UNIVERSIDADE FEDERAL DO AMAZONAS
INSTITUTO DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

JOSIANE RODRIGUES DA SILVA

RECUPERAÇÃO DE IMAGEM COM MÚLTIPLOS RÓTULOS USANDO HASHING PROFUNDO

Manaus
Julho de 2022

UNIVERSIDADE FEDERAL DO AMAZONAS

INSTITUTO DE COMPUTAÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

JOSIANE RODRIGUES DA SILVA

RECUPERAÇÃO DE IMAGEM COM MÚLTIPLOS RÓTULOS USANDO HASHING PROFUNDO

Tese apresentada ao Programa de Pós-Graduação em Informática do Instituto de Computação da Universidade Federal do Amazonas como requisito parcial para a obtenção do grau de Doutora em Informática.

Orientador: Marco Antônio Pinheiro de Cristo

Manaus

Julho de 2022

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

Silva, Josiane Rodrigues da
S586r Recuperação de imagem com múltiplos rótulos usando hashing profundo / Josiane Rodrigues da Silva. 2022
122 f. : il. color; 31 cm

Orientador: Marco Antônio Pinheiro de Cristo
Tese (Doutorado em Informática) - Universidade Federal do Amazonas

1. Recuperação de imagem baseada em conteúdo. 2. Aprendizagem de máquina. 3. Aprendizagem profunda. 4. Hashing profundo. 5. Arquiteturas geradoras profundas. I. Cristo, Marco Antônio Pinheiro de. II. Título.

FOLHA DE APROVAÇÃO

"Recuperação de imagem com múltiplos rótulos usando Hashing profundo"

JOSIANE RODRIGUES DA SILVA

Tese de Doutorado defendida e aprovada pela banca examinadora constituída pelos Professores:

Prof. Marco Antonio Pinheiro de Cristo - PRESIDENTE

Profa. Eulanda Miranda dos Santo - MEMBRO INTERNO

Prof. André Luiz da Costa Carvalho - MEMBRO INTERNO

Prof. Thierson Couto Rosa - MEMBRO EXTERNO

Prof. Raoni Simões Ferreira - MEMBRO EXTERNO

Manaus, 25 de Jul de 2022

Aos meus pais.

Agradecimentos

Tão difícil quanto escrever esta tese, é conseguir expressar em poucas palavras toda minha gratidão a todas as pessoas que me ajudaram chegar até aqui. Mesmo assim, não posso deixar de agradecer a todos vocês.

Inicio agradecendo a Deus pelo dom da vida e por ter me permitido fechar mais esse ciclo.

À minha família por toda torcida e por todo apoio que contribuiu diretamente para que eu pudesse ter um caminho mais fácil nesses longos anos. Em especial, aos meus pais, Maria Rodrigues e José Gomes, que tornaram esse sonho possível e por ser o sonho deles também. Às minhas irmãs, Joélia e Jomara, por todo carinho. E ao meu noivo Rafael Sandes por todo amor.

Agradeço imensamente ao meu orientador, professor Marco, por toda paciência, orientação e por todos os ensinamentos. Quero também expressar minha admiração pelo profissional competente e amigo que se tornou.

Agradeço a todos os amigos que me apoiaram nesse período. Em particular, às amigas do coração: Lidia Lizziane, Maria Ivanilse, Joyce e Ludimila. Obrigada por todo carinho e atenção. Provavelmente vocês receberão menos reclamações agora :D

Meus agradecimentos ao professor Juan Colonna por contribuir com esta pesquisa e a todos os professores do PPGI que de alguma forma contribuíram para a minha formação.

A versão final do texto desta tese foi feita após defesa no dia 25 de julho de 2022. Gostaria de agradecer imensamente aos professores da banca: Thierson Rosa, Raoni Simões, Eulanda Miranda e André Carvalho. Obrigada pela avaliação e contribuições para versão final desta tese.

A todos vocês meus sinceros agradecimentos.

"A educação é a arma mais poderosa que você pode usar para mudar o mundo."
(Nelson Mandela)

Resumo

Recuperação de imagem baseada em conteúdo (*Content-based Image Retrieval* - CBIR) é a tarefa que visa exibir, como resultado de uma busca, imagens com os mesmos conteúdos visuais de uma consulta. Esse problema tem atraído atenção crescente na área de visão computacional. Técnicas de *hashing* baseado em aprendizado são hoje uma das abordagens mais estudadas de pesquisa aproximada de vizinhos mais próximos para recuperação de imagens em larga escala. Com o avanço das redes neurais profundas em representação de imagem, métodos de *hashing* para CBIR passaram a usar aprendizado profundo no processo de construção dos códigos binários. Tais estratégias são conhecidas genericamente como técnicas de *Hashing* Profundo (*deep hashing*). Embora uma variedade de métodos tenham sido propostos para CBIR usando *hashing* profundo, a maioria deles propõem abordagens que tratam as imagens como descrevendo um único tópico, ou seja, associadas a um único rótulo. Contudo, em busca visual é natural que as imagens tenham vários tópicos, cada um dos quais representados por diferentes rótulos que podem estar relacionados, por exemplo, com objetos de várias categorias ou diferentes conceitos associados com as imagens. Além disso, muitos desses modelos focam exclusivamente na qualidade dos *rankings* gerados, ignorando questões como a eficiência da busca e do uso do espaço disponível, aspectos importantes em recuperação de imagem. Desta forma, esta pesquisa investiga técnicas de *hashing* profundo com o intuito de tornar a tarefa de recuperação de imagem mais eficiente mantendo a qualidade do *ranking* de resposta compatível com o estado-da-arte. Além disso, focamos no cenário de múltiplos rótulos com o objetivo de gerar códigos *hash* que representem os vários níveis de similaridade entre as imagens. Mais especificamente, ao longo desta pesquisa, propusemos e estudamos arquiteturas profundas geradoras treinadas em duplas e triplas de imagens para a tarefa de recuperação de imagens multi-rótulo. As arquiteturas usadas foram autocodificadores variacionais baseados em distribuição discreta, capazes de gerar representações compactas das imagens, diretamente aplicáveis a técnicas de *hashing*, sem auxílio de processos intermediários não vinculados ao treino. Ao avaliarmos os métodos propostos em duas coleções de imagens multi-rótulo,

