COMPRESSÃO DE IMAGENS USANDO REDES NEURAIS ARTIFICIAIS

MARCELO GUEDES FRANÇA

TRABALHO DE CONCLUSÃO DE CURSO EM ENGENHARIA ELÉTRICA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

IMAGE COMPRESSION USING ARTIFICIAL NEURAL NETWORKS

MARCELO GUEDES FRANÇA

ORIENTADOR: PROF. DR. EDUARDO PEIXOTO FERNANDES DA SILVA

TRABALHO DE CONCLUSÃO DE CURSO EM
ENGENHARIA ELÉTRICA

PUBLICAÇÃO: PPGEA.TD-001/11

BRASÍLIA/DF: NOVEMBRO - 2019

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

COMPRESSÃO DE IMAGENS USANDO REDES NEURAIS ARTIFICIAIS

MARCELO GUEDES FRANÇA

TRABALHO DE CONCLUSÃO DE CURSO SUBMETIDO AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL.

APROVADO POR:

_____

Prof. Dr. Eduardo Peixoto Fernandes da Silva – ENE/Universidade de Brasília
Orientador

_____

Prof. Dr. Dr. Edson Mintsu Hung – ENE/Universidade de Brasília
Membro Interno

_____

Prof. Dr. L. Diogo – CIC/Universidade de Brasília
Membro Interno

BRASÍLIA, 27 DE NOVEMBRO DE 2019.

FICHA CATALOGRÁFICA

FRANÇA, MARCELO
Compressão de Imagens Usando Redes Neurais Artificiais [Distrito Federal] 2019.
xiv, 73p., 210 x 297 mm (ENE/FT/UnB, Bacharel, Engenharia Elétrica, 2019).
Trabalho de Conclusão de Curso – Universidade de Brasília, Faculdade de Tecnologia.
Departamento de Engenharia Elétrica

1. Compressão de Imagens	2. Redes Neurais Artificiais
3. Autocodificadores	4. Redes Recorrentes
I. ENE/FT/UnB	II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

FRANÇA, M. (2019). Compressão de Imagens Usando Redes Neurais Artificiais . Trabalho de Conclusão de Curso em Engenharia Elétrica, Publicação PPGEA.TD-001/11, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 73p.

CESSÃO DE DIREITOS

AUTOR: Marcelo Guedes França
TÍTULO: Compressão de Imagens Usando Redes Neurais Artificiais .
GRAU: Bacharel ANO: 2019

Marcelo Guedes França

Departamento de Engenharia Elétrica (ENE) - FT
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP 70919-970 - Brasília - DF - Brasil

À minha família.

AGRADECIMENTOS

.

RESUMO

A compressão de imagens é um tópico exaustivamente estudado dentro da ciência, em busca de representar a informação de forma compacta. Esse esforço é justificado pelo grande tráfego dessa mída pela Internet, porém limitado pelos requisitos de largura de banda. Recentemente, pesquisas de algoritmos baseados em Redes Neurais Artificiais para compressão de imagens têm alcançado resultados com enorme potencial. Em particular, a abordagem de ponta a ponta na qual codificador e decodificador são treinados em conjunto está presente nos trabalhos que superam os métodos tradicionais de compressão de imagens.

Essa abordagem é possível através de uma arquitetura de rede neural denominada autocodificador. Sob certas condições, ela é especialmente útil para aprender representações compactadas e bem estruturas dos dados. Na literatura, são relatados autocodificadores com características específicas para o problema de compressão. Dentre eles, muitos são baseados somente em camadas convolucionais, contudo esse método exige, geralmente, o treinamento de diversos modelos para diferentes taxas de compactação para que possam gerar uma curva taxa-distorção com boa pefomance. Nos autocodificadores com camadas recorrentes é possível gerar um modelo com flexibilidade para gerar taxas variadas com qualidade, ao aproveitarmos da sua capacididade para manter informações passadas. Os autocodificadores esparsos possuem alguma regularização envolvendo o espaço de código latente para encontrar represenações robustas dos dados.

Introduzimos modificações em trabalhos anteriores baseados em redes neurais convolucionais recorrentes para compressão de imagem com perdas. Primeiro, avaliamos o papel de uma base de dados com diferentes característiacas de entropia na perfomance do modelo. Em seguida, foi feito uma análise empírica para obter uma função de custo adequada no nosso problema.

A introdução de um fator de regularização, interpretado aqui, como uma estimativa da entropia de ordem zero, nessa função propocionou ganhos consideráveis. Essa função teve o papel de promover, de certa forma, esparsidade no fluxo de bits gerados pelo codificador e mudar a frequência de ocorrência dos bits (-1 e 1 para o nosso modelo). Essa estatística foi aproveitada pelo GZIP para reduzir as taxas. Por fim, propomos um esquema simples de alocação de bits para aproveita a diferença de complexidade nas regioões de uma imagem.

Os nossos resultados indicam melhor perfomance em relação ao JPEG nas medidas de PSNR, SSIM e MS-SSIM. Ao comparar esses resultados com o JPEG2000 obtemos desempenho competitivo. Contudo, o método de alocação dinâmica de bits teve o inconveniente de gerar artefatos de blocos visíveis e que degradam a qualidade perceptiva da imagem.

Palavras-chave: Compressão de Imagens, Redes Neurais Artificiais, Autocodificadores, Redes Recorrentes.

ABSTRACT

Keywords: Image Compression, Artificial Neural Networks, Autoencoder, Recurrent Networks.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÔNIMOS E ABREVIA CÕES

AC Autocodificador (*autoencoder*). 17–19

ADAM Estimação adaptativa de momento (*Adaptive Moment Estimation*). 10, 54

ADMM Método de Direção Alternativa dos Multiplicadores (*alternating direction method of multipliers*). 41

BPG Better Portable Graphics. 28, 35, 39

CBR Taxa de Bits Constante (*Constant Bitrate*). 51, 52

CCITT *Consultative Committee for International Telephony and Telegraphy*. 24

CODEC Codificador/Decodificador (*Encoder/Decoder*). xii, 20–22, 24, 28, 35, 38–40, 52, 60

Conv2D Convolução Bidimensional. 12, 33

Conv2DLSTM Rede de Mémória de Longo Prazo Convolucional Bidimensional. 16, 17, 33, 38, 42, 43

Conv2DRNR Rede Neural Recorrente Convolucional Bidimensional. 33

DCT Transformação Discreta de Cosseno (*Discrete Cosine Transform*). 25

FC-LSTM LSTM Totalmente Conectada (*Fully-Conected LSTM*). 16

GAN Redes Adversárias Generativas (*Generative adversarial network*). 39–41

GRU Unidades Recorrentes Fechadas (*Gated Recurrent Units*). 33

IDCT Transformada Discreta de Cosseno Inversa (*Inverse Discrete Cosine Transform*). 25

ISO International Organization for Standardisation. 24

ITU-T International Telecommunication Union. 24

JPEG Grupo de Especialistas em Fotografia (*Joint Photographic Experts Group*). x, 3, 22, 25, 26, 35, 38

LSTM Memória de Longo Prazo (*Long Short-Term Memory*). x, 15–17, 32–34, 38

MAE Erro Absoluto Médio (*Mean Absolute Error*). 50, 55
MLP Perceptron Multicamadas (*Multilayer Perceptron*). 10, 11, 14
MS-SSIM Índice de Similaridade Estrutural em Multi-Escalas (*Multi-scale Structural Similarity Index*). 26, 27, 35, 38, 39, 41
MSE Erro Médio Quadrático (*Mean Square Error*). 26, 41, 50, 51

PE Profunidade para Espaço (*Depth-to-Space*). 33, 42
PSNR Relação Pico-Sinal-Ruído (*Peak Signal to Noise Ratio*). 26, 38, 39, 52

ReLU Unidade Linear Retificadora (*Rectified Linear Unit*). 6, 7, 11
RGB Vermelho, Verde, Azul (*Red, Green, Blue*). 13, 23, 25, 50
RLE Codificação de Comprimento de Execução (*Run-Length Encoding*). 25
RNA Rede Neural Artificial (*Artificial Neural Network*). x, 1, 3–6, 10, 12, 28–30, 34, 38, 50
RNC Rede Neural Convolucional (*Convolutional Neural Network*). 12
RNR Rede Neural Recorrente (*Recurrent Neural Network*). 14

SSIM Índice de Similaridade Estrutural (*Structural Similarity Index*). 26, 27, 37, 38, 50

VAE autocodificador variacional (*variational autocoders*). 19, 39

YCbCr Luminância, Crominância Azul, Crominância Vermelho (*Luminance, Blue Chrominance, Red Chrominance*). 24, 50

1 INTRODUÇÃO

O século XXI tem sido marcado por grandes avanços tecnológicos que mudaram a forma do ser humano de se comunicar, relacionar, e informar, entreter, etc. Dentre os avanços, destacam-se o advento e popularização da rede mundial de computadores e das redes móveis para celulares. Hoje, elas superaram a distância física para prover comunicações rápidas entre os seres humanos. Essencial também são os avanços dos *smparthones* e computadores que dispões, cada vez mais, de recursos de *hardware* e *software* para processar grande volume de informação na forma de áudio, imagem, vídeo, etc.

Não é exagero dizer que esse cenário só foi possível com a digitalização e compressão da informação. O avanço nas qualidades de mídias como imagens e vídeo é, via de regra, seguindo por necessidade maior de dados. O aumento nos dados requer mais espaço de armazenamento, e na transmissão exigem mais consumo de energia e mais largura de banda. Todos esses recursos são escassos e dispendiosos. Uma solução viável é a elaboração de algoritmos de compressão cada vez mais eficientes. Esses algoritmos têm melhor performance quando atuam sobre uma categoria específica de informação, como textos, vídeos ou imagens.

Para imagens, temos observado o surgimento de definições em altas resoluções, como as imagens em *HD* (1280×720 pixels) e chegando até 10k (10240×4320 pixels). Aliado a isso, as imagens representam parte significativa no fluxo de informação que trafegam pela internet. Portanto, faz se necessário o esforço da comunidade científica, empresas dessa área, e demais interessados no desenvolvimento de novos codificadores para imagens ou aperfeiçoamento dos existentes. Hoje, grande parte dos trabalhos que superam o estado da arte da compressão de imagens são baseados em Rede Neural Artificial (*Artificial Neural Network*) (RNA)'s. Elas são conhecidas por ser uma ferramenta para a solução de uma variedade de problema de classificação e regressão com desempenho, em muitos desses problemas, a cima dos métodos tradicionais.

As RNA's possuem vantagens na compressão de imagens por efetuar funções extramamentes complexas e não-lineares. O problema de compressão é conhecido pela sua não linearidade. Além disso, essas abordagens possuem vantagens na especialização em compressão de imagens com conteúdo específico. O autocodificador é uma arquitetura de RNA amplamente usada na tarefa de compressão de imagens.

Nesse trabalho, tomamos como base uma arquitetura de autocodificador para compressão de imagens de alta resolução com camadas recorrentes e convolucionais proposto em [1] e realizamos algumas análises e alterações na tentativa de melhorar a sua performance.

Considerando essa arquitetura, são objetivos desse trabalho:

1. Analisar a performance da arquitetura em função da entropia e variabilidade da base de dados de treinamento;
2. Avaliar o desempenho pela alteração da função de custo;
3. Adicionar uma penalização a uma dada estatística dos bits gerados a fim de reduzir a taxa de compressão em um passo seguinte.
4. Propor uma técnica para alocação dinâmica dos bits em uma etapa pós treinamento

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, primeiro apresentaremos uma visão geral das redes neurais artificiais, desde o neurônio artificial, passando pelos principais tipos de arquitetura e chegando até os autocodificadores variacionais. No segundo momento, fundamentamos brevemente a teoria de compressão de dados, métodos para medida de informação e representação de imagens. Por fim, explicamos o funcionamento do JPEG e citamos as principais métricas objetivas para avaliar a qualidade entre imagens.

2.1 REDES NEURAIS ARTIFICIAIS

Nessa seção, descrevemos a ideia básica de treinamento de qualquer RNA, suas vantagens e seu elemento essencial: o neurônio artificial.

O aprendizado de máquina, do inglês *machine learning*, é um campo de conhecimento interdisciplinar que refere-se à detecção automatizada de padrões significativos nos dados através da construção de programas de computador que melhoram com a experiência (aprendizagem) [2]. Na etapa de aprendizado, o modelo de aprendizagem de máquina utiliza os dados de entrada para realizar predições dos resultados. Uma medida de erro é calculado entre a previsão e o resultado esperado. Essa medição é usada como um sinal de retroalimentação (*feedback*) para ajustar as regras do algoritmo de modo a minimizar o erro. Após o aprendizado do modelo ele deve ser capaz de fazer inferências com novos exemplos dos dados de entrada [3].

Na programação clássica, os seres humanos inserem regras (um programa) e dados a serem processados de acordo com essas regras para obter as respostas - não há processo de aprendizado. A figura em 2.1 resume a diferença essencial entre a programação convencional e o aprendizado de máquina [3].

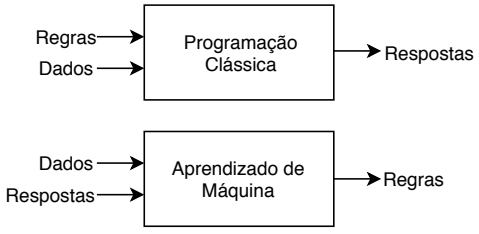


Figura 2.1 – Aprendizado de Máquina: um novo paradigma de programação. Adaptado de [3].

As RNA's são técnicas específicas de aprendizado de máquina inspiradas no funcionamento do cérebro humano. A semelhança com esse órgão humano advém de dois aspectos

principais [4]:

1. O conhecimento é adquirido pela rede a partir do seu ambiente através de um processo de aprendizagem;

2. Forças de conexão de neurônios, conhecidos como pesos sinápticos, são utilizados para armazenar o conhecimento adquirido.

Na literatura, o termo aprendizado profundo (*deep learning*) é usado para se referir às redes neurais com muitas camadas de processamento [5, 6]. Uma camada consiste em um módulo que realiza uma transformação matemática sobre uma entrada e gera um conjunto de dados de saída. Os pesos são os valores numéricos responsáveis por parametrizar as transformações em cada camada [3]. As camadas são conectadas em cadeia (uma após a outra) de forma que a última camada forneça uma representação numérica útil para o modelo realizar as predições. Após a previsão, a função de perdas calcula um valor do quão bem as previsões da rede correspondem ao esperado. Tal valor é usado por um algoritmo de otimização (otimizador) para realizar a atualização dos pesos. O aprendizado ou treinamento de uma RNA usando amostras dos dados e os seus respectivos rótulos é denominado de aprendizado supervisionado. A Figura 2.2 apresenta um esquemático do procedimento para o treinamento de uma rede neural artificial de 2 camadas.

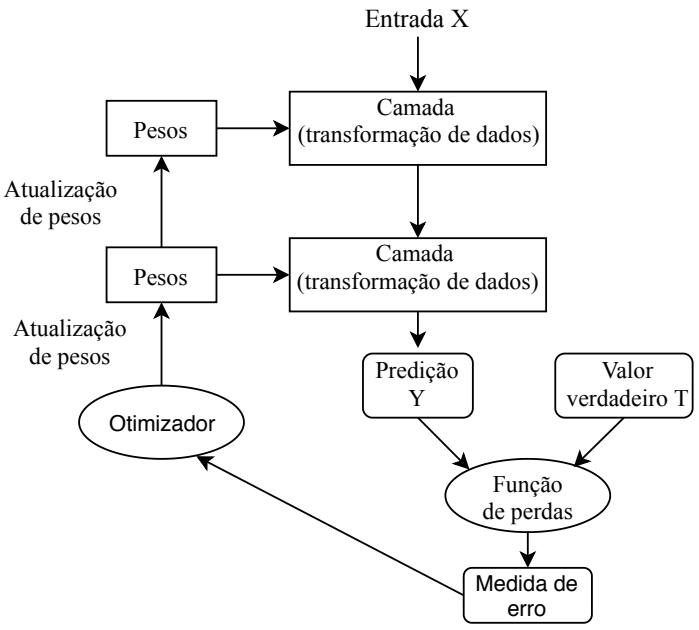


Figura 2.2 – A rede neural composta por 2 camadas encadeadas mapeia os dados de entrada para as previsões. A função de perdas compara essas previsões às metas, produzindo um valor de erro. O otimizador usa esse valor para atualizar os pesos da rede. Adaptado de [3].

O neurônio artificial, também chamado perceptron, é a unidade básica de processamento de uma rede neural e a interligação em massa dessas unidades computacionais a permite

alcançar desempenho cada vez melhor [4]. As principais propriedades úteis das RNA's em diversas tarefas de tomada de decisão são [4]:

1. A possibilidade de não linearidade dos neurônio, o que permite a rede resolver problemas não-lineares e complexos;

2. Adaptabilidade: os pesos das RNA's são facilmente adaptados sendo possível retreinar uma rede em um novo ambiente com pequenas modificações em relação ao original. Ademais, é possível encontrar um equilíbrio para que os pesos sejam modificados se a fonte de dados é não-estacionária e capaz de ignorar perturbações espúrias;

3. Resposta a evidência: nas tarefas de classificação a rede pode ser projetada para indicar o grau de certeza nas suas previsões. Isso permite ao modelo descartar informações ambíguas e aprimorar a performance de classificação;

4. Informação contextual: cada neurônio é potencialmente afetado pelo comportamento dos demais de forma a proporcionar um conhecimento de contexto ao modelo.

2.1.1 Neurônio Artificial

O neurônio é a unidade mais básica da RNA. O seu modelo está ilustrado na Figura 2.3.

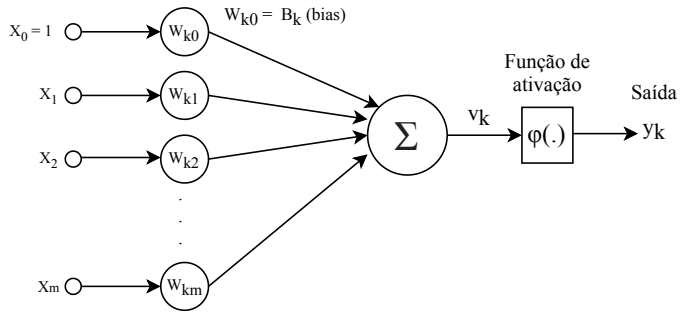


Figura 2.3 – Representação do neurônio artificial formado por um vetor de peso W de tamanho m. A entrada X_0 é artificial e o peso associada a ela é denominado viés. O neurônio funciona como um operador matemático simples que realiza um mapeamento de muitos para 1. Adaptado de [4].

A sua descrição matemática é dada pelas seguintes equações.

$$v_k = \sum_{j=0}^{m} w_{kj} x_j \tag{2.1}$$

$$y_k = \phi(v_k) \tag{2.2}$$

Em que x_0 é uma entrada fictícia que será multiplicada pelo viés ou do inglês *bias*, $w_{k0} = b_k$; $x_1, x_2, ..., x_m$ são os valores reais de entrada, $w_{k1}, x_{k2}, ..., x_{km}$ são os pesos sinápticos do neurônio k; v_k é a saída do produto escalar entre os vetores **w** e **x**; $\phi()$ é denominada função de ativação e y_k é o sinal de saída do neurônio. O viés é um termo independente dos sinais de entrada e tem o objetivo de fornecer mais um grau de liberdade ao neurônio.

2.1.1.1 Funções de ativação

A função de ativação é uma transformação matemática adicional que ocorre no perceptron. O seu uso permite a solução de problemas complexos e não-lineares, tais como tarefas de visão computacional e processamento de linguagem natural [7]. O neurônio com a função limiar foi utilizado no primeiro modelo de perceptron [8, 4] e contém a propriedade "tudo ou nada". As funções com formato de S e não-lineares são chamadas de sigmóides e amplamente usadas em RNA's. Entre elas há a função logística que restringe a saída no intervalo [0,1] e a tangente hiperbólica cuja saída pertence ao intervalo [-1,1]. A função *softmax* também é uma função sigmóide usada para prever as probabilidades associadas a uma classificação multi-classe [9]. Ela toma um vetor K-dimensional correspondentes aos K neurônios de uma camada e produz outro vetor K-dimensional com valores reais no intervalo (0, 1) que somam 1. A função *softmax* é idealmente usada na camada de saída do classificador [9]. A Unidade Linear Retificadora (*Rectified Linear Unit*) (ReLU) [10], é uma função não-linear que mapeia uma entrada para zero se ela for negativa ou retorna a própria entrada se ela for positiva. Diferentemente das funções sigmóides, ela é uma função não saturada e em decorrência disso possui duas vantagens principais: minimiza o problema denominado "gradiente de explosão e fuga" e acelerar a velocidade de convergência da rede [11]. As equações das funções de ativações mencionadas aqui e a suas derivadas estão apresentadas na Tabela 2.1.

O processo de aprendizado de uma rede neural consiste em determinar um vetor de pesos que seja capaz de prever corretamente a saída y para o máximo dos exemplos de treinamento fornecidos [7]. Além disso, o modelo deve ser geral o suficiente para realizar previsões corretamente em novos exemplos. Esse processo pode ser alcançado através dos algoritmos de retropropagação (conhecido do inglês como *backpropagation*) [12, 9] e de descida do gradiente. O primeiro toma a medida do erro, calculado na saída da rede neural, e computa o seu gradiente em relação a cada um dos pesos da rede [9]. O segundo é usado para atualizar os pesos por meio do gradiente calculado [7]. A descida do gradiente é um algoritmo específico dos otimizadores. A seguir, apresentamos a atuação desses algoritmos no treinamento do neurônio artificial. A Equação 2.3 é uma função de perdas relativa aos exemplos

Tabela 2.1 – Funções de ativações usadas em redes neurais artificiais e suas respectivas derivadas. Todas são funções de uma variável, exceto a função *softmax* que atua em um conjunto de entrada e retorna um grupo de saída.

Função de ativação	**Equação**	**Derivada**
Limiar	$\beta(v) = \begin{cases} 1, & \text{se } v \geq 0; \\ 0, & \text{se } v < 0 \end{cases}$	$\beta'(v) = \begin{cases} 0, & \text{se } v \neq 0 \\ \text{Não definida}, & \text{se } v = 0 \end{cases}$
Logística	$\alpha(v) = \dfrac{1}{1+e^{-v}}$	$\alpha'(v) = \alpha(v) \times (1 - \alpha(v)$
Tangente Hiperbólica	$tanh(v) = \dfrac{e^{v} - e^{-v}}{e^{v} + e^{-v}}$	$tanh'(v) = 1 - tanh^2(v)$
ReLU	$relu(v) = \begin{cases} v, & \text{se } v \geq 0; \\ 0, & \text{se } v < 0 \end{cases}$	$relu'(v) = \begin{cases} 1, & \text{se } v > 0 \\ 0, & \text{se } v < 0 \end{cases}$
Softmax	$S(\mathbf{v})_j = \dfrac{e^{v_j}}{\sum_{k=1}^{K} e^{v_k}}$	$\dfrac{\partial S_j}{\partial v_i} = \begin{cases} S_j(1 - S_i), & \text{se } i = j \\ -S_j S_i, & \text{se } i \neq j \end{cases}$

de treinamento [7].

$$E(\mathbf{w}) = \frac{1}{2} \sum_{d=0}^{D} (t_d - y_d)^2 \tag{2.3}$$

Onde, D é o conjunto de exemplos de treinamento, t_d é a saída de destino para o exemplo de treinamento d e y_d é a predição do perceptron. Por essa definição, caracterizamos E como uma função de **w** [7]. A Figura 2.4 ilustra a superfície do erro hipotético em função do espaço de hipóteses para o vetor de pesos de tamanho igual a 2. O espaço de hipótese é formado pelo plano $w_0 w_1$ enquanto o erro para cada vetor de pesos está no eixo vertical. Para qualquer ponto nessa superfície o algoritmo de descida de gradiente indica a direção de redução mais acentuada do erro. Com essa informação o vetor de pesos é atualizado sucessivamente até encontrar um mínimo na superfície de possibilidades do erro. Idealmente, desejamos que esse o mínimo seja o global, entretanto para muitas situações não há como garantir isso [7].

