USO DE ÁRVORES DE DECISÃO EM UM TRANSCODIFICADOR DE VÍDEO DE H.264/AVC PARA H.265/HEVC

THIAGO DANTAS DE MORAES ALMEIDA

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

USO DE ÁRVORES DE DECISÃO EM UM TRANSCODIFICADOR DE VÍDEO DE H.264/AVC PARA H.265/HEVC

THIAGO DANTAS DE MORAES ALMEIDA

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

USO DE ÁRVORES DE DECISÃO EM UM TRANSCODIFICADOR DE VÍDEO DE H.264/AVC PARA H.265/HEVC

THIAGO DANTAS DE MORAES ALMEIDA

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. Daniel Guerreiro e Silva – ENE/Universidade de Brasília
Membro Interno

_____

Prof. Dr. Bruno L. Macchiavello Espinoza – CIC/Universidade de Brasília
Membro Interno

BRASÍLIA, 26 DE NOVEMBRO DE 2019.

FICHA CATALOGRÁFICA

DANTAS, THIAGO
Uso de Árvores de Decisão em um Transcodificador de Vídeo de H.264/AVC para H.265/HEVC [Distrito Federal] 2019.
xv, 75p., 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. Árvores de Decisão	2. Transcodificação de Vídeo
3. H.264/AVC	4. H.265/HEVC
I. ENE/FT/UnB	II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

DANTAS, T. (2019). Uso de Árvores de Decisão em um Transcodificador de Vídeo de H.264/AVC para H.265/HEVC . Trabalho de Conclusão de Curso em Engenharia Elétrica, Publicação PPGEA.TD-001/11, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 75p.

CESSÃO DE DIREITOS

Thiago Dantas de Moraes Almeida

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

À minha família.

AGRADECIMENTOS

Agradeço primeiramente a Deus pela saúde e pelas oportunidades de crescer como pessoa e como profissional. Agradeço aos meus familiares que sempre me apoiaram em minhas escolhas e que sempre me incentivaram a buscar ser minha melhor versão. Agradeço aos meus amigos e colegas de curso por fazerem da jornada que é a faculdade algo mais leve e descontraído. Agradeço a todos os professores que sempre estiveram dispostos a ajudar, tirar dúvidas e ensinar.

RESUMO

A maioria do conteúdo em vídeo consumido atualmente, seja por meio de serviços de *streaming* (*Netflix, Youtube*), cinema, mídias de DVD ou *Blu-Ray*, utiliza da tecnologia de compressão de vídeo para representar o vídeo de forma mais compacta, a fim de tornar factível tanto o armazenamento como a transmissão desse tipo de conteúdo. Transmissão de vídeo corresponde à maior parcela do tráfego mundial da *internet* e continua a crescer substancialmente. De acordo com o *Cisco Visual Networking Index* [**??**], o tráfego de vídeo de IP vai quadruplicar no período de 2017-2022, subindo de 75% do tráfego total da *internet* em 2017 para 82% por volta de 2022. Com esta crescente, mostram-se necessárias não só melhores técnicas de compressão de vídeo como também formas de permitir a interoperabilidade entre sistemas que utilizam diferentes padrões de codificação.

Um standardo de codificação de vídeo é a especificação de como representar uma sequência de vídeo em um arquivo ou *bitstream* [**??**]. Uma sequência de vídeo codificada em um padrão pode ser decodificada apenas pelo decodificador daquele padrão, não sendo compatível com outro decodificador. Para que sistemas que utilizam diferentes padrões se comuniquem é necessária uma mudança de padrão, chamada de transcodificação, que consiste na conversão de um *bitstream*, codificado com um determinado standardo (chamado de standardo fonte) para outro *bitstream*, codificado com outro standardo (chamado de standardo alvo) [**??**].

O padrão mais utilizado atualmente, o padrão internacional H.264 Advanced Video Coding (H.264/AVC), está sendo substituído por um novo padrão, chamado H.265 High Efficiency Video Coding (H.265/HEVC). O H.265/HEVC oferece uma redução de 30% a 50% na taxa de bits, para uma mesma qualidade, comparado com o H.264/AVC [**??**]. Assim, um transcodificador de vídeo do standardo H.264/AVC para o standardo H.265/HEVC tem uma motivação dupla: promover a inter-operabilidade entre sistemas que utilizam esses standardos e aproveitar o maior ganho de compressão do novo padrão.

Neste trabalho, comparamos a performance de 10 transcodificadores diferentes em um conjunto de sequências já conhecidas e com diversos resultados publicados. Entre os 10 transcodificadores, 8 utilizam Árvores de Decisão, sendo 4 com treinamento *offline* e 4 com treinamento *online*. Dois dos transcodificadores testados utilizam LDFs, ambos com treinamento *online*. Em específico, comparamos dois transcodificadores em que a única diferença é o algoritmo de classificação e o transcodificador com Árvores de Decisão obteve um BD-Rate médio 2.22 vezes menor que o que utiliza LDFs para uma *speed up* 1.11 vezes menor. Além disso, propôs-se uma novo critério de classificação para o modelo das Árvores de Decisão que é baseado no custo taxa-distorção do HEVC. Essa técnica mostrou-se promissora, apresentando resultados muito satisfatórios para algumas sequências.

Para testar os transcodificadores de forma eficiente e rápida, foram utilizados os serviços de computação na nuvem oferecidos pelo *Google Cloud Platform*. A utilização dessa ferramenta foi crucial para o desenvolvimento deste trabalho, uma vez que permitiu que fossem feitos diversos testes em paralelo.

A implementação das Árvores de Decisão foi feita utilizando a biblioteca *Scikit-Learn* do *Python*. Todo o código desenvolvido neste trabalho está disponível de forma aberta e gratuita neste link.

Palavras-chave: Árvores de Decisão, Transcodificação de Vídeo, H.264/AVC, H.265/HEVC.

ABSTRACT

Most video content consumed today, whether through streaming (Netflix, Youtube), cinema, DVD media or Blu-Ray uses some video compression technology to represent the video more compactly to make it feasible both the storage and transmission of such content. Video broadcasting accounts for the largest share of the internet world traffic and continues to grow substantially. According to the Cisco Visual Networking Index [**??**], IP video traffic will grow four-fold in the period of 2017-2022, rising from 75% of the total internet traffic in 2017 to 82% by 2022. Therefore, there is a need not only for more efficient video compression, but also for other tools that facilitate the use of such compression technology. Video transcoding is one such tool in this context. By enabling inter-operability between different codecs and systems, video transcoding allows for the use of newer, more efficient codecs.

A video encoding standard is the specification of how to represent a video sequence in a file or bitstream [**??**]. A video sequence encoded in one standard can be decoded only by that standard's decoder and is not compatible with another decoder. For systems that use different standards to communicate, a change of standard, called transcoding, is required, which consists of converting a bitstream encoded with a given standard (called the source standard) to another bitstream, encoded with another standard (called the target standard) [**??**].

The most widely used standard, the international standard H.264/AVC, is being replaced by a new standard called H.265/HEVC. H.265/HEVC offers a 30% to 50% reduction in bitrate for the same quality, compared to H.264/AVC [**??**]. So a video transcoder of the standard H.264/AVC for the standard H.265/HEVC has a twofold motivation: to promote interoperability between systems using these standards and to take advantage of the higher compression gain of the new standard.

In this work, we compare the performance of 10 different transcoders in a set of known sequences and with several published results. Among the 10 transcoders, 8 use Decision Trees, 4 with offline training and 4 with online training. Two of the transcoders tested use LDFs, both with online training. Specifically, we compared two transcoders where the only difference is the classification algorithm and the Decision Tree transcoder obtained an average BD-Rate 2.22 times lower than using LDFs for a 1.11 times lower speed up. In addition, a new classification criterion was proposed for the Decision Tree model that is based on the H.265/HEVC rate-distortion cost. This technique was promising, presenting very satisfactory results for some sequences.

In order to test transcoders efficiently and quickly, we used the cloud computing services offered by Google Cloud Platform. The use of this tool was crucial for the development of this work, as it allowed several tests to be done in parallel.

Decision Trees were implemented using Python's Scikit-Learn library. All code developed in this work is available free and open source at this link.

Keywords: Decision Trees, Video Transcoding, H.264/AVC, H.265/HEVC.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÔNIMOS E ABREVIA CÕES

AC	Autocodificador (*autoencoder*). 15–17
CCITT	*Consultative Committee for International Telephony and Telegraphy*. 22
CODEC	Codificador/Decodificador (*Encoder/Decoder*). 18, 22
Conv2D	convolução Bidimensional. 10
Conv2DLSTM	LSTM Convolucional Bidimensional. 15
DCT	Transformação Discreta de Cosseno (*Discrete Cosine Transform*). 22
FC-LSTM	LSTM Totalmente Conectada (*Fully-Conected LSTM*). 14
H.264/AVC	H.264 Advanced Video Coding. i
H.265/HEVC	H.265 High Efficiency Video Coding. i
IDCT	Transformada Discreta de Cosseno Inversa (*Inverse Discrete Cosine Transform*). 23
ISO	International Organization for Standardisation. 22
ITU-T	International Telecommunication Union. 22
JPEG	Grupo de Especialistas em Fotografia (*Joint Photographic Experts Group*). x, 1, 23
LSTM	Memória de Longo Prazo (*Long Short-Term Memory*). x, 13, 14
MLP	Perceptron Multicamadas (*Multilayer Perceptron*). 8, 9, 11
MS-SSIM	Índice de Similaridade Estrutural em Multi-Escalas (*Multi-scale Structural Similarity Index*). 24, 25
MSE	Erro Médio Quadrático (*Mean Square Error*). 24
PSNR	Relação Pico-Sinal-Ruído (*Peak Signal to Noise Ratio*). 23, 24
ReLU	Unidade Linear Retificadora (*Rectified Linear Unit*). 4, 5, 9
RGB	Vermelho, Verde, Azul (*Red, Green, Blue*). 11, 20, 21

RLE Codificação de Comprimento de Execução (*Run-Length Encoding*). 23

RNA Rede Neural Artificial (*Artificial Neural Network*). x, 1–4, 8, 10

RNC Rede Neural Convolucional (*Convolutional Neural Network*). 10

RNR Rede Neural Recorrente (*Recurrent Neural Network*). 12

SSIM Índice de Similaridade Estrutural (*Structural Similarity Index*). 24, 25

VAE autocodificador variacional (*variational autocoders*). 17

YCbCr Luminância, Crominância Azul, Crominância Vermelho (*Luminance, Blue Chrominance, Red Chrominance*). 22

1 FUNDAMENTAÇÃO TEÓRICA

Nesse capítulo, primeiro apresentaremos uma visão geral das redes neurais artificiais, desde o neurônio artificial, os principais tipos de arquitetura e chegando até os autocodificadores variacionais. No segundo momento, fundamentamos brevemente a teoria de compressão de dados, 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.

1.1 REDES NEURAIS ARTIFICIAIS

Nessa seção, descrevemos a ideia básica de qualquer Rede Neural Artificial (*Artificial Neural Network*) (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) [5]. 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 [6].

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 1.1 resume a diferença essencial entre a programação convencional e o aprendizado de máquina [6].

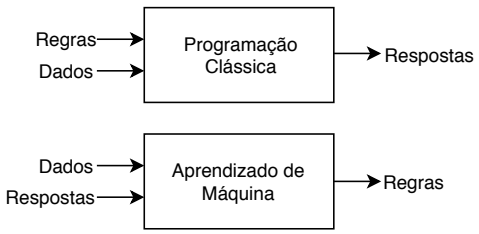


Figura 1.1 – Aprendizado de Máquina: um novo paradigma de programação. Adaptado de [6].

As RNA's são técnicas específicas de aprendizado de máquina inspiradas no funciona-

mento do cérebro humano. A semelhança com esse órgão humano advém de dois aspectos principais [7]:

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 [8, 9]. 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 [6]. 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 1.2 apresenta um esquemático do procedimento para o treinamento de uma rede neural artificial de 2 camadas.

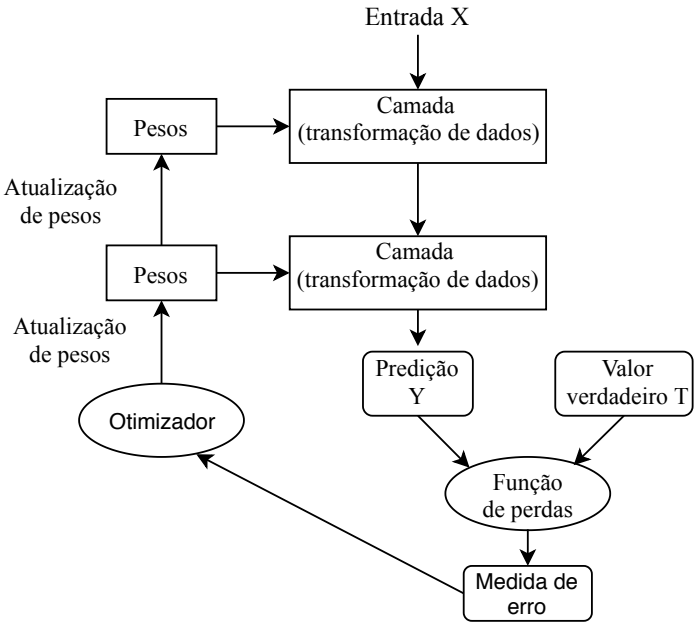


Figura 1.2 – A rede neural composta por 2 camadas encadeadas mapeia os dados de entrada para as previsões. A função de perdas compara essas previsões às metas, produzindo um valor de erro. O otimizador usa esse valor para atualizar os pesos da rede. Adaptado de [6].

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 [7]. As principais propriedades úteis das RNA's em diversas tarefas de tomada de decisão são [7]:

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.

1.1.1 Neurônio Artificial

O neurônio é a unidade mais básica da RNA. O seu modelo está ilustrado na Figura 1.3.

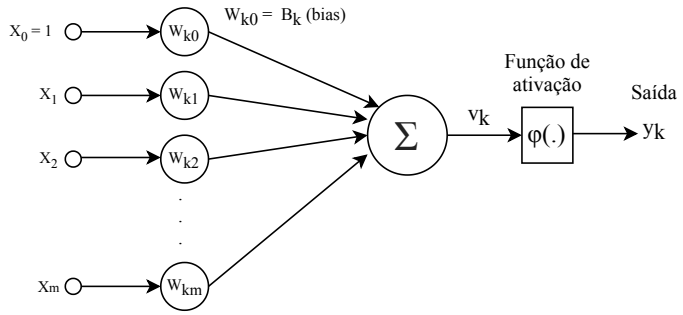


Figura 1.3 – Representação do neurônio artificial formado por um vetor de peso W de tamanho m. A entrada X_0 é artificial e o peso associada a ela é denominado viés. O neurônio funciona como um operador matemático simples que realiza um mapeamento de muitos para 1. Adaptado de [7].

A sua descrição matemática é dada pelas seguintes equações.

$$v_k = \sum_{j=0}^{m} w_{kj} x_j \tag{1.1}$$

$$y_k = \phi(v_k) \tag{1.2}$$

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.

1.1.1.1 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 [10]. O neurônio com a função limiar foi utilizado no primeiro modelo de perceptron [11, 7] 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]. Elas são usadas em neurônios das camadas intermediárias ou na última para problemas de classificação binária ou regressão. A função *softmax* também é uma função sigmóide usada para prever as probabilidades associadas a uma classificação multi-classe [12]. 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 [12]. A Unidade Linear Retificadora (*Rectified Linear Unit*) (ReLU) [13], 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 [14]. As equações das funções de ativações mencionadas aqui e a suas derivadas estão apresentadas na Tabela 1.1.

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 [10]. 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*) [15, 12] 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 [12]. O segundo é usado para atualizar os pesos por meio do gradiente calculado [10]. A descida do gradiente é um algoritmo específico dos otimizadores. A seguir, apresentamos a atuação desses algoritmos no treinamento do neurônio artificial. A Equação 1.3 é uma função de perdas relativa aos exemplos

Tabela 1.1 – 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 [10].

$$E(\mathbf{w}) = \frac{1}{2} \sum_{d=0}^{D} (t_d - y_d)^2 \tag{1.3}$$

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** [10]. A Figura 1.4 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 [10].

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.

$$\nabla E(\mathbf{w}) \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, ..., \frac{\partial E}{\partial w_m} \right] \tag{1.4}$$

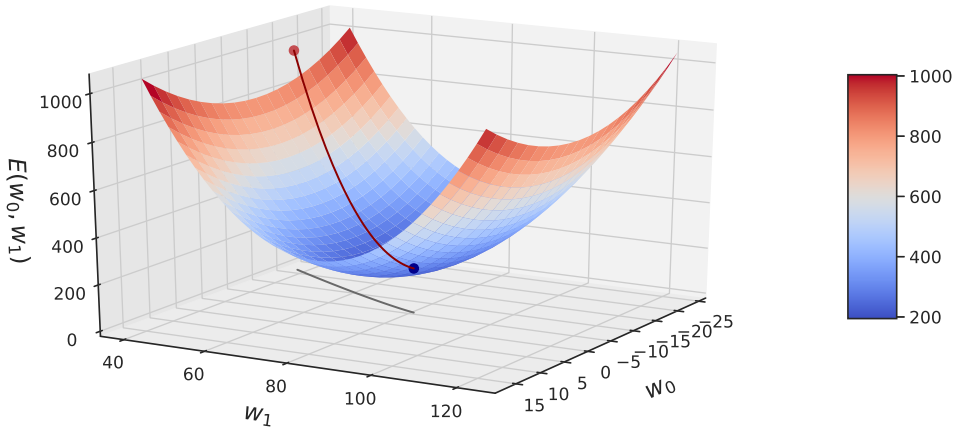


Figura 1.4 – Visualização do espaço de hipóteses dos pesos e atuação do algoritmo de descida de gradiente. Os pontos vermelho e azul indicam o início e o fim do aprendizado, respectivamente. A linha cinza é a projeção da curva de aprendizado no plano $w_0 w_1$.