uma sintética e outra real, observamos que os modelos são capazes de gerar códigos *hash* binários efetivos em termos da qualidade dos *rankings* criados além de eficientes em termos do uso do espaço de *hashing*.

Palavras-chave: Recuperação de Imagem Baseada em Conteúdo, Aprendizagem de Máquina; Aprendizagem Profunda; *Hashing* Profundo; Arquiteturas Geradoras Profundas; Recuperação de Imagens com Múltiplos Rótulos; Autocodificadores Variacionais.

Abstract

Content-based Image Retrieval (CBIR) is the task of retrieving images as result of an image search, such that the retrieved images have the same visual contents as the query image. This problem has attracted increasing attention in the area of computer vision. Learning-based hashing techniques are among the most studied approaches to nearest-neighbor approximate search for large-scale image retrieval. With the advancement of deep neural networks in image representation, hashing based methods for CBIR have adopted deep learning in the process of outputing binary hash codes. Such strategies are known generically as Deep Hashing techniques. Although a variety of methods have been proposed for CBIR using deep hashing, most of them deal with single-labeled images. However, in visual search it is natural for images to have several topics, each of which is represented by a different label that may be related, for example, with objects of various categories or different concepts associated with the images. Furthermore, many of these models focus exclusively on the quality of the generated rankings, ignoring issues such as search efficiency and the use of the available space, which are important aspects to consider in Image Retrieval. In this way, we investigate deep hashing techniques which enable efficient image retrieval while achieving a high-quality response ranking. In addition, we focus on the multiple-label scenario so that the generated hash codes capture the various levels of similarity among the images. More specifically, throughout this research, we propose and study deep generative architectures trained on pairs and triples of images for the task of multi-label image retrieval. To this, we adopt variational autoencoders based on discrete distributions. These models can generate compact image representations, directly applicable to hashing techniques, without intermediate processes unrelated to training. When evaluating the proposed methods in two collections of multi-label images, we observed that they are capable of generating effective binary hash codes. Such codes can be used to produce high-quality rankings while enabling an efficient use of the hashing space.

Keywords: Content-based Image Retrieval; Machine Learning; Deep Learning; Deep

Hashing; Deep Generative Architectures; Multi-label Image Retrieval; Variational Autoencoders.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A disponibilidade massiva de imagens na *web* requer o desenvolvimento de técnicas efetivas de representação de conteúdo para viabilizar a sua recuperação pelos usuários. Como consequência, a recuperação de imagem baseada em conteúdo (*Content-based Image Retrieval* - CBIR), que visa exibir, como resultado de uma busca, imagens com os mesmos conteúdos visuais de uma consulta, tem atraído atenção crescente na área de visão computacional. Uma variedade de métodos eficientes de busca têm sido propostos com o intuito de tornar essa tarefa mais efetiva.

Em geral, a tarefa de recuperação de imagem é baseada na busca dos vizinhos mais próximos (ANN) [Wang et al., 2014]. Contudo, determinar o conjunto exato de vizinhos mais próximos tem alto custo computacional, especialmente se considerarmos que em cenários reais as coleções são de grande escala. Consequentemente, a maior parte da literatura [Wang et al., 2018] adota técnicas de busca aproximada (também conhecidas como pesquisa por similaridade, pesquisa por proximidade ou pesquisa por item próximo), que são mais eficientes e mostram-se suficientemente úteis em muitos problemas práticos.

As soluções mais estudadas para resolução do problema de pesquisa por similaridade envolvem o uso de métodos de *hashing*. Nestes métodos, dados de alta dimensão são transformados em códigos binários compactos de forma que códigos similares sejam mapeados para itens de conteúdo similar. Devido à eficiência, tanto na velocidade quanto no armazenamento, muitos métodos de *hashing* foram propostos nos últimos anos. Estes normalmente são classificados em métodos independentes de dados e dependentes de dados. Para a primeira categoria, geralmente são empregadas projeções aleatórias para mapear amostras em um espaço de características e, em seguida, os vetores numéricos resultantes são binarizados. Os métodos representativos nesta categoria incluem *Locatily Sensitive Hashing* (LSH) [Andoni & Indyk, 2006] e suas extensões

discriminativas e baseadas em *kernel* [Jain et al., 2008; Kulis & Grauman, 2009; Raginsky & Lazebnik, 2009]. Para a segunda categoria, várias técnicas de aprendizado de máquina são exploradas para aprender funções de *hashing* capazes de mapear amostras em códigos binários. Esta categoria inclui *spectral hashing* [Weiss et al., 2009], *binary reconstructive embedding* (BRE) [Kulis et al., 2009], *iterative quantization* (ITQ) [Gong et al., 2013b], *K-means hashing* (KMH) [Lu et al., 2015], *minimalloss hashing* (MLH) [Norouzi & Blei, 2011] e *sequential projection learning hashing* (SPLH) [Wang et al., 2012]. Comparados aos métodos de *hashing* independentes de dados, os dependentes alcançam desempenho de busca e acurácia comparável ou melhor com códigos mais curtos, por utilizarem dados de treino.