A direção de maior crescimento de uma função é obtido pelo cálculo do vetor gradiente em relação às variáveis independentes. No neurônio artificial, o algoritmo de retropropagação obtém esse vetor de forma direta, conforme a próxima equação.

$$\nabla E(\mathbf{w}) \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, ..., \frac{\partial E}{\partial w_m} \right] \tag{2.4}$$

Portanto, o negativo do vetor $\nabla E(\mathbf{w})$ informa a direção de maior decaimento do erro e a

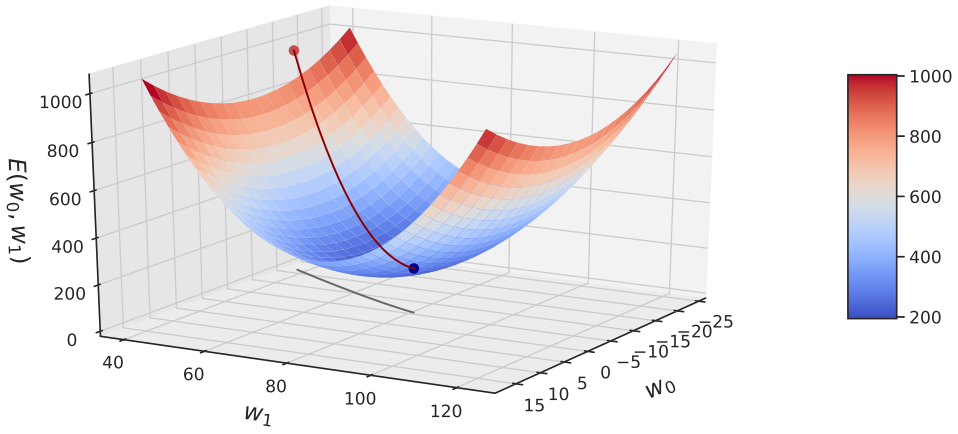


Figura 2.4 – Visualização do espaço de hipóteses dos pesos e atuação do algoritmo de descida de gradiente. Os pontos vermelho e azul indicam o início e o fim do aprendizado, respectivamente. A linha cinza é a projeção da curva de aprendizado no plano $w_0 w_1$.

regra de treinamento para a descida do gradiente é:

$$\begin{aligned} \mathbf{w}^{t+1} &= \mathbf{w}^t + \Delta(\mathbf{w}^t) \\ \Delta(\mathbf{w}^t) &= -\eta \times \nabla E(\mathbf{w}^t) \end{aligned} \tag{2.5}$$

Nessa equação, o subíndice é usado para diferenciar o vetor de pesos atual, (w^t), do próximo (w^{t+1}). O parâmetro $\eta > 0$ é a taxa de aprendizado, responsável por controlar o tamanho do passo da atualização dos pesos. O sinal negativo é empregado para obter o sentido oposto ao vetor gradiente. Essa regra de treinamento também pode ser escrita para cada componente do vetor de pesos:

$$w_i^{t+1} = w_i^t - \eta \times \frac{\partial E}{\partial w_i^t} \tag{2.6}$$

O desenvolvimento matemático da Equação 2.6 nos fornece o valor do gradiente para

cada peso do neurônio. Lembrando que $v = \sum_{j=0}^{m} w_j x_j$, teremos.

$$
\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{d=0}^{D}(t_d - y_d)^2 \\
\frac{\partial E}{\partial w_i} &= \frac{1}{2}\sum_{d=0}^{D}\frac{\partial}{\partial w_i}(t_d - y_d)^2 \\
\frac{\partial E}{\partial w_i} &= \frac{1}{2}\sum_{d=0}^{D}2(t_d - y_d)\frac{\partial}{\partial w_i}(t_d - y_d) \\
\frac{\partial E}{\partial w_i} &= \sum_{d=0}^{D}(t_d - y_d)\frac{\partial}{\partial w_i}(t_d - \phi(v)) \\
\frac{\partial E}{\partial w_i} &= \sum_{d=0}^{D}(t_d - y_d)(0 - \phi'(v)\frac{\partial v}{\partial w_i}) \\
\frac{\partial E}{\partial w_i} &= -\sum_{d=0}^{D}(t_d - y_d)\phi'(v)\frac{\partial}{\partial w_i}\sum_{j=0}^{m} w_j x_j) \\
\frac{\partial E}{\partial w_i} &= -\sum_{d=0}^{D}(t_d - y_d)\phi'(v)x_i
\end{aligned}
\tag{2.7}
$$

Aqui, vemos a necessidade da função de ativação seja diferenciável para que o algoritmo de retropropagação possa calcular o gradiente. Agora, a variação no valor de cada peso w_i será dada por:

$$
\begin{aligned}
w_i^{t+1} &= w_i^t + \Delta w_i^t \\
\Delta w_i^t &= \eta \times \sum_{d=0}^{D}(t_d - y_d)\phi'(v)x_i
\end{aligned}
\tag{2.8}
$$

Quando ajustamos os pesos somente após a apresentação de todos os exemplos de treinamento dizemos que a aprendizagem foi feita no modo lote (do inglês *batch*) [4]. A regra de treinamento de descida de gradiente apresentada na Equação 2.8 realiza as atualizações nesse modo. Todavia, à medida que o tamanho do conjunto de treinamento aumenta para bilhões de exemplos, o tempo para calcular o gradiente se torna proibitivamente longo [9].

Para contornar esse problema, modificamos o procedimento de aprendizado para atualizar os pesos após a apresentação de pequenos conjuntos de amostras denominados minilote ou *minibatches*, e extraídos aleatoriamente do conjunto de treinamento [9]. Nesse cenário, aproximamos o método de descida do gradiente pela sua versão estocástica. A regra de treinamento modificada é semelhante à Equação 2.8, exceto que à medida que iteramos em cada

minilote de tamanho m, atualizamos o peso de acordo com:

$$\Delta w_i^t = \eta \times \sum_{d=0}^{m} (t_d - y_d)\phi'(v)x_i \tag{2.9}$$

O otimizador Estimação adaptativa de momento (*Adaptive Moment Estimation*) (ADAM) **??** implementa um método de otimização por descida estocástica do gradiente e é amplamente usado em redes neurais artificiais.

2.1.2 Redes *Feedforward*

As redes neurais *feedforward* são uma classe de RNA formada pela composição de funções distintas. O modelo está associado a um gráfico acíclico direcionado que descreve como as funções são compostas [9]. Em uma rede *feedforward* com 2 funções, $f^{(1)}$ e $f^{(2)}$, a saída **y** devido a entrada **x** será dada pela conexão em cadeia das funções, $\mathbf{y} = f^{(2)}(f^{(1)}(\mathbf{x}))$ [9]. Na prática essas funções são modeladas por camadas. A última camada é denominada camada de saída e as demais são camadas ocultas. A profundidade da RNA é dada pelo número de camadas.

De forma análoga ao neurônio simples, em uma rede *feedforward* há a fase de propagação (*do inglês forward pass*), onde as entradas são passadas através da rede e as previsões obtidas. No passo para trás, do inglês *backward pass*, primeiro calculamos o gradiente da função de perdas em relação aos pesos da última camada da rede.

Nas camadas ocultas, o algoritmo de retropropagação não sabe qual saída cada neurônio deve fornecer. Portanto, aqui, devemos calcular a derivada do erro da última camada em relação as pesos das camadas ocultas. Isso é obtido aplicando-se a regra da cadeia. É possível provar que o gradiente calculado na última camada é "passado" até chegar a primeira camada oculta. Em redes com camadas ocultas a regra da cadeia possibilita computar o quanto um peso qualquer contribuiu para o sinal de erro calculado pela função de custo.

2.1.2.1 Perceptron Multicamadas

A Figura 2.5 apresenta uma rede *feedforward* conhecida como Perceptron Multicamadas (*Multilayer Perceptron*) (MLP). Essa rede se caracteriza pela combinação de neurônios organizados em duas camadas ou mais.

O teorema da aproximação universal [13] estabelece que uma camada oculta é suficiente para uma rede *feedfoward* aproximar qualquer função contínua em um subconjunto fechado e limitado de R^n por uma rede neural e uma quantidade de erro diferente de zero [9].

Seja m_0 o número de entradas de uma MLP e M o número de neurônios na camada de

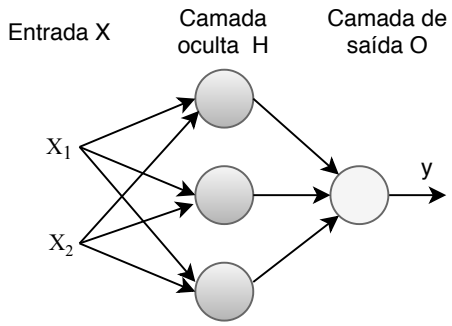


Figura 2.5 – Na rede *feedforward* as informações fluem através das camadas. Não há conexões de retroalimentação nas quais as saídas do modelo sejam realimentadas. Adaptado de [9].

saída. A relação de entrada-saída da rede define um mapeamento de um espaço de entrada euclidiano de dimensão m_0 para um espaço de saída euclidiano de dimensão M, que é infinitamente diferenciável desde que a função de ativação também tenha essa propriedade. Então o teorema diz:

Suponha que $\phi(.)$ seja uma função contínua não-constante, limitada e monotonamente crescente. Suponha que I_{m_0} represente o hipercubo unitário $[0,1]^{m_0}$ de dimensão m_0. O espaço das funções contínuas em I_{m_0} é representado por $C(I_{m_0})$. Então, dada qualquer função $f \in C(I_{m0})$ e $\epsilon > 0$, existe um inteiro m_1 e conjuntos de constantes reais α_i, β_i e w_{ij}, onde $i = 1, ...m_1$ e $j = 1, .., m_0$ tal que podemos definir:

$$F(x_1, .., x_{m_0}) = \sum_{i=1}^{m_1} \alpha_i \phi(\sum_{i=1}^{m_0} w_{ij} x_j + b_i) \tag{2.10}$$

como uma realização aproximada da função $f(.)$; isto é,

$$|F(x_1, ..., x_{m_0}) - f(x_1, ..., x_{m_0})| < \epsilon \tag{2.11}$$

para todo $x_1, x_2, .., x_{m_0}$ que se encontre no espaço de entrada.

A equação apresentada em 2.10 representa uma rede MLP com apenas uma camada oculta com m_1 neurônios. A ativação da camada oculta pode ser a função logística, uma vez que ela é não constante, limitada e monotonamente crescente. Os vetores **w** e **b** são os pesos e viés dos neurônios da camada oculta. A saída da rede é uma combinação linear das saídas dos neurônios ocultos, com α definindo o vetor de pesos sinápticos da camada de saída [4].

Vale ressaltar que em novos trabalhos o teorema de aproximação universal foi provado para uma classe mais ampla de funções de ativação, como a ReLU [14]. Uma rede neural também pode aproximar qualquer mapeamento de função de qualquer espaço discreto dimensional finito para outro [9].

O teorema da aproximação universal sintetizado pela Equação 2.10 generaliza as apro-

ximações por série de Fourier [4]. Contudo, ele não define o mínimo número de neurônios da camada oculta para cada ϵ. Na pior das hipóteses, pode ser necessário um número exponencial de unidades ocultas (possivelmente uma unidade oculta correspondente a cada configuração de entrada que precise ser distinguida) [9]. Além disso, o teorema também não diz se essa abordagem é ótima no sentido de tempo de de aprendizagem, facilidade de implementação e especialmente na generalização do modelo [4]. Na prática, as redes com mais de uma camada oculta se mostram mais concebíveis computacionalmente e generalizam melhor [9].

As principais considerações de arquitetura de uma RNA são a sua profundidade e o número de neurônios em cada camada. Redes mais profundas costumam usar menos unidades por camada e menos parâmetros. Elas normalmente generalizam para o conjunto de testes, contudo são frequentemente difíceis de otimizar. A arquitetura de rede ideal para uma tarefa deve ser encontrada por meio de experimentação guiada pelo monitoramento do erro do conjunto de validação [9].

2.1.2.2 Rede Convolucional

A Rede Neural Convolucional (*Convolutional Neural Network*) (RNC) é *feedforward* apropriada para extrair atributos de imagens através da aplicação de convoluções bidimensionais em que os pesos dos filtros são aprendidos durante o treinamento. Ela é projetada especificamente para reconhecer formas bidimensionais com um alto grau de invariância quanto a translação, inclinação e outros modos de distorção. Isso significa que após aprender um certo padrão em uma posição específica de uma imagem a rede convolucional o reconhece em qualquer lugar da imagem [4]. A Convolução Bidimensional (Conv2D) é usada em redes neurais para operar sobre mapas de características (estruturas bidimensionais). Tais mapas estão organizados em pilhas formando tensores tridimensionais. Tensores podem ser interpretados como uma matriz multidimensional [15], generalizando vetores e matrizes.

Nesses tensores, a primeira e segunda dimensão são a altura e largura do mapa e a terceira dimensão é o número de canais ou, profundidade, ou ainda, o número de mapas recursos. A saída da convolução também é montada em tensores tridimensionais. Os parâmetros para definir a operação de Conv2D são o passo (*stride*), preenchimento (*padding*), as dimensões espaciais do filtro e o número de canais de saída. O passo controla como o filtro se desloca pelos mapas. Através do preenchimento podemos adicionar um número apropriado de linhas e colunas em cada lado dos mapas. As duas principais configurações de preenchimento são as do tipo "válida" *(*valid) e 'mesma" *same*. Na primeira, não há preenchimento (somente locais válidos do mapa sofrem convolução pelo filtro), enquanto na segunda o preenchimento é feito de maneira a ter uma saída com a mesma largura e altura que a entrada [3]. As dimensões espaciais do filtro se referem a sua altura e largura. O número de canais na saída

da camada de convolução é igual à quantidade de filtros que foi escolhido para essa camada. O número de canais do filtro é sempre igual ao número de canais do tensor 3D de entrada.

Matematicamente, a convolução de um tensor **I** com dimensões $H \times W \times 1$, por um filtro $K(M \times N \times 1)$, com passo 1 e preenchimento válido, resulta no tensor **S** calculado segundo a equação a seguir.

$$S(i,j) = (I * K)(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} I(i+m, j+n) K(m,n) \tag{2.12}$$

onde $i \in [0, 1, ..., M - H + 1]$ e $j \in [0, 1, ..., N - W + 1]$.

A Figura 2.6 ilustra a convolução 2D entre um mapa de recursos com um filtro $K = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$.

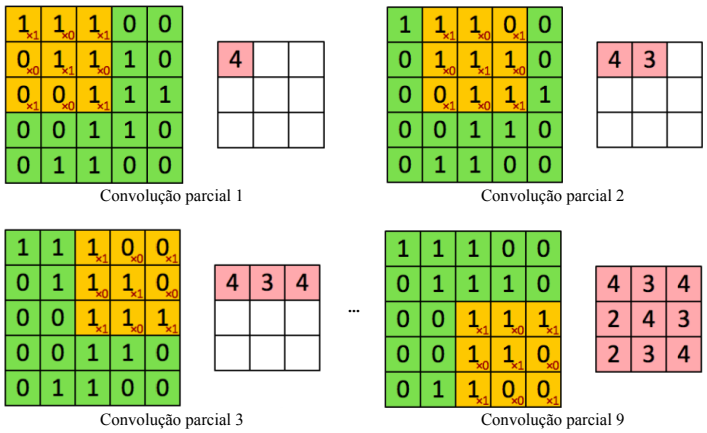


Figura 2.6 – Convolução de um tensor $5 \times 5 \times 1$, em verde, com um filtro $3 \times 3 \times 1$, em amarelo. À medida que o filtro se desloca pela entrada obtemos progressivamente um novo mapa de característica, em rosa, com dimensões $3 \times 3 \times 1$. O filtro K se desloca 9 vezes, sempre executando uma operação de multiplicação ponto a ponto entre K e a parte P do mapa sobreposto pelo filtro. Em seguida, somamos todos os valores de K $\odot$ P para preencher o mapa de característica da saída [16].

Se a entrada for uma imagem com múltiplos canais, por exemplo, RGB, então cada filtro da camada convolucional terá 3 canais. Então, aplicamos a Equação 2.12 em cada canal e todos os resultados são somados com o viés para fornecer uma saída. A Figura 2.7 ilustra uma etapa da convolução entre uma imagem RGB com um filtro 3D para o cálculo do primeiro valor do mapa de característica. Uma função de ativação não-linear se aplica após a convolução.

Em uma rede convolucional, a unidade de processamento é representado por um filtro que recebe seus sinais de entrada de um campo receptivo local na camada anterior, aprendendo características locais [4, 272]. Os pesos dos filtros são os mesmos para cada mapa gerado.

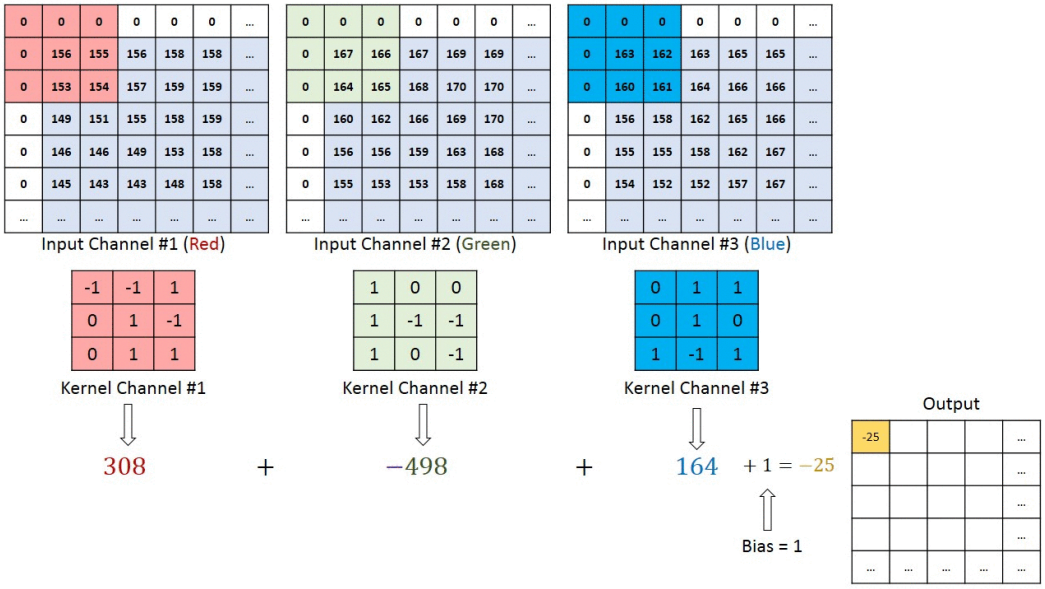


Figura 2.7 – Convolução de uma imagem $5 \times 5 \times 3$, preenchido de zeros nas bordas, com um filtro K $3 \times 3 \times 1$. Para cada canal da imagem é feito uma convolução 2D com o respectivo canal do filtro. O resultado das convoluções são somados entre si e com o viés para fornecer a saída gradualmente. O número de canais na saída é apenas 1 porque estamos utilizando 1 filtro [16].

Esse compartilhamento de pesos é uma restrição estrutural que proporciona redução drástica do número de parâmetros livres em comparação com uma rede MLP [4, 272].

2.1.3 Redes Recorrentes

A Rede Neural Recorrente (*Recurrent Neural Network*) (RNR) [17] é uma classe de redes neurais que inclui conexões de retroalimentação na sua arquitetura para processar dados sequenciais ou temporais. Para uma sequência $S = \{x_1, x_2, ..., x_n\}$ a RNR itera pelos seus elementos, mantendo um estado com informações relativas ao que foi visto até agora. Portanto, essa rede é projetada para ter memória [3] e processar um conjunto de entrada arbitrariamente grande e de tamanho variável [18]. A Figura 2.8 apresenta uma rede recorrente genérica em que a saída atual é uma função das saídas anteriores. Essa formulação recorrente resulta no compartilhamento de parâmetros através do tempo essencial para a viabilidade e eficácia dessa rede [9].

Podemos visualizar o compartilhamento dos pesos ao longo do tempo e o efeito dos resultados anteriores na saída atual pela próxima equação.

$$\mathbf{h}_t = \phi(\mathbf{x}_t\mathbf{W} + \mathbf{h}_{t-1}\mathbf{U}) \tag{2.13}$$

Onde h_{t-1} e h_t são os estados ocultos ou saídas da unidade recorrente no instante de

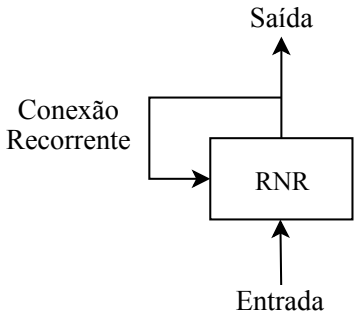


Figura 2.8 – Arquitetura básica para uma rede neural recorrente. As saídas anteriores são usadas na predição do estado atual. Adaptado de [3].

tempo t-1 e t, respectivamente. x_t é a entrada da etapa t da sequência, **W** e **U** são os vetores de pesos que multiplicam x_t e h_{t-1}, respectivamente. Os pesos são compartilhados durante as iterações que compõem uma sequência de dados. A função de ativação está representada genericamente por $\phi(\odot)$. Devido ao ciclo da rede, o estado oculto, no instante t, depende de todos os estados ocultos obtidos anteriormente.

O compartilhamento de parâmetros torna possível estender e aplicar o modelo a sequências de diferentes comprimentos, e identificar uma informação específica em qualquer posição dentro da sequência [9]. Para treinar uma RNR é preciso executar o algoritmo de retropropagação ao longo das etapas para cada sequência. Isso requer desenrolar a RNR em uma rede profunda. O treinamento de uma rede neural profunda pode sofrer com lentidão e problemas de gradientes que tendem a zero ou a valores muito elevados. A solução mais simples e comum para esse problema é desenrolar a RNR apenas em um número limitado de etapas durante o treinamento. Isso é chamado de retropropagação truncada ao longo do tempo [15].

Atualmente, um dos modelos de sequências mais eficazes usados em aplicações práticas é a Memória de Longo Prazo (*Long Short-Term Memory*) (LSTM) [19]. A sua arquitetura favorece a rápida convergência e detecção de dependências de longo prazo nos dados [15]. Essas redes contém têm "células LSTM" que possuem uma recorrência interna além da recorrência externa. Veja a Figura 2.9 para observar essas duas recorrências. Cada célula possui as mesmas entradas e saídas que uma rede recorrente comum, mas contém mais parâmetros e um sistema de unidades de controle que coordena o fluxo de informações. A formulação matemática da LSTM possui 4 camadas de rede neural sendo 3 delas unidades

de controle ou portas.

$$
\begin{aligned}
\mathbf{i}_t &= \sigma_i(\mathbf{W}_i\mathbf{x}_t + \mathbf{U}_i\mathbf{h}_{t-1} + \mathbf{b}_i) \\
\mathbf{f}_t &= \sigma_g(\mathbf{W}_f\mathbf{x}_t + \mathbf{U}_f\mathbf{h}_{t-1} + \mathbf{b}_f) \\
\mathbf{o}_t &= \sigma_o(\mathbf{W}_o\mathbf{x}_t + \mathbf{U}_o\mathbf{h}_{t-1} + \mathbf{b}_o) \\
\mathbf{c}'_t &= tanh(\mathbf{W}_c\mathbf{x}_t + \mathbf{U}_c\mathbf{h}_{t-1} + \mathbf{b}_c) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{c}'_t \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t)
\end{aligned}
\tag{2.14}
$$

Onde, $\mathbf{x}_t$ é a entrada atual da sequência. As variáveis **W**, **U** e **b** são as matrizes de pesos e viés aprendidas por cada camada. O vetor $\mathbf{i}_t$ é a porta de entrada que nos diz quais novas informações vamos armazenar no estado da célula, $\mathbf{c}_t$. A porta do esquecimento, $\mathbf{f}_t$, controla quais informações serão "esquecidas" do estado da célula anterior, $\mathbf{c}_{t-1}$. A porta de saída $\mathbf{o}_t$ é usada para fornecer a ativação para a saída, $\mathbf{h}_t$. O vetor $\mathbf{c}'_t$ é o estado da célula candidata e $\mathbf{c}_t$ é o estado de célula atual. Este último é o termo que contém a "memória" de longo prazo da LSTM. Nessa equação, o símbolo $\odot$ significa multiplicação ponto a ponto entre dois vetores. O cálculo do estado de célula é uma função recursiva que ocorre no interior da célula. Por fim, a saída é um vetor $\mathbf{h}_t$. A função logística é adotada na ativação das 3 portas. Dessa forma, as portas podem inibir uma informação na medida que seus elementos sejam próximos de zero (saturação inferior) ou mantê-la se os seus elementos são próximos a um (saturação superior). A lista a seguir detalha as variáveis da LSTM quando o vetor $\mathbf{x}_t$ tem tamanho d e escolhemos h unidades ocultas.

1. $\mathbf{x}_t \in \mathbb{R}^d$
2. $\mathbf{f}_t, \mathbf{i}_t, \mathbf{o}_t, \mathbf{h}_t, \mathbf{c}'_t, \mathbf{c}_t \in \mathbb{R}^h$
3. $\mathbf{W} \in \mathbb{R}^{h \times d}$
4. $\mathbf{U} \in \mathbb{R}^{h \times h}$
5. $\mathbf{b} \in \mathbb{R}^h$

A formulação da LSTM apresentada aqui é também denominada por LSTM Totalmente Conectada (*Fully-Conected LSTM*) (FC-LSTM).

A sua principal desvantagem, no tratamento de dados espaço-temporais, é o fato que não leva em consideração uma correlação espacial devido ao uso de conexões completas nas operações internas. Para superar esse problema, defini-se a Rede de Mémória de Longo Prazo Convolucional Bidimensional (Conv2DLSTM) [21, 1] na qual todas as entradas($x_1, ..., x_t$), estados de células ($c_1, ..., c_t$), estados ocultos($h_1, ..., h_t$) e portas(i_t, f_t, o_t) são tensores 3D

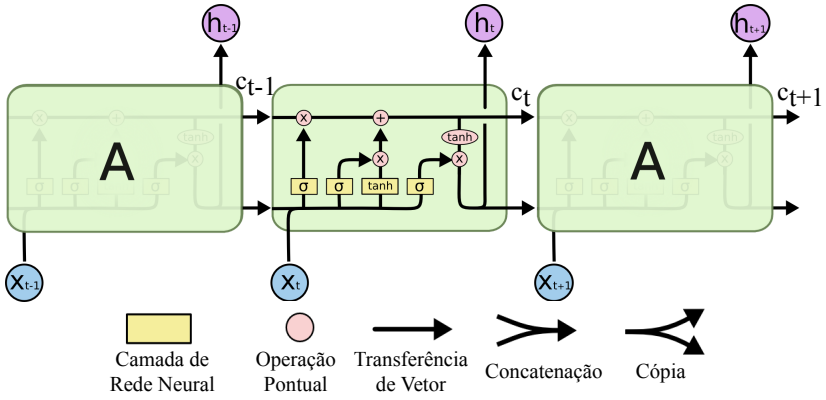


Figura 2.9 – Célula LSTM desenrolada no tempo. O estado de célula $\mathbf{c}_t$ e estado oculto $\mathbf{h}_t$ são usados para realimentar a célula a cada etapa de tempo. O primeiro é calculado em um ciclo interno enquanto o segundo é obtido por recorrência externa à LSTM. Adaptado de [20].

cujas duas últimas dimensões são dimensões espaciais (linhas e colunas). Isso pode ser alcançado usando um operador de convolução nas transições de estado para estado e de entrada para estado. As principais equações do Conv2DLSTM são mostradas abaixo, onde '*' denota o operador de convolução.

$$
\begin{aligned}
\mathbf{i}_t &= \sigma_i(\mathbf{W}_i * \mathbf{x}_t + \mathbf{U}_i * \mathbf{h}_{t-1} + \mathbf{b}_i) \\
\mathbf{f}_t &= \sigma_g(\mathbf{W}_f * \mathbf{x}_t + \mathbf{U}_f * \mathbf{h}_{t-1} + \mathbf{b}_f) \\
\mathbf{o}_t &= \sigma_o(\mathbf{W}_o * \mathbf{x}_t + \mathbf{U}_o * \mathbf{h}_{t-1} + \mathbf{b}_o) \\
\mathbf{c}'_t &= tanh(\mathbf{W}_c * \mathbf{x}_t + \mathbf{U}_c * \mathbf{h}_{t-1} + \mathbf{b}_c) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{c}'_t \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t)
\end{aligned}
\tag{2.15}
$$

2.1.4 Autocodificadores

O Autocodificador (*autoencoder*) (AC), é uma rede neural que mapeia a entrada x para um espaço vetorial latente, z, por meio de um módulo de codificador E, e depois a decodifica para uma saída x' com as mesmas dimensões da entrada original, por meio do módulo de decodificador D [3]. Esse processo é resumido por:

$$
\begin{aligned}
z &= E(x) \\
x' &= D(z)
\end{aligned}
\tag{2.16}
$$

Nessa rede, o processo de aprendizagem visa minimizar uma função de perdas:

$$
L = d(x, D(E(x))) \tag{2.17}
$$

Onde d é uma função de perda que penaliza $D(E(x))$ por ser diferente de x. Na prática, os ACes clássicos não levam a espaços latentes particularmente úteis ou bem estruturados [3].