Portanto, o negativo do vetor $\nabla E(\mathbf{w})$ informa a direção de maior decaimento do erro e a regra de treinamento para a descida do gradiente é:

$$\begin{aligned} \mathbf{w}^{t+1} &= \mathbf{w}^t + \Delta(\mathbf{w}^t) \\ \Delta(\mathbf{w}^t) &= -\eta \times \nabla E(\mathbf{w}^t) \end{aligned} \tag{1.5}$$

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:

$$w_i^{t+1} = w_i^t - \eta \times \frac{\partial E}{\partial w_i^t} \tag{1.6}$$

O desenvolvimento matemático da Equação 1.6 nos fornece o valor do gradiente para

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

$$
\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{d=0}^{D}(t_d - y_d)^2 \\
\frac{\partial E}{\partial w_i} &= \frac{1}{2}\sum_{d=0}^{D}\frac{\partial}{\partial w_i}(t_d - y_d)^2 \\
\frac{\partial E}{\partial w_i} &= \frac{1}{2}\sum_{d=0}^{D}2(t_d - y_d)\frac{\partial}{\partial w_i}(t_d - y_d) \\
\frac{\partial E}{\partial w_i} &= \sum_{d=0}^{D}(t_d - y_d)\frac{\partial}{\partial w_i}(t_d - \phi(v)) \\
\frac{\partial E}{\partial w_i} &= \sum_{d=0}^{D}(t_d - y_d)(0 - \phi'(v)\frac{\partial v}{\partial w_i}) \\
\frac{\partial E}{\partial w_i} &= -\sum_{d=0}^{D}(t_d - y_d)\phi'(v)\frac{\partial}{\partial w_i}\sum_{j=0}^{m} w_j x_j) \\
\frac{\partial E}{\partial w_i} &= -\sum_{d=0}^{D}(t_d - y_d)\phi'(v)x_i
\end{aligned}
\tag{1.7}
$$

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:

$$
\begin{aligned}
w_i^{t+1} &= w_i^t + \Delta w_i^t \\
\Delta w_i^t &= \eta \times \sum_{d=0}^{D}(t_d - y_d)\phi'(v)x_i
\end{aligned}
\tag{1.8}
$$

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*) [7]. A regra de treinamento de descida de gradiente apresentada na Equação 1.8 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 [12].

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 [12]. Nesse cenário, aproximamos o método de descida do gradiente pela sua versão estocástica. A regra de treinamento modificada é semelhante à Equação 1.8, exceto que à medida que iteramos em cada

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

$$\Delta w_i^t = \eta \times \sum_{d=0}^{m} (t_d - y_d)\phi'(v)x_i \tag{1.9}$$

1.1.2 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 [12]. 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, $\mathbf{y} = f^{(2)}(f^{(1)}(\mathbf{x}))$ [12]. 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.

1.1.2.1 Perceptron Multicamadas

A Figura 1.5 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 [16] 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 [12].

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

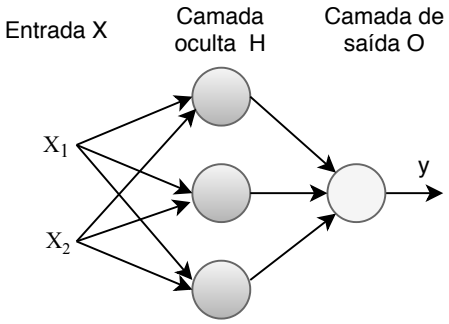


Figura 1.5 – Na rede *feedforward* as informações fluem através das camadas. Não há conexões de retroalimentação nas quais as saídas do modelo sejam realimentadas. Adaptado de [12].

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:

$$F(x_1, .., x_{m_0}) = \sum_{i=1}^{m_1} \alpha_i \phi(\sum_{i=1}^{m_0} w_{ij} x_j + b_i) \tag{1.10}$$

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

$$|F(x_1, ..., x_{m_0}) - f(x_1, ..., x_{m_0})| < \epsilon \tag{1.11}$$

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

A equação apresentada em 1.10 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 [7].

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 [17]. Uma rede neural também pode aproximar qualquer mapeamento de função de qualquer espaço discreto dimensional finito para outro [12].

O teorema da aproximação universal sintetizado pela Equação 1.10 generaliza as aproximações por série de Fourier [7]. 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) [12]. 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 [7]. Na prática, as redes com mais de uma camada oculta se mostram mais concebíveis computacionalmente e generalizam melhor [12].

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 [12].

1.1.2.2 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 [7]. 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 [18], 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 [6]. 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 $\mathbf{I}$ 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 $\mathbf{S}$ calculado segundo

a equação a seguir.

$$S(i,j) = (I * K)(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} I(i+m, j+n) K(m,n) \tag{1.12}$$

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

A Figura 1.6 ilustra a convolução 2D entre um mapa de recursos com um filtro $K = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$.

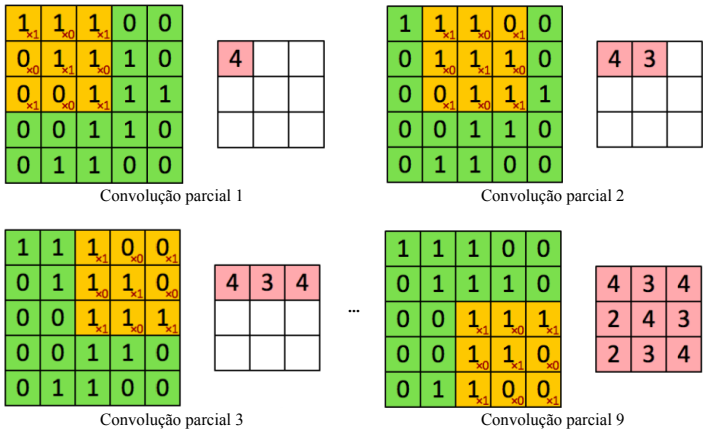


Figura 1.6 – Convolução de um tensor $5 \times 5 \times 1$, em verde, com um filtro $3 \times 3 \times 1$, em amarelo. À medida que o filtro se desloca pela entrada obtemos progressivamente um novo mapa de característica, em rosa, com dimensões $3 \times 3 \times 1$. O filtro K se desloca 9 vezes, sempre 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 [19].

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 1.12 em cada canal e todos os resultados são somados com o viés para fornecer uma saída. A Figura 1.7 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 [7, 272]. Os pesos dos filtros são os mesmos para cada mapa gerado. Esse compartilhamento de pesos é uma restrição estrutural que proporciona redução drástica do número de parâmetros livres em comparação com uma rede MLP [7, 272].

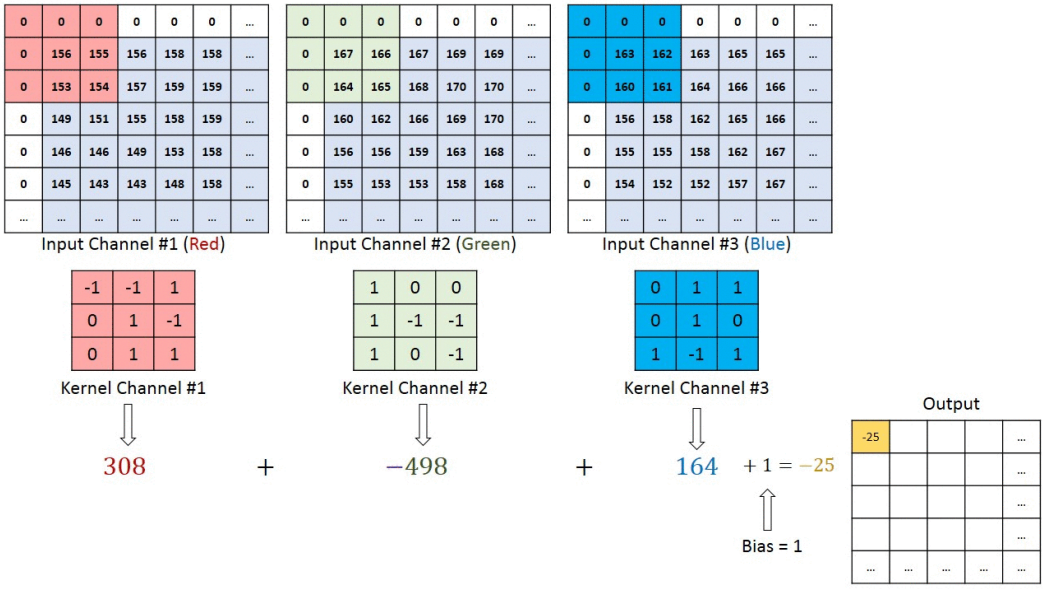


Figura 1.7 – Convolução de uma imagem $5 \times 5 \times 3$, preenchido de zeros nas bordas, com um filtro K $3 \times 3 \times 1$. Para cada canal da imagem é feito uma convolução 2D com o respectivo canal do filtro. O resultado das convoluções são somados entre si e com o viés para fornecer a saída gradualmente. O número de canais na saída é apenas 1 porque estamos utilizando 1 filtro [19].

1.1.3 Redes Recorrentes

A Rede Neural Recorrente (*Recurrent Neural Network*) (RNR) [20] é uma classe de redes neurais que incluem 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 [6] e processar um conjunto de entrada arbitrariamente grande e de tamanho variável [21]. A Figura 1.8 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 [12].

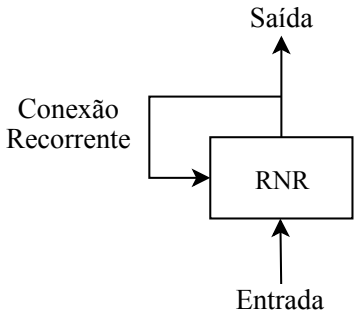


Figura 1.8 – Arquitetura básica para uma rede neural recorrente. As saídas anteriores são usadas na predição do estado atual. Adaptado de [6].

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

$$\mathbf{h}_t = \phi(\mathbf{x}_t\mathbf{W} + \mathbf{h}_{t-1}\mathbf{U}) \tag{1.13}$$

Onde h_{t-1} e h_t são os estados ocultos ou saídas da unidade recorrente no instante de 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 [12]. 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 [18].

Atualmente, um dos modelos de sequências mais eficazes usados em aplicações práticas é a Memória de Longo Prazo (*Long Short-Term Memory*) (LSTM) [22]. A sua arquitetura favorece a rápida convergência e detecção de dependências de longo prazo nos dados [18]. Essas redes contém têm "células LSTM" que possuem uma recorrência interna além da recorrência externa. Veja a Figura 1.9 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.

$$\begin{aligned}
\mathbf{i}_t &= \sigma_i(\mathbf{W}_i\mathbf{x}_t + \mathbf{U}_i\mathbf{h}_{t-1} + \mathbf{b}_i) \\
\mathbf{f}_t &= \sigma_g(\mathbf{W}_f\mathbf{x}_t + \mathbf{U}_f\mathbf{h}_{t-1} + \mathbf{b}_f) \\
\mathbf{o}_t &= \sigma_o(\mathbf{W}_o\mathbf{x}_t + \mathbf{U}_o\mathbf{h}_{t-1} + \mathbf{b}_o) \\
\mathbf{c}'_t &= tanh(\mathbf{W}_c\mathbf{x}_t + \mathbf{U}_c\mathbf{h}_{t-1} + \mathbf{b}_c) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{c}'_t \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t)
\end{aligned} \tag{1.14}$$

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 o 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. $\mathbf{b} \in \mathbb{R}^h$

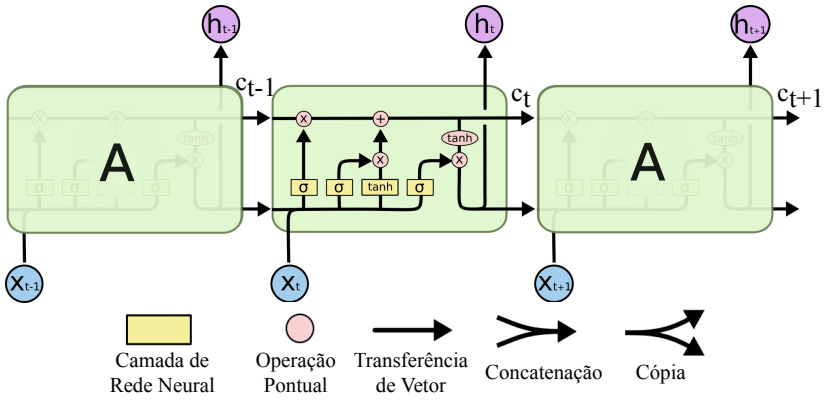


Figura 1.9 – Célula LSTM desenrolada no tempo. O estado de célula $\mathbf{c}_t$ e estado oculto $\mathbf{h}_t$ são usados para realimentar a célula a cada etapa de tempo. O primeiro é calculado em um ciclo interno enquanto o segundo é obtido por recorrência externa à LSTM. Adaptado de [23].

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 LSTM Convolucional Bidimensional (Conv2DLSTM) [24, 2] 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 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

$$\begin{aligned}
\mathbf{i}_t &= \sigma_i(\mathbf{W}_i * \mathbf{x}_t + \mathbf{U}_i * \mathbf{h}_{t-1} + \mathbf{b}_i) \\
\mathbf{f}_t &= \sigma_g(\mathbf{W}_f * \mathbf{x}_t + \mathbf{U}_f * \mathbf{h}_{t-1} + \mathbf{b}_f) \\
\mathbf{o}_t &= \sigma_o(\mathbf{W}_o * \mathbf{x}_t + \mathbf{U}_o * \mathbf{h}_{t-1} + \mathbf{b}_o) \\
\mathbf{c}'_t &= tanh(\mathbf{W}_c * \mathbf{x}_t + \mathbf{U}_c * \mathbf{h}_{t-1} + \mathbf{b}_c) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{c}'_t \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t)
\end{aligned} \tag{1.15}$$

1.1.4 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 de volta para uma saída x' com as mesmas dimensões da entrada original, por meio de um módulo de decodificador D [6]. Esse processo é resumido por:

$$\begin{aligned}
z &= E(x) \\
x' &= D(z)
\end{aligned} \tag{1.16}$$

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

$$L = d(x, D(E(x))) \tag{1.17}$$

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 [6].

Um AC é dito incompleto quando a dimensão de saída do codificador é menor que a dimensão de entrada [12]. 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 dos dados de entrada. Aprender uma

representação incompleta força o *autoencoder* a capturar os recursos mais destacados dos dados de treinamento [12].

1.1.4.1 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 [12]:

$$L = d(x, D(E(x))) + \Omega(z) \tag{1.18}$$

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 [25]. 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:

$$\hat{\rho}_j = \frac{1}{M} \sum_{i=1}^{M} z_j \tag{1.19}$$

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

$$\hat{\rho}_j = \rho \tag{1.20}$$

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) [26] entre duas variáveis aleatórias de Bernoulli com médias ρ e $\hat{\rho}_j$: [25].

$$D_{KL}(\rho||\hat{\rho}_j) = \sum_{j=1}^{h} \rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_j} \tag{1.21}$$

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 [26]. Então podemos reescrever a Equação 1.18 como:

$$L = d(x, D(E(x))) + \lambda \sum_{j=1}^{s_2} D_{KL}(\rho||\hat{\rho}_j) \tag{1.22}$$

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 [6].

1.1.4.2 Autocodificador Variacional

O autocodificador variacional (*variational autocoders*) (VAE) [27, 28] é um modelo generativo profundo capaz de aprender representações latentes não supervisionadas de dados [29]. 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 [6].

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 $Enc_\phi : X \rightarrow Z$ e o decodificador $Dec_\theta : Z \rightarrow X$, 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) [30]. O objetivo idealizado do VAE é obter a função de verossimilhança logarítmica (*log-likelihood*) marginalizada:

$$\sum_{i=1}^{N} \log(p(x^i)) \tag{1.23}$$

Entretanto, esse objetivo não é tratável e é aproximado pelo limite inferior da evidência (ELBO) [27]. 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:

$$\mathcal{L}(q) = \mathbb{E}_{z\sim q(z|x^i)} \log p(x_i|z) - D_{KL}(q(z|x^i)||p(z)) \leq \log p(x^i) \tag{1.24}$$

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:

$$Enc_\theta(x) \sim q_\phi(z|x) = \mathcal{N}(\mu_\phi(x), diag\ \ \sigma^2_\phi(x)) \tag{1.25}$$

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 [31]. Além disso, todas as expectativas na Equação 1.24 podem ser aproximadas pela amostragem de Monte Carlo [12]. Uma vez concluído o treinamento, a distribuição a *posterior* aproximado $q_\phi(z|x)$ funciona como um codificador.

1.2 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 ocupado por dados que representam uma dada informação. Os dados são os meios pelos quais as informações são transmitidas [32]. 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:

$$\begin{aligned} C &= \frac{b}{b'} \\ Rd &= 1 - \frac{1}{C} \end{aligned} \tag{1.26}$$

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 1.10.

Esses dois algoritmos formam o CODEC. Ele pode ser projetado para tratar com catego-

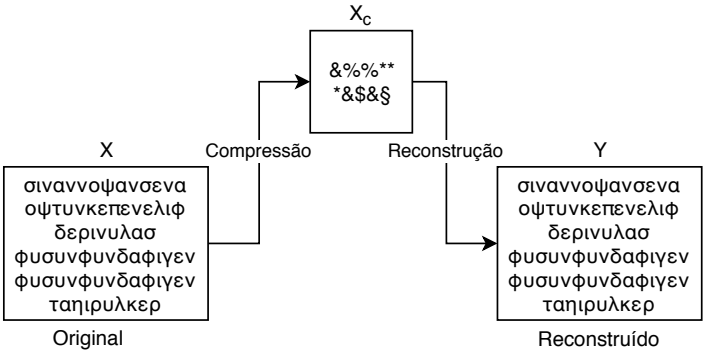


Figura 1.10 – O algoritmo de compressão (codificador) recebe uma entrada X e gera uma representação X_c que requer menos bits. O algoritmo de reconstrução (decodificador) opera na representação compactada X_c para gerar a reconstrução Y. Adaptado de [33].

rias 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 os dados recuperados são exatamente iguais aos originais [33].