Com o avanço das redes neurais profundas em representação de imagem [Krizhevsky et al., 2012], métodos de *hashing* para CBIR passaram a usar aprendizado profundo no processo de construção dos códigos binários. Tais estratégias são conhecidas genericamente como técnicas de *Hashing* Profundo (*Deep Hashing*). Elas têm sido amplamente investigadas por combinar as vantagens das redes neurais convolutivas profundas com o baixo custo computacional e de armazenamento de técnicas de *hashing*. Assim, os métodos resultantes mapeiam os complexos espaços de representação de imagens aprendidos por modelos profundos para espaços binários compactos, preservando a similaridade semântica dos espaços originais.

Embora uma variedade de métodos tenham sido propostos para CBIR usando *hashing* profundo [Zhu et al., 2016; Cao et al., 2016; Wang et al., 2016b; Shen et al., 2017; Yang et al., 2018; Cakir et al., 2018], a maioria deles propõem abordagens que tratam as imagens como descrevendo um único rótulo, ou seja, associada a um único rótulo. Contudo, em busca visual é natural que as imagens tenham vários rótulos, cada um dos quais representados por diferentes rótulos que podem estar associados, por exemplo, com objetos de várias categorias. Além disso, múltiplos rótulos podem facilitar a captura de informações sobre os relacionamentos entre as imagens, não necessariamente identificáveis apenas por conteúdo visual. Por exemplo, é possível perceber, através de rótulos, que duas imagens representam pinturas, mesmo que elas tenham conteúdos visuais muito distintos. Dessa forma, definir semelhanças semânticas sem considerar os vários rótulos das imagens pode ter impacto negativo na tarefa de recuperação.

Observe que abordagens para recuperação de imagens com múltiplos rótulos também são aplicáveis às imagens que contêm apenas um rótulo, uma vez que o problema de rótulo único pode ser pensado como um caso particular de múltiplos rótulos. Logo, esta abordagem é mais geral.

Desse modo, o foco desta pesquisa é a recuperação eficiente de imagem com múl-

tiplos rótulos. A abordagem proposta consiste em um modelo de *hashing* dependente dos dados usando redes neurais profundas. Métodos que utilizam esse tipo de abordagem visam aprender funções de *hashing* a partir de um conjunto de dados específicos para que o resultado da pesquisa de vizinho mais próximo no espaço de codificação seja a mais similar possível ao resultado no espaço original. Além disso, eles devem preservar a similaridade semântica dos vários rótulos, com baixo tempo de pesquisa bem como uso efetivo de espaço.

1.1 Motivação e Justificativa

Os modelos baseados em aprendizado profundo, propostos para solucionar o problema de recuperação de imagem com múltiplos rótulos, têm atraído cada vez mais atenção devido ao ganho de precisão obtido comparados aos modelos clássicos. Recentemente, a combinação de aprendizado profundo com o uso de técnicas de *hashing* tem tornado esses modelos mais eficientes. Contudo, apesar do avanço nessa área, tais técnicas ainda necessitam de processamento computacional elevado na etapa de recuperação devido à utilização de arquiteturas profundas, o que torna esses modelos naturalmente mais lentos. Além disso, há algumas propriedades dos códigos *hash* que podem ser úteis em termos de eficiência de processamento e uso de memória, como a independência e balanceamento dos bits nos códigos binários gerados que tem impacto direto sobre o uso efetivo do espaço de representação [Lu et al., 2017]. Estas propriedades, contudo, como vimos ao longo desta pesquisa, são pouco exploradas pelos métodos ao gerar códigos para imagens com múltiplos rótulos. Nesse sentido, aprofundar os estudos nessa área, explorando técnicas eficazes e eficientes, se apresenta como um caminho promissor. Através dele, é possível melhorar a capacidade de representação dos códigos gerados bem como reduzir o custo associado à etapa de recuperação de imagens, tornando o sistema mais eficiente sem comprometer a qualidade de resposta da busca, alcançando assim resultados competitivos em relação aos outros métodos CBIR que utilizam aprendizagem profunda.

1.2 Descrição do Problema

O problema objeto desta tese consiste em, dada uma imagem descrita por um ou mais rótulos, obter uma representação numérica desta imagem que possa ser usada de forma efetiva como um código *hash*. Assim, para que tal código seja usado para recuperar a imagem correspondente, através de um método de busca baseado em *hashing*, espera-se

que ele tenha propriedades que facilitem processos de busca e armazenamento, proporcionando (1) *eficiência*: os códigos gerados devem ser compactos e facilitar processos eficientes de recuperação com uso adequado do espaço de representação, buscando-se o menor número de colisões para cada comprimento de código escolhido; e (2) *boa capacidade de representação conceitual*: imagens similares, seja por seus aspectos visuais ou múltiplos rótulos em comum, devem receber códigos similares.

