- [2] Kodak., E.: *Kodak lossless true color image suite*. <http://r0k.us/graphics/kodak/>, 2014. [Online; accessed 25-June-2019]. ix, x, xi, xii, xiii, xiv, 8, 24, 35, 37, 38, 39, 43, 44, 45, 51, 52, 54
- [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, 10
- [4] Computerphile: *The problem with jpeg - computerphile*. <https://youtu.be/yBX8GFqt6GA?t=48>, 2015. [Online; accessed 30-June-2019]. ix, 11
- [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, 14
- [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, 15, 34
- [8] 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]. x, xi, xiii, xiv, 23, 33, 35, 39, 40, 42, 43, 44, 45
- [9] Sayood, Khalid: *Introduction to data compression*. Morgan Kaufmann, 2017. 1, 2, 3
- [10] 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. 3
- [11] Google, WebP: *Compression techniques*. <https://developers.google.com/speed/webp/docs/compression>. Accessed: 2019-06-16. 3
- [12] Bellard, Fabrice: *BPG image format*. <https://bellard.org/bpg/>. Accessed: 2019-06-13. 3

- [13] Simonyan, Karen e Andrew Zisserman: *Very deep convolutional networks for large-scale image recognition.* Relatório Técnico, Cornell University Library, 2014. arXiv:1409.1556. 4
- [14] 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
- [15] 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
- [16] 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
- [17] 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
- [18] 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
- [19] 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
- [20] 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
- [21] Krizhevsky, Alex e Geoffrey E Hinton: *Using very deep autoencoders for content-based image retrieval.* Em *ESANN*, 2011. 4
- [22] 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
- [23] 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
- [24] Theis, Lucas e Matthias Bethge: *Generative image modeling using spatial lstms.* Em *Advances in Neural Information Processing Systems*, páginas 1927–1935, 2015. 4
- [25] Ahmed, Nasir, T\_ Natarajan e Kamisetty R Rao: *Discrete cosine transform.* IEEE transactions on Computers, 100(1):90–93, 1974. 7

- [26] Huffman, David A: *A method for the construction of minimum-redundancy codes*. Proceedings of the IRE, 40(9):1098–1101, 1952. 12
- [27] Pennebaker, WB, JL Mitchell *et al.*: *Arithmetic coding articles*. IBM J. Res. Dev, 32(6):717–774, 1988. 12
- [28] Nielsen, Michael A.: *Neural Networks and Deep Learning*. Determination Press, 2015. 13
- [29] Rumelhart, David E, Geoffrey E Hinton, Ronald J Williams *et al.*: *Learning representations by back-propagating errors*. Cognitive modeling, 5(3):1, 1988. 13
- [30] Kingma, Diederik P e Jimmy Ba: *Adam: A method for stochastic optimization*. Relatório Técnico, Cornell University Library, 2014. arXiv:1412.6980. 15
- [31] 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. 15
- [32] Academy, Data Science: *Deep learning book*. <http://www.deeplearningbook.com>. br, 2019. [Online; accessed 30-June-2019]. 16
- [33] 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. 18
- [34] Wang, Zhou, Eero P Simoncelli e Alan C Bovik: *Multiscale structural similarity for image quality assessment*. Em *The Thirly-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, volume 2, páginas 1398–1402. Ieee, 2003. 18
- [35] 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
- [36] 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
- [37] Group, Multimedia Signal Processing: *Ultra-eye: Uhd and hd images eye tracking dataset*. <https://mmspgr.epfl.ch/downloads/ultra-eye/>, 2014. [Online; accessed 25-June-2019]. 23