1.2.1 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 [34]:

$$I(E) = log_b \frac{1}{P(E)} = -\log(P(E)) \tag{1.27}$$

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:

$$\bigcup A_i = S \tag{1.28}$$

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

$$H(x) = -\sum P(A_i) \log_b(P(A_i)) \tag{1.29}$$

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 [33]. 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 [33]. 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.

1.3 IMAGENS

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 [32]. 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 [35].

1.3.1 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 [34]. 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 [35]. O objetivo dos modelos de cores é permitir a especificação de cores em um formato padronizado e aceito por todos [32].

1.3.1.1 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 [35]. 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 [32].

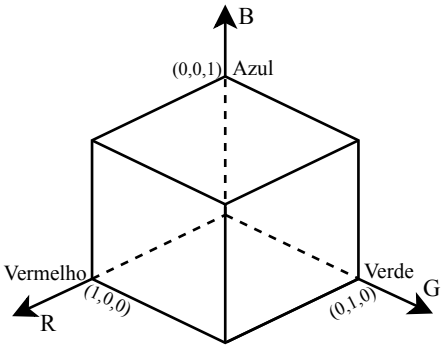


Figura 1.11 – Espaço de cores RGB representado em um cubo. Adaptado de [32].

1.3.1.2 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 [35]. O componente de luminância é calculado como uma média R, G e B ponderada pelos pesos k_r, k_g e k_b, respectivamente.

$$Y = k_r R + k_g G + k_b B \tag{1.30}$$

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

$$\begin{aligned} Cr &= R - Y \\ Cb &= B - Y \\ Cg &= G - Y \end{aligned} \tag{1.31}$$

1.3.2 Compressão de Imagens

No contexto da compressão digital de imagens os três principais tipos de redundância de dados que podem ser identificados e explorados são [32]:

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 neces-

sário 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.

1.3.2.1 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 [33]. 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 [33].

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) [36].

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 [36]. A DCT-2D para uma imagem $m \times n$ é calculada por:

$$
\begin{aligned}
G_{ij} &= \sqrt{\frac{2}{m}}\sqrt{\frac{2}{n}} C_i C_j \sum_{x=0}^{n-1}\sum_{y=0}^{m-1} p_{xy} cos\left[\frac{(2y+1)j\pi}{2m}\right] cos\left[\frac{(2x+1)i\pi}{2n}\right] \\
C_f &= \begin{cases} \frac{1}{\sqrt{2}} & \text{if } f = 0 \\ 1 & \text{if } f > 0 \end{cases}
\end{aligned}
\tag{1.32}
$$

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 da quantização [33]. Cada valor quantizado é representado por um rótulo I_{ij} e obtido a partir do respectivos coeficiente G_ij e do elemento da matriz de quantização Q_{ij} conforme descrito pela próxima equação.

$$I_{ij} = \left\lfloor \frac{G_{ij}}{Q_{ij}} + 0.5 \right\rfloor \tag{1.33}$$

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 [36].

O esquema decodificação é o inverso dos passos da codificação [33]. Dessa forma, a partir do conjunto de bits efetuaremos a decodificação de entropia, reescalonamento ou "quantização inversa",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 1.12 resume o esquemático de codificação e decodificação de uma imagem usando o padrão JPEG.

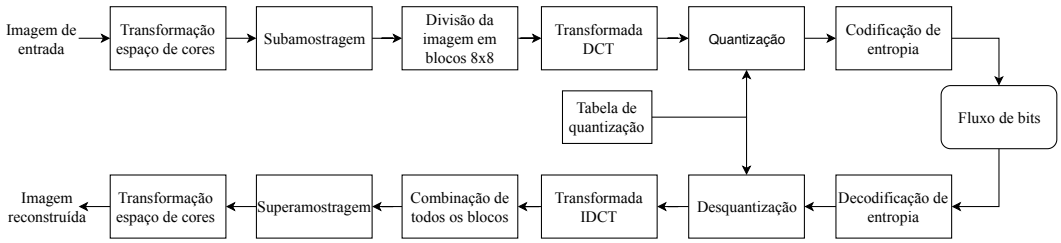


Figura 1.12 – Etapas da codificação e decodificação de uma imagem através do JPEG.

1.3.3 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, é influenciada por muitos fatores subjetivos que dificultam a obtenção de uma medida de qualidade completamente precisa [35]. 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) [35]. As duas últimas são métricas que levam em consideração a percepção humana.

1.3.3.1 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:

$$MSE(x, x') = \frac{1}{nm} \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} (x(i,j) - x'(i,j))^2 \tag{1.34}$$

$$PSNR_{dB}(x, x') = 10 \log \frac{(2^P - 1)^2}{MSE(x, x')} \tag{1.35}$$

1.3.3.2 SSIM

O SSIM foi projetado para modelar qualquer distorção da imagem como uma combinação da perda de correlação, distorção da luminância e do contraste [37]. Ele é definido por:

$$SSIM(x, x') = l(x, x') \times c(x, x') \times s(x, x') \tag{1.36}$$

Onde

$$\begin{aligned} l(x, x') &= \frac{2\mu_x \mu_{x'} + C_1}{\mu_x^2 + \mu_{x'}^2 + C_1} \\ c(x, x') &= \frac{2\sigma_x \sigma_{x'} + C_2}{\sigma_x^2 + \sigma_{x'}^2 + C_2} \\ s(x, x') &= \frac{\sigma_{xx'} + C_3}{\sigma_x \sigma_{x'} + C_3} \end{aligned} \tag{1.37}$$

O primeiro termo em 1.37 é a função que mede a proximidade da luminância média de duas imagens [37]. Os valores de μ_x e $\mu_{x'}$ são as médias das imagens x e x' que podem ser vistas como estimativas das luminâncias [38].

O segundo é a função de comparação de contraste entre duas imagens [37]. Aqui, o contraste é aproximado pelo desvio padrão σ_x e $\sigma_{x'}$. O terceiro termo é a função de compa-

ração de estrutura que mede o coeficiente de correlação entre as duas imagens x e x' [37]. 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, $x = x'$.

1.3.3.3 MS-SSIM

O MS-SSIM [38] é uma variação do SSIM para várias escalas de imagem e conveniente para incorporar detalhes das imagens 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 [38]. 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).

2 REVISÃO BIBLIOGRÁFICA

Neste capítulo, apresentamos uma revisão bibliográfica de *codecs* de imagem desenvolvidos com técnicas de redes neurais artificiais. O autocodificador é o tipo de RNA presente em todos os trabalhos. As diferenças estão na arquitetura dos autocodificadores, na modelagem do problema, emprego de análises probabilísticas, generativas, de segmentação semântica etc.

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 [39].

Devido a isso, os *codecs* tradicionais aplicam transformações lineares para gerar uma representação adequada sobre a qual podemos considerar a otimização conjunta de distorção e taxa [3]. Por exemplo, o JPEG realiza a DCT e a decomposição ortogonal em várias escalas de *Wavelets* é usada pelo JPEG2000.

Entretanto, não há razão para esperar que uma função linear seja ideal para comprimir todo o espectro de imagens naturais e levando em consideração os formatos de mídia emergentes [40]. Uma alternativa é substituir as transformações lineares por redes neurais artificiais a fim de obter um *codec* que generaliza melhor sobre as imagens.

Há 3 principais abordagens para incluir RNAs na tarefa de compressão de imagens. A primeira é o desenvolvimento completo do par codificador-decodificador. A segunda consiste na implementação de redes neurais dos algoritmos de compactação de imagem tradicionais. A terceira abordagem é usar RNA para substituir, aperfeiçoar ou adicionar alguma etapa na compressão de imagens realizadas por codecs convencionais. Aqui o objetivo dela é fornecer melhorias adicionais a esses algoritmos [41]. Hoje, a primeira abordagem é a predominantes e vem obtendo os melhores resultados.

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 (sigla do inglês *Better Portable Graphics* e considerado o estado da arte de compressão de imagens) em alguns trabalhos [**??**42].

A natureza da compressão com perdas pressupõe a utilização de quantização dos dados. Tal operação não é diferenciável e impõe uma restrição ao uso de RNA para a compressão. Outra preocupação é fornecer ao codec flexibilidade similar aos codecs tradicionais de imagens. Em geral, deseja-se que a rede seja capaz de fornecer reconstruções progressivas, controlar a qualidade e a taxa, codificar e decodificar em tempo similar os codecs convencionais etc.

2.1 AUTOCODIFCADORES EM CADEIA

Em [1], Toderici et. al propuseram arquiteturas ou modelos de redes neurais voltados para compressão imagem 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 em termos de programação e, 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. Logo, aqui temos um cascateamento em laço onde os pesos do autocodificador são compartilhados nas iterações. O número de autocodificadores (implícitos ou explícitos) define o número de iterações do nosso modelo.

Durante a execução (treinamento ou teste) de qualquer uma das arquiteturas apresentadas em [1] 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 a partir daqui realiza o mesmo procedimento que a primeira iteração, entretanto o resultado gerado pelo D_2 (e das próximas iterações) pode ser a reconstrução da informação residual (reconstrução aditiva) ou a reconstrução de uma nova versão da imagem original (reconstrução única), como veremos. Após as k iterações, passamos a processar a próxima imagem e o ciclo se repete.

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

$$F_t(r_{t-1}) = D_t(B(E_t(r_{t-1}))) \tag{2.1}$$

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 treinado 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.

2.1.1 Binarização

O trabalho [1] 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.

$$\begin{aligned} b(x) &= x + \epsilon \\ b(x) &\in \{-1, 1\} \end{aligned} \tag{2.2}$$

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

$$\epsilon \sim \begin{cases} 1 - x, \text{com probabilidade } \frac{1+x}{2} \\ -x - 1, \text{com probabilidade} \frac{1-x}{2} \end{cases} \tag{2.3}$$

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 2.3 como:

$$\epsilon = \begin{cases} 1 - x, \text{ se } u \leq \frac{1+x}{2} \\ -x - 1, \text{ se } u > \frac{1+x}{2} \end{cases} \tag{2.4}$$

A função de binarização B é dada por:

$$B(x) = b(\tanh W^{bin} + b^{bin}) \tag{2.5}$$

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:

$$\frac{\partial \mathbf{E}[b(x)]}{\partial x} = \frac{\partial x}{\partial x} = 1, \ \forall x \in [-1, 1] \tag{2.6}$$

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 $b(x)$:

$$b^{inf}(x) = \begin{cases} -1, \text{se } x < 0, \\ +1, \text{caso contrário.} \end{cases} \tag{2.7}$$

2.1.2 Redes não-recorrentes

Seguindo o trabalho [1], quando o modelo é composto apenas por camadas não-recorrentes o encadeamento de autocodificadores é explícito e F_t sempre fará uma estimativa do resíduo. Logo:

$$r'_t = F_t(r_t)//r_{t+1} \quad = r'_t - r_t \tag{2.8}$$

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.

$$x' = \sum_{t=0}^{k-1} r'_t \tag{2.9}$$

A Figura 2.1 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.

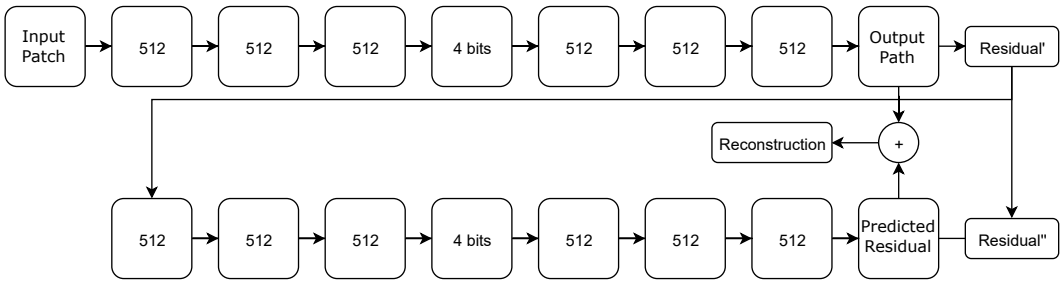


Figura 2.1 – Um autoencoder residual totalmente conectado com função de ativação $\tanh$. Esta figura representa uma arquitetura de duas iterações. As primeiras iterações codificam a imagem original. Os resíduos da reconstrução são passados para a segunda iteração. Cada nível de iteração produz 4 bits [1].

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. Consulte [1] para obter a formulação matemática da convolução transposta. Pela própria definição da Conv2D a taxa gerada na binarização é independente do tamanho da imagem de entrada. Os autores definiram essa taxa a 2 bits por pixel por iteração. A figura 2.2 mostra essa arquitetura (exceto a conversão RGB).

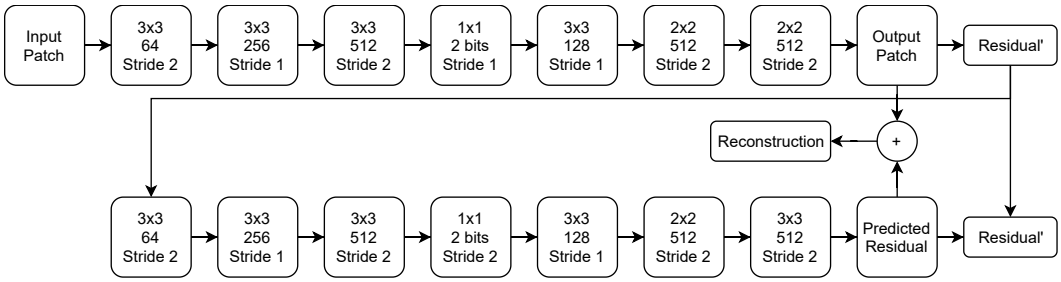


Figura 2.2 – O codificador residual convolucional/deconvolucional. As camadas convolucionais são representadas como retângulos pontiagudos, enquanto as camadas deconvolucionais são representadas como retângulos arredondados. [1].

2.2 REDES RECORRENTES

Em autocodificadores recorrentes há a noção de memória entre os estágios de tempo. Essa característica permite a função F_t 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 2.8. Nessas abordagens o encadeamento de autocodificadores é implícito (uso recursivo do autocodificador). A motivação de usar camadas recorrentes foram utilizados para capturar as dependências entre os resíduos gerados em cada iteração.

Ainda no trabalho [1], 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 1.14.

A Figura 2.3 em mostra um desenrolar do LSTM (menos a conversão RGB), necessária para treinamento, em duas etapas de tempo. As conexões verticais entre os estágios LSTM no desenrolar 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.

Nesse trabalho, os autores usaram um banco de dados de referência com 216 milhões de imagens coloridas e coletadas aleatoriamente da internet. Essas imagem foram reduzidas para blocos de 32×32 que elimina artefatos de compactação preexistentes para a maioria das imagens [1]. Para o treinamento dos modelos LSTM, 90% das imagens foram utilizadas; os restantes 10% foram reservados para avaliação. A função de custo utilizada foi a métrica L_2 nos resíduos.

Ela é definida substituindo $p = 2$ na Equação a seguir [43]:

$$||v||_p = \left(\sum_{i=1}^{n} |v_i|^p \right)^{\frac{1}{p}} \tag{2.10}$$

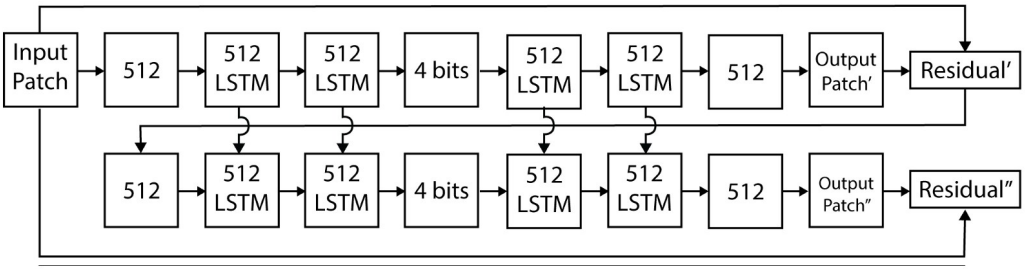


Figura 2.3 – O codificador residual LSTM totalmente conectado. Esta figura mostra um desenrolamento do LSTM, necessário para o treinamento, em duas etapas. Ao todo, a rede foi treinada com 16 níveis de resíduo, para gerar representações de 64 bits. As conexões verticais entre os estágios LSTM no desenrolamento mostram o efeito da memória persistente. Nessa arquitetura LSTM, cada etapa prevê a saída real. [1].

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

Em [2] foi proposto um modelo convolucional-recorrente como resultado de melhorias do trabalho em [1]. 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 2.11.

$$\begin{aligned} r_0 &= x \\ b_t &= B(E_t(r_{t-1})) \\ \hat{x}_t &= D_t(b_t) + \gamma \hat{x}_{t-1} \\ r_t &= x - \hat{x}_t \end{aligned} \tag{2.11}$$

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 convolucionais bidimensionais (Conv2D); hibridas do tipo convolucional-recorrente (ConvRNN); e camadas estáticas de binarização e profundidade para espaço (PE). A figura 2.4 apresenta um fluxograma da arquitetura para uma iteração.

As camada ConvRNN seguem formulação matemática dependente do tipo de rede recorrente. Em um dos modelos, os autores empregam a Conv2DLSTM descrita pela Equação 1.15. Também foram testadas a LSTM associativa e a Unidades Recorrentes Fechadas do inglês *Gated Recurrent Units* (GRU). Independentemente da camada recorrente dentre as 3 citadas, as RNNConv 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:

$$\begin{aligned} dim(X) &= (N, L, H_{in}, W_{in}) \\ L &= C \times \text{uf}^2 \\ Y &= pixel\_shuffle(X, \text{uf}) \\ dim(Y) &= (N, C, H_{in} \times \text{uf}, W_{in} \times \text{uf}) \end{aligned} \tag{2.12}$$

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.