Mais formalmente, considere $\mathcal{X} = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N] \in \mathbb{R}^{d \times N}$ um conjunto de N imagens, onde $\mathbf{x}_n \in \mathbb{R}^d$ $(1 \leq n \leq N)$ é a n-ésima imagem em $\mathcal{X}$.[1] Considere ainda um conjunto de classes $\mathcal{L} = \{1, ..., C\}$, tal que cada $\mathbf{x}_n \in \mathbb{R}^d$ é associada a um subconjunto de rótulos $\mathcal{Y}_n \subseteq \mathcal{L}$. Métodos de *hashing*, como os que focamos nesta tese, devem aprender funções de *hashing* para mapear cada imagem em códigos compactos. Dessa forma, podemos assumir que existem funções de *hashing* a serem aprendidas, que mapeiam cada $\mathbf{x}_n$ com rótulos $\mathcal{Y}_n$ a um vetor de código com K valores numéricos. Entre estas funções, queremos aprender as que proporcionem representações:

1. Eficientes: uma imagem $\mathbf{x}_n$ será sempre representada pelo mesmo código $\mathbf{y}_n$ de comprimento K, onde $\mathbf{y}_n = [y_{n1}, y_{n2}, ..., y_{nK}] \in \{0, 1\}^{K \times 1}$, ou seja, uma representação binária compacta. Dadas duas imagens distintas, $\mathbf{x}_i$ e $\mathbf{x}_j$, espera-se que estas tenham representações diferentes $(\mathbf{y}_i \neq \mathbf{y}_j)$ se há menos imagens que o espaço de representação, ou seja, para $N < 2^K$. Considerando a dificuldade de atingir tal objetivo, espera-se que o número de colisões $(\mathbf{y}_i = \mathbf{y}_j$ para imagens distintas $\mathbf{x}_i$ e $\mathbf{x}_j)$ seja o menor possível;

2. Conceituais: imagens similares são representadas através de códigos similares, ou seja, $(\mathbf{y}_i - \mathbf{y}_j) < (\mathbf{y}_i - \mathbf{y}_k)$ se $\mathbf{x}_i$ é mais similar a $\mathbf{x}_j$ que $\mathbf{x}_k$, considerando tanto seus conteúdos visuais quanto os rótulos de descrição;

Em suma, o desafio é obter códigos para imagens de múltiplos rótulos que conciliem eficiência com a melhor capacidade de representação conceitual.

1.3 Hipótese e Questões de Pesquisa

O problema apresentado nesta tese tem sido abordado tipicamente como um problema de representação de imagens, onde a representação é aprendida de forma supervisionada usando um modelo profundo. Normalmente, um processo de quantização é usado para

[1] Em termos práticos, $\mathbf{x}_n$ é normalmente um tensor de m canais (de cores) de d pixels. Sem perda de generalidade, se considerarmos uma imagem em tons de cinza (ou seja, com um canal), $\mathbf{x}_n \in \mathbb{R}^d$.

transformar a representação numérica encontrada pelo modelo no código binário a ser usado por um método de busca baseado em *hashing*.

Uma boa solução para este problema deve endereçar aspectos de qualidade e eficiência. Em termos de qualidade, o código gerado deve capturar aspectos semânticos da imagem de maneira que imagens de múltiplos rótulos semanticamente similares são representadas por códigos igualmente similares. Em termos de eficiência, estes códigos devem ser compactos e mapeados de maneira que o espaço de representação seja usado de forma tal que se evitem colisões desnecessárias.

Em relação à qualidade, os melhores modelos para aprendizagem de representação são os profundos [Rodrigues et al., 2020]. Modelos profundos tem sido amplamente desenvolvidos nos últimos anos para capturar de forma efetiva a semântica de imagens especialmente vinculada aos seus aspectos visuais. Além disso, dada que a nossa aplicação de interesse é busca de imagens, teoricamente seria mais efetivo treinar estes modelos em uma tarefa de aprendizagem supervisionada de *ranking*. Entre potenciais vantagens desta abordagem, citamos:

1. Aprender diretamente que uma imagem é mais similar a uma imagem de consulta que outra;

2. A busca por imagens com conteúdo cujos rótulos nunca foram vistos pelos modelos [Zhuang et al., 2016; Schroff et al., 2015]. Isso é interessante, pois em cenários reais, o conjunto de conteúdo de imagem (e rótulos) disponível geralmente é muito maior do que o usado para treinamento. Portanto, é desejável que o sistema produza respostas relevantes para as consultas não vistas no treinamento que provavelmente serão realizadas;

3. Lidar de forma melhor com dados desbalanceados já que esta é uma caraterística inerente ao aprendizado de *rankings* [Menon & Elkan, 2011; Cruz et al., 2016];

Tipicamente, o aprendizado de *ranking* envolve a análise de duplas, triplas e listas de instâncias [Wang et al., 2018]. Considerando o custo deste tipo de treinamento para listas de instâncias, poderíamos considerar o aprendizado de ordem baseado em pares ou triplas de instâncias, modificando adequadamente estas técnicas para tirar proveito das informações de múltiplos rótulos.