Um AC é dito incompleto quando a dimensão de saída do codificador é menor que a dimensão de entrada [9]. Nesse caso, o codificador realiza sucessivas transformações de filtragens e reduções na dimensionalidade dos dados de entrada, em um processo chamado de subamostragem, do inglês (*downsampling*), e produz o vetor latente. O vetor latente segue para o decodificador que irá aumentar a dimensionalidade dos dados na etapa de superamostragem (*upsampling*) e produz uma versão aproximada da entrada. Aprender uma representação incompleta força o *autoencoder* a capturar os recursos mais destacados dos dados de treinamento [9].

2.1.4.1 Autocodificador Esparso

Um AC esparso possui um critério de treinamento que envolve uma penalidade de esparsidade $\Omega(z)$ na camada da representação do vetor latente, além do erro de reconstrução [9]:

$$L = d(x, D(E(x))) + \Omega(z) \tag{2.18}$$

Se impusermos uma restrição de esparsidade às unidades do latente, o codificador automático descobrirá uma estrutura interessante nos dados, mesmo que o número de unidades ocultas seja grande [22]. Para uma função de ativação logística, considera-se um neurônio como sendo "ativo" se seu valor de saída for próximo de 1 ou como "inativo" se seu valor de saída for próximo de 0. Sendo z_j a ativação de uma unidade do vetor latente z para uma entrada específica x_m podemos definir uma média da ativação dessa unidade para um conjunto com M exemplos de treinamento:

$$\hat{\rho}_j = \frac{1}{M} \sum_{i=1}^{M} z_j \tag{2.19}$$

Em um AC esparso desejamos obter neurônios inativos na maioria das vezes, isto é, queremos impor a restrição:

$$\hat{\rho}_j = \rho \tag{2.20}$$

onde ρ é um parâmetro de esparsidade, normalmente um valor pequeno próximo a zero. Então, escolhemos uma função que penaliza $\hat{\rho}_j$ por ser diferente de ρ. Uma opção, é usar a divergência de de Kullback-leibler (KL) [23] entre duas variáveis aleatórias de Bernoulli

com médias ρ e $\hat{\rho}_j$: [22].

$$D_{KL}(\rho||\hat{\rho}_j) = \sum_{j=1}^{h} \rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_j} \tag{2.21}$$

Aqui, h é o número de elementos do latente. A divergência KL é uma função padrão para medir a diferença entre duas distribuições diferentes. Ela tem a propriedade que $D_{KL}(\rho||\hat{\rho}_j) = 0$ se $\rho = \hat{\rho}_j$ e, caso contrário, aumenta monotonicamente à medida que $\hat{\rho}_j$ diverge de ρ [23]. Então podemos reescrever a Equação 2.18 como:

$$L = d(x, D(E(x))) + \lambda \sum_{j=1}^{s_2} D_{KL}(\rho||\hat{\rho}_j) \tag{2.22}$$

Onde λ controla o peso do termo de penalidade de esparsidade. Geralmente, restringimos o latente a ser de baixa dimensão e esparso para que o codificador atue para compactar os dados de entrada em menos bits de informação [3].

2.1.4.2 Autocodificador Variacional

O autocodificador variacional (*variational autocoders*) (VAE) [24, 25] é um modelo generativo profundo capaz de aprender representações latentes não supervisionadas de dados [26]. Um VAE transforma uma entrada em parâmetros de uma distribuição estatística. Isso significa assumir que a entrada foi gerada por um processo estatístico e que a aleatoriedade desse processo deve ser levada em consideração durante a codificação e decodificação [3].

Seja $X = \{x^i\}_{i=1}^N$ uma base de dados consistindo de N amostras independentes e identicamente distribuídas, gerada de uma variável aleatória x. Um AC opera com dois mapeamentos, o codificador opera $Enc_\phi : X \to Z$ e o decodificador $Dec_\theta : Z \to X$, onde Z é o espaço latente. No caso do VAE, ambos os mapeamentos são probabilísticos e uma distribuição fixa a *prior* $p(z)$ sobre Z é assumida. Como a distribuição de x também é fixa (distribuição de dados real $q(x)$), os mapeamentos Enc_ϕ e Dec_θ induzem distribuições conjuntas $q(x,z) = q_\phi(z|x)q(x)$ e $p(x,z) = p_\theta(x|z)p(z)$, respectivamente (omitindo a dependência com os parâmetros θ e ϕ) [27]. O objetivo idealizado do VAE é obter a função de verossimilhança logarítmica (*log-likelihood*) marginalizada:

$$\sum_{i=1}^{N} \log(p(x^i)) \tag{2.23}$$

Entretanto, esse objetivo não é tratável e é aproximado pelo limite inferior da evidência (ELBO) [24]. Para um x_i fixo, o logarítmico de verossimilhança $\log p(x^i)$ tem o limite

inferior dado por $\mathcal{L}(q)$ conforme a seguinte equação:

$$\mathcal{L}(q) = \mathbb{E}_{z\sim q(z|x^i)} \log p(x_i|z) - D_{KL}(q(z|x^i)||p(z)) \leq \log p(x^i) \tag{2.24}$$

onde o primeiro termo corresponde à perda de reconstrução e o segundo à divergência KL entre a representação latente $q(z|x_i)$ e a distribuição a *prior* $p(z)$. Por fim, $p(z)$ é definido como uma distribuição gaussiana multidimensional com média zero: $\mathcal{N}(0, \mathcal{I})$ onde $\mathcal{I}$ é a matriz identidade. O codificador também assume o formato de uma distribuição gaussiana dada por:

$$Enc_\theta(x) \sim q_\phi(z|x) = \mathcal{N}(\mu_\phi(x), diag\ \ \sigma^2_\phi(x)) \tag{2.25}$$

onde μ_ϕ e σ_ϕ são mapeamentos determinísticos que dependem dos parâmetros ϕ. A matriz de covariância é aplicada para ser diagonal. Os parâmetros de geração e inferência, θ e ϕ, são treinados em conjunto pela maximização de $\mathcal{L}$, onde usamos o truque de reparametrização para aplicar a descida estocástica do gradiente através das variáveis latentes gaussianas [28]. Além disso, todas as expectativas na Equação 2.24 podem ser aproximadas pela amostragem de Monte Carlo [9]. Uma vez concluído o treinamento, a distribuição a *posterior* aproximado $q_\phi(z|x)$ funciona como um codificador.

2.2 COMPRESSÃO DE DADOS

Nessa seção, apresentamos o problema da compressão de dados e o papel desempenhado pelo par Codificador/Decodificador (*Encoder/Decoder*) (CODEC). Além disso, também é tratado, superficialmente, um desenvolvimento probabilístico proposto por Shanon para mensurar a informação e estabelecer, dentro das suas hipóteses, um valor mínimo ótimo de taxa de compressão sem perdas.

A compressão de dados compreende o processo de reduzir o espaço em memória ocupada por dados que representam uma informação. Os dados são os meios pelos quais as informações são transmitidas [29]. Sejam b e b' o número de bits de duas representações das mesmas informações, a taxa de compressão C e a redundância relativa de dados, Rd, da representação b em relação a b' são:

$$\begin{aligned} C &= \frac{b}{b'} \\ Rd &= 1 - \frac{1}{C} \end{aligned} \tag{2.26}$$

Considerando que a informação representada pelos b bits contém s símbolos, então,

defini-se a taxa dessa representação ou número médio de bits como $R = \frac{b}{s}$ bits por símbolo (bps).

O codificador é o algoritmo utilizado para explorar as redundâncias na informação e gerar um conjunto de dados comprimidos. O algoritmo do decodificador realiza o processo de reconstrução dos dados. Ele toma os dados comprimidos e tenta recuperar os dados originais que representam a informação. Esse processo está ilustrada na Figura 2.10.

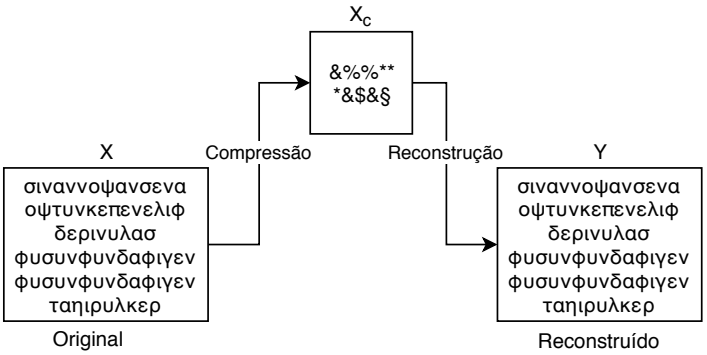


Figura 2.10 – O algoritmo de compressão (codificador) recebe uma entrada X e gera uma representação X_c que requer menos bits. O algoritmo de reconstrução (decodificador) opera na representação compactada X_c para gerar a reconstrução Y. Adaptado de [30].

Esses dois algoritmos formam o CODEC. Ele pode ser projetado para tratar com categorias de informações específicas que incluem, por exemplo, imagens, áudios e vídeos. A compressão pode ser com perdas e sem perdas. No primeiro tipo o codificador introduz restrições que faz o decodificador recuperar dados que aproximam a representação da informação original ao passo que na compressão sem perdas eles são recuperados iguais aos originais [30].

2.2.1 Medidas de informação

A geração da informação pode ser modelada como um processo probabilístico medido de maneira intuitiva. Conforme essa suposição, dizemos que um evento aleatório E que ocorra com probabilidade $P(E)$ contém $I(E)$ unidades de informação dada por [31]:

$$I(E) = log_b \frac{1}{P(E)} = -\log(P(E)) \tag{2.27}$$

onde a base b do logaritmo define a unidade utilizada para medir as informações. Se a base 2 for selecionada, a unidade de informação é o bit. $I(E)$ também é denominado de auto-informação. Se tivermos um conjunto de eventos independentes A_i provenientes de

algum experimento S, tais que:

$$\bigcup A_i = S \tag{2.28}$$

sendo S o espaço amostral, então a auto-informação média associada ao experimento aleatório é fornecida por:

$$H(x) = -\sum P(A_i) \log_b(P(A_i)) \tag{2.29}$$

Essa quantidade é chamada entropia associada ao experimento. Se o experimento é uma fonte que gera os símbolos A_i de um conjunto $\mathcal{A}$ denominado alfabeto da fonte, a entropia é uma medida do número médio de símbolos binários necessários para codificar a saída da fonte [30]. Ademais, essa quantidade é a mínima necessária para que qualquer método de compressão codifique essa fonte sem perdas.

Contudo, se o conjunto de eventos gerados são dependentes devemos considerar tais dependências para pode estimar de forma mais precisa a entropia real da fonte. Uma possibilidade para realizar essa estimação é observar distribuições conjuntas de sequências de símbolos cada vez mais longas [30]. Para uma sequência de comprimento n, podemos definir a entropia de ordem n que se aproxima da entropia real da fonte à medida que n aumente.

2.3 IMAGENS

Nessa seção explicamos como as imagens digitais podem ser representadas, quais são as suas principais características que são exploradas pelos CODECs. Em seguida, apresentamos o algoritmo base do padrão JPEG e as métricas objetivas para avaliar imagens comparativamente,

Uma imagem pode ser definida como uma função bidimensional, $f(x, y)$, em que x e y são coordenadas espaciais (plano), e a amplitude de f em qualquer par de coordenadas (x, y) é chamada de intensidade ou valor do *pixel* nesse ponto [29]. Em uma imagem digital os valores de x,y e f são discretos e finitos. Uma imagem monocromática requer apenas um número para indicar a intensidade de cada amostra espacial. As imagens coloridas, por outro lado, exigem pelo menos três números por posição de pixel para representar com precisão as cores [32].

2.3.1 Espaço de Cores

As três características normalmente utilizadas para distinguir as cores entre si são: brilho ou luminância (*brightness*), matiz (*hue*) e saturação (*saturation*). O brilho representa a noção de intensidade luminosa da radiação, o matiz é uma propriedade associada ao comprimento de onda predominante na combinação das várias ondas visíveis, enquanto a saturação expressa a pureza do matiz ou o grau de mistura do matiz original com a luz branca [31]. O matiz e a saturação são denominados conjuntamente de cromaticidade. O método escolhido para representar brilho e cor é descrito como um espaço de cores [32]. O objetivo dos modelos de cores é permitir a especificação de cores em um formato padronizado e aceito por todos [29].

2.3.1.1 RGB

No espaço de cores Vermelho, Verde, Azul (*Red, Green, Blue*) (RGB), a imagem colorida é representada com três números que indicam as proporções relativas de vermelho, verde e azul, as três cores primárias aditivas da luz [32]. Combinar vermelho, verde e azul em proporções variadas pode criar qualquer cor. O modelo pode ser visto como um cubo onde três de seus vértices são as cores primárias, o vértice junto à origem é o preto e o mais afastado da origem corresponde à cor branca [29].

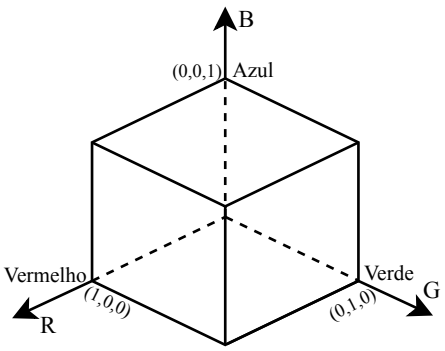


Figura 2.11 – Espaço de cores RGB representado em um cubo. Adaptado de [29].

2.3.1.2 YCbCr

O sistema visual humano é menos sensível à cor do que à luminância (Y). No espaço de cores RGB, as três cores são igualmente importantes e, geralmente, armazenadas na mesma resolução. Contudo, é possível representar uma imagem colorida com mais eficiência, separando a luminância das informações de cores e representando Y com uma resolução maior que a cor [32]. A componente de luminância é calculado como uma média entre R, G e B,

ponderadas pelos pesos k_r, k_g e k_b, respectivamente.

$$Y = k_r R + k_g G + k_b B \tag{2.30}$$

Cada componente de crominância é a diferença entre R, G ou B e Y.

$$\begin{aligned} Cr &= R - Y \\ Cb &= B - Y \\ Cg &= G - Y \end{aligned} \tag{2.31}$$

2.3.2 Compressão

No contexto da compressão digital de imagens os três principais tipos de redundância de dados que podem ser identificados e explorados são [29]:

1. Redundância de codificação. Os códigos de 8 bits utilizados para representar as intensidades na maioria dos arranjos de intensidade 2-D contêm mais bits do que o necessário para representar as intensidades.

2. Redundância espacial. Como os *pixels* da maioria dos arranjos de intensidade 2-D são correlacionados no espaço (isto é, cada pixel é similar aos *pixels* vizinhos ou dependente deles), as informações são desnecessariamente replicadas nas representações dos *pixels* correlacionados.

3. Informações irrelevantes. A maioria dos arranjos de intensidade 2-D contém informações ignoradas pelo sistema visual humano e/ou irrelevantes para a utilização pretendida da imagem.

2.3.2.1 JPEG

O JPEG é o acrônimo de *Joint Photographic Experts Group*. Foi concebido por um esforço conjunto do *Consultative Committee for International Telephony and Telegraphy* (CCITT) (agora International Telecommunication Union (ITU-T)) e da International Organization for Standardisation (ISO). Ele é um dos padrões mais conhecidos para compactação de imagem com perdas, amplamente usado na *Web* e em máquinas fotográficas [30]. As principais etapas desse CODEC estão descritas a seguir.

1. A primeira etapa consiste em transformar uma imagem colorida para o espaço de cores de luminância e crominância, como YCbCr. Em seguida subtraímos cada elemento da imagem por 2^{P-1}, onde P é o número de bits usado para representar cada pixel [30].

2. As componentes de crominância das imagens coloridas podem ser subamostradas reduzindo as suas resoluções espaciais. A redução da amostragem pode ser feita por um fator de 2 na direção vertical e horizontal (amostragem 4:2:0) ou por um fator de 2 somente na horizontal (amostragem 4:2:2) [33].

3. Os canais de cor são recortadas em blocos de 8×8 *pixels* e compactados separadamente.

4. Aplicamos a Transformação Discreta de Cosseno (*Discrete Cosine Transform*) (DCT) a cada bloco para criar um mapa 8×8 dos componentes de frequência. Isso prepara os dados da imagem para a etapa crucial de perda de informações. Como a DCT envolve alguma perda de informação, normalmente pequena, devido à precisão limitada da aritmética do computador [33]. A DCT-2D para uma imagem $m \times n$ é calculada por:

$$\begin{aligned} G_{ij} &= \sqrt{\frac{2}{m}}\sqrt{\frac{2}{n}} C_i C_j \sum_{x=0}^{n-1}\sum_{y=0}^{m-1} p_{xy} cos\left[\frac{(2y+1)j\pi}{2m}\right] cos\left[\frac{(2x+1)i\pi}{2n}\right] \\ C_f &= \begin{cases} \frac{1}{\sqrt{2}} & \text{if } f = 0 \\ 1 & \text{if } f > 0 \end{cases} \end{aligned} \tag{2.32}$$

   Para $0 \leq i \leq n-1$ e $0 \leq j \leq m-1$. O primeiro coeficiente, G_{00} é denominado coeficiente DC e os demais são chamados coeficientes AC.

5. O JPEG usa quantização uniforme para quantificar os coeficientes da DCT. Os tamanhos das etapas do quantizador são organizados em uma tabela chamada tabela de quantização e podem ser vistos como a parte fixa desse processo [30]. Cada valor quantizado é representado por um rótulo I_{ij} e obtido a partir dos respectivos coeficientes G_ij e do elemento da matriz de quantização Q_{ij}, conforme descrito pela próxima equação.

$$I_{ij} = \left\lfloor \frac{G_{ij}}{Q_{ij}} + 0.5 \right\rfloor \tag{2.33}$$

6. Os rótulos são lidos em zigue-zague para fornecer uma representação unidimensional e agrupar os coeficientes quantizados de maior importância no início da sequência. Os 64 coeficientes de frequência quantizados para cada unidade de dados são codificados usando uma combinação de codificação Codificação de Comprimento de Execução (*Run-Length Encoding*) (RLE) e Huffman [33].

O esquema de decodificação é o inverso dos passos da codificação [30]. Dessa forma, a partir do conjunto de bits efetuaremos: a decodificação de entropia, reescalonamento dos dados, transformada IDCT, recuperação da imagem a partir dos blocos, superamostragem, somar 2^{P-1} em cada *pixe*l e, por fim, transformar a imagem para o espaço de cor RGB. A Figura

2.12 resume o esquemático de codificação e decodificação de uma imagem usando o padrão JPEG.

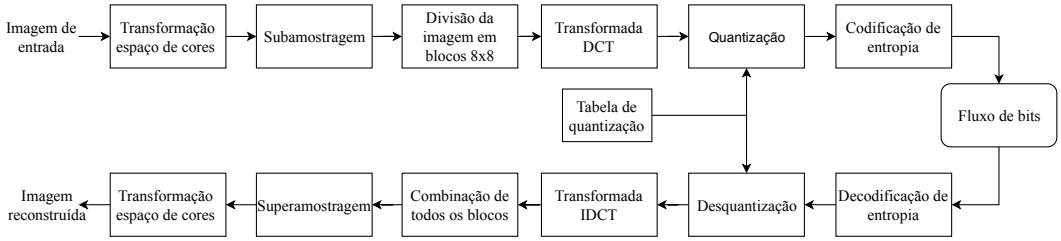


Figura 2.12 – Etapas da codificação e decodificação de uma imagem através do JPEG.

2.3.3 Métricas de Qualidade

As métricas de qualidade são utilizadas para avaliar a fidelidade entre a informação que passou por um processo de compressão com perdas com a informação original. A qualidade visual é inerentemente subjetiva e, portanto, influenciada por muitos fatores subjetivos que dificultam a obtenção de uma medida de qualidade completamente precisa [32]. Em imagens, as principais métricas objetivas (calculadas automaticamente) são a Relação Pico-Sinal-Ruído (*Peak Signal to Noise Ratio*) (PSNR), Índice de Similaridade Estrutural (*Structural Similarity Index*) (SSIM) e Índice de Similaridade Estrutural em Multi-Escalas (*Multi-scale Structural Similarity Index*) (MS-SSIM) [32]. As duas últimas são métricas que tentam levar em consideração a percepção humana de qualidade.

2.3.3.1 PSNR

A PSNR depende do valor máximo teórico de um pixel (255 para imagens de 8 bits) e do Erro Médio Quadrático (*Mean Square Error*) (MSE) entre a imagem original e a sua versão reconstruída. Para uma imagem monocromática x de dimensões $n \times m$ representada por P bits e sendo x' a sua reconstrução, teremos:

$$MSE(x, x') = \frac{1}{nm} \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} (x(i,j) - x'(i,j))^2 \tag{2.34}$$

$$PSNR_{dB}(x, x') = 10 \log \frac{(2^P - 1)^2}{MSE(x, x')} \tag{2.35}$$

2.3.3.2 SSIM

O SSIM foi projetado para modelar qualquer distorção da imagem como uma combinação da perda de correlação, distorções da luminância e do contraste [34]. Ele é definido

por:

$$SSIM(x, x') = l(x, x') \times c(x, x') \times s(x, x') \tag{2.36}$$

Onde

$$\begin{aligned} l(x, x') &= \frac{2\mu_x\mu_{x'} + C_1}{\mu_x^2 + \mu_{x'}^2 + C_1} \\ c(x, x') &= \frac{2\sigma_x\sigma_{x'} + C_2}{\sigma_x^2 + \sigma_{x'}^2 + C_2} \\ s(x, x') &= \frac{\sigma_{xx'} + C_3}{\sigma_x\sigma_{x'} + C_3} \end{aligned} \tag{2.37}$$

O primeiro termo em 2.37 é a função que mede a proximidade da luminância média de duas imagens [34]. Os valores de μ_x e $\mu_{x'}$ são as médias das imagens x e x' que podem ser vistas como estimativas das luminâncias [35].

O segundo é a função de comparação de contraste entre duas imagens [34]. Aqui, o contraste é aproximado pelo desvio padrão σ_x e $\sigma_{x'}$. O terceiro termo é a função de comparação de estrutura que mede o coeficiente de correlação entre as duas imagens x e x' [34]. O termo $\sigma_{xx'}$ é a covariância entre as imagens. As constantes positivas C_1, C_2 e C_3 são usadas para evitar um denominador nulo. O SSIM varia de 0 a 1, onde 0 significa que não existe similaridade estrutural, e 1 significa que a similaridade estrutural é máxima, isto é, $x = x'$.

2.3.3.3 MS-SSIM

O MS-SSIM [35] é uma variação do SSIM para várias escalas de imagens e conveniente para incorporar seus detalhes em diferentes resoluções. A imagem original e a reconstruída estão na escala 1. As imagens na escala 2 são obtidas após aplicar um filtro passa-baixas nas imagens da escala 1 (escala anterior). Esse filtro realiza subamostragem nas imagens por um fator de 2 [35]. Esse processo se repete até obtermos as imagens de escala M. Essas imagens são usadas para compor o cálculo do MS-SSIM. Essa métrica retorna valores entre 0 (ausência de similaridade) e 1 (reconstrução sem perdas).

3 REVISÃO BIBLIOGRÁFICA

Neste capítulo, apresentamos uma revisão bibliográfica de CODEC's de imagem desenvolvidos com técnicas de redes neurais artificiais. O autocodificador é uma RNA presente na maioria dos trabalhos. As diferenças estão na arquitetura dos autocodificadores, na modelagem do problema, emprego de análises probabilísticas, generativas, de segmentação semântica, etc.

3.1 INTRODUÇÃO

As técnicas de redes neurais para compressão de imagens é um tema que vem despertando o interesse de muitos pesquisadores e já apresenta resultados competitivos com o BPG (considerado o estado da arte de compressão de imagens) em alguns trabalhos [36, 37, 38].

Há 3 principais abordagens para incluir RNA's na tarefa de compressão de imagens [39]. A primeira é o desenvolvimento completo do par codificador-decodificador. A segunda consiste na implementação de redes neurais dos algoritmos tradicionais de compactação de imagem. A terceira abordagem é usar uma RNA para substituir, aperfeiçoar ou adicionar alguma etapa na compressão de imagens realizadas por CODEC' conhecidos. Aqui, o objetivo é fornecer melhorias adicionais a esses algoritmos [39]. Hoje, a primeira abordagem é a predominantes e vem obtendo os melhores resultados.

O problema central na compressão de imagens com perdas é a otimização conjunta da taxa e distorção. Esse objetivo se torna intratável para imagens ou espaços de alta dimensão sem a imposição de alguma restrição [40]. Devido a isso, os CODEC's tradicionais aplicam transformações lineares para gerar uma representação adequada sobre a qual podemos considerar a otimização conjunta de distorção e taxa [41]. Entretanto, não há razão para esperar que uma função linear seja ideal para comprimir todo o espectro de imagens naturais e ainda considerando os formatos de mídia emergentes [42].