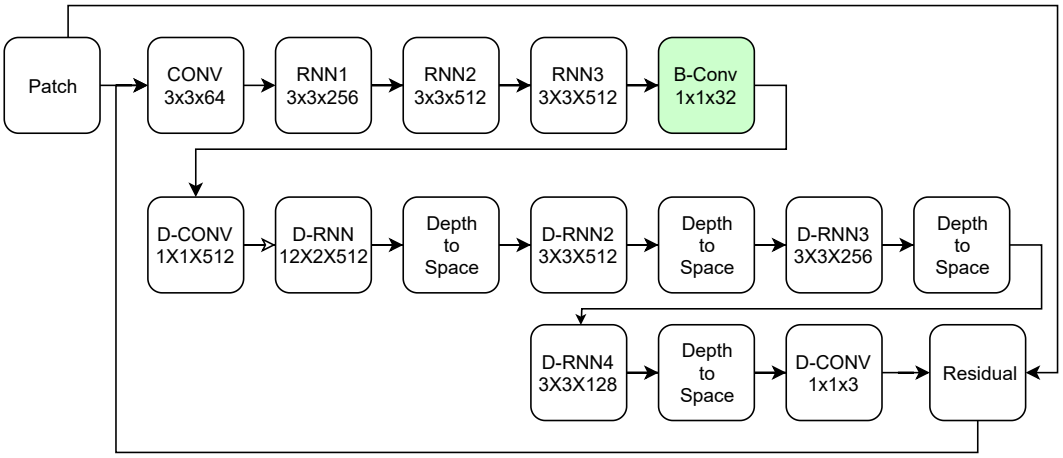


Figura 2.4 – The flow of the iterations in an RNN architecture. In this case, the prefix $D-$ refers to the "inverse" operation in the decoder. Also, the depth to space is a shuffle of the pixels as defined in the sub-pixel operations [2].

Durante o treinamento dos modelos em [2] 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. Para treinar o modelo foram usados dois conjuntos de imagens de $32 \times 32 \times 3$. O primeiro corresponde a base de dados usados em [1]. Para obter o segundo conjunto de dados, primeiro reuniu-se uma amostra aleatória de 6 milhões de imagens de 1280×720 da *Web* que foram decompostas em blocos em $32 \times 32 \times 3$ sem sobreposição e codificadas com PNG. Então os 100 blocos de mais alta entropia por imagem, isto é, com a pior taxa de compactação, foram novamente amostrados e reunidos no segundo conjunto de dados. A partir daqui diversas melhorias foram propostas na tentativa de contornar limitações impostas pelo modelo.

Ainda, em [2] 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. O fluxo da informção usando essa rede de ganho é mostrado na Figura **??**.

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 2.4 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 nosso 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 codecs 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 [45].

No trabalho apresentado por Minnen et. al. em [46] 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.

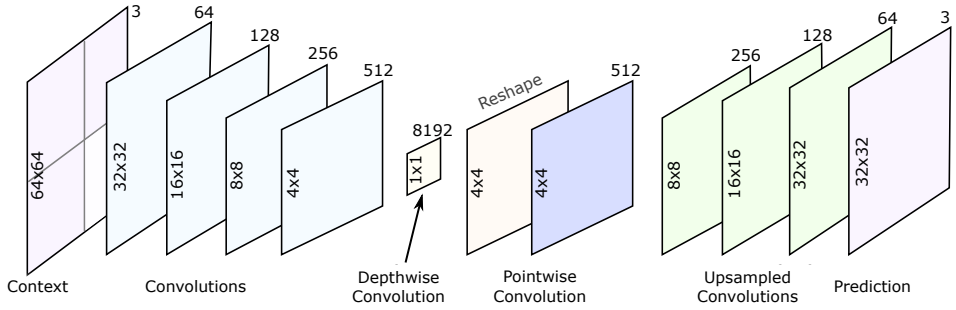


Figura 2.5 – A rede DPCM é um codificador automático que recebe uma imagem de 64×64, com zeros no bloco atual e gera esse bloco ausente[46].

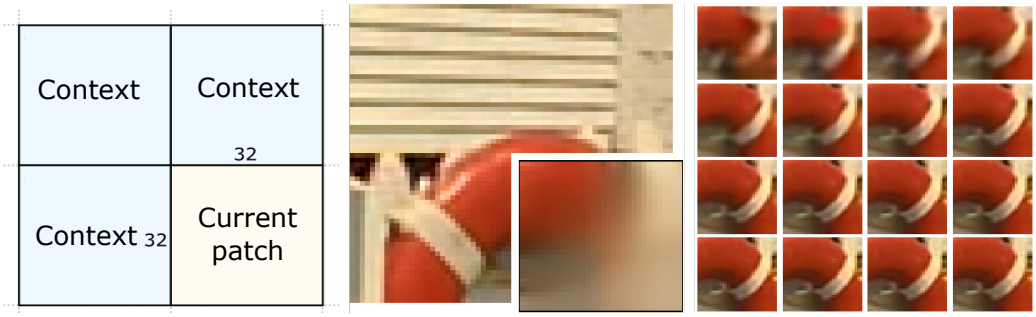


Figura 2.6 – Para cada 32×32, o modelo DPCM tenta prever com base nos blocos vizinhos. Os resíduos são propagados para o autocodificador [46].

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 2.5, 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 2.6 ilustra o processo de previsão descrito até aqui.

A segunda rede, irá complementar a tarefa da primeira ao reconstruir a informação residual. Ela corresponde a rede 2.4 com reconstrução aditiva e usando unidades Con2DLSTM.

No artigo [47], Johnston et al. implementaram 3 modificações na forma de treinar o modelo que o fez superar os codecs 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 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 [47].

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. Por exemplo, para representar partes da imagens de baixa frequência (por exemplo o céu sem nuvens) são necessários menos bits que informações de alta frequência (por exemplos confetes).

A solução proposta pelos autores em [47] 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 2.13.

$$\begin{aligned} L(x, x') &= w(x, x') \times L_1(x - x') \\ w(x, x') &= \frac{S(x, x')}{\bar{S}} \end{aligned} \tag{2.13}$$

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 2.1 apresenta as principais características dos autocodificadores em cadeia relatos nesse trabalho.

Tabela 2.1 – Tabela com informações sintetizadas do *codes* 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 codecs tradicionais
Toderici et. al. [1]	- 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. [2]	- 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 [1] 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. [46]	- 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 [2]	- 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. [47]	- 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)

2.3 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.

O autocodificador variacional foi empregado em [3] 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 2.7.

Eles usaram uma módulo de não-linearidade denominado normalização divisiva generalizada (GDN) que tem se mostrado eficaz na gaussianização de densidades de imagem.

A transformação de análise g_a consiste em três estágios de convolução, subamostragem e normalização divisiva. Analogamente, a transformada de síntese g_s consiste em três estágios, com a ordem das operações invertida dentro de cada estágio: IGDN (inverso aproximado da GDN), superamostragem e convolução.

Esta transformação é seguida por quantização escalar uniforme (isto é, cada elemento é arredondado para o número inteiro mais próximo), o que implementa efetivamente uma forma paramétrica de quantização vetorial no espaço da imagem original. A imagem comprimida é reconstruída a partir destes valores quantificados usando uma transformação inversa não linear paramétrica aproximada.

Para qualquer ponto desejado ao longo da curva de taxa-distorção, os parâmetros de ambas as transformações de análise e síntese são otimizados em conjunto usando a descida de gradiente estocástica. Para conseguir isso na presença de quantização (que produz zero gradientes em quase todos os lugares), usamos uma função de perda de proxy baseada em um relaxamento contínuo do modelo de probabilidade, substituindo a etapa de quantização por ruído uniforme aditivo. Aqui, definimos a transformação perceptual g_p para a identidade e usamos o erro quadrático médio (MSE) como a métrica (ou seja, d (z, z^) = kz - z^k22).

A Figura 2.7, extraída desse trabalho, mostra a arquitetural geral. Para esse fim, a seguinte função objetivo é usada:

Em vez de tentar uma quantização ótima diretamente no espaço da imagem, que é intratável devido à alta dimensionalidade, assumimos um quantizador escalar fixo uniforme no espaço de código, e pretendemos que as transformações não-lineares distorcem o espaço de maneira apropriada, implementando efetivamente uma forma paramétrica de quantização vetorial.

As taxas reais obtidas por um código de entropia apropriadamente projetado são apenas ligeiramente maiores que a entropia (Rissanen e Langdon, 1981), e assim definimos o objetivo funcional diretamente em termos de entropia:

$$L[g_a, g_s, P_q] = -\mathbb{E}[\log_2 P_q] + \lambda \mathbb{E}[d(z, \hat{z})] \tag{2.14}$$

Portanto, o primeiro termo corresponde à entropia do código, modelado por P_q e o segundo termo corresponde à perda de reconstrução. Ambas as expectativas serão aproximadas por médias ao longo de um conjunto de imagens.

onde ambas as expectativas serão aproximadas por médias de um conjunto de imagens de treinamento. Dado um conjunto suficientemente poderoso de transformações, podemos supor sem perda de generalidade que o tamanho do escaninho de quantização é sempre um e os valores representativos estão no centro dos escaninhos. Isso é,

$$\hat{y}_i = q_i = round(y_i) \tag{2.15}$$

Aqui, fazemos uso de uma transformação de normalização divisiva generalizada (GDN) com parâmetros otimizados, que anteriormente mostramos ser altamente eficientes na Gaussianização de estatísticas de junções locais de imagens naturais, muito mais do que cascatas de transformações lineares seguidas por não linearidades pontuais (Ballé, Laparra e Simoncelli, 2015).

Note que alguns algoritmos de treinamento para redes convolucionais profundas incorporam a "normalização de lotes", reescalonando as respostas dos filtros lineares na rede, de modo a mantê-lo em uma faixa operacional razoável (Ioffe e Szegedy, 2015). Esse tipo de normalização é diferente do controle de ganho local, pois o fator de redimensionamento é idêntico em todos os locais espaciais. Além disso, quando o treinamento é concluído, os parâmetros de escala são tipicamente fixos, o que transforma a normalização em uma transformação afim em relação aos dados - ao contrário da GDN, que é espacialmente adaptável e pode ser altamente não-linear.

Finalmente, em vez de relatar estimativas de entropia diferencial ou discreta, implementamos um código de entropia e relatamos o desempenho usando taxas de bits reais, demonstrando assim a viabilidade de nossa solução como um método completo de compactação com perdas.

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 $P(z)$ e é feita uma amostragem.

Portanto, o codificador e o decodificador são as funções usadas para induzir a distribuição

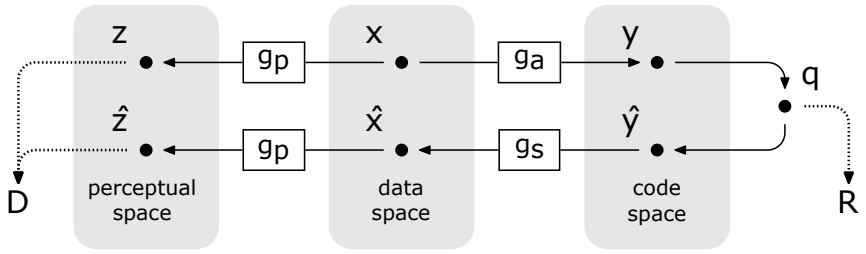


Figura 2.7 – O vetor de imagem $x \in \mathbb{R}^N$ é mapeado para o espaço do código com $y = g_a(x, \phi)$. y é quantizado, produzindo um vetor $q \in \mathbb{Z}^M$. A taxa R é limitada pela entropia do setor quantizado, $H[P_q]$. Em seguida, os elementos discretos q são interpretados como $\hat{y}$ de valor contínuo, que é transformado em espaço de dados com $\hat{x} = g_s(\hat{y}, \theta)$. A comparação da reconstrução é feita em um espaço desejado mapeado por g_p. Os parâmetros ϕ e θ são otimizados para minimizar a distorção da taxa $R + \lambda D$ [3].

e mapear os gaussianos fatorados para a distribuição real, reconstruindo a amostra.

Embora os VAE sejam usados no paradigma generativo e aparentemente não tenham nada a ver com codificação, essas arquiteturas podem ser usadas para fins de codificação. Na tentativa de modelar diretamente a distorção da taxa, $R + \lambda D$, VAE são usados em [3]. A Figura 2.7, extraída desse trabalho, mostra a arquitetura geral. Para esse fim, a seguinte função objetivo é usada:

$$L[g_a, g_s, P_q] = -\mathbb{E}[\log_2 P_q] + \lambda \mathbb{E}[d(z, \hat{z})] \tag{2.16}$$

Assim, o primeiro termo corresponde à entropia do código, modelado por P_q e o segundo termo corresponde à perda de reconstrução. Ambas as expectativas serão aproximadas por médias sobre um conjunto de imagens. A quantização tem tamanho de caixa de um:

$$\hat{y}_i = q_i = round(y_i) \tag{2.17}$$

e a densidade marginal de $\hat{y}_i$ é dada por um trem de massas de probabilidade discretas com pesos iguais à função de massa de probabilidade de q_i:

$$P_{q_i}(n) = \int_{n-\frac{1}{2}}^{n+\frac{1}{2}} p_{y_i}(t)dt, \forall n \in \mathbb{Z} \tag{2.18}$$

Um problema apontado pelos autores é que, uma vez que essa modelagem depende de valores quantizados, os derivados são zero em quase todos os lugares. Então eles substituem o quantizer com ruído uniforme i.i.d aditivo, $\tilde{y} = y + \Delta y$, que é um relaxamento contínuo do pmf de q:

$$p_{\tilde{y}}(n) = P_q(n), \forall n \in \mathbb{Z}^M \tag{2.19}$$

Ele permite que a entropia diferencial de $\tilde{y}$ seja usada como uma aproximação da entropia de q. Para modelar a probabilidade de $\tilde{y}$ os autores usam funções lineares em sentido de sentido sinuoso. Assim, a perda específica a ser otimizada pode ser colocada da seguinte forma [3]:

$$\begin{aligned} L(\theta, \phi) = \mathbb{E}_{x,\Delta y}[-\sum_i \log_2 p_{\tilde{y}_i}(g_a(x;\phi) + \Delta y; \psi^{(i)}) \\ +\lambda d(g_p(g_s(g_a(x;\phi) + \Delta y; \theta)), g_p(x))] \end{aligned} \tag{2.20}$$

Esta formulação, derivada do problema de distorção de taxa, assemelha-se ao contexto de autoencodificadores variacionais, uma vez que uma função de perda contínua é usada. Como explicado pelos autores, em inferência variacional bayesiana, dadas as observações de variáveis x juntamente com $p_{xy}(x||y)$ um posterior $p_{y|x}(y|x)$ é perseguido, que não pode ser expresso na forma fechada. Usando a divergência Kullback-Leibler

$$\begin{aligned} D_{KL}[q||q_{y|x}] &= \mathbb{E}_{y\sim q} \log q(y|x) - \mathbb{E}_{y\sim q} \log p_{y|x}(y|x) \\ &= \mathbb{E}_{y\sim q} \log q(y|x) - \mathbb{E}_{y\sim q} \log p_{x|y}(x|y) - \mathbb{E}_{y\sim q} \log p_y(y) + const. \end{aligned} \tag{2.21}$$

Os autores mostram que essa função objetiva é equivalente ao problema de otimização de distorção de taxa relaxado, com distorção medida como MSE, definindo o modelo da seguinte forma:

$$p_{x|\tilde{y}}(x|\tilde{y}; \lambda, \theta) = \mathcal{N}(x; g_s(\tilde{y}; \theta), (2\lambda)^{-1}1) \tag{2.22}$$

$$p_{\tilde{y}}(\tilde{y}; \psi^{(0)}, \psi^{(1)}, ...) = \prod_i p_{\tilde{y}_i}(\tilde{y}_i; \psi^{(i)}) \tag{2.23}$$

e o posterior como:

$$q(\tilde{y}|x; \phi) = \prod_i \mathcal{U}(\tilde{y}_i; y_i; 1) \text{ with } y = g_a(x; \phi) \tag{2.24}$$

onde $\mathcal{U}(\tilde{y}_i; y_i; 1)$ é a densidade uniforme no intervalo da unidade centrada em y_i. Com esta formulação, o primeiro termo em KL-divergência é constante, o segundo é a distorção eo terceiro é a taxa.

Para a configuração de treinamento, foram utilizadas 6507 imagens do ImageNet. Essa otimização foi realizada para cada λ e o codificador de entropia foi uma implementação inspirada no CABAC. A avaliação foi realizada no conjunto de dados da Kodak. Muitos outros trabalhos são baseados na formulação apresentada neste trabalho. Por exemplo, em

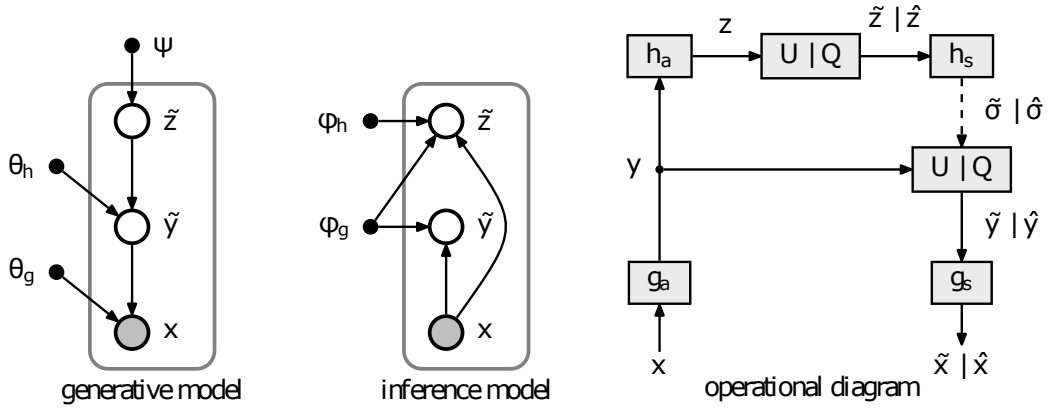


Figura 2.8 – The description of the model considering the hyperprior [4].