Finalmente, quanto à eficiência, processos de quantização separados do processo de aprendizagem de representação, para o aprendizado de códigos binários, podem levar a soluções sub-ótimas. Isso ocorre porque o modelo não foi criado para produção de códigos binários, de forma que aspectos sutis da representação podem se perder

em um processo de quantização que ocorre de maneira apartada da otimização feita durante a criação do modelo. Uma forma de lidar com este problema seria aprender uma representação binária (discreta) diretamente. Isto é difícil em modelos profundos baseados em redes neurais porque estes usam técnicas de otimização baseadas em funções diferenciáveis (contínuas). Uma forma de contornar este problema seria usar um modelo profundo de geração baseada em uma distribuição de probabilidades discretas, para o qual um processo de reparametrização diferenciável contínuo é conhecido. Este é o caso, por exemplo, do uso de modelos profundos conhecidos como auto-codificadores variacionais baseados em distribuição de Bernoulli [Dadaneh et al., 2020] ou Gumbel-Softmax [Silveira et al., 2018]. Entre estes, modelos baseados em distribuição de Bernoulli são vantajosos para o contexto de *hashing* por já produzirem representações binárias.

Considerando o exposto nesta seção, nesta pesquisa a seguinte hipótese será considerada para solucionar o problema proposto:

> **Hipótese**: *Um modelo de recuperação de imagem com múltiplos rótulos deve alcançar melhor desempenho computacional sem grande perda da qualidade de busca se for construído a partir de uma arquitetura de rede neural profunda (i) capaz de extrair eficientemente características de imagens explorando seus múltiplos rótulos, (ii) representando-as diretamente através de códigos binários em um espaço que facilite a utilização de algoritmos de vizinhança baseados em hashing e (iii) treinando o modelo em uma tarefa supervisionada de aprendizagem de ranking de imagens de custo razoável.*

Com base nessa hipótese, surgem as seguintes questões de pesquisa:

- Como garantir representações que possibilitem *hashing* eficiente em um cenário com múltiplos rótulos?
- Entre os modelos de representação obtidos através de aprendizagem de *ranking*, seria melhor usar aqueles baseados em pares ou triplas de instâncias, considerando imagens de múltiplos rótulos?
- Quão vantajoso é o aprendizado direto do código binário, usando um modelo gerador baseado em uma distribuição de Bernoulli em comparação com técnicas alternativas de quantização?
- Os modelos construídos obtém códigos binários que usam de forma mais efetiva o espaço de representação e tem propriedades de interesse como códigos com bits independentes e balanceados?

- Como os modelos propostos se comparam com métodos estado-da-arte quanto ao seu desempenho computacional e qualidade de recuperação?

1.4 Objetivos

O objetivo geral deste trabalho consiste em aprimorar técnicas de *hashing* profundo para tarefas de recuperação de imagem com múltiplos rótulos, aprendendo diretamente códigos binários por meio de um modelo gerador baseado em uma distribuição de Bernoulli.

1.4.1 Objetivos Específicos

Para alcançar o objetivo deste trabalho, os seguintes objetivos específicos foram identificados:

1. Avaliar o impacto de diferentes arquiteturas com respeito às formas de tratamento das imagens de entrada durante treino (pares de pontos ou trios de pontos) no desempenho da tarefa de recuperação de imagens em termos de qualidade e eficiência.

2. Explorar propriedades e características de *hashing* que garantam representações que possibilitem busca eficiente em um cenário com múltiplos rótulos;

3. Verificar o impacto do aprendizado direto do código binário, usando um modelo gerador baseado em uma distribuição de Bernoulli em comparação com técnicas alternativas de quantização;

4. Obter um modelo de *hashing* profundo para recuperação de imagens com múltiplos rótulos capaz de melhorar as propriedades de *hashing* que iremos considerar de interesse, mantendo a qualidade do *ranking* de resposta compatível ou superior ao estado-da-arte;

1.5 Contribuições

Entre as principais contribuições deste trabalho, citamos:

1. Uma descrição detalhada da literatura de recuperação de imagens com múltiplos rótulos usando *hashing* profundo [Rodrigues et al., 2020]. Esse trabalho, "*Deep hashing for multi-label image retrieval: a survey*", foi publicado no periódico

Artificial Intelligence Review. O Capítulo 3 descreve os trabalhos discutidos nesse *survey*.

2. Dois métodos de *hashing* profundo para imagens de múltiplos rótulos, treinados para aprender *ranking*. O primeiro, baseado em pares de instâncias, é capaz de aprender códigos binários diretamente. Este método é descrito no Capítulo 4 desta tese. O segundo, diferente do anterior, é baseado em triplas de instâncias e usa uma função de perda *triplet* modificada para lidar com múltiplos rótulos. Este trabalho é descrito no Capítulo 5 desta tese. Vale ressaltar que, diferente dos métodos anteriormente propostos na literatura, ambos os métodos aprendem códigos binários diretamente, por usar uma distribuição de Bernoulli que elimina a etapa de quantização. Além disso, estes modelos exploram e avaliam propriedades de *hashing* como balanceamento e independência de bits, com o intuito de melhorar a eficiência dos mesmos. Um artigo está sendo preparado com estes resultados para publicação em revista.