As abordagens baseadas em RNA [43, 36, 44, 36, 1, 41, 37, 45, 46, 47] substituem as transformações lineares por funções básicas aprendidas capazes de aprender recursos das imagens. Nesses trabalhos, a arquitetura do autocodificador contém ao menos uma camada não-linear, e o modelo como um todo é não-linear.

3.2 AUTOCODIFCADORES EM CADEIA

Nessa seção, apresentamos trabalhos que usam arquiteturas baseadas em cascateamento de autocodificadores para compressão de imagens. Várias camadas de RNA são testadas na tentativa de aprimorar a peromance do modelo.

Em [43], Toderici et. al propuseram arquiteturas de redes neurais voltados para compressão de imagens em miniaturas. As arquiteturas são baseadas no uso de autocodificadores encadeados e treinadas pelas imagens e suas informações residuais obtidas progressivamente. Esse encadeamento pode ser explícito, isto é, todos os autocodificadores foram explicitamente definidos, portando, cada um possui seus próprios pesos e camadas. No encadeamento implícito apenas um autocodificador é projetado, porém, durante a execução do modelo, conexões de retroalimentação são realizadas nesse autocodificador. Assim temos um cascateamento devido ao *loop*, onde os pesos do autocodificador são compartilhados nas iterações. O número escolhido de autocodificadores (implícitos ou explícitos) definem a quantidade de iterações do modelo.

Durante a execução (treinamento ou teste) de qualquer uma das arquiteturas apresentadas em [43] a primeira iteração segue o seguinte procedimento: toma uma imagem $x = r_0$ passa pela função E_1 (codificador 1), o resultado é transformado em códigos binários por uma função B (função de binarização), em seguida a rede do decodificador D_1 (decodificador 1) cria uma estimativa da imagem de entrada original com base no código binário recebido. Então, calcula-se o resíduo da primeira iteração, dado por $r_1 = r_0 - D_1(B(E_1(r_0)))$. Na próxima iteração, o E_2 recebe a informação residual e uma saída é gerada por D_2. Entretanto, a nova saída nessa iteração e das próximas, podem ser a informação residual reconstruída ou uma nova versão da imagem original (reconstrução única), como veremos. Após o modelo realizar o número defnido de etapas, passamos a processar a próxima imagem e o ciclo se repete.

Podemos resumir a ideia do encadeamento de autocodificadores com a próxima Equação:

$$F_t(r_{t-1}) = D_t(B(E_t(r_{t-1}))) \tag{3.1}$$

onde E_t e D_t são o codificador e decodificador da iteração t-ésima para a qual r_{t-1} é a entrada. A função de binarização B é o mesmo em todas as iterações. A equação para obter o resíduo r_t depende da informação de saída do autocodificador F_t. Os pares codificador-decodificador são treinados de ponta a ponta, mas durante a implantação, eles são normalmente usados independentemente.

Antes de apresentar os modelos com mais detalhes, discutiremos o processo de binarização adotado.

3.2.1 Binarização

O trabalho [43] propõe uma operação de binarização e um método para transpor a limitação que esse processo impõe ao treinamento de uma RNA. Nessa proposta, uma camada totalmente conectada com ativação de $\tanh$ é usada para produzir as saídas no intervalo contínuo $[1, -1]$. Em seguida, uma abordagem estocástica usada durante o treinamento da rede é aplicada para converter desses valores para o conjunto discreto $\{-1, 1\}$. Sendo x um valor a ser quantizado e ϵ o erro de quantização devemos ter.

$$\begin{aligned} b(x) &= x + \epsilon \\ b(x) &\in \{-1, 1\} \end{aligned} \tag{3.2}$$

O valor ϵ é obtido dentre dois valores possíveis e obedece a seguinte regra estocástica:

$$\epsilon \sim \begin{cases} 1 - x, \text{com probabilidade } \frac{1+x}{2} \\ -x - 1, \text{com probabilidade} \frac{1-x}{2} \end{cases} \tag{3.3}$$

Na prática, definimos um valor u entre 0 e 1 obtido de uma distribuição uniforme, isto é, $u \in \mathcal{U}[0, 1]$ e podemos reescrever 3.3 como:

$$\epsilon = \begin{cases} 1 - x, \text{ se } u \leq \frac{1+x}{2} \\ -x - 1, \text{ se } u > \frac{1+x}{2} \end{cases} \tag{3.4}$$

A função de binarização B é dada por:

$$B(x) = b(\tanh W^{bin} + b^{bin}) \tag{3.5}$$

onde, W^{bin} e b^{bin} são os pesos e viés lineares da camada imediatamente anterior à binarização. Para permitir a retropropagação do erro pela rede, substituímos o gradiente da binarização (que possui valor nulo ou indefinido) pelo gradiente da expectativa da binarização:

$$\frac{\partial \mathbf{E}[b(x)]}{\partial x} = \frac{\partial x}{\partial x} = 1, \ \forall x \in [-1, 1] \tag{3.6}$$

Esse resultado significa que o algoritmo de retropropagação ignora a operação de binarização no cálculo do gradiente de todos os pesos da rede. Durante o teste do modelo (após o seu treinamento) a binarização é simplificada pelo resultado mais provável de $b(x)$:

$$b^{inf}(x) = \begin{cases} -1, \text{se } x < 0, \\ +1, \text{caso contrário.} \end{cases} \tag{3.7}$$

3.2.2 Redes não-recorrentes

Seguindo com o trabalho [43], quando a arquitetura é composta apenas por camadas não-recorrentes o encadeamento de autocodificadores é explícito e F_t sempre fará uma estimativa do resíduo. Logo:

$$r'_t = F_t(r_t)//r_{t+1} \quad = r'_t - r_t \tag{3.8}$$

No primeiro modelo concebido, as redes do codificador e decodificador são formadas por camadas totalmente conectadas com 512 unidades (neurônios) exceto a última camada do codificador (às vezes denominado de camada de binarização) que é formada por 4 neurônios. Assim, a rede fornece um vetor latente de 4 bits por iteração. A reconstrução da imagem original é obtida progressivamente pela soma dos resíduos gerados, conforme Equação a seguir.

$$x' = \sum_{t=0}^{k-1} r'_t \tag{3.9}$$

A Figura 3.1 ilustra tal arquitetura em duas iterações e omitindo a última camada do decodificador. Ela é responsável por converter os dados da camada anterior para uma imagem RGB de mesma dimensão que a entrada.

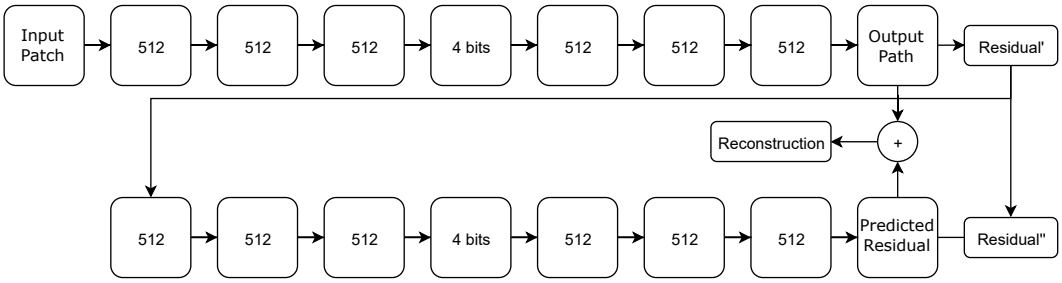


Figura 3.1 – Um autocodificador residual totalmente conectado com função de ativação $\tanh$. Esta figura representa uma arquitetura de duas iterações. As primeiras iterações codificam a imagem original. Os resíduos da reconstrução são passados para a segunda iteração. Cada nível de iteração produz 4 bits [43].

Os autores estenderam essa arquitetura substituindo as camadas totalmente conectadas pelos operadores de convolução no codificador E e pelos operadores "deconvolucionais" ou de convolução transposta no decodificador D [43].

3.2.3 Redes Recorrentes

Em autocodificadores recorrentes podemos incluir a memória da rede para capturar as dependências entre os resíduos gerados em cada iteração. Além disso, essa característica

permite ao modelo estimar diretamente a imagem original em cada estágio t, isto é, $F_t(r_{t-1})$ = x'_t ou obter a reconstrução dos resíduos por iteração como apresentado na Equação 3.8. Nessas abordagens, o encadeamento de autocodificadores é implícito (uso recursivo do autocodificador). Outra motivação de usar camadas recorrentes é a redução no número de parâmetros do modelo a serem aprendidos.

Ainda no trabalho [43], explorou-se o uso de modelos recorrentes em duas arquiteturas. Na primeira, o codificador é composto por uma camada totalmente conectada seguida por duas camadas LSTM empilhadas. O decodificador tem a estrutura oposta: duas camadas LSTM empilhadas seguidas por uma camada totalmente conectada com uma não-linearidade que prediz os valores RGB. Aqui, o modelo da LSTM segue a formulação da Equação 2.14.

A Figura 3.2 mostra um desenrolar dessa rede com camadas LSTM (menos a conversão RGB), necessária para o treinamento, em duas etapas de tempo. As conexões verticais entre os estágios LSTM no mostram o efeito de propagação de memória. Na segunda abordagem, os autores combinam os operadores convolucionais e deconvolucionais com a LSTM. Tal arquitetura obteve a melhor performance dentre os testes realizados.

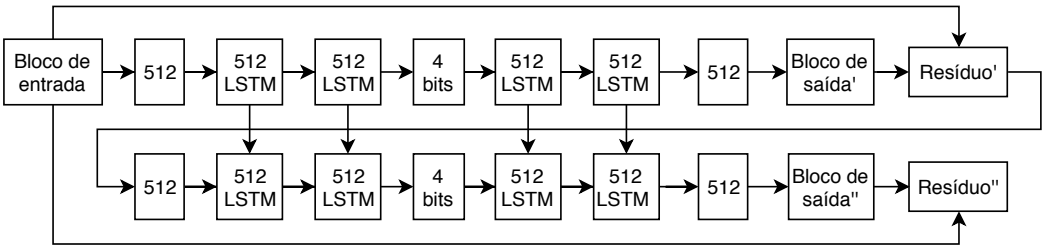


Figura 3.2 – Esta figura mostra o autocodificador com camadas LSTMem duas iterações de treinamento. Ao todo, a rede foi treinada com 16 níveis de resíduo, para gerar representações de 64 bits. As conexões verticais entre os estágios LSTM no desenrolamento mostram o efeito da memória persistente. Nessa arquitetura LSTM, cada etapa prevê a saída real. [43].

A função de custo utilizada foi a métrica L_2 nos resíduos. Ela é definida substituindo $p = 2$ na Equação a seguir [48]:

$$||v||_p = \left(\sum_{i=1}^{n} |v_i|^p \right)^{\frac{1}{p}} \tag{3.10}$$

onde, $v \in \mathbb{R}^n$ é um vetor sobre o qual estamos aplicando a norma $L - p$.

Em [1] foi proposto um modelo convolucional-recorrente como resultado de melhorias do trabalho em [43]. A ideia central permanece a mesma: encadeamento de autocodificadores treinados sobre os resíduos e com taxa de compressão fixa por iteração. Tal arquitetura é

representada compactamente através da Equação 3.11.

$$\begin{aligned} r_0 &= x \\ b_t &= B(E_t(r_{t-1})) \\ \hat{x}_t &= D_t(b_t) + \gamma \hat{x}_{t-1} \\ r_t &= x - \hat{x}_t \end{aligned} \tag{3.11}$$

onde E_t, D_t, b_t, $\hat{x}_t$ e r_t representam, respectivamente, codificador, decodificador, fluxo de bits, reconstrução progressiva da imagem original x, e o residual entre x e $\hat{x}_t$ na iteração t. Se o valor de γ é zero a reconstrução é do tipo única ("um disparo"), caso contrário, $\gamma = 1$, e teremos a reconstrução aditiva. Na reconstrução única cada iteração sucessiva tem acesso a mais bits gerados pelo codificador, o que permite uma melhor reconstrução.

O modelo é composto por camadas Conv2D; hibridas do tipo convolucional-recorrente (Conv2DRNR); e camadas estáticas de binarização e de Profunidade para Espaço (*Depth-to-Space*) (PE). A figura 3.3 apresenta um fluxograma da arquitetura para uma iteração.

As camadas Conv2DRNR seguem formulação matemática dependente do tipo de rede recorrente. Nela temos 4 camadas no : 3 convolucionais e 1 RNN-Conv; 1 camada de binarização do tipo convolucional; e 6 camadas no enconder: 2 convolucionais e 4 unidades RNN-Conv Em um dos modelos, os autores empregam a Conv2DLSTM descrita pela Equação 2.15. Também foram testadas a LSTM associativa e a Unidades Recorrentes Fechadas (*Gated Recurrent Units*) (GRU). Independentemente da camada recorrente, dentre as 3 citadas, as Conv2DRNR incluem duas convoluções 2D: uma no vetor de entrada, x_t, outra no vetor de estado oculto anterior, h_{t-1}, que será referida como "convolução oculta" realizada pelo "filtro oculto". Os resultados das convoluções são somados e seguem para uma etapa de processamento (dependente da rede recorrente) para fornecer h_t e c_t.

A PE interpola os *pixels* em diferentes canais para formar imagens de alta resolução de acordo com fator de escala, $upscale\_factor$. Essa operação também é conhecida como *PixelShuffle*. Para exemplificar, seja uma entrada X e fator de escala uf, então obteremos a saída Y segundo:

$$\begin{aligned} dim(X) &= (N, L, H_{in}, W_{in}) \\ L &= C \times \text{uf}^2 \\ Y &= pixel\_shuffle(X, \text{uf}) \\ dim(Y) &= (N, C, H_{in} \times \text{uf}, W_{in} \times \text{uf}) \end{aligned} \tag{3.12}$$

onde $dim\,(\odot)$ retornar a dimensão de um tensor. A PE reduz o número de canais e aumenta a dimensão espacial, por isso pode ser usada em substituição a convolução transposta.

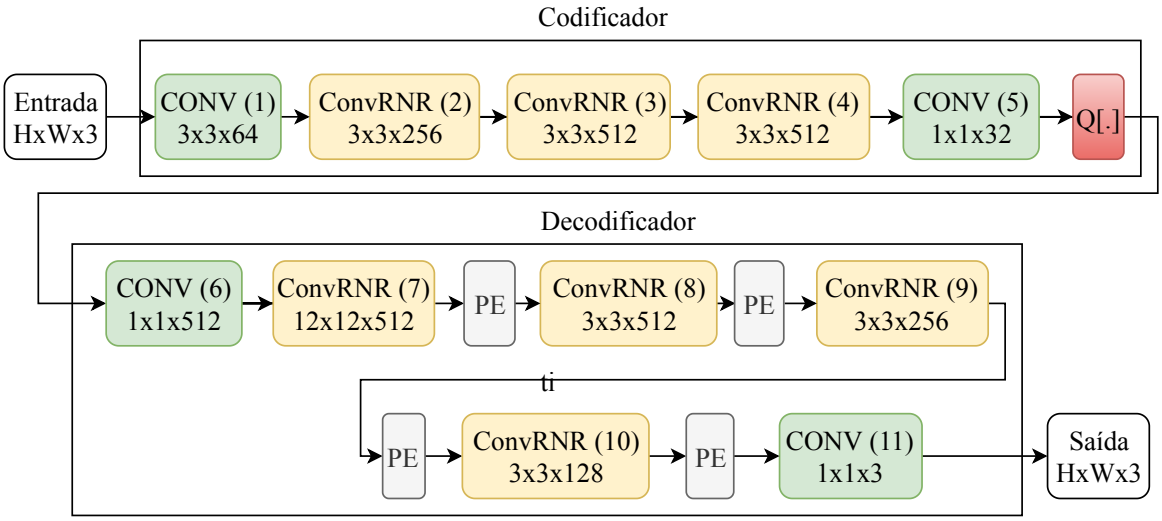


Figura 3.3 – Esta Figura mostra a arquitetura de um autocodificador com camadas recorrentes e convolucionais para compressão de imagens de alta resolução. Nas camadas convolucionais, em verde, está indicada o formato do seu filtro convolucional (altura x largura x número de canais). Para cada camada ConvRNR o formato de filtro mostrado contém as dimensões espaciais dos núcleos aplicados no tensor de entrada e o número de filtros desses núcleos e dos núcleos ocultos. As dimensões espaciais dos filtros ocultos nas ConvRNR são todas 1x1, exceto as camadas 9 e 10 onde são 3x3. Em vermelho está indicado a função de binarização. As camadas de profundidade para espaço reduzem por um fator de 4 o número de canais do tensor de entrada e aumenta cada dimensão espacial por um fator de 2. Todas as operações de convolução são bidimensionais. Figura adaptada de [1]

.

Durante o treinamento dos modelos em [1] uma perda ponderada de L_1 é calculada sobre os resíduos gerados em cada iteração, isto é, a perda total é $\beta \sum_t |r_t|$. Apenas ao final das iterações de recorrência é que ocorre a atualização dos pesos. A partir desse ponto diversas melhorias foram propostas na tentativa de contornar limitações impostas pelo modelo.

Ainda, em [1] os autores usam uma rede neural específica, G, para promover um fator de ganho nos resíduos. A saída dela é condicionada pela reconstrução e iteração, isto é, $g_t = G(x'_t)$. A intenção é impedir que o erro residual se torne ínfimo a ponto de comprometer a convergência do modelo. Outra melhoria proposta foi a adição de uma nova RNA de codificação de entropia para melhorar a taxa de compactação. De forma simplificada, é empregado uma rede recorrente binária, baseada em uma LSTM, que induz um modelo de probabilidade a um codificador aritmético.

Em [44] os autores modificaram a rede 3.3 para usar 0/1, em vez de -1/1 a fim de evitar artefatos de limite nas bordas das imagens. Além disso, eles implementaram um método para que a rede possa alocar mais bits nos blocos mais difíceis de reconstruir em comparação com os mais simples. Para que isso fosse possível uma camada com máscara é utilizada sobre o código binário gerado e aplicado após a primeira iteração da rede, no tempo de treinamento. O mascaramento pode ser transparente sobre esse código binário e, nesse caso, a saída dessa

camada coincide com o código binário. Contudo, se em uma dada iteração da rede pelo menos uma das seguintes condições forem satisfeitas, a máscara levará todos os bits para zero.

- Onde a qualidade de reconstrução do bloco excede o nível de qualidade desejado;
- A saída do codificador é acidentalmente zero;
- Os códigos foram mascarados em uma iteração anterior.

Essa lógica de mascaramento permite tratar um código de bits zero como um sinal de parada e evitar o envio de qualquer informação subsequente do codificador para o decodificador. Neste trabalho, outra mudança atingiu a função de perdas da rede. Para essa função, foi considerada a perda L_1 sobre os resíduos (erro de reconstrução) e uma função (não detalhada no referido artigo) proporcional a quantidade de bits diferente de zero. O propósito dessa segunda parcela é tornar o código binário com maior taxa de 0 e possibilitar compactação de entropia mais eficaz. A codificação de entropia usada foi a LempelZiv (LZ77). Os CODEC's de imagens que apresentam melhor desempenho se valem do conteúdo espacial da imagem para alocar os bits disponíveis de forma otimizada. Tal técnica é empregada, por exemplo, pelo JPEG2000, WebP e BPG [49]. No trabalho apresentado por Minnen et. al. em [50] é discutido um método para implementar uma alocação de bits com base na complexidade de regiões de imagens. Na prática, essa adaptação da taxa se dá pela correlação entre os blocos próximos de uma imagem.

Aqui, o CODEC proposto é composto por duas redes neurais para realizar tarefas específicas. A primeira rede deverá gerar previsões de blocos da imagem a partir dos blocos vizinhos, a segunda recebe o resíduo dessa previsão para reconstruí-lo progressivamente. Ao final, a reconstrução da imagem corresponde a primeira previsão somada aos resíduos reconstruídos. A primeira rede é um autocodificador convolucional, ilustrado na Figura 3.4, que recebe 1 bloco 64×64 ou, de forma equivalente, 4 blocos 32×32, sendo 3 deles da imagem original e um bloco com pixels zeros. Este último é o alvo que a rede tentará prever na saída ao minimizar o erro L_1 entre essa previsão e o bloco original. A figura 3.5 ilustra o processo de previsão descrito.

A segunda rede, irá complementar a tarefa da primeira ao reconstruir a informação residual. Ela corresponde a rede 3.3 com reconstrução aditiva e usando unidades Con2DLSTM.

No artigo [36], Johnston et al. implementaram 3 modificações na forma de treinar o modelo que o fez superar os CODEC's BPG(4:2:0), WebP, JPEG2000 e JPEG medidos na métrica MS-SSIM.

A primeira mudança foi na execução de iterações falsas na rede (passos adicionais) a fim de inicializar os estados ocultos das camadas recorrentes (h_t e c_t) com valor diferente

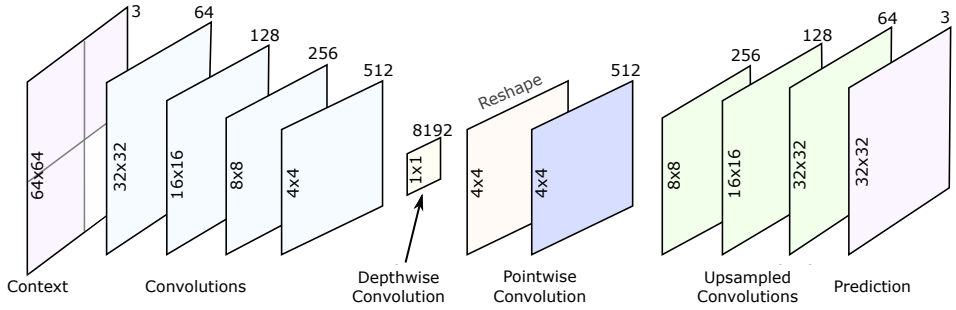


Figura 3.4 – A rede de previsão com base no contexto espacial é um codificador automático que recebe uma imagem de 64×64, com zeros no bloco atual e gera esse bloco ausente[50].

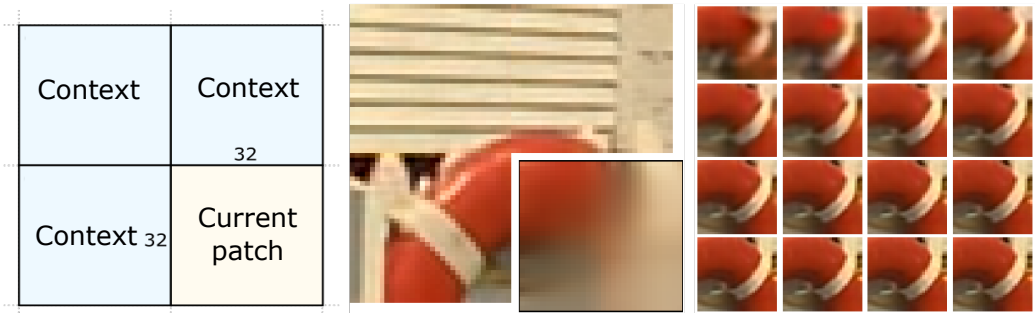


Figura 3.5 – Para cada 32×32, o modelo DPCM tenta prever com base nos blocos vizinhos. Os resíduos são propagados para o autocodificador [50].

de zero - essa técnica foi chamada de *k-priming* de estado oculto. Nessa técnica, um lote de imagens passa k vezes pela rede do codificador descartando os bits gerados nas primeiras k-1 vezes (falsas iterações). Todavia as alterações nos tensores de estado oculto das unidades recorrentes do codificador são mantidas. Quando o lote passa pela k-ésima vez pelo codificador teremos uma iteração real e os bits são transmitidos para o decodificador. De forma análoga, na rede do decodificador esses bits são usados para reconstruir k vezes o conjunto de imagens. Dessa forma, a rede consegue inicializar os estados ocultos do decodificador e na iteração k a reconstrução será válida para continuar o treinamento.

O *k-priming* pode ser executado, também, entre as iterações e é especificamente denominado por *k-difusão*. Essa técnicas proporcionaram melhores resultados, mas ao custo de exigir mais tempo de execução e recursos computacionais [36].

Como foi discutido a abordagem de redes neurais em cascata adiciona uma quantidade fixa de bits em cada iteração. Esse fato torna a rede ineficiente na medida que, independentemente da complexidade da imagem, o mesmo número de bits são utilizados para representá-la. A solução proposta pelos autores em [36] é, dada uma qualidade de destino, atribuir a cada bloco de 16×16 pixels uma quantidade de bits suficientes para atender ou exceder a qualidade de destino até o máximo suportado pelo modelo. Esse processo ocorre no tempo de teste da rede (após o seu treinamento) e algumas heurísticas foram adotadas para evitar

artefatos de compressão. O bloco pode ser subdividido em 4 blocos 8×8, sobre os quais obtém-se a norma L_1 do erro residual. A qualidade do bloco 16×16 será tomado como o valor máximo dentre as 4 medidas de erro. Cada bloco 16×16 usará entre 50% e 120% da taxa de bits de destino (arredondado para a iteração mais próxima). O codificador terá que criar uma matriz para indicar quantos bits foram reservados para cada localização da imagem. O algoritmo do decodificador requer que todos os bits estejam presentes para reconstruir uma imagem, logo ele preenche com um valor numérico as regiões nas quais o codificador reservou menos bits que o máximo possível. O valor adotado pelos autores foi 0, o valor médio da binarização (-1 e 1). Por fim, a terceira mudança recaiu na função de custo. Os autores adotaram uma função para considerar, além do erro tradicional de L_1, uma métrica perceptual de qualidade. Ela corresponde a métrica $L1$ ponderada por uma função que mede a dissimilaridade perceptual entre duas imagens 3.13.

$$\begin{aligned} L(x, x') &= w(x, x') \times L_1(x - x') \\ w(x, x') &= \frac{S(x, x')}{\bar{S}} \end{aligned} \tag{3.13}$$

onde $S(x, x')$ é uma medida perceptual da dissimilaridade entre as imagens x e x' e $\bar{S}$ é ajustada para a média móvel de $S(x, x')$. Intuitivamente, essa perda está realizando amostragem de importância dinâmica: compara a distorção perceptual de uma imagem com a distorção perceptiva média e pesa mais as imagens com alta distorção perceptual e menos pesadamente as imagens para as quais a rede de compressão já funciona bem. Na prática, $S(x, x')$ foi modela pela medida de dissimilaridade $D(x, x') = 0,5 \times (1 - SSIM(x, x'))$. A Tabela 3.1 apresenta as principais características dos autocodificadores em cadeia, relatos neste trabalho.