[4] o modelo acima é melhorado usando hiperpriors. A Figura reffig:balle2, apresentada nesse trabalho, mostra o novo modelo utilizado.

Neste trabalho, cada $\tilde{y}_i$ é modelado como um Gaussian zero-médio com um desvio padrão σ_i, obtido por um VAE auxiliar. Então:

$$p_{\tilde{y}|\tilde{z}}(\tilde{y}|\tilde{z}) = \prod_i (\mathcal{N}(0, \tilde{\sigma}_i^2) * \mathcal{U}(-\frac{1}{2}, \frac{1}{2}))(\tilde{y}_i) \tag{2.25}$$

com $\tilde{\sigma} = h_s(\tilde{z}; \theta_h)$. Este modelo é ampliado ainda mais por empilhamento h_a em cima de y criando um posterior variacional fatorado:

$$q(\tilde{y}, \tilde{z}|x, \phi_g, \phi_h) = \prod_i \mathcal{U}(\tilde{y}_i|y_i - \frac{1}{2}, y_i + \frac{1}{2}). \prod_j \mathcal{U}(\tilde{z}_j|z_i - \frac{1}{2}, z_i + \frac{1}{2}) \tag{2.26}$$

com $y = g_a(x; phi_g)$ e $z = h_a(y; phi_h)$. A modelagem de z, o hiperprior, usa o modelo totalmente fatorado usado no trabalho anterior por $\tilde{y}$:

$$p_{\tilde{z}|\psi}(\tilde{z}|\psi) = \prod_i (p_{\tilde{z}_i|\psi^{(i)}}(\psi^{(i)}) * \mathcal{U}(-\frac{1}{2}, \frac{1}{2}))(\tilde{z}_i) \tag{2.27}$$

Considerando esta formulação, a perda é definida por [4] como:

$$\begin{aligned} \mathbb{E}_{x\sim p_x} D_{KL}[q||p_{\tilde{y},\tilde{z}|x}] = \mathbb{E}_{x\sim p_x} \mathbb{E}_{\tilde{y},\tilde{z}\sim q}[\log q(\tilde{y}, \tilde{z}|x) - \log p_{x|\tilde{y}}(x|\tilde{y}) \\ - \log p_{\tilde{y}|\tilde{z}}(\tilde{y}|\tilde{z}) - \log p_{\tilde{z}}(\tilde{z})] + const. \end{aligned} \tag{2.28}$$

onde, novamente, o primeiro termo é zero, uma vez que q é um produto de densidades uniformes de largura da unidade. O segundo termo é a distorção, enquanto o terceiro e quarto mandato representam as entropias cruzadas de $\tilde{y}$ e $\tilde{z}$. O quarto termo pode ser visto como informação lateral.

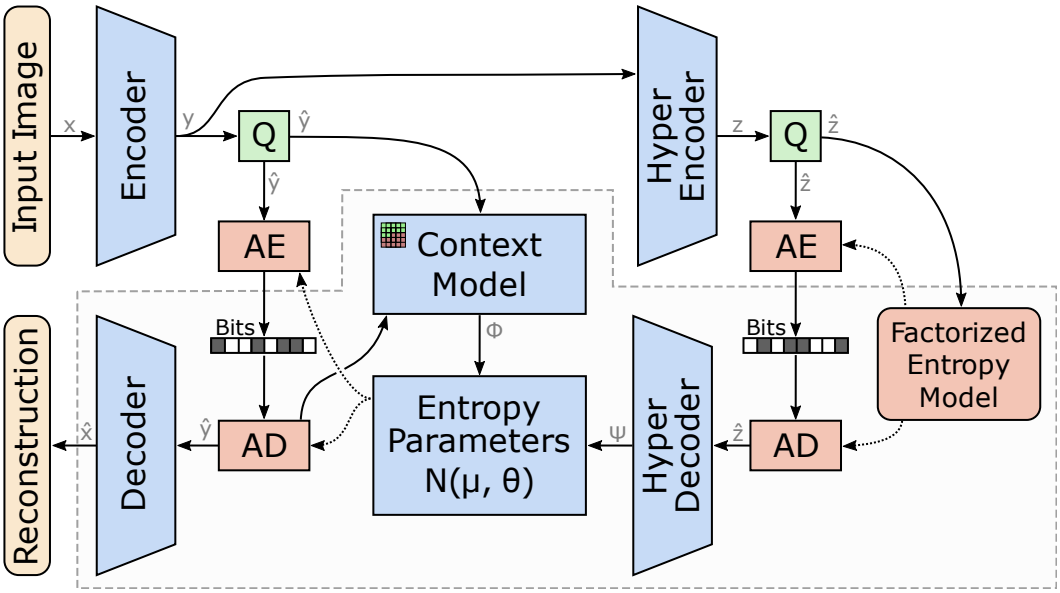


Figura 2.9 – The context model is the autoregressive component which predicts latents from context. There's also the hyperprior, seen in earlier proposal of this architecture. Both models feed a network of entropy parameters which outputs information about the distribution of the code model to the basic VAE [**??**].

A configuração de treinamento deste novo trabalho compreende um conjunto de 1 milhão de imagens JPEG da web, com tamanho 3000 x 5000. Estas imagens foram amostradas para 640 x 1200 pixels e 256 x 256 culturas das imagens foram extraídas. O lote é composto por 8 amostras e a taxa de aprendizagem é de 10^{-4}. Em [4] os autores mencionam que a normalização em lotes e alguns horários de taxa de aprendizagem não tiveram efeitos benéficos neste caso. Um total de 16 modelos com hiperpriors foram treinados.

Vale a pena mencionar, como com o trabalho anterior, que uma desvantagem dessa abordagem é a necessidade de treinar muitos modelos para diferentes distorções de taxa. Um problema não presente nas abordagens descritas nas sessões anteriores. A configuração do teste é, novamente, o conjunto de dados kodak, onde melhores resultados foram obtidos neste trabalho do que no anterior. Uma tabela que descreve estes resultados será exibida em uma sessão mais atrasada.

Outra melhoria no modelo apresentado em [4] é vista em [**??**]. Este trabalho traz um modelo de estrutura mais complexo usando um modelo autoregressivo através de redes neurais e, novamente, o hiperprior. Esses modelos são combinados para alcançar uma modelagem mais robusta do modelo de código. A Figura reffig:balle3, extraída deste trabalho, mostra toda a arquitetura.

Nesta proposta, o modelo para latente $\hat{y}_i$ é novamente definido como um gaussian envolvido com uma distribuição uniforme da unidade. Mas desta vez, o modelo é expandido considerando que os componentes de média e escala dos gaussians são previstos pelo mo-

delo:

$$
\begin{aligned}
p_{\hat{y}}(\hat{y}|\hat{z}, \theta_{hd}, \theta_{cm}, \theta_{ep}) &= \prod_i (\mathcal{N}(\mu_i, \sigma_i^2) * \mathcal{U}(-\frac{1}{2}, \frac{1}{2}))(\hat{y}_i) \\
\mu_i, \sigma_i^2 &= g_{ep}(\psi, \phi_i; \theta_{ep}) \\
\psi &= g_h(\hat{z}; \theta_{hd}) \\
\phi_i &= g_{cm}(\hat{y}_{<i}; \theta_{cm})
\end{aligned}
\tag{2.29}
$$

onde g_{ep} é a rede de parâmetros de entropia, g_h é a rede hiperprior, ϕ_i é a rede de modelos de contexto e $\hat{y}_{<i}$ é o contexto causal de cada latente $\hat{y}_i$. As versões chapéu aqui correspondem à quantização das variáveis, que, novamente, é relaxado no tempo de treinamento usando $\mathcal{U}$, o ruído uniforme aditivo. Como não há suposições sobre a distribuição dos hiper-latents, um modelo não paramétrico e totalmente fatorado é usado. A função de perda, neste caso, é semelhante à versão do trabalho anterior:

$$
R + \lambda D = \mathbb{E}_{x \sim p_x}[-\log_2 p_{\hat{y}}(\hat{y})] + \mathbb{E}_{x \sim p_x}[-\log_2 p_{\hat{z}}(\hat{z})] + \lambda \mathbb{E}_{x \sim p_x} ||x - \hat{x}||_2^2 \tag{2.30}
$$

onde o primeiro termo é a taxa dos latents, o segundo termo é a taxa dos hiper-latents e λ-term é a distorção. Este trabalho foi o primeiro a superar o BPG em métricas de distorção PSNR e MS-SSIM.

Algumas abordagens apresentadas na literatura são baseadas nessas obras apresentadas anteriormente. Por exemplo, [48] propõe uma estrutura de compressão de imagem com base em [3]. Os autores se concentram na eficiência computacional, uma vez que os trabalhos baseados em VAE exigem treinamento de muitos modelos, para cada λ, que é computacionalmente caro. Uma primeira diferença na arquitetura diz respeito à quantização. Diferente de [3], os autores não substituem totalmente a quantização por ruído uniforme. Em vez disso, apenas os gradientes são substituídos enquanto a quantização é realizada no tempo de treinamento.

Conforme formulado pelos autores, em vez dos gradientes mal definidos da quantização, o derivado é o de uma aproximação suave, r, que depende de y:

$$
\frac{d}{dy}[y] = \frac{d}{dy} r(y) \tag{2.31}
$$

Os autores afirmam que não há uma substituição totalmente com a aproximação suave, porque o decodificador pode aprender a inverter a aproximação suave. O resultado é que ele pode remover o gargalo de informações que força a rede a comprimir informações. Empiricamente, eles usam a identidade $r(y) = y$, que funciona bem como escolhas sofisticadas.

Desta forma, os gradientes podem ser propagados sem modificações.

Uma vez que a arquitetura é baseada em VAE, há também o problema de usar um modelo de entropia discreto Q que não tem gradientes. Assim, uma aproximação contínua e diferenciada q é usada, definindo um limite superior à entropia de Q:

$$\begin{aligned} Q(z) &= \int_{[-.5,.5[^M} q(z+u)du \\ -\log_2 Q(z) &= -\log_2 \int_{[-.5,.5[^M} q(z+u)du \leq \int_{[-.5,.5[^M} -\log_2 q(z+u)du \end{aligned} \tag{2.32}$$

com o segundo passo após a desigualdade de Jensen. A principal diferença é proposta para as taxas de bitvariável. A maneira padrão, até agora, é treinar modelos diferentes para curvas diferentes da taxa-distorção. Para reduzir os requisitos computacionais, os autores aperfeiçoam um autocodificador pré-treinado atraindo um $\lambda \in mathbbR^M$

$$-\log_2 q([f(x) \odot \lambda] + u) + \beta d(x, g(\frac{[f(x) \odot \lambda]}{\lambda})) \tag{2.33}$$

onde $\odot$ representa multiplicação pontiaguda. E para reduzir o número de escalas treináveis, elas também são compartilhadas entre as dimensões. Com este λ é possível controlar a quantidade de informação rejeitada na quantização. Por exemplo, se o valor 0.3583 aparecer em um latente, a operação redonda renderia 0. Em vez disso, se for multiplicado por US 10, a operação redonda devolveria US 3. Considerando uma escala de 100, o valor arredondado seria 35. No decodificador, uma vez que a operação de escala é reversível, há mais informações que passam do codificador para o decodificador.

A ideia de usar esses parâmetros de escala é implementada da seguinte forma: o autocodificador é treinado otimizando um β. Depois disso, este autocodificador é ajustado novamente com um pequeno número de iterações para cada um dos λ's. Além disso, para aumentar a densidade do λ's, uma interpolação é realizada nesses pontos.

Para modelar as distribuições dos coeficientes e estimar a taxa de bits, são utilizadas misturas de escala gaussiana:

$$\log_2 q(z+u) = \sum_{i,j,k} \log_2 \sum_s \pi_{ks} \mathcal{N}(z_{kij} + u_{kij}; 0, \sigma_{ks}^2) \tag{2.34}$$

Adam é o otimizador aplicado em lotes de 32 imagens de $128vezes128$ pixels. Esse treinamento é realizado usando uma estratégia diferente, que os autores chamam de treinamento "incremental". Isso é feito introduzindo uma máscara binária m:

$$-\log_2 q([f(x)] \odot m + u) + \beta d(x, g([f(x)] \odot m)) \tag{2.35}$$

Esta máscara começa com apenas duas entradas não-zero e esse valor é aumentado cada vez que um limite de desempenho é atingido pela rede. Depois que a máscara é totalmente habilitada, a taxa de aprendizagem é reduzida de 10^{-4} para 10^{-5}. O treinamento foi realizado com 10^6 iterações. A configuração de treinamento do λ fine-tune foi uma taxa de aprendizagem inicial de 10^{-3} aumentada por um fator de $\frac{.8^\kappa}{(1000+t)^\kappa}$.

As escalas foram otimizadas, como explicado anteriormente, para 10000 iterações. Os resultados alcançados superam o JPEG2000 em termos de SSIM.

2.4 AUTOCODIFICADORES BASEADO EM GAN

No artigo [42] 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 GANS 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 codecs nas métricas PSNR e MS-SSIM em uma ampla faixa de taxas de bits. Além disso, como o mapa de segmentação semântica está incluído no fluxo de bits, o esquema proposto pode facilitar muitas outras tarefas, como pesquisa de imagens e compactação de imagem adaptativa baseada em objetos [42].

Em [49], 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* [50]. 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.

$$L = d(x, x') + \lambda R(z) \tag{2.36}$$

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

(Método de Direção Alternativa dos Multiplicadores) [51].

No artigo [40] 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 [40]. Nesse trabalho, os resultados não supera os codecs 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 de entropia de comprimento variável tradicionais.

Em [53], Rippel et al. proporam um modelo de treinamento adversarial 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 GANs 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.

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 GANs 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 decodifi-

cador 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, $\mathcal{L}_{gan}$. Em relação à interação entre a perda GAN e a perda MSE, observa-se que a perda 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.

3 METODOLOGIA

Nesse seção apresentamos a Metodologia.

Usamos o modelo da rede convolucional-LSTM proposto por [2] cuja arquitetura foi apresentada em 2.4 com algumas modificações.

O método de reconstrução de uma imagem x será aditivo e a rede aprende a reconstruir a informação residual. Além disso, em um dado estágio ou iteração t, o autocodificador recebe a diferença entre o resíduo da iteração anterior r_{t-1} e a sua previsão r'_{t-1}. Essas duas características estão resumidas na próxima Equação, em um estágio:

$$\begin{aligned} r_0 &= x \\ x'_0 &= 0 \\ b_t &= B(E_t(r_{t-1})) \\ r'_{t-1} &= D_t(b_t) \\ r_t &= r_{t-1} - r'_{t-1} \\ x'_t &= r'_{t-1} + x'_{t-1} \end{aligned} \tag{3.1}$$

Usamos as camadas Conv2DLSTM descritas pela Equação 1.15 para capturar as dependências espaciais dos resíduos. A figura 3.1 detalha essa camada com k mapas de recursos.

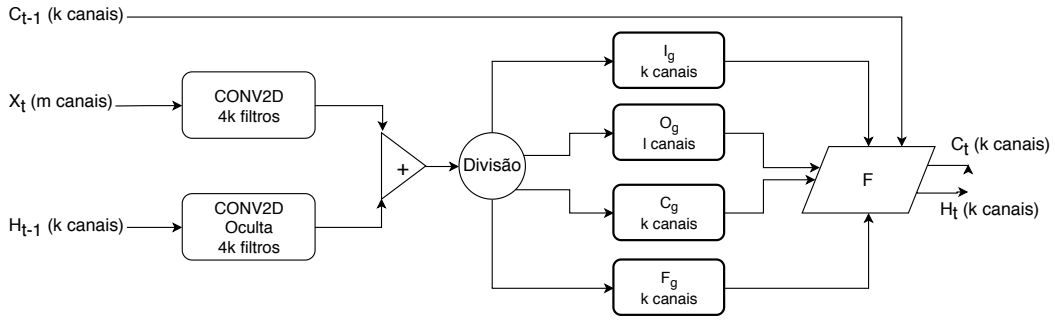


Figura 3.1 – Camada convolucional recorrente com k filtros. Sendo X_t o elemento atual da nossa sequencia com m canais que está sendo processado, H_{t-1} e c_{t-1} representam a saída e estado de célula gerados por essa unidade na etapa anterior. As saídas de ambas as convoluções tem o mesmo número de 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 as portas e em conjunto com C_{t-1} para fornecer $C_t e H_t$ segundo as equações em 1.15.

A Figura 3.2 mostra a rede "desenrolada" no tempo. Para simplificar a representação omitimos as camadas profundidade para espaço.