3. Uma coleção sintética, para estudo de métodos de recuperação de imagens com múltiplos rótulos, que possibilita ciclos rápidos de treino e avaliação. Este coleção é descrita no Capítulo 6 e será disponibilizada publicamente em um repositório do Github.

1.6 Estrutura da Tese

Além deste capítulo introdutório, esta tese está organizada como segue:

- O Capítulo 2 aborda os conceitos básicos necessários para compreensão desta pesquisa;
- No Capítulo 3, são discutidos os trabalhos relacionados ao problema de recuperação de imagem com múltiplos rótulos considerados relevantes para o contexto desta proposta;
- Os Capítulos 4 e 5 descrevem os modelos desenvolvidos nesta pesquisa;
- O Capítulo 6 apresenta o protocolo dos experimentos realizados bem como os resultados obtidos;
- O Capitulo 7 apresenta as conclusões e trabalhos futuros.

Capítulo 2

Fundamentação Teórica

A pesquisa desenvolvida neste trabalho está relacionada a duas grandes áreas. A primeira consiste na Recuperação de Imagem Baseada em Conteúdo. A segunda está relacionada às técnicas de Aprendizagem Profunda para representação da imagem e aprendizado de *ranking* com *hashing*. Portanto, para melhor entendimento desta tese, este capítulo apresenta conceitos introdutórios referentes a essas áreas. Ao final, são apresentadas as métricas mais comumente utilizadas para avaliar sistemas CBIR.

2.1 Recuperação de Imagem Baseada em Conteúdo

Segundo da Silva Torres & Falcao [2006], um sistema de recuperação de imagem baseada em conteúdo (*Content-Based Image Retrieval*, CBIR) tem como principal fundamento a noção de similaridade entre imagens, ou seja, dado um banco com um grande número de imagens, o usuário deseja recuperar as imagens mais similares a um padrão de consulta. O processo de recuperação é baseado na comparação das imagens por meio de descritores.

A Figura 2.1 mostra uma arquitetura típica de um sistema de recuperação de imagem baseado em conteúdo. Nesse esquema, existem duas funcionalidades principais: a inserção das imagens no banco de dados e o processamento da consulta.

Na parte de inserção dos dados o sistema é responsável por extrair vetores de características representativas das imagens e armazená-las no banco de imagens. Normalmente, esse processo é feito uma única vez para cada imagem e de forma *off-line*. Mais tarde esses vetores armazenados serão usados no processamento das consultas.

No processamento da consulta, a imagem também passa por um módulo que extrai os vetores de características da imagem. Em seguida, é aplicada uma métrica (Distância Euclidiana, por exemplo) para medir a similaridade entre a imagem de

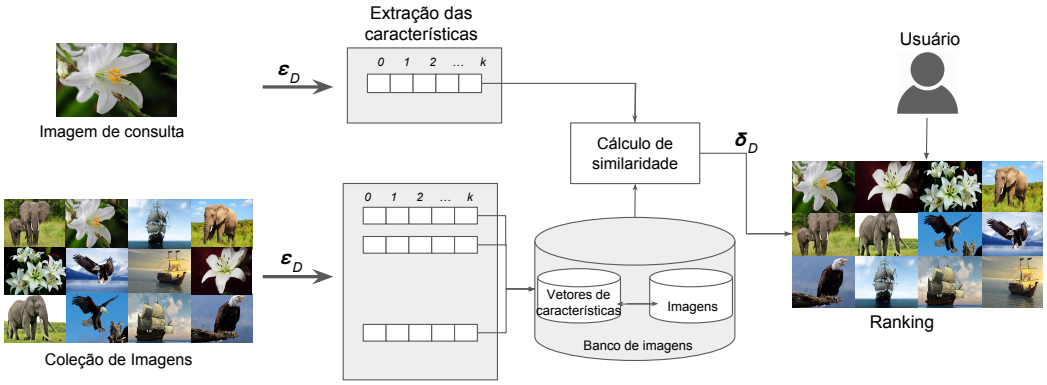


Figura 2.1. Arquitetura típica de um sistema CBIR. Figura adaptada de [Santos et al., 2016]

consulta e as imagens contidas no banco de imagens na etapa de cálculo de similaridade. Com base nessas medidas, é possível ordenar as imagens do banco de imagem em ordem decrescente de acordo com seu grau de similaridade com a imagem de consulta, e retornar as mais similares no *ranking* de resposta para o usuário.

2.1.1 Descritores de Imagem

O descritor de imagem em um sistema CBIR é o componente responsável por descrever as propriedades intrínsecas da imagem, tais como cor, forma e textura, que normalmente são representadas por vetores. Esse processo de descrição das características das imagens torna possível a comparação, e consequente indexação e busca de imagens.

No trabalho proposto por da Silva Torres & Falcao [2006], um descritor D é definido como um par (ϵ_D, δ_D), onde:

- ϵ_D - é a função responsável por extrair as características da imagem e representá-las em um vetor de códigos.
- δ_D - é a função de similaridade responsável por comparar dois vetores de características baseada em uma métrica de distância, definindo o quão similar são as duas imagens.

Por meio destas duas funções (ϵ_D, δ_D), um descritor D irá computar a similaridade ente duas imagens. Primeiro, a função de extração ϵ_D computa um vetor de características para cada imagem. Em seguida, a função de similaridade δ_D determina os valores de similaridade entre as imagens, conforme ilustrado na Figura 2.1. A extração das características mais representativas da imagem em conjunto com a utilização

de uma medida de similaridade adequada resultará em um descritor eficaz no processo de recuperação.