Tabela 3.1 – Tabela com informações sintetizadas do CODEC's de imagens baseados em autocodificadores em cadeia.

Autores	Autocodificador	Descrição	Função de custo e Base de treinamento	Controle de taxa e /ou qualidade	Comparação com CODEC's tradicionais
Toderici et. al. [43]	- Totalmente conectado - Recorrente - Convolucional - Convolucional LSTM - Encadeamento implícito ou explícito de ACs	- Modelo opera sobre os resíduos em 16 iterações - Reconstrução aditiva ou única	- Norma L_2 aplicada nos resíduos - Base com 216 milhões de imagens obtidas da internet e subamostradas para blocos 32×32	- Controle da taxa é feito pelo número de iterações na reconstrução dos blocos.	- O modelo convolucional LSTM supera o JPEG em SSIM dentro do intervalo 0,125 a 1,375 bpp.
Toderici et. al. [1]	- Convolucional LSTM - Convolucional LSTM associativa - Convolucional GRU - Encadeamento implícito	- Modelo opera sobre os resíduos em 16 iterações - Reconstrução aditiva ou única - RNA para camada de fator de ganho dos resíduos - RNA para gerar contexto de codificação aritmética	- Norma L_1 aplicada nos resíduos - Base reunida em [43] e um novo conjunto de blocos 32×32 de alta entropia recortados de 6 milhões de imagens 1280×720 que foram coletadas aleatoriamente da internet	- Controle da taxa é feito pelo número de iterações na reconstrução dos blocos - Codificação de entropia reduz a taxa	- O modelo Convolucional GRU de reconstrução única supera o JPEG em MS-SSIM e PSNR-HVS dentro do intervalo 0,1 a 1,5 bpp (aproximadamente)
Covell et. al. [44]	- Convolucional LSTM - Encadeamento implícito	- Modelo opera sobre os resíduos em 16 iterações. - Reconstrução aditiva com ganho residual - Implementação de códigos de parada - LZ77 é usado para codificação de entropia	- Norma L_1 aplicada nos resíduos e penalidade pela ocorrência de bits diferentes de zero - Base de treinamento formada por blocos 32×32	- Controle da taxa é feito pelo número de iterações na reconstrução dos blocos - A qualidade alvo é usada para adaptar as taxas de bits por blocos - Codificação de entropia reduz a taxa	- O modelo supera o JPEG em PSNR dentro do intervalo 0.125 a 1,375 bpp (aproximadamente)
Minnen et. al. [50]	- Convolucional LSTM - Encadeamento implícito	- Modelo opera sobre os resíduos em 16 iterações. - AC para predição de contexto - Reconstrução aditiva	- AC de contexto: norma L_1 sobre o resíduo - AC Conv2DLSTM: função de custo não especificada - Base de treinamento composta por blocos 64×64 de alta entropia recortados de 6 milhões de imagens 1280×720, seguindo método em [1]	- Controle da taxa é feito pelo número de iterações na reconstrução dos blocos	- O modelo supera o JPEG em PSNR dentro do inverva-lo de 0,25 a 1,5 bpp
Jonhston et. al. [36]	- Convolucional LSTM - Encadeamento implícito	- Codificação com uma arquitetura de 16 iterações que memória. - iterações falsas - Método de alocação de bits pós treinamento	- Norma L_1 sobre os resíduos ponderada pela DSSIM - Base de treinamento é formada por blocos 128×128 amostrados aleatoriamente de 6 milhões de imagens 1280×720 da internet	- Controle da taxa é feito pelo número de iterações na reconstrução dos blocos - A qualidade alvo é usada para adaptar as taxas de bits por blocos	O modelo supera o BPG no MS-SSIM dentro do intervalo 0,125 a 2 bpp (aproximadamente)

3.3 AUTOCODIFICADORES VARIACIONAIS

Diferentemente dos autocodificadores que produzem diretamente z no gargalo, os VAE produzem duas matrizes, representando média e variância, para caracterizar os gaussianos. Essa matriz representa o $P(z)$ (PDF da variável latente z) e é feita uma amostragem. Portanto, o codificador e o decodificador são as funções usadas para induzir a distribuição e mapear os gaussianos fatorados para a distribuição real, reconstruindo a amostra [51]. O autocodificador variacional foi empregado em [41] na tarefa de compressão de imagens ao tentar modelar taxa e distorção, $R + \lambda D$. Os autores desenvolveram uma estrutura para otimização de ponta a ponta de um modelo de compressão de imagem baseado em transformações não-lineares. Eles usaram um módulo de não-linearidade denominado: normalização divisiva generalizada (GDN) que se mostrou eficaz na gaussianização de densidades de imagem. Para qualquer ponto desejado ao longo da curva de taxa-distorção, os parâmetros das transformações de análise e síntese do modelo são otimizados em conjunto. Para conseguir isso na presença de quantização, os autores usam um relaxamento contínuo do modelo de probabilidade, e substituem a etapa de quantização por ruído uniforme aditivo.

Esse trabalho foi aprimorada em [46, 37]. Entre as inovações, destaca-se modelo autoregressivo através de redes neurais [37] e distribuição anterior em um hiperparâmetro [37, 46]. Duas desvantagens da abordagem usando VAE para compressão de imagens, presentes nesses trabalhos, são a necessidade de treinar muitos modelos e exigência muito recurso computacional no treinamento.

3.4 AUTOCODIFICADORES BASEADOS EM GAN

No artigo [38] é proposto uma estrutura de aprendizagem profunda de compressão de imagem aliada a segmentação semântica e um modelo de geração de dados. Nessa abordagem, o codificador gera 3 tipos de informação que irão compor o fluxo de bits. A primeira se refere ao mapa de segmentação semântica da imagem de entrada. A segunda diz respeito a uma representação decimada da imagem de entrada. Ainda no codificador, o mapa de segmentação e a versão compacta da imagem são entradas do gerador de uma GAN que sintetiza uma reconstrução grosseira da imagem. O residual entre a entrada e a reconstrução grosseira é a terceira camada de informação codificada. Arquitetura é composta de 3 redes: *SegNet*, *CompNet* e *FineNet* (gerador). O uso de GAN's é com múltiplas escalas de discriminadores. A função de custo é formada por medidas de perdas em pixels, perdas perceptivas e treinamento antagônico.

Resultados experimentais mostram que a estrutura proposta supera o BPG baseado em H.265/HEVC e outros CODEC's nas métricas PSNR e MS-SSIM em uma ampla faixa de

taxas de bits [38].

No artigo [42] propõe-se uma estrutura unificada de aprendizado de ponta a ponta para aprender representações compressíveis, otimizando conjuntamente os parâmetros do modelo, os níveis de quantização e a entropia do fluxo de símbolos resultante. O objetivo é na compressão de imagens em miniatura. Adota-se um paradigma da compressão generativa, onde treina-se primeiro a transformação de síntese como modelo generativo. Os autores utilizaram a rede DCGAN [52] para esse fim. Essa transformação de síntese é então usada como um decodificador não adaptável como parte da configuração de um autocodificador. Assim, limita-se o espaço de pesquisa das reconstruções a um conjunto compacto menor de imagens naturais que leva em consideração a redundância semântica [42]. Nesse trabalho, os resultados não supera os CODEC's padrão, no entanto, é consideravelmente mais resiliente às taxas de erros de bits (por exemplo, de canais sem fio ruidosos) do que os esquemas de codificação tradicionais de entropia de comprimento variável.

Em [53], Rippel et al. propuseram um modelo de treinamento baseado em GAN's em várias escalas para incentivar as reconstruções nítidas e próximas à imagem original, mesmo com taxas de bits muito baixas. Foi o primeiro trabalho trabalho a trabalhar com GAN's para compactação de imagem.

De forma resumida, a primeira operação do modelo é extrair os recursos da imagem através de redes convolucionais. Os recursos correspondem a um vetor compactado com as características úteis da imagem original. Para isso, a partir da imagem original são obtidos novas imagens em escalas diferentes. Dessas imagens são obtidos os recursos formando uma "decomposição piramidal". Em seguida há um procedimento de alinhamento que explora a estrutura compartilhada de tais recursos [53].

O segundo módulo é responsável por compactar ainda mais os recursos extraídos. Ele quantiza os recursos e os codifica por meio de um esquema de codificação aritmética adaptativa aplicado em suas expansões binárias. Uma regularização adaptativa do comprimento do código é introduzida para penalizar a entropia dos recursos, que o esquema de codificação explora para obter uma melhor compactação. A função custo é composta pela distorção entre o alvo e sua reconstrução e a perda do discriminador devido ao treinamento antagônico com GAN's para buscar reconstruções realistas.

Agustsson et. al. no artigo [54], propuseram uma estrutura baseada em GAN para compactação generativa aprendida, além disso foi apresentado o primeiro estudo completo dessa estrutura para compactação de imagem em resolução total. Para comprimir uma imagem x, seguiram a formulação de autocodificador composto por um codificador E, decodificador/-gerador G e um quantizador finito q. Aqui, o codificador E mapeia a imagem para um mapa de características latentes z, cujos valores são quantizados para L níveis [$C = c_1, ..., c_L \in \mathbb{R}$] para obter uma representação $\hat{z} = q(E(x))$. O decodificador é representado pela rede

geradora de uma GAN condicional [55] que tenta recuperar a imagem formando uma reconstrução $\hat{x}$. Para poder retropropagar o sinal de erro através da quantização é usado um relaxamento diferenciável para q. Nessa formulação, controla-se a entropia, e portanto a taxa, pela escolha do número de níveis de quantização, uma vez que a entropia obedece $H(\hat{z}) \leq dim(\hat{z}) \log_2(L)$ [54].

A função de perdas usada como sinal para atualizar os pesos é composta por um termo de distorção da informação reconstruída e a perda do discriminador D da GAN, $\mathcal{L}_{gan}$. Observa-se que a perda por MSE estabiliza o treinamento, pois penaliza o colapso do GAN [54]. Nesse trabalho, os resultados mostram que, para baixas taxas de bits, essa compactação generativa pode proporcionar uma economia drástica de taxa de bits em comparação com os métodos de ponta anteriores otimizados para objetivos clássicos, como MS-SSIM e MSE, quando avaliados em termos de qualidade visual em um estudo de usuário.

Em [47], os autores adotaram um autocodificador compressivo em que a base do codificador são camadas convolucionais e o decodificador espelha a estrutura do codificador, exceto que usa camadas convolucionais *sub-pixel* [56]. O problema de otimização que a rede tenta alcançar é, a já conhecida, distância d entre as imagens original e reconstruída e o número de bits necessários R para armazenar o código latente z.

$$L = d(x, x') + \lambda R(z) \tag{3.14}$$

Para otimizar R ao invés de usar um estimador de entropia H é empregado uma medida que penaliza o sinal de erro quando o número de elementos diferente de zero no latente for superior a um número *l* desejado. Esse problema foi resolvido aplicando o algoritmo ADMM [57].

4 METODOLOGIA

Neste capítulo explicaremos com mais detalhes a arquitetura do trabalho [1], o método para elaboração da base de dados de treinamento, uma função de custo para promover esparsidade e baixa entropia no código binário. Além disso, abordamos um método simples para alocação de bits com base em um alvo de qualidade

.

4.1 ARQUITETURA

Nessa seção explicaremos o método para treinamento do modelo ao longo das iterações, os formatos dos tensores nas saídas de cada camada da rede, e como se dá a formação do código binário de uma imagem.

Usamos o autocodificador com camada LSTM-convolucional proposto por [1] e apresentado na Figura 3.3. Decidimos que a rede irá receber a informação residual do estágio anterior, dada por $r_t = r_{t-1}$ - r'_{t-1}, e será treinada para reconstruí-la. Dessa forma, a reconstrução de uma imagem x será progressivamente. Essas duas características estão resumidas na próxima Equação, em um estágio:

$$\begin{aligned} r_0 &= x \\ x'_0 &= 0 \\ b_t &= B(E_t(r_{t-1})) \\ r'_{t-1} &= D_t(b_t) \\ r_t &= r_{t-1} - r'_{t-1} \\ x'_t &= r'_{t-1} + x'_{t-1} \end{aligned} \tag{4.1}$$

Esse método difere um pouco do apresentado em [1] e descrito pela Equação 3.11 pois nele os autores usam o valor de $x - x'_{t-1}$ como a informação residual de entrada no estágio t, ou seja, o resíduo em relação à imagem original. Originalmente, esse método foi apresentado em [43] para treinar autocodificadores de camadas não-recorrente. A Figura 4.1 mostra a rede "desenrolada" no tempo. Para simplificar a representação omitimos as camadas PE.

Usamos as camadas Conv2DLSTM descritas pela Equação 2.15 para capturar as dependências espaciais dos resíduos que formam uma sequência. A figura 4.2 detalha essa camada com k mapas de recursos.

Por construção, a arquitetura da rede determina que a saída do codificador fornece um

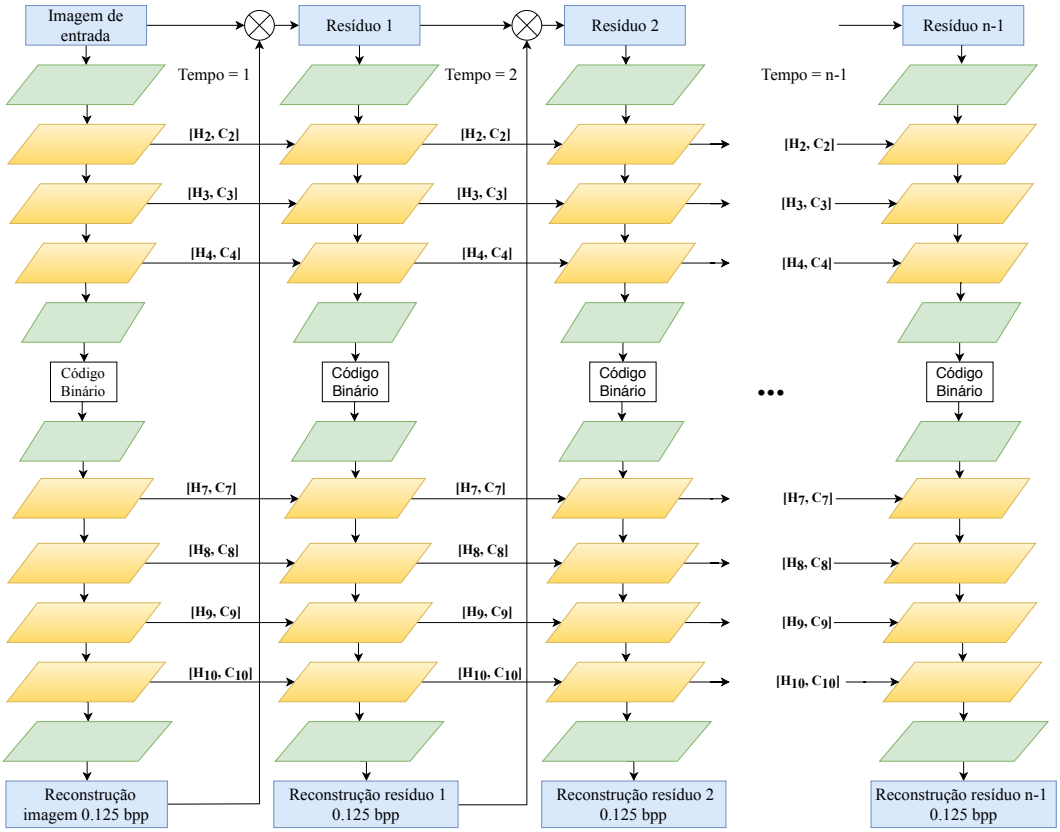


Figura 4.1 – Representação do autocodificador no decorrer de n estágios. Os losangos verdes e laranjas representam as camadas Conv2d e Conv2DLSTM, respectivamente. Na primeira iteração os valores de $C_t e H_t$ para cada Conv2DLSTM são zeros e as setas horizontais evidenciam o $loop$ dessas camadas. Em cada interação o autocodificador recebe o resíduo da reconstrução anterior como entrada e tenta reconstrói-lo ao final da iteração usando 0,125 bits por pixel.

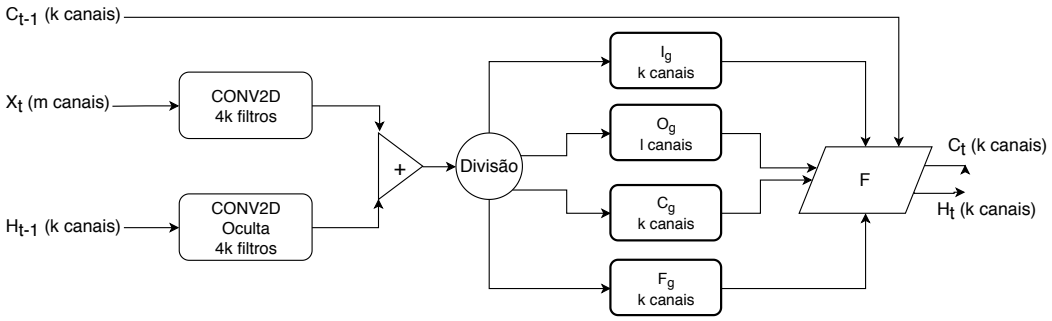


Figura 4.2 – Camada convolucional recorrente com k filtros. Sendo X_t o elemento atual da nossa sequência, com m canais que está sendo processado, H_{t-1} e c_{t-1} representam a saída e estado de célula gerados por essa unidade na etapa anterior. As saídas de ambas as convoluções têm o mesmo número de canais,$4 \times k$, para serem somadas ponto-a-ponto. Em seguida, o novo tensor é separado em 4 novos tensores de k canais. Eles representarão as portas de entrada, esquecimento, célula e saída. A função F irá operar nas portas e no tensor C_{t-1} para fornecer $C_t e H_t$, segundo as equações em 2.15.

conjunto de bits da forma $\frac{H}{16} \times \frac{W}{16} \times 32$ por iteração em que H e W são, respectivamente, a altura e largura da imagem de entrada. Isso nos fornece $\frac{H \times W}{8}$ bits e podemos calcular a taxa nominal por iteração R_{it} em bits por pixel:

$$R_{it} = \frac{\frac{H \times W}{8}}{H \times W} = \frac{1}{8} \text{ bpp} \tag{4.2}$$

Então, obtemos a taxa nominal R na iteração $k > 0$ como $R = \frac{k}{8}$. Dessa forma podemos controlar a quantidade de bits por pixel que enviamos ao decodificador com o passo de $1/8$. Aplicando a Equação 2.26 obtemos a taxa compressão nominal C na iteração k:

$$C = \frac{24 \times H \times W}{k \times \frac{H \times W}{8}} = \frac{192}{k} \tag{4.3}$$

Os detalhes do autocodificador podem ser consultados na Tabela 4.1 para uma entrada com formato $32 \times 32 \times 3$. O tamanho do lote está omitido dessa representação, contudo, ele ocupa mais uma dimensão, e possui valor fixo (pode ser diferente no último lote de uma imagem) durante todas as operações realizadas pela rede.

4.1.1 Binarização

O processo de binarização adotado foi o descrito na seção 3.2.1 e proposto em [43]. Na prática, para tornar esse processo estocástico, definimos um valor u entre 0 e 1 obtido de uma distribuição uniforme, isto é, $u \in \mathcal{U}[0, 1]$ e podemos reescrever a Equação 3.3 para obter o

Tabela 4.1 – Parâmetros das operações no autocodificador.

Camada	Formato de saída	Núcleo Convolucioal Normal/Oculto	Passo	Preenchimento	Função de Ativação	Número de Parâmetros
Conv2D	16x16x64	3x3/-	2	“Mesmo”	Identididade	1728
Conv2DLSTM	8x8x256	3x3/1x1	2	“Mesmo”	Sigmoid	851968
Conv2DLSTM	4x4x512	3x3/1x1	2	“Mesmo”	Sigmoid	5767168
Conv2DLSTM	2x2x512	3x3/1x1	2	“Mesmo”	Sigmoid	10485760
Conv2D	2x2x32	1x1/-	1	“Válida”	Tanh	16384
Binarizador	2x2x32	-				
Conv2D	2x2x512	1x1/-	1	“Mesmo”	Identididade	16384
Conv2DLSTM	2x2x512	3x3/1x1	1	“Mesmo”	Sigmoid	10485760
P.E	4x4x128	-				
Conv2DLSTM	4x4x512	3x3/1x1	1	“Mesmo”	Sigmoid	3407872
P.E	8x8x128	-				
Conv2DLSTM	8x8x256	3x3/1x1	2	“Mesmo”	Sigmoid	1441792
P.E	16x16x64	-				
Conv2DLSTM	16x16x128	3x3/1x1	2	“Mesmo”	Sigmoid	360448
P.E	32x32x32	-				
Conv2D	32x32x3	1x1/-	1	“Mesmo”	Tanh	96
Total de parâmetros						32835360

erro ϵ da quantização:

$$\epsilon = \begin{cases} 1 - x, \text{ se } u \leq \frac{1+x}{2} \\ -x - 1, \text{ se } u > \frac{1+x}{2} \end{cases} \tag{4.4}$$

A binarização será feita com valores -1 e 1, e usamos o truque apresentado em 3.2.1 para conseguir treinar o modelo com essa camada não diferenciável.

4.1.2 Formato do fluxo de bits

A arquitetura será otimizada para reconstruir blocos de 32×32 pixels, uma vez que os modelos serão treinados com imagens dessa dimensão. Portanto, durante os testes, devemos dividir as imagens em blocos 32×32 pixels.

Considerando que o número de blocos gerados é igual a n e desejamos agrupa-los em lotes de tamanho l e reconstruídos passando k vezes pela rede, então teremos:

1. Na primeira iteração, o modelo irá tomar os primeiros l blocos da imagem na ordem direita para esquerda e de cima para baixo.

2. Eles são codificados pela rede do codificador para gerar o código binário parcial de formato $l \times 2 \times 2 \times 32$ conforme a tabela 4.1;

3. Em seguida, decodificamos esse lote e computamos o resíduo;

4. Esse resíduo é codificado para gerar um novo código binário $l \times 2 \times 2 \times 32$ que é concatenado com o anterior para formar $2 \times l \times 2 \times 2 \times 32$;

5. Esse processo se repete até concluirmos as k iterações na rede, e ao final da codificação de um lote o nosso código binário é da forma $k \times l \times 2 \times 2 \times 32$.

O número de vezes que devemos realizar o processo descrito acima para codificar a imagem inteira é igual a $\frac{n}{l}$. Portanto, o código binário final é da forma $\frac{n}{l} \times k \times l \times 2 \times 2 \times 32$. Obviamente, o número de bits desse código independe do tamanho de lote escolhido.

4.2 BASE DE DADOS

Nesta seção explicamos o processo de construção das bases de dados de treinamento seguindo um método utilizado em [58].

Idealmente, desejamos obter uma base de dados de treinamento a fim de tornar o modelo capaz de comprimir satisfatoriamente (desempenho semelhante ao JPEG200) imagens naturais com diferentes características de tamanho, componentes de frequência, entropia etc. Essa é uma das principais escolhas para treinar qualquer algoritmo de redes neurais. Primeiro, selecionamos imagens de bases de dados distintas que não passaram por nenhum processo de compressão com perdas, assim, preservarmos as componentes de frequência das imagens naturais. Essas imagens foram obtidas dos seguintes conjuntos de dados:

1. Conjunto de dados CLIC [59] (*Challenge on Learned Image Compression*) - Imagens de alta qualidade.

   Professional - validação: 41 imagens

   Professional - treinamento: 585 imagens;

   Mobile - validação: 61 imagens;

   Mobile - treinamento: 1048 imagens.

2. Conjunto de dados DIV2K [60] (*DIVerse 2K resolution high. quality image*) - Imagens de alta resolução

   Treinamento: 800 imagens;

   Validação: 100 imagens.

3. Conjunto de dados Ultra-Eye [61] (*Ultra-Eye: UHD and HD images eye tracking datase*) - Alta qualidade e alta resolução.

   HD: 38 imagens;

   UHD: 40 imagens.

Esse procedimento gerou 6.231.440 blocos. Apresentamos na Figura 4.3 um histograma do tamanho desses blocos. Essa medida é usada aqui como uma estimativa da entropia do bloco. O bloco que apresentou maior entropia foi definido como uma referência de 100% de entropia. A partir dessa referência, e tendo em mente que a entropia das imagens de treinamento poderia impactar na performance do modelo, separamos os blocos em 5 base de dados observando os seguintes critérios:

1. Banco de dados 0 (BD0). Todos os blocos cuja entropia seja menor que 20% do valor da entropia de referência. Total: 1.248.978 blocos.

2. Banco de dados 1 (DB1). Todos os blocos no intervalo de 40% e 60% de entropia de referência. Total: 1.251.421 blocos.

3. Banco de Dados 2 (DB2). Todos os blocos correspondentes a pelo menos 80% da entropia de referência. Total: 1.248.725 blocos.