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

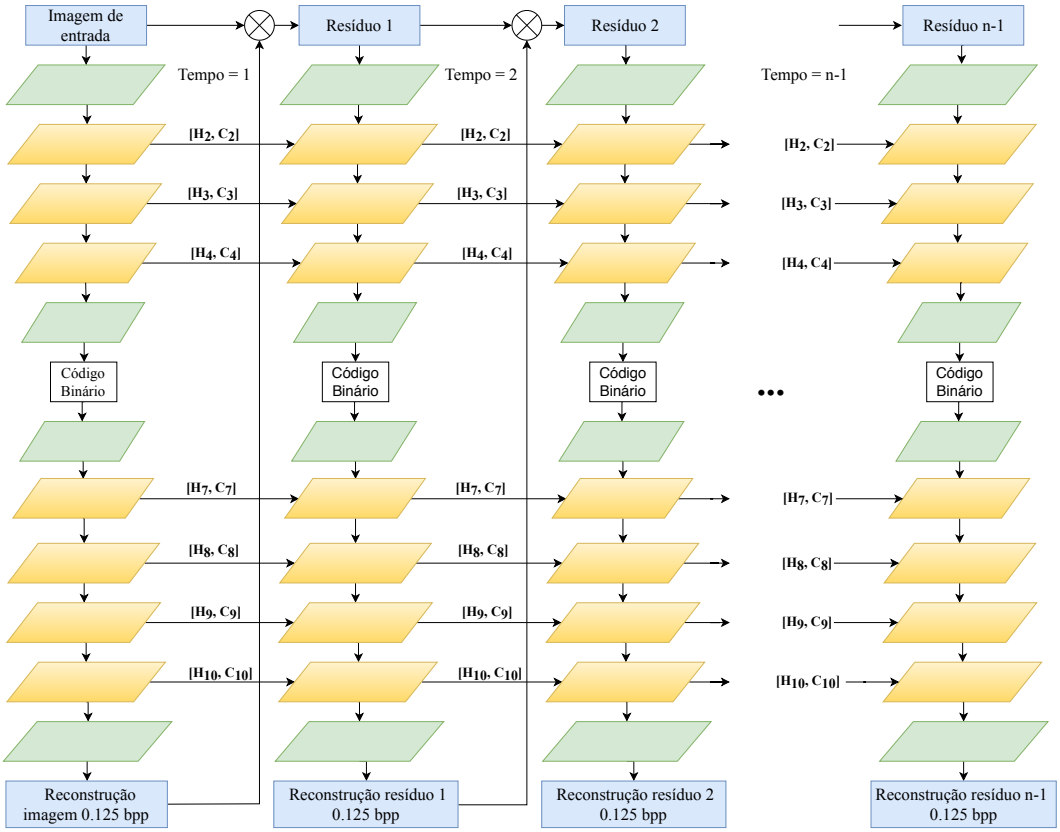


Figura 3.2 – Representação do autocodificador no decorrer de n estágios. Os losangos verdes e laranjas representam as camadas Conv2d e Conv2DLSTM, respectivamente. Na primeira iteração os valores de $C_t e H_t$ para cada Conv2DLSTM são zeros e as setas horizontais evidenciam o $loop$ dessas camadas. 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.

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:

$$R_{it} = \frac{\frac{H \times W}{8}}{H \times W} = \frac{1}{8} \text{ bpp} \tag{3.2}$$

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 1.26 obtemos a taxa compressão nominal C na iteração k:

$$C = \frac{24 \times H \times W}{k \times \frac{H \times W}{8}} = \frac{192}{k} \tag{3.3}$$

Os detalhes do autocodificador podem ser consultados na Tabela 3.1 considerada entradas formadas por blocos de imagens $32 \times 32 \times 3$. O tamanho do minilote está omitido dessa representação, contudo, ele ocupa mais uma dimensão de valor fixo durante todas as operações realizadas pela rede.

Tabela 3.1 – 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

O processo de binarização adotado foi o descrito na seção 2.1.1 e proposto em [1].

3.1 FORMATO DO FLUXO DE BITS

A arquitetura é otimizada para reconstruir blocos de $32 \times 32 \times 3$ uma vez que foi treinada com imagens dessa dimensão. Portanto, dividimos nossa imagem para gerar blocos 32×32. Vamos nos referir a cada um dos blocos por b_p em que $p \in \{0, ..., n-1\}$ em que n foi o número de blocos obtidos. Considerando um lote de tamanho l no qual vamos fazer k iterações na rede. Na primeira iteração da rede o codificador irá tomar os primeiros l blocos na ordem direita para esquerda e de cima para baixo e passar pela rede do codificador para gerar o código binário parcial de formato $l \times 2 \times 2 \times 32$ conforme a tabela 3.1. Em seguida, decodificamos esse lote e computamos o resíduo. 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 $2 \times l \times 2 \times 2 \times 32$. 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 temos repetir o processo descrito acima para codificar a imagem inteira é de $\frac{n}{l}$. 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.

3.2 BASE DE DADOS

Idealmente, desejamos obter uma base de dados de treinamento a fim de tornar o modelo capaz de comprimir satisfatoriamente (desempenho semelhante ao JPEG) 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. Intuitivamente, formaremos a base de dados de treinamento com imagens que não passaram por nenhum processo de compressão com perdas. Dessa forma, acreditamos ela seja mais representativa por preservar todas as componentes de frequência das imagens naturais. A partir dela são construídas novas bases de dados com blocos de características distintas de entropia. A hipótese é, baseado em [1, 2], que a performance do autocodificador está relacionado com essa característica. O processo de construção das bases de dados a seguir foram obtidas seguindo o relatório técnico em [56].

Primeiro, uma base de dados principal foi obtida usando todas as imagens dos seguintes conjuntos de dados:

1. Conjunto de dados CLIC [57] (*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 [58] (*DIVerse 2K resolution high. quality image*) - Imagens de alta resolução

   Treinamento: 800 imagens;

   Validação: 100 imagens.

3. Conjunto de dados Ultra-Eye [59] (*Ultra-Eye: UHD and HD images eye tracking datase*) - Alta qualidade e alta resolução.

   HD: 38 imagens;

   UHD: 40 imagens.

As imagens foram separadas em blocos de $32 \times 32 \times 3$ os quais foram salvos usando compressão sem perdas no padrão PNG. Esse procedimento gerou 6.231.440 blocos. Apresentamos na Figura 3.3 um histograma do tamanho desses blocos. Essa medida será usado como uma estimativa da entropia do bloco.

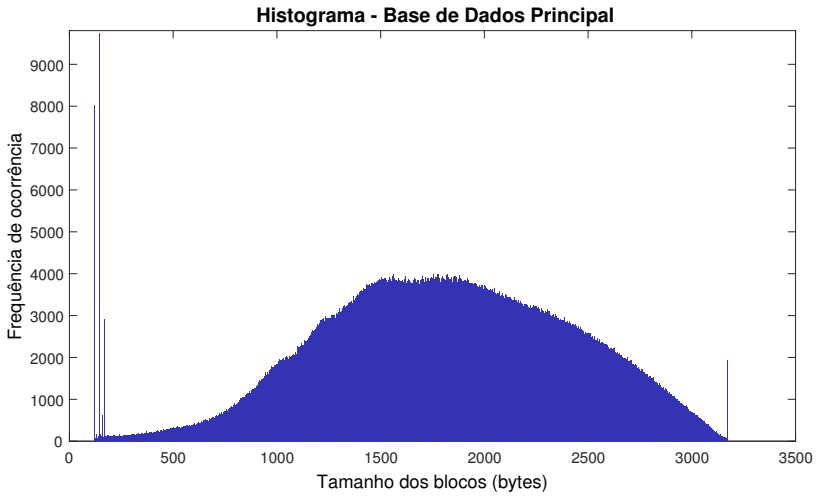


Figura 3.3 – Histograma de todo o banco de dados

O bloco que apresentou maior entropia foi definido como uma referência de entropia igual a 100%. A partir dessa referência, e tendo em mente que a entropia das imagens de treinamento impactaria 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.

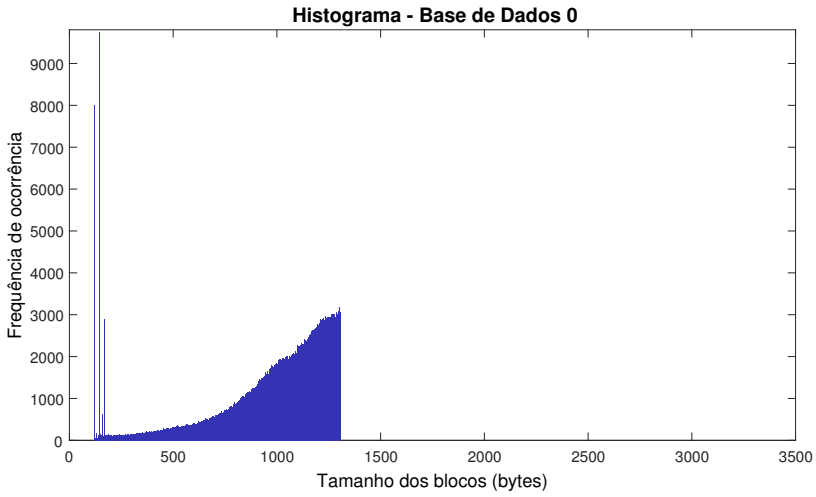


Figura 3.4 – Histograma da baixa entropia DB0.

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

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 3.4, 3.5, 3.6, 3.7 e 3.8.

A base de dados da Kodak [60] formada por 24 imagens naturais foi empregada nos testes do autocodificador.

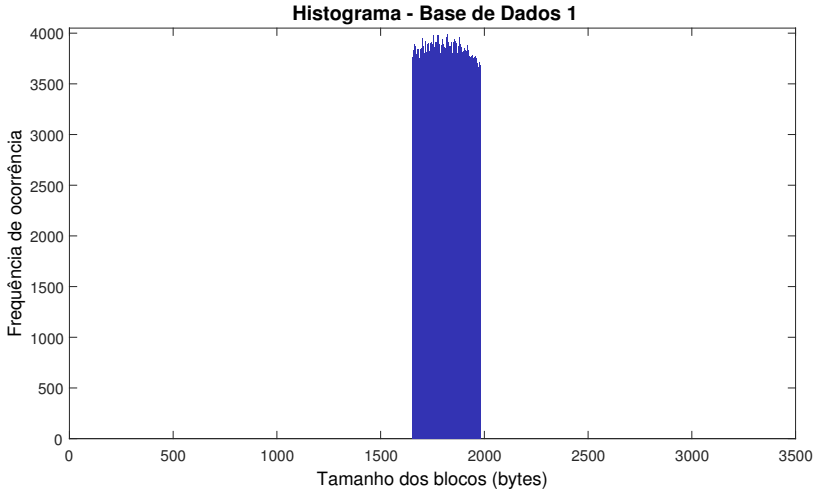


Figura 3.5 – Histograma do DB1 de entropia média.

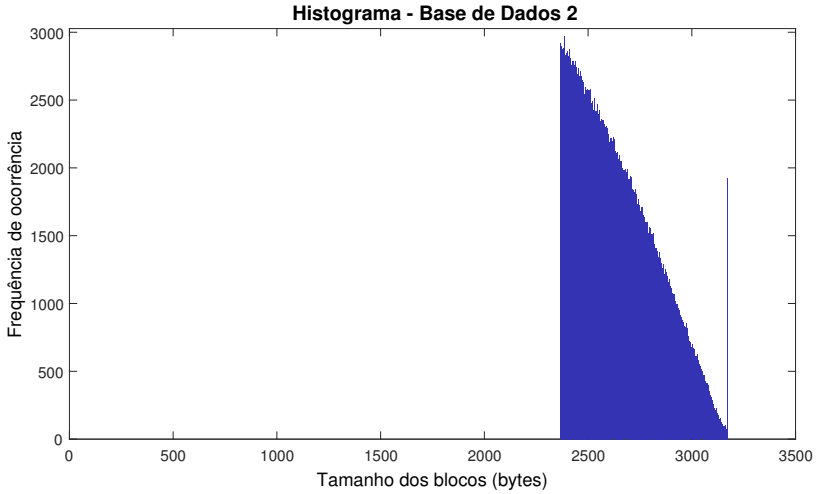


Figura 3.6 – Histograma do DB2 de alta entropia.

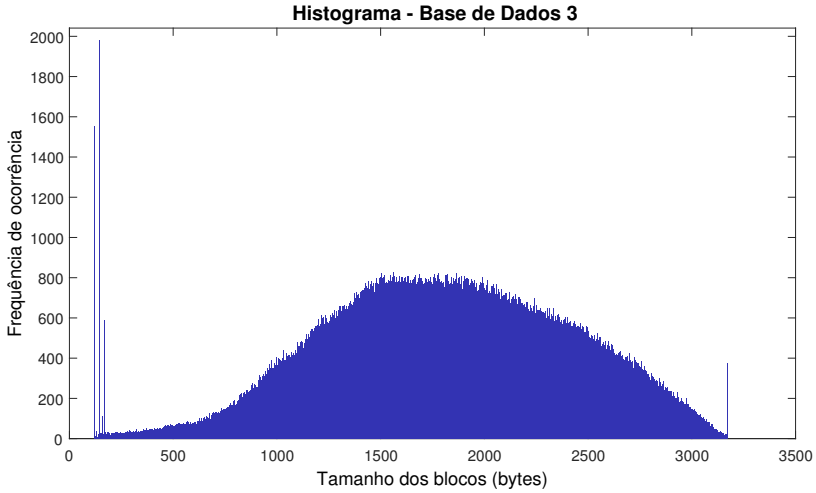


Figura 3.7 – Histograma da entropia mista DB3.

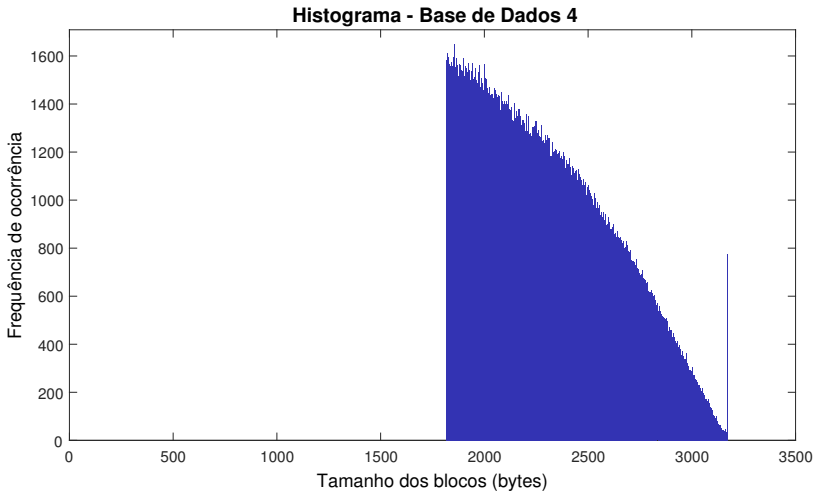


Figura 3.8 – Histograma da entropia alta DB4.

3.3 FUNÇÃO DE CUSTO

A função de custo é outra escolha importante nos treinamento 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 [47, 3]. Algumas funções conhecidas apresentam relações da medida de distorção entre uma imagem e sua reconstrução, tais como MSE, MAE, e versões diferenciáveis de $1-SSIM$ e $1-MS-SSIM$. Em nossos testes, dedicamos atenção especial na escolha da função de custo conveniente para o problema de compressão de imagens. No primeiro momento, avaliamos a performance do modelo com distorção das reconstruções.

Vamos definir algumas variações e combinações dessas funções e comparar os resultados em treinamentos curtos. A função de custo L é calculada em cada estágio e ao final das iterações computamos a sua média para finalmente aplicar os algoritmos de retropropagação e atualização dos pesos, isto é:

$$l = \frac{1}{k}\sum_{t=0}^{k} L(r_t, r'_t) \tag{3.4}$$

Até aqui, não estamos tendo controle sobre a entropia do código binário gerado. Se pudermos otimizar a rede para gerar a sequência de bits com baixa entropia teremos ganhos relevantes ao aplicador um codificador de entropia.

Portanto, em um segundo momento adicionamos uma função de regularização R para otimizar implicitamente a taxa do código binário, inspirado no trabalho [49]. Para calcula-lo fazemos uma cópia do código binário fornecido pela função de binarização em um dada

iteração t, $B(E_t(r_{t-1}))$, 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 como um medida de distorção para obter a nova função de perdas L:

$$L = MSE(r, r') + \lambda(t) \times \sum_{t=1}^{n} Z \tag{3.5}$$

onde n é o número de bits do código binário por nível. Em nossa rede esse número é igual a 128 por bloco $32x32$. 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 regularização que controla a proporção de R sobre a função de custo e que depende da iteração. Em nossos testes, verificamos é interessante que λ seja uma função decrescente da iteração t. Para um mini-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 é esperado para a função R e então controlamos o seu impacto em L usando um valor de regularização menor. Dessa forma, tentamos balancear o impacto da distorção e taxa. Nessa abordagem, aplicamos uma codificação sem perdas dada pelo GZIP para aproveitar a estatística de distribuição do bits que esperamos ser de baixa entropia. Portanto, na reconstrução de uma imagem existe a taxa nominal r_n e fixa por iteração, caracterizando um método de taxa r_r constante (do inglês *Constant bitrate* (CBR))), e há a taxa real fornecida pelo GZIP. O ganho percentual de taxa em virtude da codificação de entropia é descrito pela Equação a seguir:

$$G = 100 \times \frac{r_n - r_r}{r_n} \tag{3.6}$$

3.4 ALOCAÇÃO DINÂMICA DE BITS

Uma desvantagem do modelo é que ele reserva um número fixo de bits para reconstruir qualquer bloco de imagem, independentemente da complexidade do seu conteúdo. Tal problema foi levantando em [47] e descrito na revisão desse trabalho. Para tentar contornar essa desvantagem, propomos uma heurística simples e semelhante a [47].