Além disso, é importante ressaltar que vários descritores podem ser combinados em um mais robusto, permitindo que múltiplas propriedades das imagens sejam codificadas ao mesmo tempo [da S. Torres et al., 2005].

Escolher o descritor mais adequado para uma determinada aplicação é crucial para o sucesso de um sistema CBIR. Por isso, é importante que sejam conduzidos experimentos comparativos utilizando diferentes descritores com o intuito de utilizar aquele que alcança o melhor desempenho.

Em CBIR tradicional, definir quais tipos de evidência serão codificadas nos vetores de características depende diretamente do contexto onde o descritor será aplicado. Cada evidência representa aspectos específicos da imagem que devem ser considerados de acordo com a necessidade da aplicação e, normalmente, são obtidos através de engenharia humana. Em geral, as evidências mais exploradas por métodos CBIR tradicional são: cor, forma e textura. Contudo, todos esses descritores requerem processos que envolvem engenharia humana. Ou seja, especialistas humanos projetam tais descritores para serem usados na representação das imagens, com base em seu conhecimento prévio e expectativa de utilidade de tais descritores. Este é um processo inerentemente difícil dado o enorme espaço de possíveis descritores de imagens e dos conceitos de alto nível percebidos por seres humanos nestas imagens.

Uma alternativa a este processo, que tem avançado muito em anos recentes, é o *aprendizado automático de representação*. Neste caso, as imagens, representadas de forma simples através de pixels, são dadas a algoritmos que determinam automaticamente os descritores mais adequados para representá-las de acordo com a tarefa a ser resolvida. A ideia mais bem sucedida é a da aprendizagem de representação hierárquica ou aprendizagem profunda, que consiste em obter descritores complexos da combinação de descritores mais simples, através de múltiplos níveis de combinação. A principal vantagem dessa abordagem é a seleção de descritores baseada em quão efetivos eles são na execução da tarefa que deve ser realizada.

De forma geral, o aprendizado de tais descritores envolve o uso de entradas formadas por pares de imagem (ou fragmentos de imagem) constituídas de imagens relacionadas e não relacionadas. Essa entrada é fornecida a um modelo de aprendizado profundo, normalmente uma rede neural profunda, que será usado para extrair descritores das imagens de entrada. Para tanto, o modelo é projetado com o propósito de fazer com que descritores de características semelhantes fiquem próximos enquanto descritores de características diferentes, distantes uns dos outros. Além disso, os modelos são projetados para serem robustos a vários tipos de transformações observadas

nas imagens.

A arquitetura de rede profunda mais utilizada para extrair automaticamente representações das imagens é a rede neural convolutiva, discutida na seção 2.2.3.

2.2 Aprendizagem Profunda

Aprendizagem Profunda é uma subárea de Aprendizagem de Máquina que investiga técnicas para simular o comportamento do cérebro humano em tarefas como reconhecimento visual, reconhecimento de fala e processamento de linguagem natural. Este tipo de solução permite que computadores aprendam a partir de experiências anteriores e compreendam o mundo em termos de uma hierarquia de conceitos, no qual os conceitos mais complexos são definidos e compreendidos em termos de sua relação com conceitos mais simples e já conhecidos [Goodfellow et al., 2016].

Ao ganhar conhecimento por meio de experiências, esta abordagem evita a necessidade de intervenção humana para especificar formalmente todo o conhecimento necessário ao processo de aprendizagem, *i.e.*, não há a necessidade da criação de estruturas de dados complexas para resolver problemas previamente conhecidos, mesmo que estes problemas tenham um grau de dificuldade alto. Para tanto, faz-se necessário construir e treinar uma grande rede neural artificial com muitas camadas ocultas entre a entrada e a saída da rede. É devido à existência de muitas camadas ocultas que esse tipo de rede neural é chamada de "profunda".

Com o intuito de entender o funcionamento dessas redes, a seguir são discutidos conceitos básicos relacionados às redes neurais. Em seguida, é apresentado uma breve discussão sobre redes de convolução e *hashing* profundo, que compõem a estrutura base do modelo proposto nesta pesquisa.

2.2.1 Redes Neurais Artificiais

Redes neurais artificiais (RNA) são inspiradas na forma como o cérebro humano processa informação. A comunicação entre os neurônios no cérebro se dá por meio da transmissão do sinal de um neurônio a outro definida por um processo químico, onde substâncias específicas são liberadas pelo neurônio transmissor. Em consequência disso há um aumento ou queda no potencial elétrico do neurônio receptor. Se esse potencial atingir o limite de ativação do neurônio, um sinal ou uma ação de potência é enviado para outros neurônios. De maneira similar ao cérebro humano, uma RNA define um modelo de neurônios artificiais, também conhecidos como unidades de processamento [Bezerra, 2016].