4. Banco de Dados 3 (DB3). Sorteio de 20% dos blocos que compõem toda a base de dados. Total: 1.247.033 blocos

5. Banco de Dados 4 (DB4). Sorteio de 20% dos blocos cuja entropia varia de 50% a 100%. Total: 1.246.698 blocos

6. Banco de Dados 5 (DB5). Todos os blocos cuja entropia varia de 50% a 100% em relação a nossa referência. Total: 2.287.520 blocos

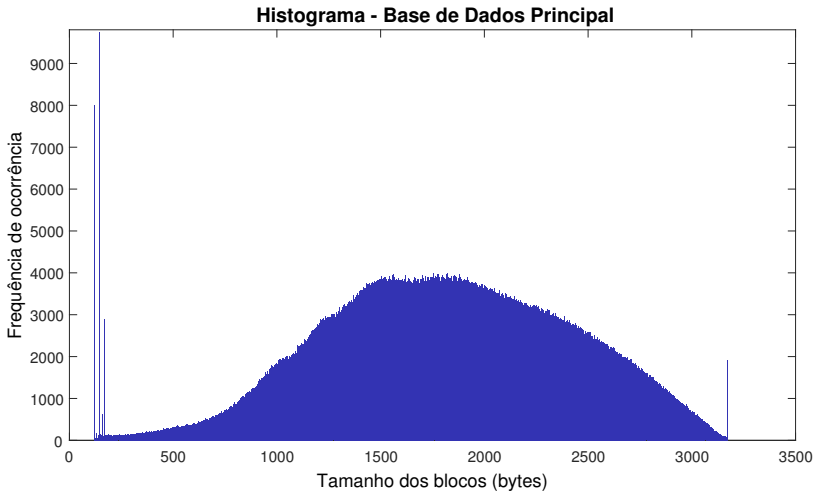


Figura 4.3 – Histograma de todo o banco de dados

Por construção, não há sobreposição entre os bancos de dados 0, 1 e 2. No entanto, existem sobreposições de informações entre os bancos de dados 3 e 4. Os histogramas dessas bases, exceto a 5, estão ilustrados nas Figuras 4.4, 4.5, 4.6, 4.7 e 4.8.

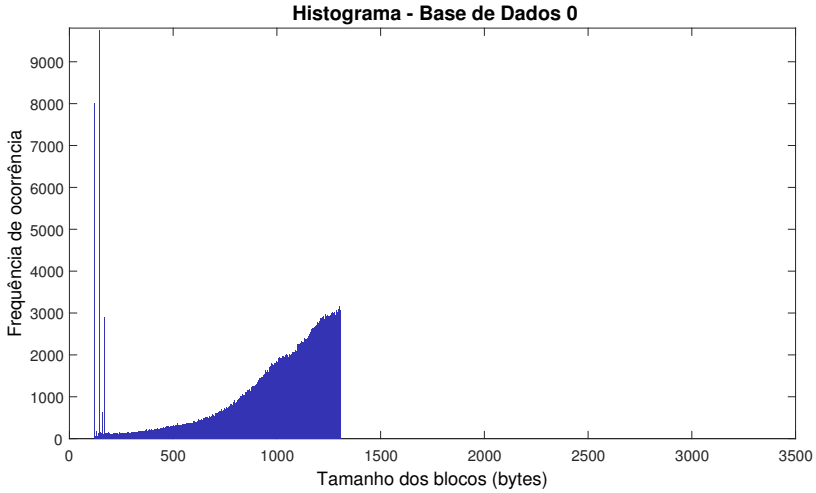


Figura 4.4 – Histograma da baixa entropia DB0.

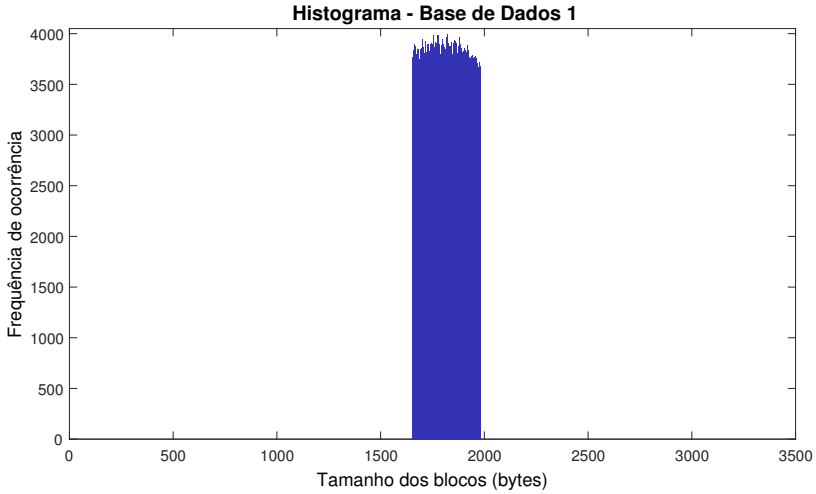


Figura 4.5 – Histograma do DB1 de entropia média.

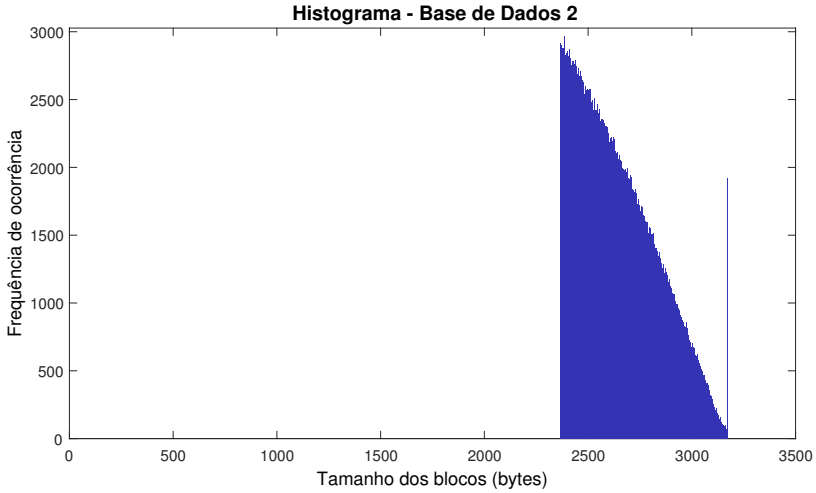


Figura 4.6 – Histograma do DB2 de alta entropia.

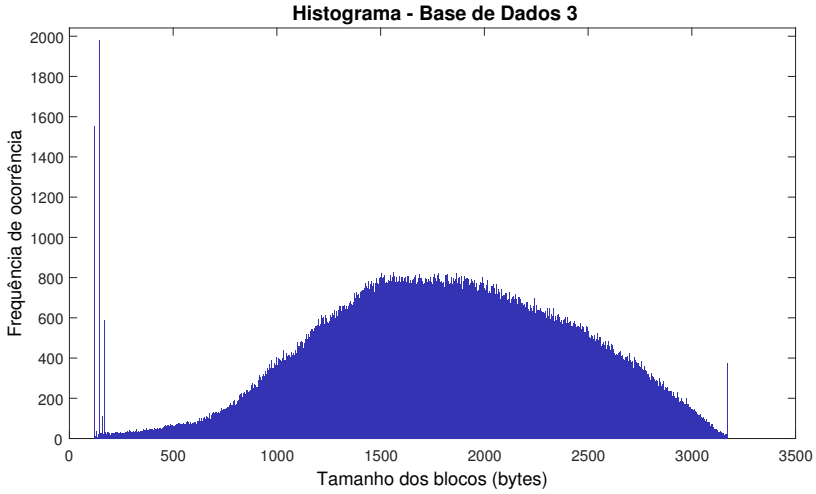


Figura 4.7 – Histograma da entropia mista DB3.

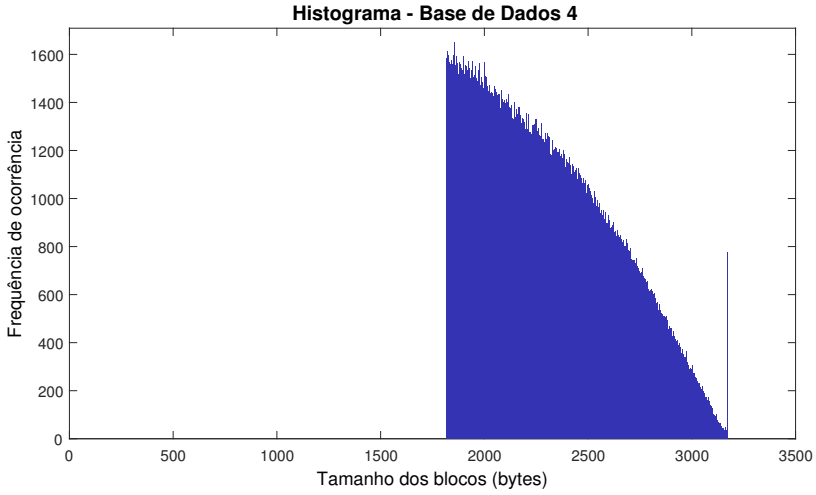


Figura 4.8 – Histograma da entropia alta DB4.

Os testes consistiram em treinar a rede usando cada uma das bases e com os mesmos hiperparâmetros. Os modelos gerados são comparados entre si usando as métricas de qualidade.Veremos que a DB4 obteve o melhor resultado, e justificou a elaboração de uma nova base de dados obedecendo a regra:

1. Banco de Dados 5 (DB5). Todos os blocos cuja entropia varia de 50% a 100% em relação a nossa referência. Total: 2.287.520 blocos

A base de dados da Kodak [62] formada por 24 imagens naturais foi empregada em todos os testes realizados nesse trabalho.

4.3 FUNÇÃO DE CUSTO

Nessa seção, discutimos o procedimento para a escolha da função de custo (ou de perdas) da arquitetura. Em princípio, consideramos apenas a distorção, em segundo momento adicionamos uma regularização na tentativa de controla a entropia do código latente gerado.

4.3.1 Otimização por distorção

A função de custo é uma escolha importante nos treinamentos de RNA. No problema de compressão de imagens a escolha dessa função tem o agravante de não haver consenso entre os pesquisadores de uma métrica ideal para avaliar a qualidade entre uma imagem original e a sua versão reconstruída como relatado em [36, 41]. Em nossa análise, usamos as funções conhecidas para medir distorção entre uma imagem e sua reconstrução, tais como MSE, MAE, e uma versão diferenciável de 1-SSIM. Em dois testes, modificamos o espaço de cores dos blocos de RGB para YCbCr. Outra modificação foi alterar a formulação em 4.1 para 3.11. Para comparar a performance de cada função, realizamos treinamentos curtos usando a base de dados 4.

A função de custo L é calculada em cada estágio e, ao final das iterações, computamos a sua média para obter L_{mean}.

$$L_{mean} = \frac{1}{k} \sum_{t=0}^{k} L(r_t, r'_t) \tag{4.5}$$

É sobre esse sinal que o algoritmo de retropropagação irá calcular os gradientes.

4.3.2 Otimização por distorção e taxa

Até aqui, não estamos tendo controle sobre a entropia do código binário gerado. Se pudermos otimizar a rede para gerar uma sequência de bits z com baixa entropia teremos ganhos relevantes ao aplicador um codificador de entropia. Em um segundo momento, adicionamos uma função de regularização $R(z)$ para otimizar implicitamente a taxa do código binário, inspirado no trabalho [47]. Para calculá-lo em uma iteração, copiamos o código binário fornecido pela função de binarização para a variável Z. Nessa variável, substituímos os bits -1 por 0 e aplicamos a norma L_1 sobre Z. Ademais, empregamos a função MSE para fornecer uma medida de distorção. Assim, temos:

$$L = MSE(r, r') + \lambda(t) \times \sum_{t=1}^{n} Z \tag{4.6}$$

onde n é o número de bits do código binário por nível. Em nossa rede esse número é igual a 128 para cada bloco 32×32. A ideia é que ao penalizarmos a ocorrência do bit 1 estamos, essencialmente, reduzindo a entropia de ordem 0 do código binário. Esperamos que isso reflita na redução da taxa ao aplicarmos um codificador de entropia.

O fator e $\lambda(t) > 0$ é uma função de iteração da t responsável por controlar a proporção de R sobre a função de custo. Em nossos testes, definimos $\lambda(t)$ como uma função decrescente. Para um lote de blocos de treinamento, à medida que as iterações prosseguem o resíduo reconstruído se torna cada vez menor e, consequentemente, a distorção medida pela MSE decaí. Todavia, tal comportamento não ocorre, necessariamente, em R. Então controlamos o peso de R sobre L através de um valor de regularização menor. Dessa forma, tentamos balancear o impacto da distorção e taxa. Podemos, encarar essa formulação com um autocodificador esparso, discutido em 2.1.4.1, onde bit -1 significa neurônio "desativado" e saída 1 "ativada".

Nessa abordagem, aplicamos uma codificação sem perdas usando o GZIP para aproveitar a estatística de distribuição dos bits (que esperamos ser de baixa entropia). Portanto, na reconstrução de uma imagem existe a taxa nominal r_n (fixa por iteração). Nessa configuração podemos falar em um método Taxa de Bits Constante (*Constant Bitrate*) (CBR). Ao aplica a codificação de entropia geramos uma nova taxa denominada, nesse trabalho, por taxa real. O ganho percentual de taxa em virtude da codificação de entropia e em relação à taxa nominal é descrito pela Equação a seguir:

$$G = 100 \times \frac{r_n - r_r}{r_n} \tag{4.7}$$

4.4 ALOCAÇÃO DINÂMICA DE BITS

Aproveitamos a arquitetura do CODEC que é baseado na reconstrução por blocos de forma progressiva para apresentar uma proposta de alocação dinâmica de bits semelhante a [36]. Nesse método, consideramos as regiões nas quais o modelo consegue reconstruir bem para economizar bits, e nas regiões mais difíceis usar mais bits.

Uma desvantagem do método CBR é usar a mesma taxa independentemente da complexidade dos blocos. Tal problema foi levantado em [36]. Para tentar contornar essa desvantagem, propomos uma heurística simples. O método de alocação dinâmica de bits é uma etapa realizada após o treinamento da rede. Nele, ao final de cada iteração e para cada lote de blocos 32×32 que são passados pela rede do codificador e decodificador calculamos a sua média de PSNR e desse conjunto também obtemos a menor PSNR. Se essa média for maior ou igual a um alvo de qualidade (P_a) e a menor PSNR for maior ou igual a $P_{ct}\%$ do valor desse alvo então interrompemos a reconstrução nesse ponto e passamos a codificar o próximo lote. Se chegarmos no número máximo iterações (k_{max}) codificaremos o próximo lote. Como veremos, essa prática gera imagens com artefatos de blocos que comprometam a qualidade perceptual ainda que a PSNR indique melhora. Por isso, adicionamos mais uma restrição: independentemente de P_a haverá um número mínimo e máximo de iterações que os blocos estão sujeitos. O fluxograma na Figura 4.9 resume o método de alocação dinâmica de bits.

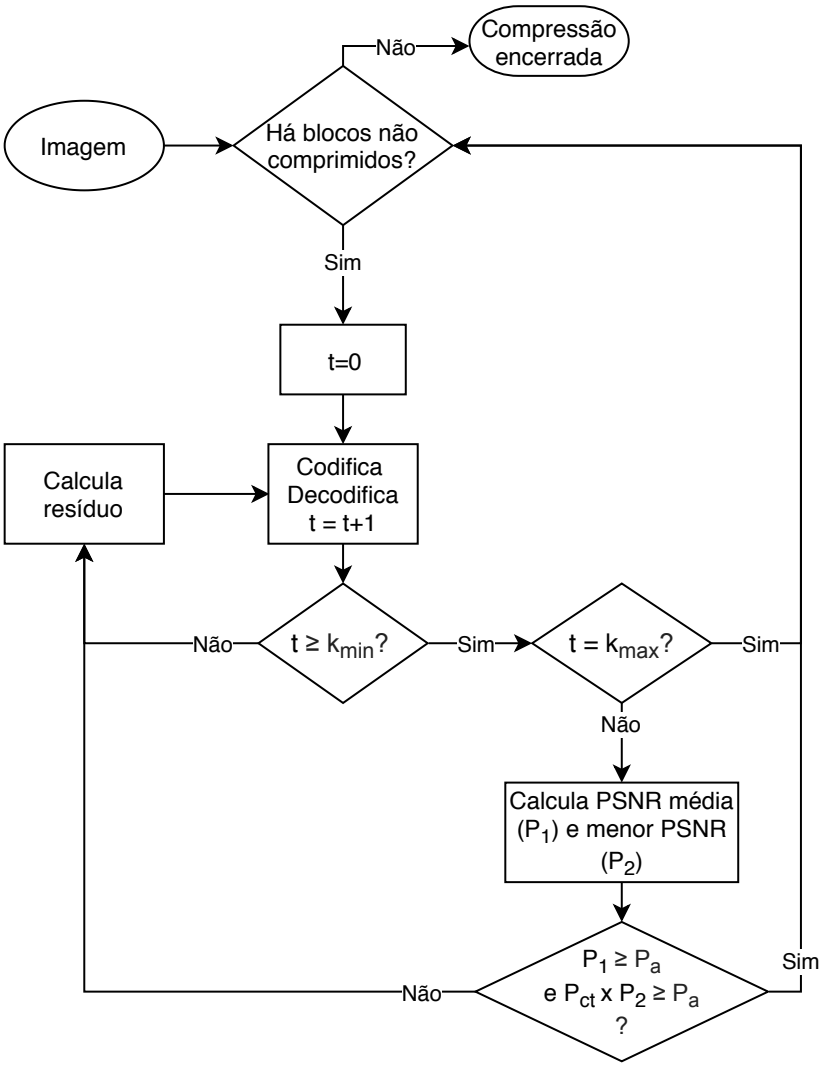


Figura 4.9 – Alocação dinâmica de bits.

4.5 IMPLEMENTAÇÃO

O autocodificador foi implementado usando a biblioteca PyTorch da empresa Facebook e baseado na implementação disponível em [63]. Essa biblioteca tem a vantagem de ter suporte a redes neurais construídas de forma dinâmica. Isso a torna a programação e depuração mais fácil e intuitiva em relação à biblioteca TensorFlow da Google. Nas demais tarefas como cálculo de métricas de qualidade e obtenção de curvas utilizamos, principalmente, a linguagem Python, e Matlab na minoria das vezes. O treinamento foi realizado em computadores com GPU GTX 1080Ti ou GPU GTX 2080Ti. Em todos os testes, normalizamos os dados de entrada para o intervalo $[-0.5, 0.5]$ para minimizar a saturação devida as ativações sigmódes presentes no modelo. O otimizador ADAM foi usado em todos os testes.

5 RESULTADOS

Nessa seção apresentamos os principais resultados na elaboração desse trabalho. Eles são referentes a análise da base de dados, função de custo, autocodificador esparso e alocação dinâmica de bits.

5.1 ANÁLISE BASE DE DADOS

Treinamos 5 modelos do autocodificador para cada base de dados apresentada em 4.2 e usando a função MAE sobre os resíduos como função de custo. Além disso, os seguintes hiper-parâmetros da rede foram adotados nesse experimento:

1. Tamanho do lote: 32;
2. Número de lotes de treinamento por época: 38 mil (aproximadamente);
3. Número de iterações: 16;
4. Número de épocas: 1;
5. Taxa de aprendizado: 5×10^{-4}

Calculamos os valores médios de PSNR, SSIM e MS-SSIM das imagens da Kodak reconstruídas na taxa nominal de 2 bits por pixel. Os resultados estão disponíveis na tabela 5.1.

Tabela 5.1 – Comparação das base de dados.

Base de dados	PSNR(dB)	SSIM	MS-SSIM
0	25,4268	0,6978	0,9177
1	31,9458	0,9201	0,9874
2	34,3622	0,9444	0,9900
3	32,9632	0,9327	0,9890
4	**34,3903**	**0,9473**	**0,9913**

Conforme esperado, uma base de dados de alta entropia é indicada para melhorar a performance do modelo. A hipótese é que essa característica otimiza a rede a aprender padrões não triviais das imagens e evitar o sobre-ajuste do modelo. A base de dados 4, nessa tabela, obteve o melhor resultado, por uma pequena margem em relação à BD2. Para temos uma comparação mais ampla, calculamos as áreas abaixo das curvas de PSNR $\times$ taxa, SSIM $\times$

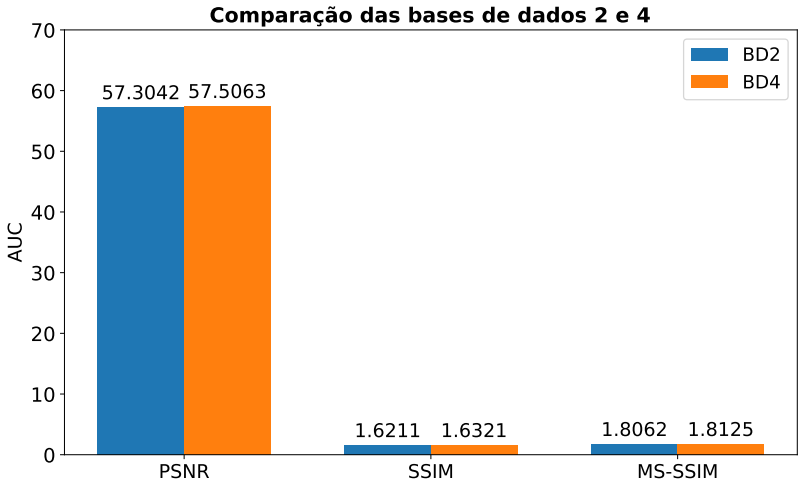


Figura 5.1 – Essa Figura mostra as áreas abaixo da curvas de PSNR, SSIM e MS-SSIM cada uma em função das taxas nominais do modelo com 16 iterações.

taxa e MS-SSIM $\times$ taxa dos modelos treinados por DB2 e DB4. Todas as curvas têm 16 pontos, com taxas nominais no intervalo de 0,125 a 2,0 bpp e incrementos constantes de 0,125 bpp. A Figura 5.1 é mais um indicativo que o modelo treinado pela BD4 apresenta a melhor performance.

Por esse motivo, calculamos a área abaixo da curva

A BD4 obteve os melhores resultados nas 3 métricas nesse conjunto de teste e será aplicada nas próximas análises.

O próximo passo foi analisar a performance no modelo à medida que aumentarmos o número de épocas de treinamento. Para isso, treinamos o modelo por 13 épocas usando somente a base de dados 4. Os parâmetros do novo treinamento foram definidos conforme itens citados em 5.1, exceto o número de épocas. A evolução do modelo ao passar das épocas pode ser vista na Figura 5.2.

Usando o modelo obtido na 13ª época, plotamos as curvas de distorção nas métricas SSIM e MS-SSIM em conjunto com o JPEG(4:2:0) e o trabalho de Toderici et. al. [1]. Essas Figuras estão disponíveis em 5.3 e 5.4. Observamos que nessas métricas e na média das imagens da Kodak, o nosso modelo supera o trabalho [1] e o JPEG. Acreditamos que esse resultado se deve a escolha criteriosa da base de dados de treinamento que não passaram por processos de compressão com perdas e decimação.

Todavia, a Figura 5.5 indica que na métrica PSNR o JPEG(4:2:0) ainda supera o nosso modelo para a maioria das taxas calculadas.

Pela figura 5.2 verificamos uma tendência de saturação do modelo, em torno de 36 dB na taxa de 2 bpp, ao passar das épocas de treinamento. Dessa forma, exploraremos novas técnicas que podem gerar menores taxas de compressão sem impactar na qualidade das imagens

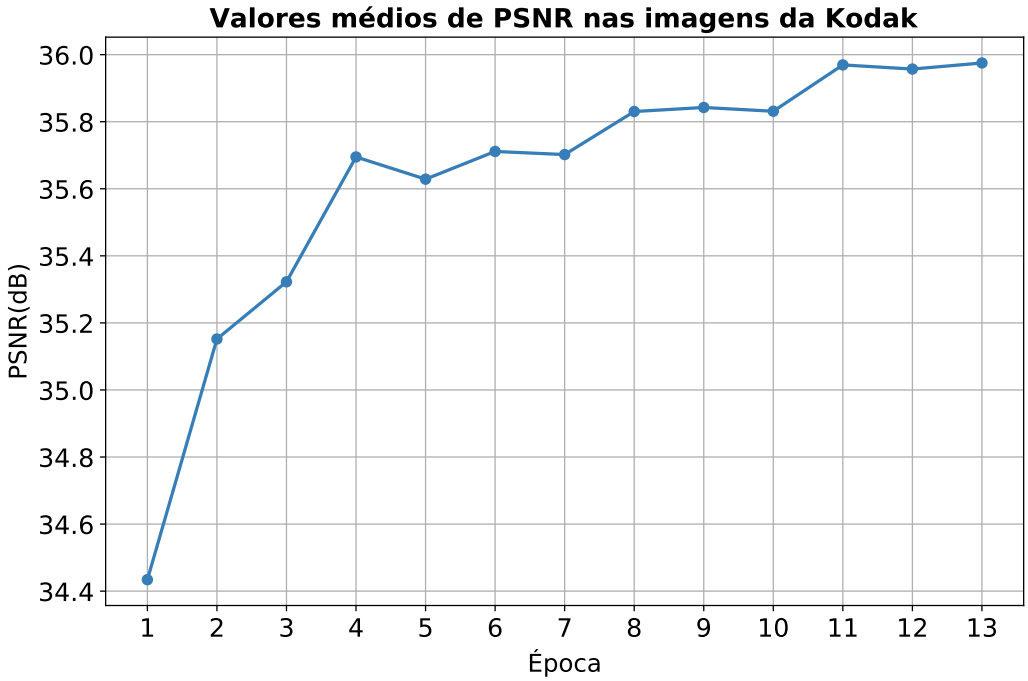


Figura 5.2 – Performance do modelo em função do número de épocas de treinamento. Para cada época, calculamos a PSNR média das imagens da Kodak reconstruídas com taxa de 2 bpp. O ganho de desempenho do autocodificador entre a primeira e 13ª época foi de um pouco mais que 1,5dB.