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 recons-

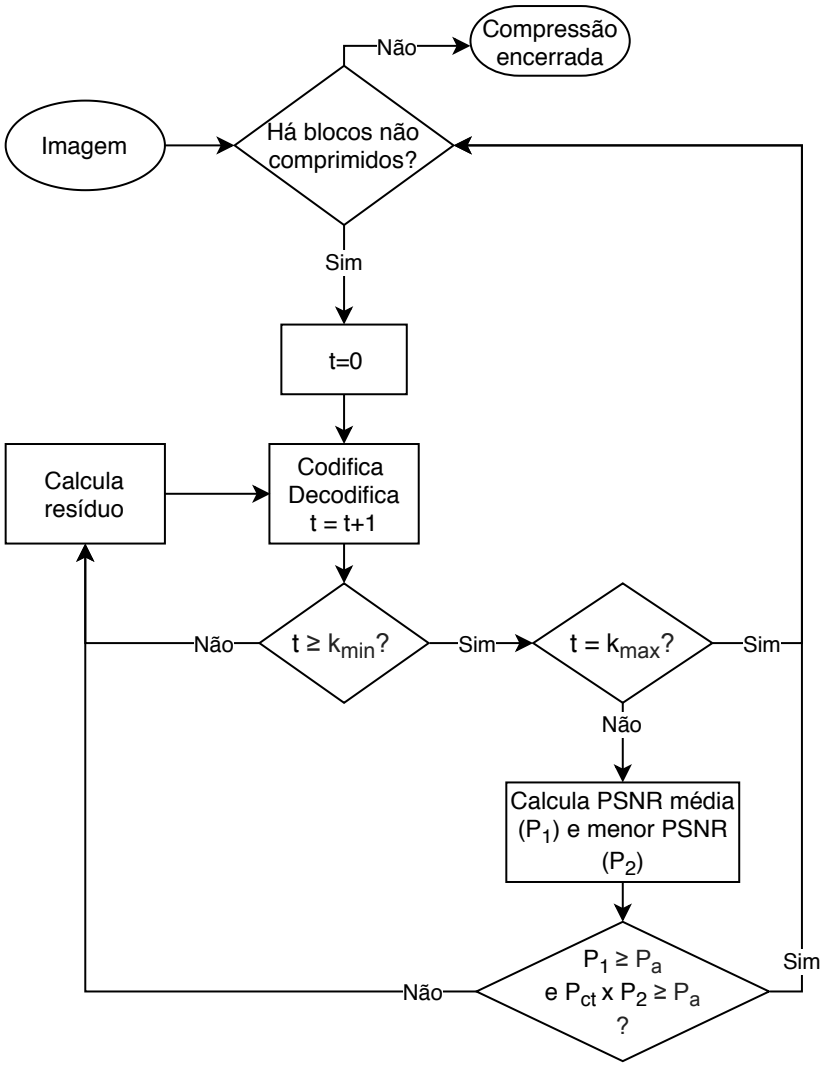


Figura 3.9 – Alocação dinâmica de bits.

trução 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 3.9 resume o método de alocação dinâmica de bits.

3.5 IMPLEMENTAÇÃO

O autocodificador foi implementado usando a biblioteca PyTorch da empresa Facebook. Ele 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 a 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.

4 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. Treinamos 5 modelos do autocodificador para cada base de dados apresentada em 3.2 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. Otimizador: Adam;
6. 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 4.1.

Tabela 4.1 – 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 margem pequena em relação a BD2. Para temos uma comparação mais ampla, calculamos as áreas abaixo das curvas de PSNR $\times$ taxa, SSIM $\times$ taxa e MS-SSIM $\times$ taxa dos modelos treinados por DB2 e DB4. Em todas as curvas temos 16 pontos com taxas nominais no intervalo de 0,125 a 2,0 bpp com incrementos constantes de 0,125 bpp. A Figura 4.1 é mais um indicativo que o modelo treinado pela BD4 apresenta a melhor performance.

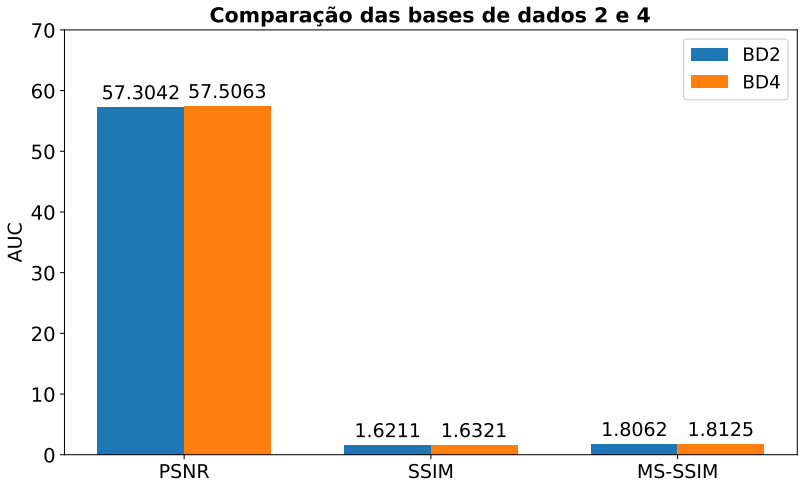


Figura 4.1 – Essa Figura mostra as áreas abaixo da curvas de PSNR, SSIM e MS-SSIM cada uma em função das taxas nominais do modelo com 16 iterações.

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 definido conforme itens citados em 4, exceto o número de épocas. A evolução do modelo ao passar das época pode ser vista na Figura 4.2.

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. [2]. Essas Figuras estão disponíveis em 4.3 e 4.4. Observamos que nessas métricas e na média das imagens da Kodak, o nosso modelo supera o trabalho de Toderici et. al. 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 4.5 indica que na métrica PSNR o JPEG(4:2:0) ainda supera o nosso modelo para a maioria das taxas calculadas.

Pela figura 4.2 verificamos uma tendência de saturação do modelo, em torno de 36 dB na taxa de 2 bpp, ao passar da épocas de treinamento. Dessa forma, exploraremos novas técnicas que podem gerar menores taxas de compressão sem impactar na qualidade das imagens reconstruídas pelo decodificador.

A Tabela 4.2 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 4. Nessa

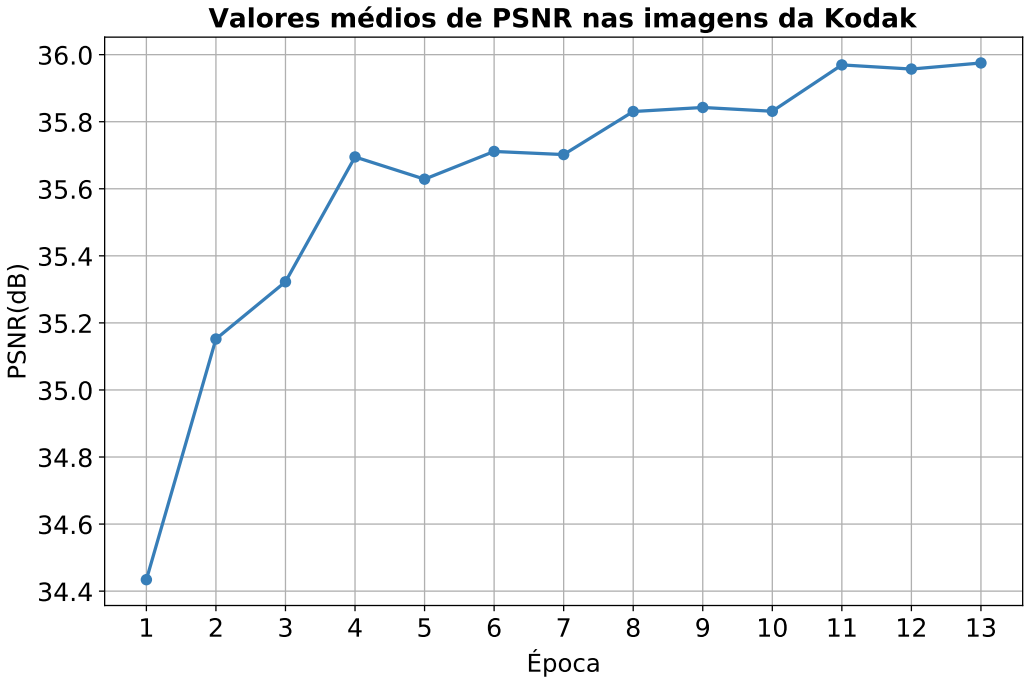


Figura 4.2 – Performance do modelo em função do número de épocas de treinamento. Para cada época, calculamos a PSNR média das imagens da Kodak reconstruídas com taxa de 2 bpp. O ganho de desempenho do autocodificador entre a primeira e 13ª época foi de um pouco mais que 1,5dB.

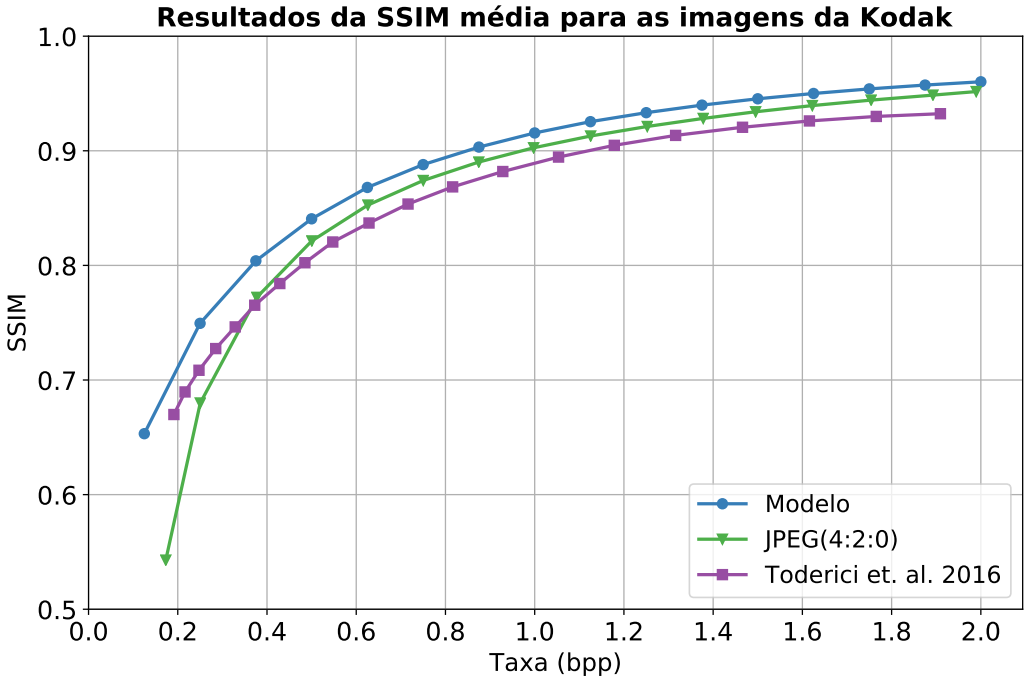


Figura 4.3 – Comparação.

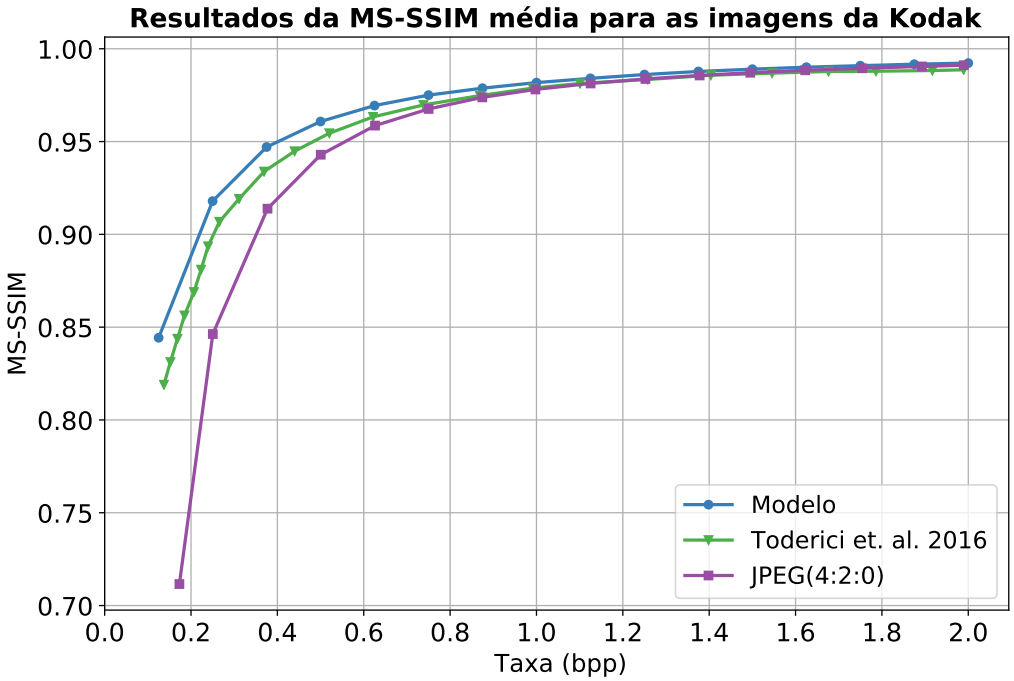


Figura 4.4 – Comparação.

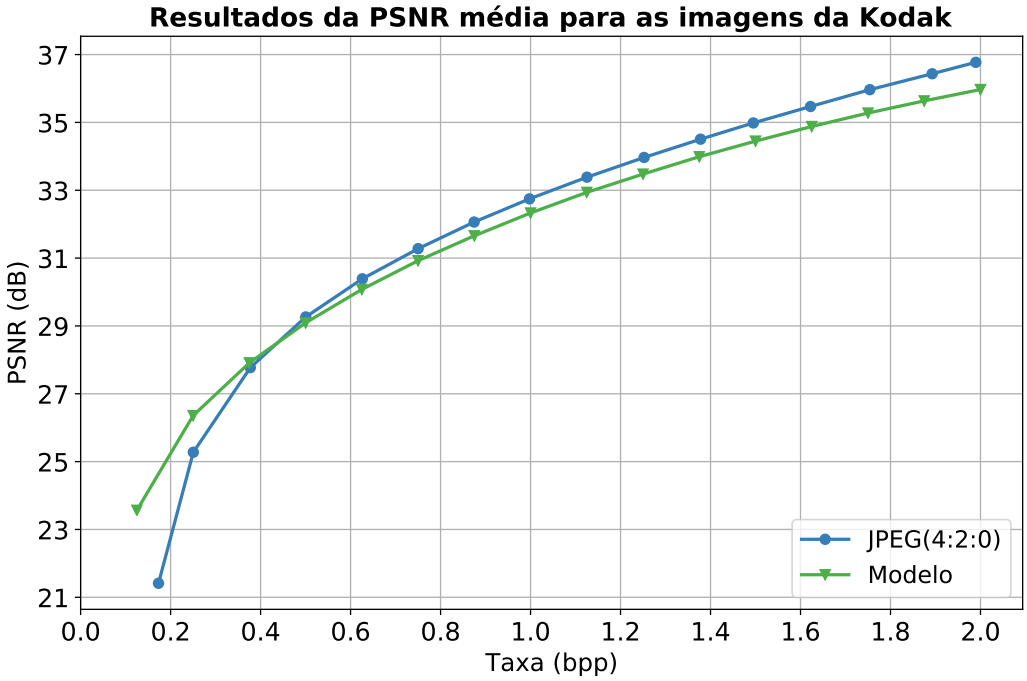


Figura 4.5 – Comparação.

Tabela 4.2 – 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

tabela, R, R', X e X' foram definidos na Equação 3.1, omitindo-se a dependência temporal. A medida $X - X'$ é o resíduo em relação a imagem como um todo, onde X é fixo. 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 [61]. 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 3.5 com um $\lambda(t)$ obtida de forma experimental e dado por:

$$\lambda(t) = 3,5 \times 10^{-7} \times e^{-0,01 \times (t+1)} \tag{4.1}$$

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;
4. Número de iterações durante os testes: 22;
5. Número de épocas: 27;
6. Otimizador: Adam;

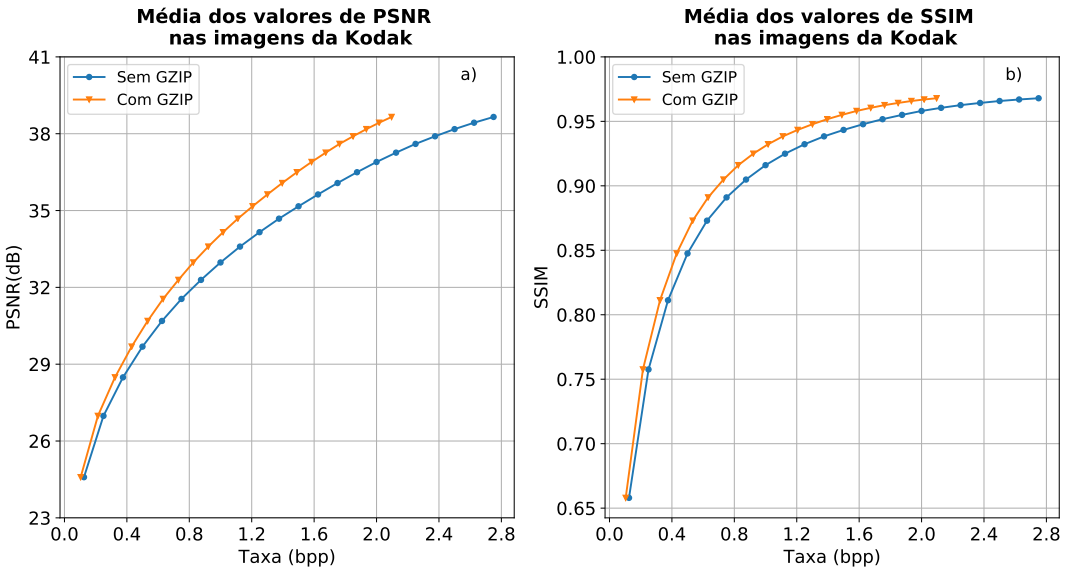


Figura 4.6 – Nessa figura, ilustramos o resultado em aplicar o GZIP na sequência binária gerada no codificador. As curvas foram plotadas usando o modelo treinado até a época 26.

7. 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 4.6 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 4.7 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 a primeira é aproximadamente de 17%. Nesse teste, a 26ª época gerou o modelo de melhor desempenho.

Nas figuras em 4.8 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.

A Figura 4.9 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.

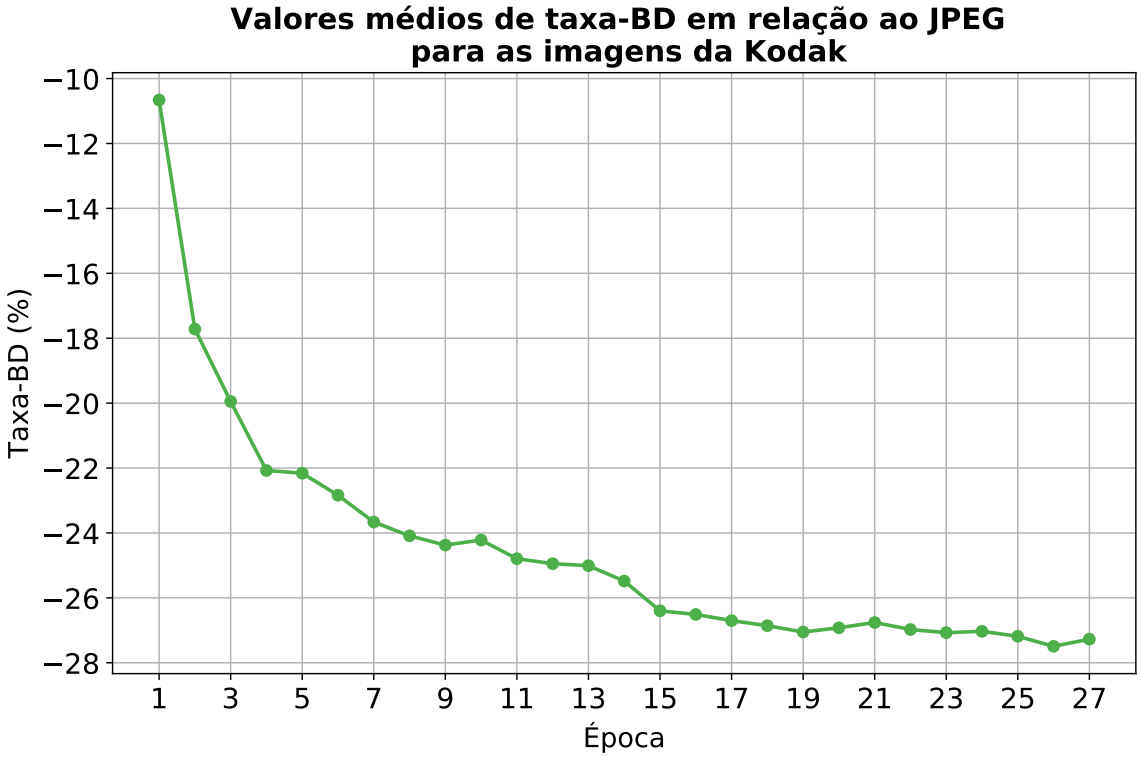


Figura 4.7 – Ilustração da porcentagem da taxa-BD média sobre a base de dados da Kodak e tendo o JPEG como âncora. Valores cada vez mais negativos indicam que a economia média de taxa de bits do nosso modelo em relação ao JPEG aumenta para uma dada qualidade equivalente de PSNR na média das imagens da Kodak.

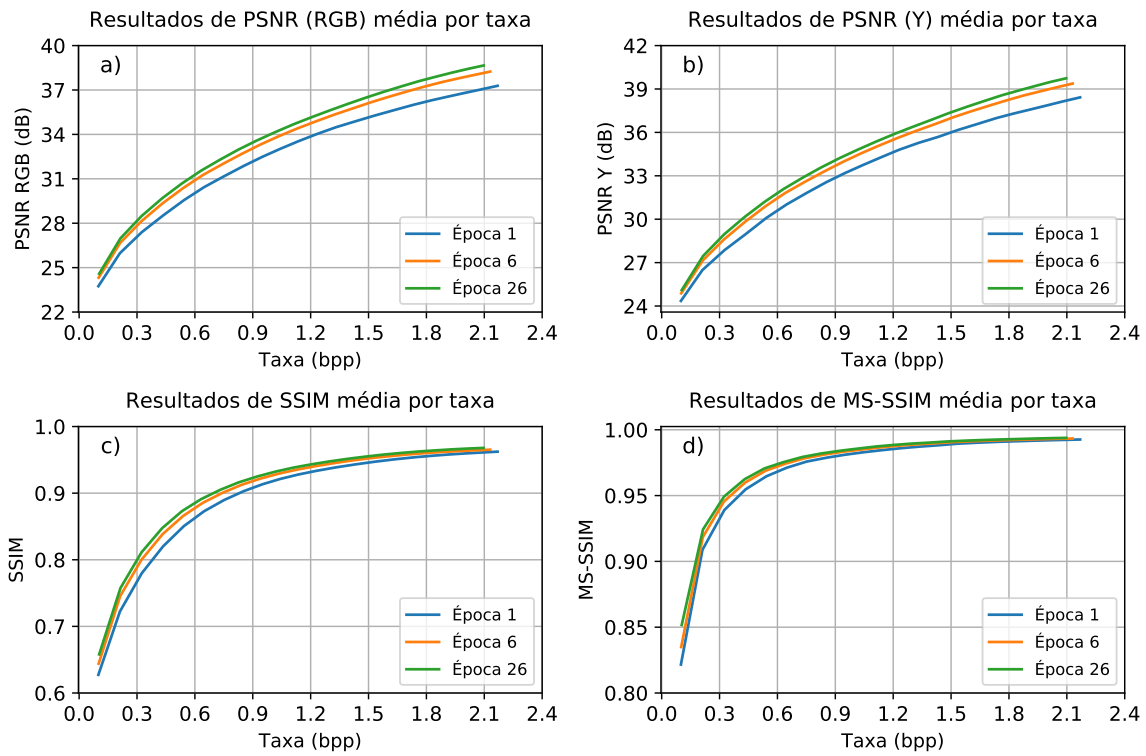


Figura 4.8 – Figuras mostrando a evolução do modelo em termos de taxa e 4 métricas de qualidade em 3 épocas distintas. Nas 4 curvas taxa e distorção apresentam melhores resultado no passar da primeira, sexta e vigésima-sexta épocas

Os resultados mostram que, aproximadamente, nos primeiros 3 níveis de reconstrução o modelo final tem ganho médio menor em relação a primeira e sexta versão do modelo. Além disso, nesses 3 níveis 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 que modela a taxa aproximadamente. Então, ao priorizar 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 no 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 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

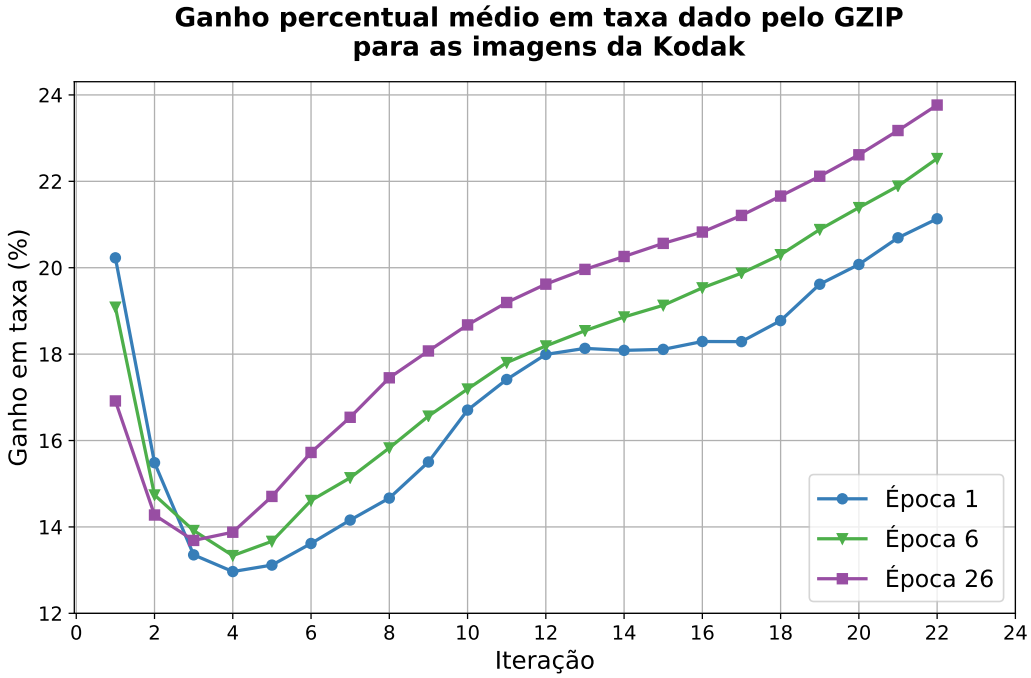


Figura 4.9 – Figura com o ganho do GZIP por nível de reconstrução e na média para as 24 imagens da Kodak em 3 codecs. De forma geral, nas primeiras iterações ocorre redução no ganho e a partir da quarta iteração o ganho de taxa é crescente.

suas reconstruções estão disponíveis ns Figura 4.11.

No método de alocação dinâmica de bits descrito em 3.4 fizemos dois testes principais. No primeiro, não havia restrição do mínimo de níveis de reconstrução ou iterações em cada bloco, isso equivale a fazer $k_min = 1$ em 3.9. Vamos no referir a ele como modeloVR0. 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 4.13

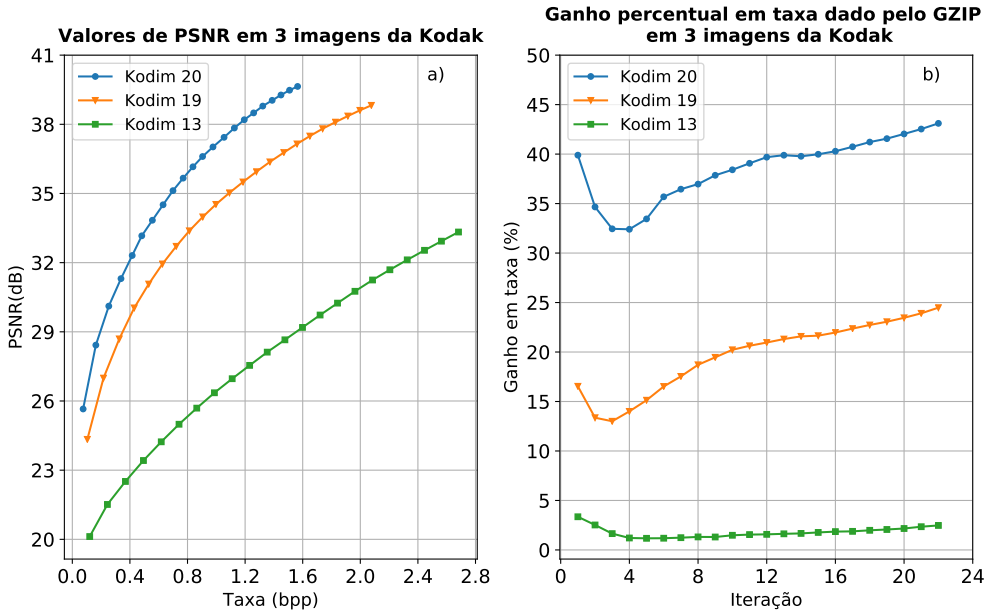


Figura 4.10

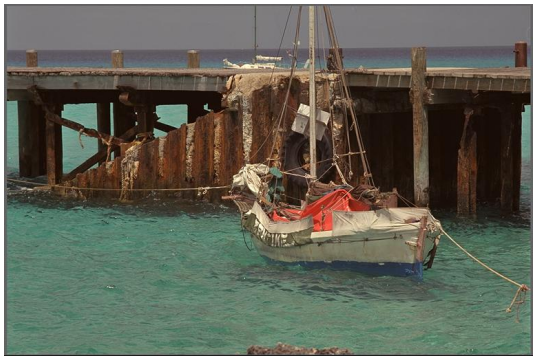
(a) Imagem original

(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.

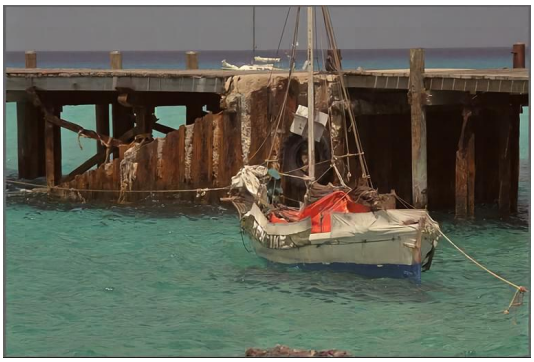
Figura 4.11 – Em (a) temos a Kodim10 original em (b) sua versão reconstruída em 10 iterações. Em (c) está representado a kodim13 e em (d) sua sua reconstruída em 8 iterações.



(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 4.12 – 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.

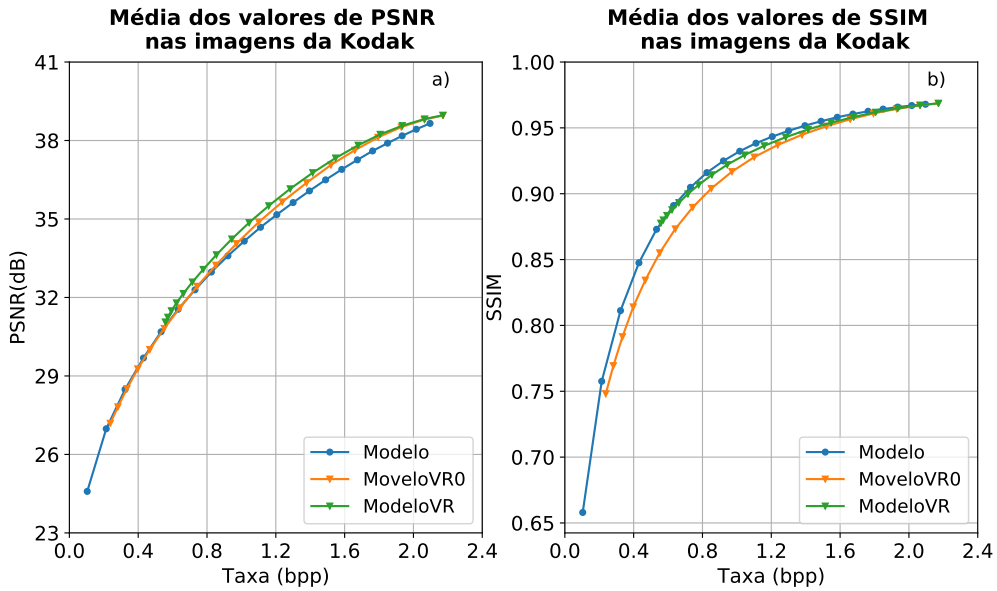


Figura 4.13

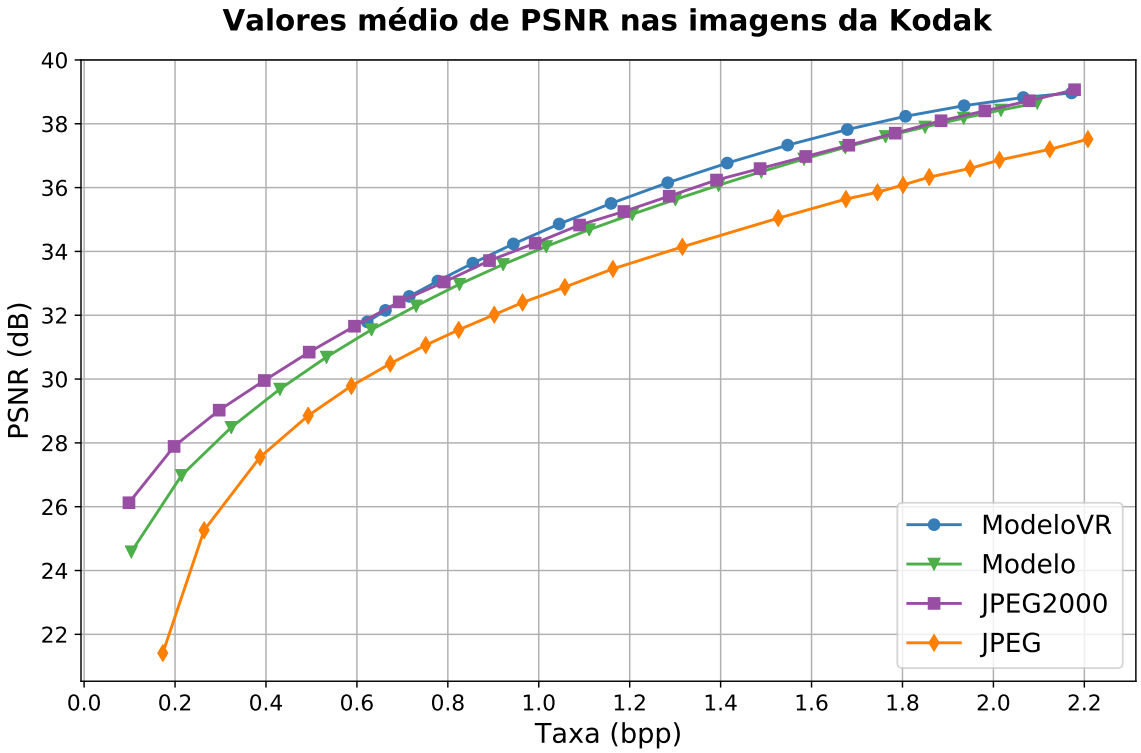

Valores médio de PSNR nas imagens da Kodak
PSNR (dB)
Taxa (bpp)
ModeloVR
Modelo
JPEG2000
JPEG

Figura 4.14

REFERÊNCIAS BIBLIOGRÁFICAS

1 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>.

2 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>.

3 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>.

4 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>.

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

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

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

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

9 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.

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

11 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.

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

13 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.

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

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

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

17 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.

18 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.

19 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>.

20 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.

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

22 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>.

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

24 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.

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

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

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

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

29 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.

30 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.

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

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

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

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

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

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

37 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.

38 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.

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

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

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

42 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.

43 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.

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

46 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>.

47 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>.

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

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

50 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.

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

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

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

58 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/>.

59 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>.

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

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