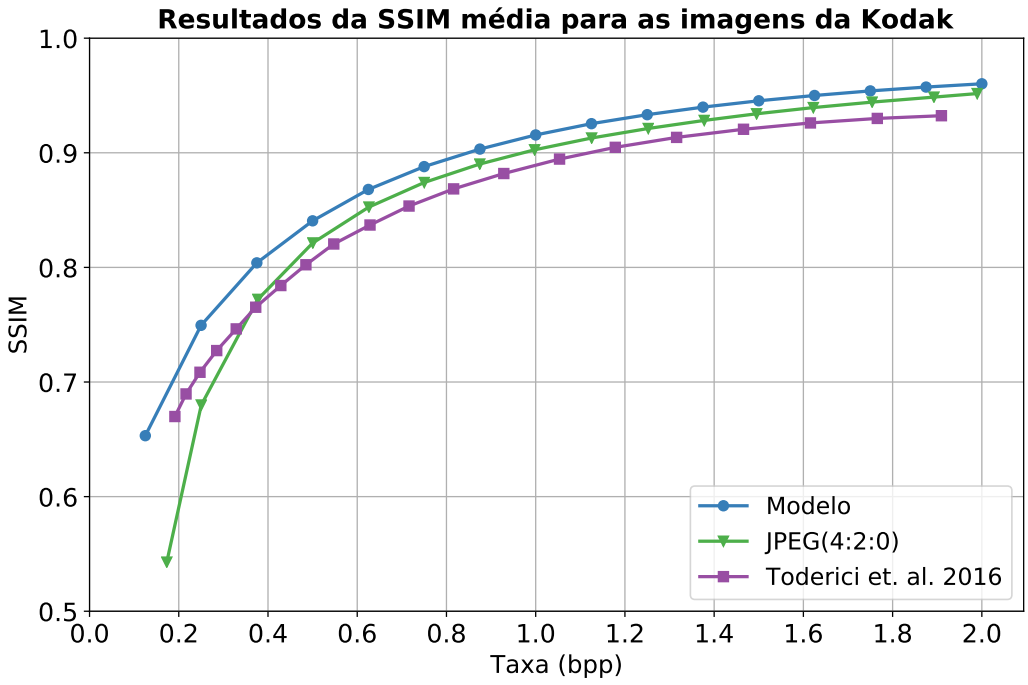


Figura 5.3 – Comparação.

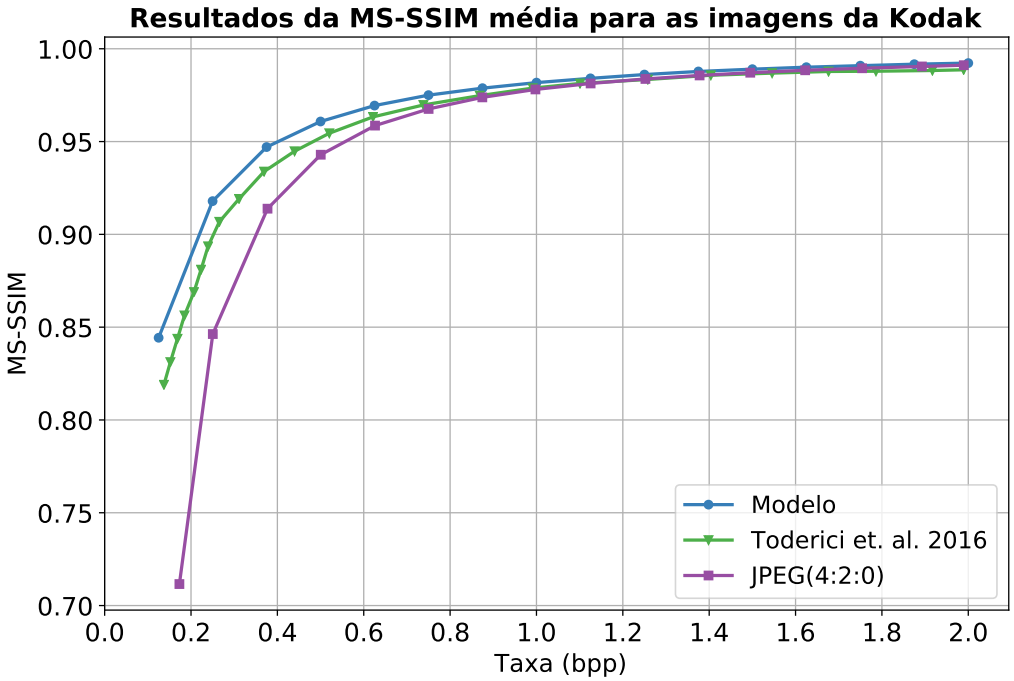


Figura 5.4 – Comparação.

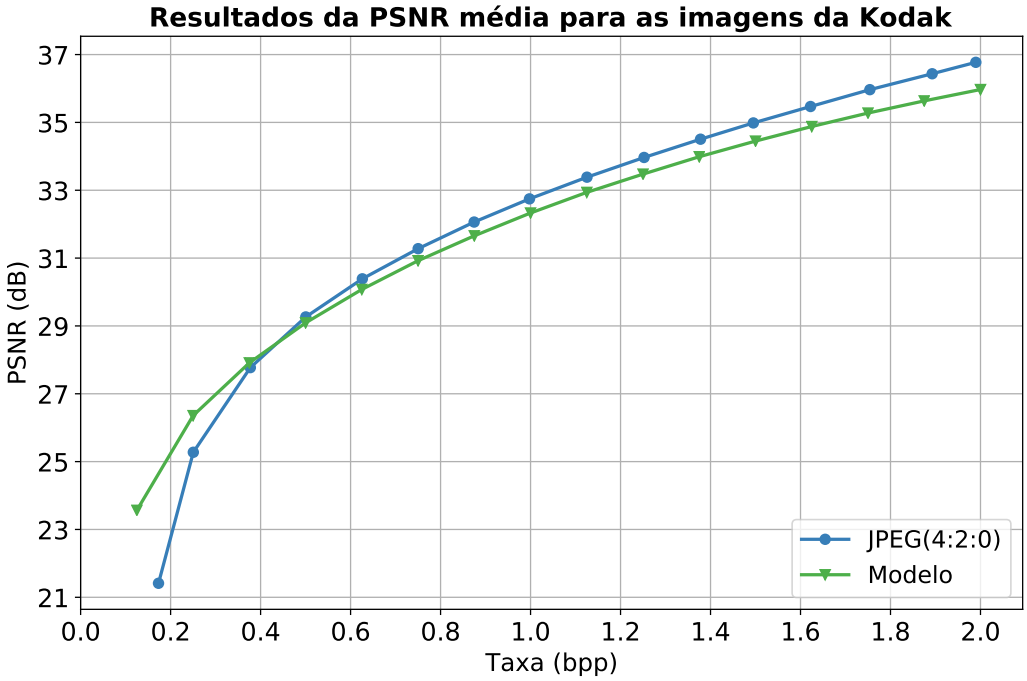


Figura 5.5 – Comparação.

Tabela 5.2 – Comparação das funções de custo.

Teste	Função de custo	Espaco	PNSR (dB)	PNSR Y (dB)	SSIM	MS-SIM
1	$MAE(R - R')$	RGB	33,0872	33,8443	0,9309	0,9876
2	$MAE^2(R - R')$	RGB	34,1766	34,9706	0,9444	0,9892
3	$MAE^2(R_y - R'_y)+$ $+0.25 \times MAE^2(R_{CbCr} - R'_{CbCr})$	YCbCr	33,3555	35,3253	0,9445	0,9857
4	$MSE(R_y - R'_y)+$ $+0.25 \times MSE(R_{CbCr} - R'_{CbCr})$	YCbCr	33,5249	35,1863	0,9390	0,9858
5	$MSE(R - R')$	RGB	**34,9259**	**35,9066**	**0,9474**	**0,9907**
6	$MSE(X - X')$	RGB	33,7662	34,6014	0,9321	0,9866
7	$1 - SSIM(X - X')$	RGB	33,2754	34,0755	0,9352	0,9896

reconstruídas pelo decodificador.

A Tabela 5.2 apresenta resultados para 7 funções de custo que mensuram a distorção entre duas imagens. Os valores foram calculados pela média das imagens da Kodak codificadas a 2 bpp. A rede foi treinada com a base de dados 4 e parâmetros listados em 5.1. Nessa tabela, R, R', X e X' foram definidos na Equação 4.1, omitindo-se a dependência temporal. A medida $X - X'$ é o resíduo em relação à imagem original X. O subíndice *y* e C_bC_r são para indicar a componente de luminância e o par crominância azul e vermelho, respectivamente. Nos testes 3 e 4, modificamos o treinamento para rede operar no espaço YC_bC_r. O objetivo foi penalizar, predominante, o erro da componente de luminância (que contém informações mais sensíveis ao olho humano) no ajuste da rede, em comparação com as componentes de crominância. No teste 7, utilizamos uma medida de dissimilaridade estrutural entre duas imagens, obtida a partir da SSIM e implementada em [64]. A distorção calculada pela MSE forneceu os melhores resultados em todas as métricas avaliadas.

A segunda etapa na escolha de uma função de perdas foi considerar, de forma implícita, a taxa ao aplicar a função 4.6 com um $\lambda(t)$ obtida de forma experimental e dado por:

$$\lambda(t) = 3,5 \times 10^{-7} \times e^{-0,01 \times (t+1)} \tag{5.1}$$

Além dessa modificação, usamos a base de dados 5 obtida conforme a regra da base de dados 4 sem a aleatoriedade no sorteio dos blocos de alta entropia e alterações o número de estágios de treinamento de 16 para 28. O aumento no número de estágios de treinamento visa permitir melhores reconstruções das imagens pelo codificador. Desejamos que o aumento da taxa nominal seja compensada pela codificação com o GZIP. Resumindo, no novo treinamento da rede teremos:

1. Tamanho do lote: 32;
2. Número de lotes de treinamento por época: 71 mil (aproximadamente);
3. Número de iterações no treinamento: 28;

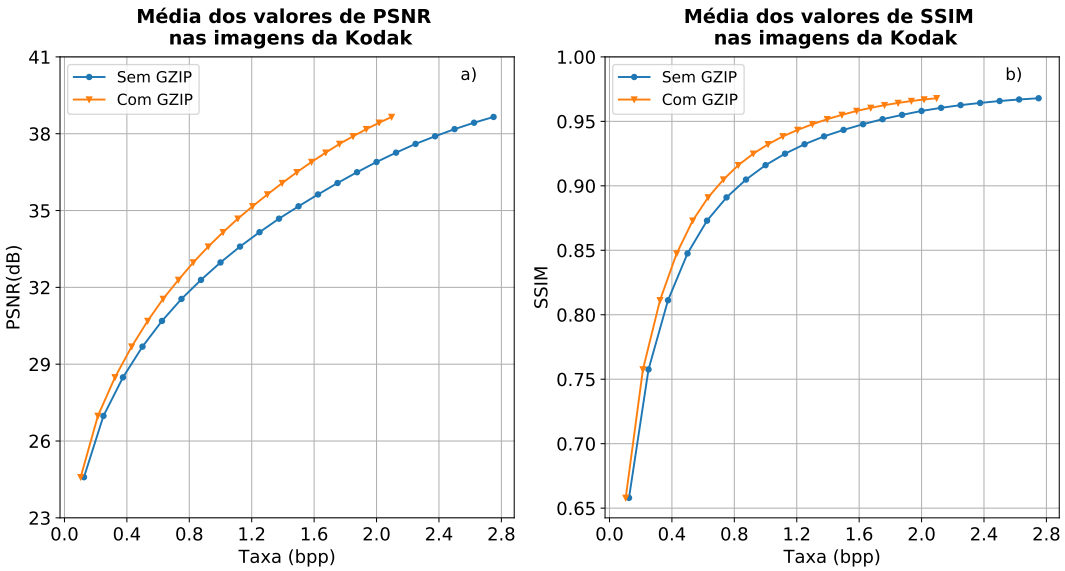


Figura 5.6 – Nessa figura, ilustramos o resultado em aplicar o GZIP na sequência binária gerada no codificador. As curvas foram plotadas usando o modelo treinado até a época 26.

4. Número de iterações durante os testes: 22;
5. Número de épocas: 27;
6. Taxa de aprendizado: 5×10^{-4} com decaimento por um fator de 0,5 caso a média da função de perdas em uma época seja maior que na época anterior.

A Figura 5.6 compara o ganho proporcionado pela codificação de entropia do GZIP ao plotarmos as curvas de PNSR (a) e SSIM (b) em função da taxa usando as imagens da Kodak. Ao usarmos a taxa real as curvas se deslocam para esquerda. Observamos, que o ganho do GZIP é maior em taxas maiores. Isso é esperado tendo em vista que, em geral, os codificadores de entropia são mais eficientes em conjuntos maiores de dados. Nos próximos testes, as taxas empregadas serão sempre a real.

A Figura 5.7 mostra como a performance em termos de taxa-BD (em relação ao jPEG) se comporta durante o treinado do modelo. O treinamento em várias épocas melhora, significativamente, o desempenho do modelo. É verdade também que a taxa com que o modelo apresenta melhores resultados se torna mais lenta ao passar das épocas. A economia da taxa na última época em relação à primeira é aproximadamente de 17%. Nesse teste, a 26ª época gerou o modelo de melhor desempenho. Vamos nos referir a esse CODEC como modelo26.

Nas figuras em 5.8 podemos observar a otimização taxa-distorção em 3 épocas. Nesse teste, a qualidade melhora e a taxa real é reduzida seguindo a época 1, 6 e 26. A qualidade aqui é modelada pelas curvas de PSNR (a), PSNR Y (b) SSIM (c) e MS-SSIM (d). Esse resultado indica que a nossa função consegue alcançar as hipóteses apresentadas, contudo não temos uma fundamentação na direção do melhor método para otimizar taxa-distorção.

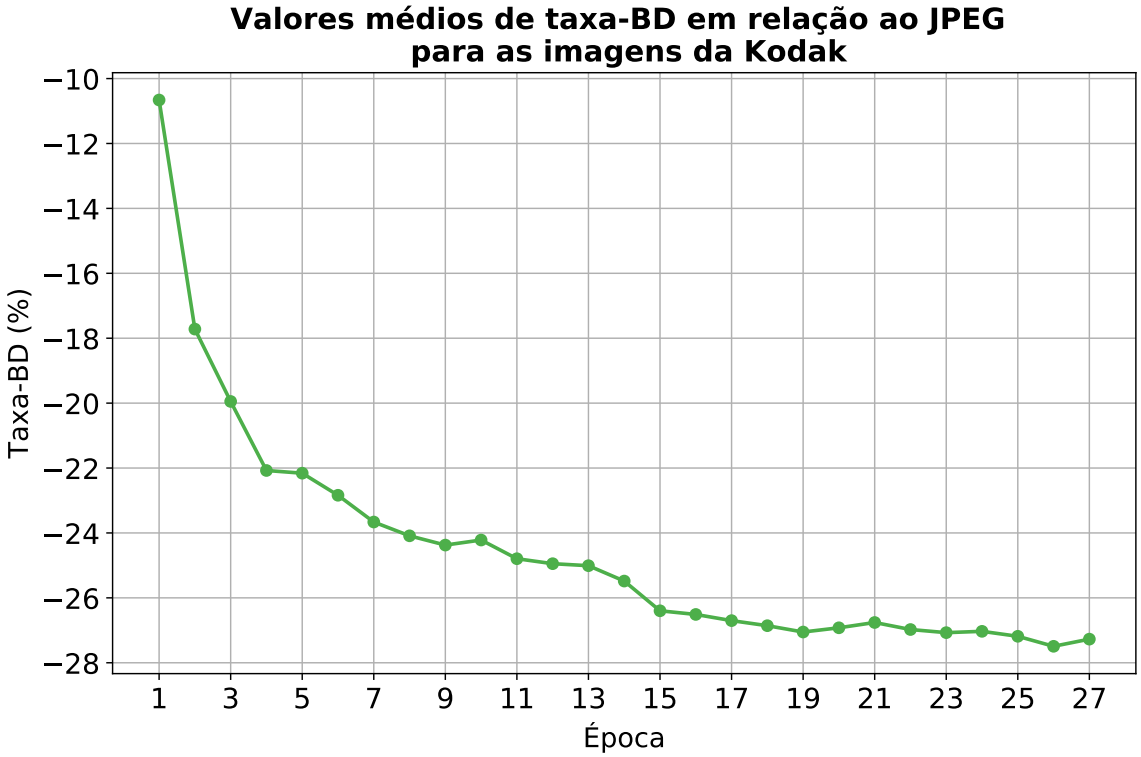


Figura 5.7 – Ilustração da porcentagem da taxa-BD média sobre a base de dados da Kodak e tendo o JPEG como âncora. Valores cada vez mais negativos indicam que a economia média de taxa de bits do nosso modelo em relação ao JPEG aumenta para uma dada qualidade equivalente de PSNR na média das imagens da Kodak.

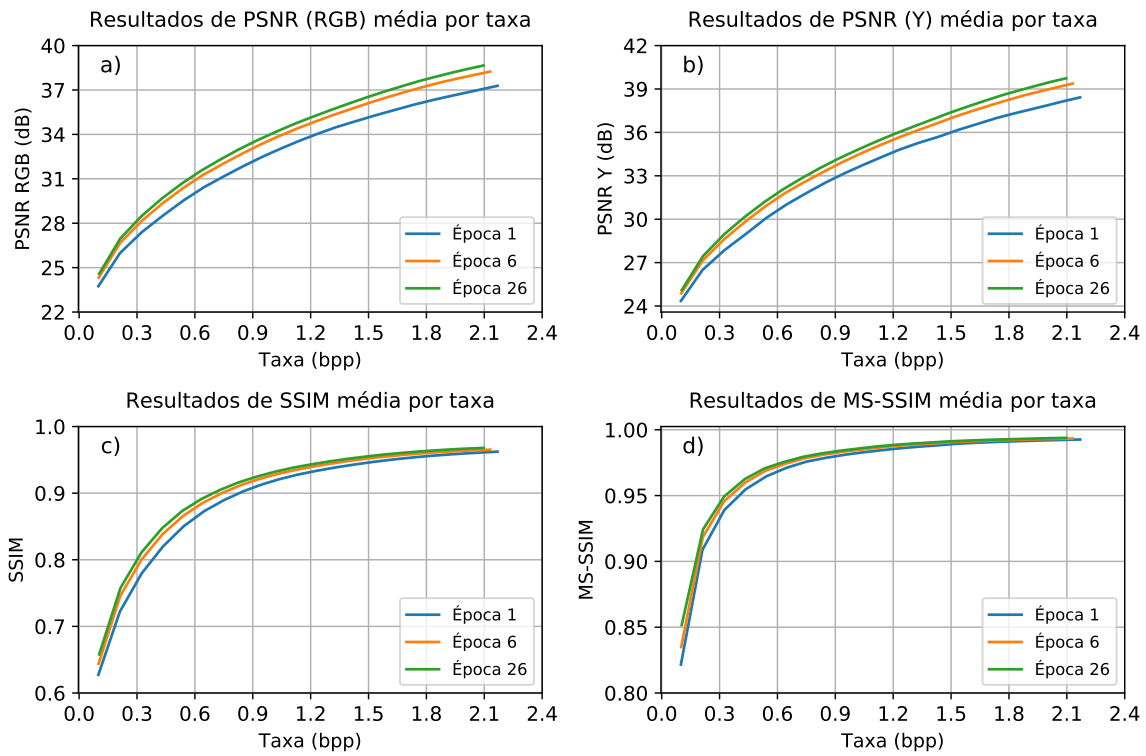


Figura 5.8 – Figuras mostrando a evolução do modelo em termos de taxa e 4 métricas de qualidade para 3 épocas distintas. Nas 4 curvas taxa e distorção apresentam melhores resultado à medida que o número de épocas aumenta.

A Figura 5.9 apresenta o ganho médio percentual da taxa real (fornecida pelo GZIP) em relação a nominal como função da iteração em que as imagens foram reconstruídas. Os resultados mostram que, aproximadamente, nos primeiros 3 níveis de reconstrução o modelo final tem ganho médio menor em relação à primeira e sexta versão do modelo.

Além disso, do primeiro para o terceiro nível o ganho é reduzido ainda que temos um conjunto maior de bits para realizar compressão. A justificativa pode ser que, durante o treinamento, para esse nível de reconstrução a função da distorção é significativamente superior ao valor da função de esparsidade. Então, quando o treinamento prioriza a reconstrução a proporção de zeros e uns é condicionada para reduzir a distorção. Esse fato não garante que a entropia de ordem 0 do fluxo de bits seja baixa. Além isso, o fator de regularização influencia de forma indireta na distorção. A partir da 4ª iteração a curva de ganho do modelo final ultrapassa as demais. Á medida que a reconstrução é feita, esperamos que, em um momento, a relevância da distorção na aprendizagem da rede seja reduzida, e a entropia de ordem 0 dos bits seja otimizada.

Nas Figuras **??** selecionamos 3 imagens para plotar as curvas de PSNR e ganho. Do nosso conjunto de teste, a koddim13 apresenta o menor ganho, a kodim20 tem o maior ganho e a kodim6 tem ganho aproximadamente próximo da média. Nesse teste, a kodim20 apresenta a maior qualidade (ou menor distorção) dentre os exemplos. Supomos que a rede

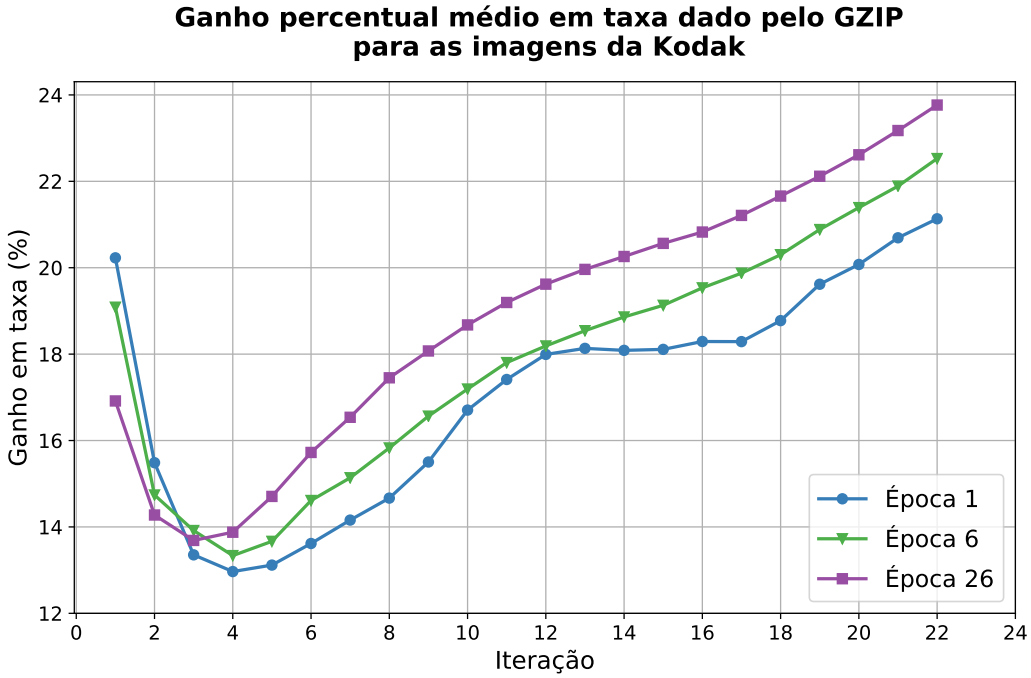


Figura 5.9 – Figura com o ganho do GZIP por nível de reconstrução e na média para as 24 imagens da Kodak em 3 codecs. De forma geral, nas primeiras iterações ocorre redução no ganho e a partir da quarta iteração o ganho de taxa é crescente.

conseguiu capturar ou "aprender" padrões com a semântica espacial presente na kodim20 com uma representação latente esparsa. Em contrapartida, acreditamos a imagem kodim13 contém muita informação de alta frequência e de "difícil compressão". Essas imagens com suas reconstruções estão disponíveis ns Figura 5.11.

No método de alocação dinâmica de bits descrito em 4.4 fizemos dois testes principais. No primeiro, sem restrição no número mínimo de níveis de reconstrução em cada bloco, isso equivale a fazer $k_min = 1$ em 4.9. Vamos no referir a ele como modeloVR0. Na tabela 5.3 apresentamos o resultado desse modelo ao aplica nas imagens da Kodak. Nessa tabela, PSNR$_d$ é o alvo de qualidade desejada e PSNR$_o$ é a qualidade obtida.

Contudo, tal procedimento gera artefatos de blocos em evidência, como podemos observar na Figura **??**. Então, fazemos $k_min = 1$ para codificar as imagens pelo modeloVR e tentar minimizar o surgimento desses artefatos de compressão. No exemplo apresentado na Figura **??**

A Figura 5.13

Em trabalhos posteriores, podemos realizar um estudo de um modelo de entropia para o conjunto de bits. Com esse estudo, podemos propor um codificador de entropia otimizado para a estatística dessa informação. Outra alternativa, consiste em propor outra função de otimização de entropia para promover padrões formados por conjuntos ou sequências de bits. Dessa forma, podemos obter ganhos maiores ao utilizar os codificadores de entropia

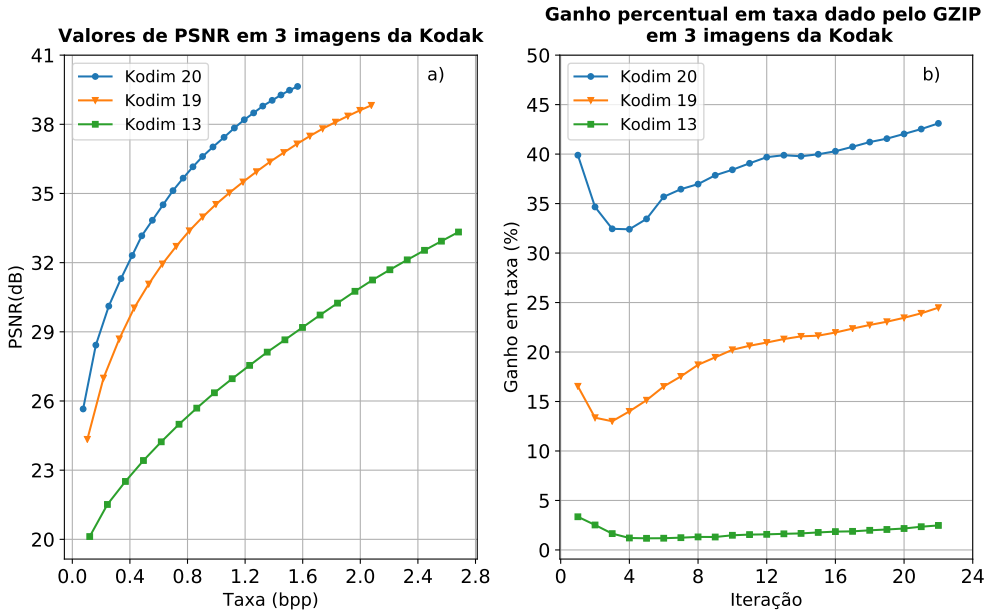


Figura 5.10

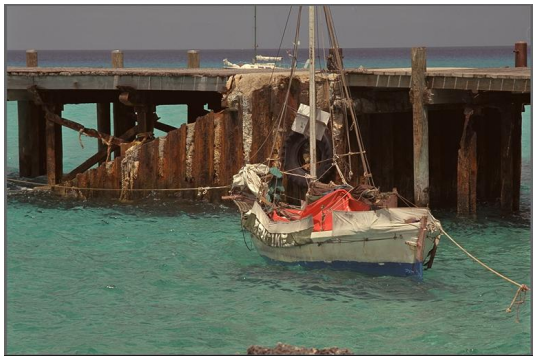
(a) Imagem original

(b) Imagem reconstruída. Taxa: 0,942 bpp. PSNR: 36,925 dB.

(c) Imagem original

(d) Imagem reconstruída. Taxa: 0,987 bpp. PSNR: 26,357 dB.

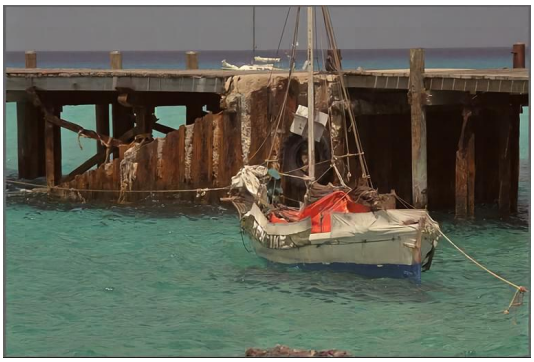
Figura 5.11 – Em (a) temos a Kodim10 original em (b) sua versão reconstruída em 10 iterações. Em (c) está representado a kodim13 e em (d) sua sua reconstruída em 8 iterações.



(a) Imagem original

(b) Imagem reconstruída. Taxa: 0,741 bpp, PSNR: 32,190dB, SSIM: 0,8675.

(c) Imagem reconstruída. Taxa: 0,782 bpp, PSNR: 32,687dB, SSIM: 0,8939



(d) Imagem reconstruída. Taxa: 0,754 bpp, PSNR: 31,621, SSIM:0,8985

Figura 5.12 – Em (a) temos a kodim11 original. A figura (b) é uma reconstrução com o modeloVR0. Em (c) a imagem é codificada pelo modeloVR de alocação de bits. Por fim, em (d) é sua reconstrução com o modelo26 usando 7 níveis de resíduo.

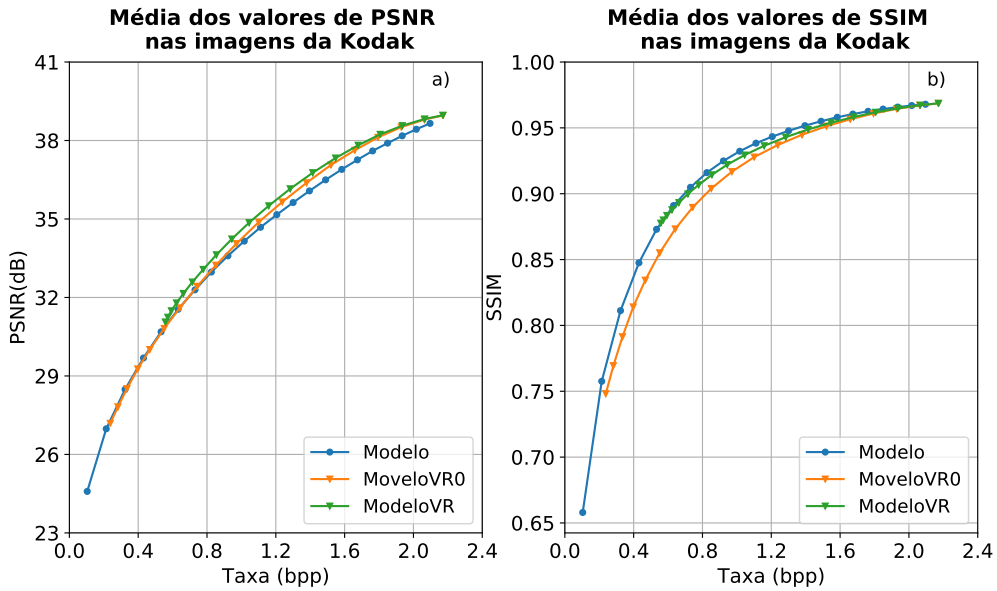


Figura 5.13

Tabela 5.3 – Resultados do Método de Alocação Dinâmica de Bits

PSNR$_d$ (dB)	**PSNR$_o$ (dB)**	**Taxa Real (bpp)**
24	27,19	0,237
25	27,82	0,282
26	28,52	0,335
27	29,27	0,397
28	30,00	0,467
29	30,81	0,551
30	31,60	0,642
31	32,42	0,742
32	33,23	0,851
33	34,05	0,971
34	34,88	1,100
35	35,65	1,237
36	36,38	1,378
37	37,06	1,520
38	37,62	1,658
39	38,12	1,793
40	38,51	1,929
41	38,79	2,063
42	38,96	2,171

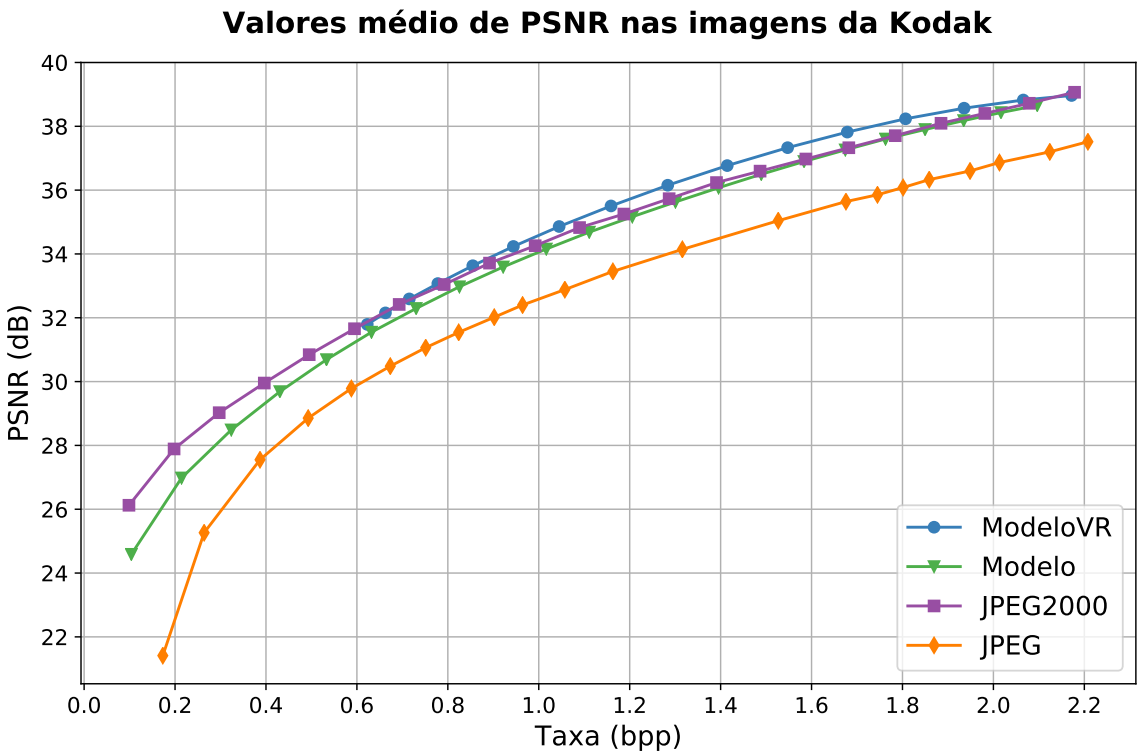


Figura 5.14

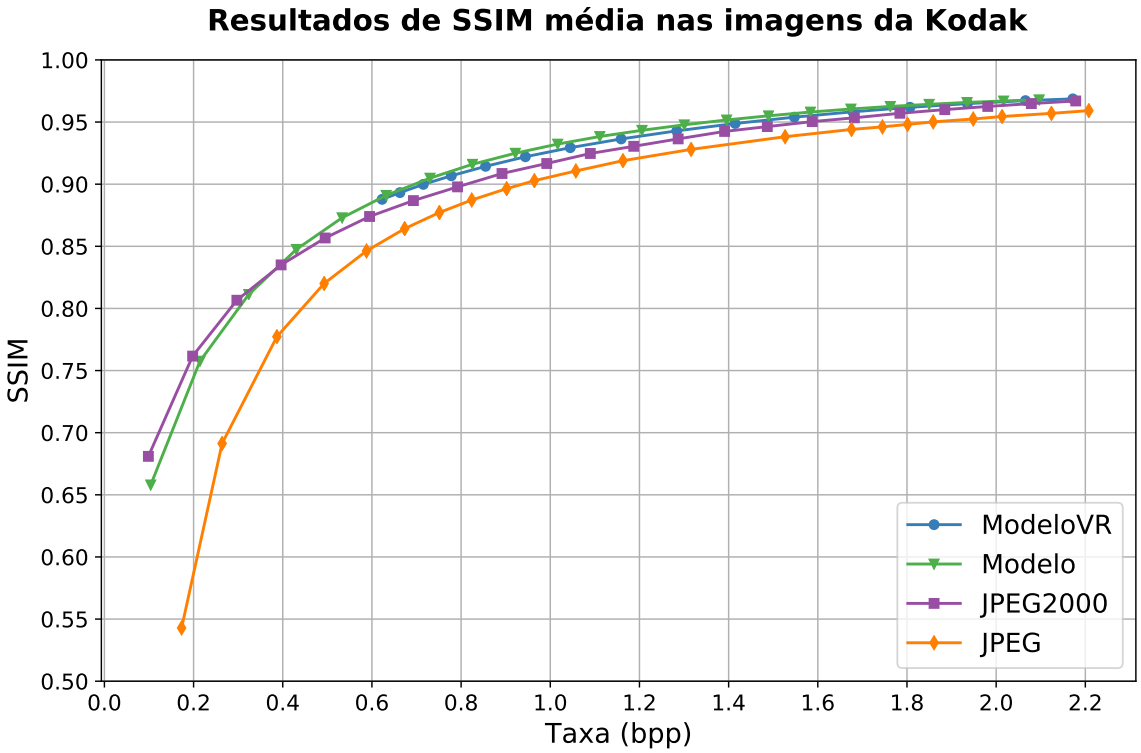


Figura 5.15

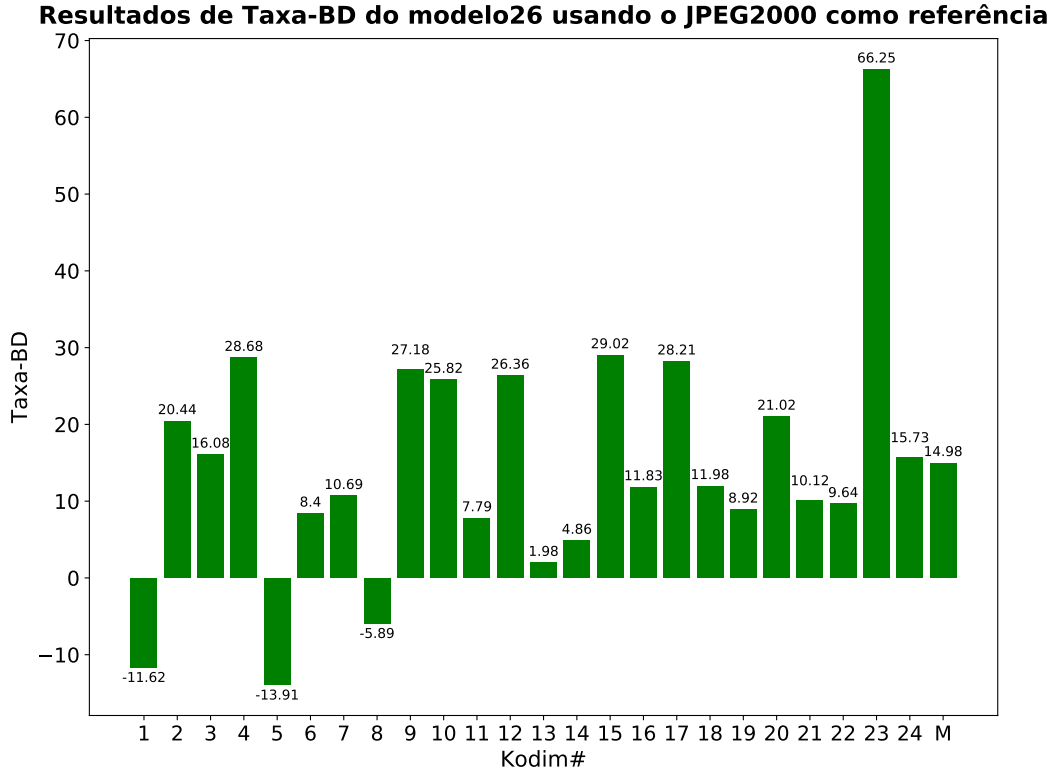


Figura 5.16

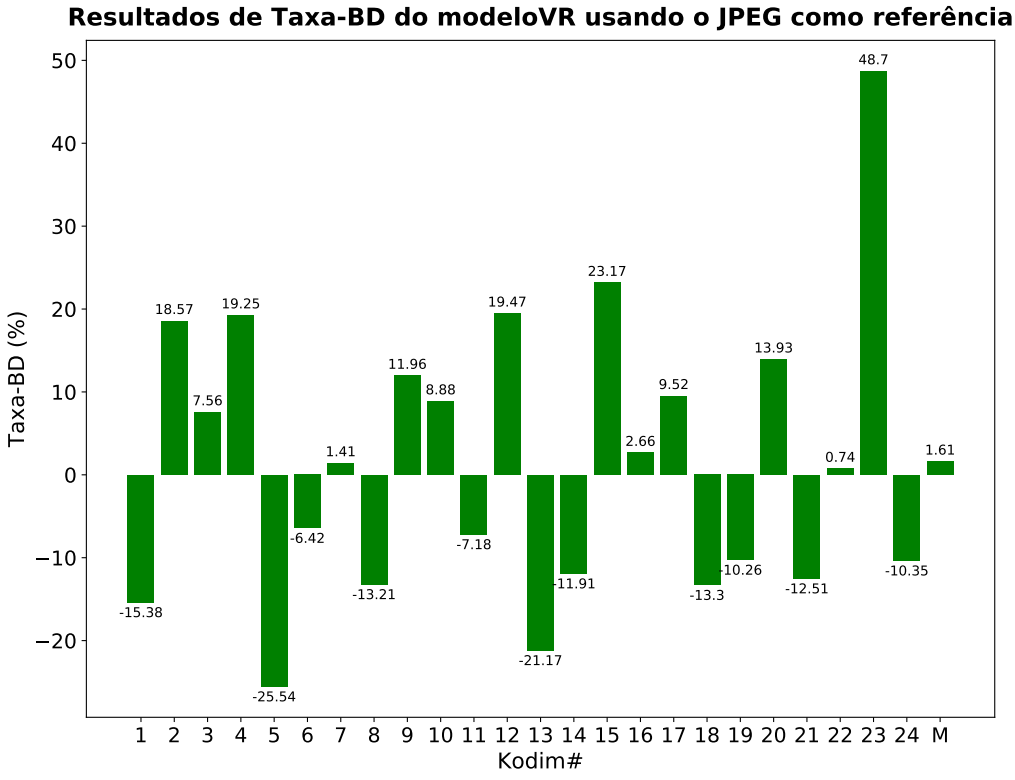

Resultados de Taxa-BD do modeloVR usando o JPEG como referência
Taxa-BD (%)
Kodim#
50
40
30
20
10
0
−10
−20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 M
-15.38
18.57
7.56
19.25
-25.54
-6.42
1.41
-13.21
11.96
8.88
-7.18
19.47
-21.17
-11.91
23.17
2.66
9.52
-13.3
-10.26
13.93
-12.51
0.74
48.7
-10.35
1.61

Figura 5.17

comerciais.

REFERÊNCIAS BIBLIOGRÁFICAS

1 TODERICI, G. et al. Full resolution image compression with recurrent neural networks. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2017. p. 5306–5314. Preprint available at <http://arxiv.org/abs/1608.05148>.

2 SHALEV-SHWARTZ, S.; BEN-DAVID, S. *Understanding machine learning: From theory to algorithms*. [S.l.]: Cambridge university press, 2014.

3 CHOLLET, F. *Deep Learning wiht Python*. [S.l.]: Maning Publications Co, 2018. ISBN 9781617294433.

4 HAYKIN, S. *Redes Neurais Princípios e Prática*. [S.l.]: Prentice Hall, 2001.

5 LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. *nature*, Nature Publishing Group, v. 521, n. 7553, p. 436–444, 2015.

6 BENGIO, Y. et al. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, Now Publishers, Inc., v. 2, n. 1, p. 1–127, 2009.

7 MITCHELL, T. M. *Machine Learning*. [S.l.]: McGraw-Hill Science, 1997.

8 MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, Springer, v. 5, n. 4, p. 115–133, 1943.

9 GOODFELLOW, S.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. <http://www.deeplearningbook.org>.

10 NAIR, V.; HINTON, G. E. Rectified linear units improve restricted boltzmann machines. In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. [S.l.: s.n.], 2010. p. 807–814.

11 XU, B. et al. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

12 RUMELHART, D. E. et al. Learning representations by back-propagating errors. *Cognitive modeling*, v. 5, n. 3, p. 1, 1988.

13 HORNIK, K.; STINCHCOMBE, M.; WHITE, H. Multilayer feedforward networks are universal approximators. *Neural networks*, Elsevier, v. 2, n. 5, p. 359–366, 1989.

14 LESHNO, M. et al. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, Elsevier, v. 6, n. 6, p. 861–867, 1993.

15 GÉRON, A. *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. [S.l.]: " O'Reilly Media, Inc.", 2017.

16 SAHA, S. *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*. Disponível em: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>.

17 RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. *Nature*, Nature Publishing Group, v. 323, n. 6088, p. 533, 1986.

18 KETKAR, N. et al. *Deep Learning with Python*. [S.l.]: Springer, 2017.

19 HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. *Neural Computation*, v. 9, n. 8, p. 1735–1780, 1997. Disponível em: <https://doi.org/10.1162/neco.1997.9.8.1735>.

20 OLAH, C. *Understanding LSTM Networks*. Disponível em: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>.

21 XINGJIAN, S. et al. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2015. p. 802–810.

22 NG, A. et al. Sparse autoencoder. *CS294A Lecture notes*, Stanford University, v. 72, n. 2011, p. 1–19, 2011.

23 KULLBACK, S.; LEIBLER, R. A. On information and sufficiency. *The annals of mathematical statistics*, JSTOR, v. 22, n. 1, p. 79–86, 1951.

24 KINGMA, D. P.; WELLING, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

25 REZENDE, D. J.; MOHAMED, S.; WIERSTRA, D. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

26 KLYS, J.; SNELL, J.; ZEMEL, R. Learning latent subspaces in variational autoencoders. In: *Advances in Neural Information Processing Systems*. [S.l.: s.n.], 2018. p. 6444–6454.

27 ROLINEK, M.; ZIETLOW, D.; MARTIUS, G. Variational autoencoders pursue pca directions (by accident). In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2019. p. 12406–12415.

28 SØNDERBY, C. K. et al. Ladder variational autoencoders. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2016. p. 3738–3746.

29 GONZALEZ, R. C.; WOODS, R. C. *Processamento digital de imagens* . [S.l.]: Pearson Educación, 2009.

30 SAYOOD, K. *Introduction to data compression*. [S.l.]: Morgan Kaufmann, 2017.

31 FILHO, O. M.; NETO, H. V. *Processamento digital de imagens*. [S.l.]: Brasport, 1999.

32 RICHARDSON, I. E. *The H. 264 Advanced Video Compression Standard*. [S.l.]: John Wiley & Sons Ltd, 2010.

33 SALOMON, D. *Data compression: the complete reference*. [S.l.]: Springer Science & Business Media, 2007.

34 HORE, A.; ZIOU, D. Image quality metrics: Psnr vs. ssim. In: IEEE. *2010 20th International Conference on Pattern Recognition*. [S.l.], 2010. p. 2366–2369.

35 WANG, Z.; SIMONCELLI, E. P.; BOVIK, A. C. Multiscale structural similarity for image quality assessment. In: IEEE. *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*. [S.l.], 2003. v. 2, p. 1398–1402.

36 JOHNSTON, N. et al. *Improved Lossy Image Compression with Priming and Spatially Adaptive Bit Rates for Recurrent Networks*. [S.l.], 2017. Disponível em: <http://arxiv.org/abs/1703.10114>.

37 MINNEN, D.; BALLÉ, J.; TODERICI, G. Joint Autoregressive and Hierarchical Priors for Learned Image Compression. In: *NIPS*. [s.n.], 2018. Disponível em: <http://arxiv.org/abs/1809.02736>.

38 AKBARI, M.; LIANG, J.; HAN, J. Dsslic: deep semantic segmentation-based layered image compression. In: IEEE. *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. [S.l.], 2019. p. 2042–2046.

39 JIANG, J. Image compression with neural networks–a survey. *Signal processing: image Communication*, Elsevier, v. 14, n. 9, p. 737–760, 1999.

40 GERSHO, A.; GRAY, R. M. *Vector quantization and signal compression*. [S.l.]: Springer Science & Business Media, 2012. v. 159.

41 BALLÉ, J.; LAPARRA, V.; SIMONCELLI, E. P. End-to-end Optimized Image Compression. *ICLR*, nov 2016. ISSN 01973975. Disponível em: <http://arxiv.org/abs/1611.01704>.

42 SANTURKAR, S.; BUDDEN, D.; SHAVIT, N. Generative compression. In: IEEE. *2018 Picture Coding Symposium (PCS)*. [S.l.], 2018. p. 258–262.

43 TODERICI, G. et al. Variable Rate Image Compression with Recurrent Neural Networks. In: *International Conference on Learning Representations (ICLR)*. [s.n.], 2015. p. 1–12. Disponível em: <http://arxiv.org/abs/1511.06085>.

44 COVELL, M. et al. *Target-Quality Image Compression with Recurrent, Convolutional Neural Networks*. [S.l.], 2017. Disponível em: <http://arxiv.org/abs/1705.06687>.

45 THEIS, L. et al. *Lossy Image Compression with Compressive Autoencoders*. [S.l.], 2017. Disponível em: <http://arxiv.org/abs/1703.00395>.

46 BALLÉ, J. et al. Variational image compression with a scale hyperprior. In: *ICLR*. [s.n.], 2018. Disponível em: <http://arxiv.org/abs/1802.01436https://arxiv.org/abs/1802.01436>.

47 ZHAO, H.; LIAO, P. *CAE-ADMM: Implicit Bitrate Optimization via ADMM-based Pruning in Compressive Autoencoders.* 1901.

48 NIE, F. et al. Efficient and robust feature selection via joint l2, 1-norms minimization. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2010. p. 1813–1821.

49 BOLIEK, M.; CHRISTOPOULOS, C.; MAJANI, E. Information technology: Jpeg2000 image coding system. *published by ISO/IEC as*, p. 15444–1, 2000.

50 MINNEN, D. et al. Spatially adaptive image compression using a tiled deep network. In: *International Conference on Image Processing*. [s.n.], 2017. Disponível em: <http://arxiv.org/abs/1802.02629>.

51 HUNG, E. M. et al. *Hybrid DPCM artificial intelligence based image codec*. [S.l.], 2019.

52 RADFORD, A.; METZ, L.; CHINTALA, S. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

53 RIPPEL, O.; BOURDEV, L. Real-Time Adaptive Image Compression. In: *ICML*. [s.n.], 2017. Disponível em: <http://arxiv.org/abs/1705.05823>.

54 AGUSTSSON, E. et al. Generative adversarial networks for extreme learned image compression. In: *Proceedings of the IEEE International Conference on Computer Vision*. [S.l.: s.n.], 2019. p. 221–231.

55 MIRZA, M.; OSINDERO, S. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

56 LI, M. et al. Learning convolutional networks for content-weighted image compression. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2018. p. 3214–3223.

57 YE, S. et al. Progressive weight pruning of deep neural networks using admm. *arXiv preprint arXiv:1810.07378*, 2018.

58 HUNG, E. M. et al. *A review of the state-of-the-art in AI-based image codecs*. [S.l.], 2019.

59 DATASET of the CVPR Workshop and Challenge on Learned Image Compression (CLIC). Online. <http://www.compression.cc>.

60 AGUSTSSON, E.; TIMOFTE, R. NTIRE 2017 challenge on single image super-resolution: Dataset and study. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. [S.l.: s.n.], 2017. DIV2K dataset: DIVerse 2K resolution high quality images as used for the challenges at NTIRE (CVPR 2017 and CVPR 2018) and at PIRM (ECCV 2018), available from <https://data.vision.ee.ethz.ch/cvl/DIV2K/>.

61 NEMOTO, H. et al. Ultra-Eye: UHD and HD images eye tracking dataset. In: *Sixth International Workshop on Quality of Multimedia Experience (QoMEX)*. Singapore: [s.n.], 2014. Available from <http://mmspg.epfl.ch/ultra-eye>. Disponível em: <http://infoscience.epfl.ch/record/200190>.

62 KODAK Image Dataset. 2019. Available from <http://www.cs.albany.edu/~xypan/research/snr/Kodak.html>.

63 PYTORCH-IMAGE-COMP-RNN. 2018. Github repository. PyTorch source code for Full Resolution Image Compression with Recurrent Neural Networks, available from <https://github.com/1zb/pytorch-image-comp-rnn>.

64 SU, P.-H. *pytorch-ssim*. [S.l.]: GitHub, 2017. <https://github.com/Po-Hsun-Su/pytorch-ssim>.