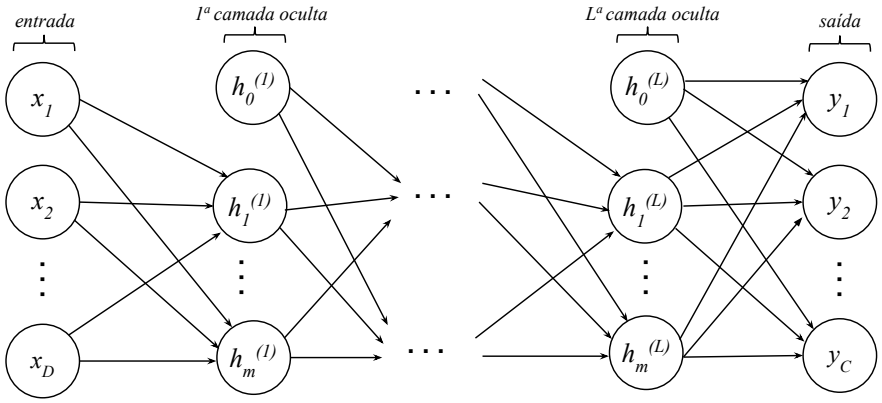


Figura 2.2. Arquitetura típica de uma RNA, contendo $(L + 1)$ camadas, D unidades na camada de entrada e C unidades na camada de saída. Fonte: [Bezerra, 2016]

A forma como os neurônios estão conectados define a arquitetura de uma RNA. A Figura 2.2 mostra a arquitetura típica de uma RNA, uma *Rede Multi-Layer Feed-forward*. Nela os neurônios são organizados em camadas consecutivas, e as conexões entre as unidades são ponderadas por valores reais denominados *pesos*. A camada que recebe os dados é chamada de *camada de entrada*, representada por um vetor $\mathbf{x}$ em um espaço de dimensão D. A última camada é chamada de *camada de saída* e representa o resultado final do processamento da rede. Na Figura 2.2, é denotado por um vetor $\mathbf{y}$ em um espaço de dimensão C. Entre as camadas de entrada e saída, pode haver uma sequência de L camadas, conhecidas como *camadas ocultas*.

É possível criar redes neurais mais simples, com apenas uma camada oculta de um único neurônio. No entanto, essas redes são muitos limitadas, por resolverem apenas problemas de natureza binária, diferente da arquitetura mostrada na Figura 2.2 que consegue resolver problemas mais complexos.

Para entender o funcionamento de uma rede neural, considere que cada neurônio de uma camada oculta recebe um vetor $\mathbf{x} = (x_1, x_2, ..., x_N)$ como entrada e realiza uma média ponderada pelos elementos de outro vetor de pesos associado entre essas camadas $\mathbf{w} = (w_1, w_2, ..., w_N)$. O valor que é gerado por essa computação conhecido, como *pré-ativação* do neurônio, é representado da seguinte forma:

$$a(\mathbf{x}) = b + \sum_i w_i x_i = \mathbf{b} + \mathbf{w}^T \mathbf{x} \tag{2.1}$$

onde $a(\mathbf{x})$ resulta em um número escalar e b é uma parcela adicional correspondente ao viés (*bias*), com peso constante (na Figura 2.2 corresponde às unidades h_0^l, $1 \leq l \leq L$).

Após calcular $a(\mathbf{x})$, o neurônio aplica uma transformação não linear sobre esse

valor por meio de uma *função de ativação*, $g(\cdot)$, da seguinte forma:

$$y = h(\mathbf{x}) = g(a(\mathbf{x})) \tag{2.2}$$

É importante ressaltar que diferentes funções de ativação resultam em diferentes comportamentos do neurônio. As funções mais comumente utilizadas são: sigmoide logística (Equação 2.3), sigmoide hiperbólica (Equação 2.4) e a função retificadora (Equação 2.5). Essas funções são normalmente utilizadas por serem todas diferenciáveis, fazendo a derivação da equação de atualização mais fácil (com exceção da ReLU em $z = 0$) e por serem funções estritamente crescentes. Além disso, uma característica importante das funções sigmoides é que elas mantêm a saída sempre entre 0 e 1.

$$g(z) = \frac{1}{1 + e^{-z}} \quad (2.3) \qquad g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \quad (2.4) \qquad g(z) = max(0, z) \quad (2.5)$$

Tanto a função de pré-ativação quanto as funções de ativação de um único neurônio podem ser estendidas para redes que contenham L camadas ocultas, com vários neurônios em cada camada. A Equação 2.6 mostra a função de pré-ativação considerando toda uma camada oculta, onde o valor de k denota a k-ésima camada intermediária, $\mathbf{W}^{(k)}$ é a matriz de pesos das conexões entre os neurônios da camada $k-1$ e k, e o uso do negrito indica que a Equação 2.1 está sendo aplicada a cada neurônio dessa camada. O mesmo ocorre para a Equação 2.7 da função de ativação.

$$\mathbf{a}^{(k)}(\mathbf{x}) = \mathbf{b}^{(k)} + \mathbf{W}^{(k-1)}(\mathbf{x}) \tag{2.6}$$

$$\mathbf{h}^{(k)}(\mathbf{x}) = g(\mathbf{a}^{(k)}(\mathbf{x})) \tag{2.7}$$

De forma geral, considerando um vetor $\mathbf{x}$ de entrada, uma rede neural computa uma função $f(\mathbf{x})$ tal que $f(\mathbf{x}) : \mathbb{R}^D \rightarrow \mathbb{R}^C$, de acordo com a Equação 2.8.

$$f(\mathbf{x}) = \mathbf{h}^{L+1}(\mathbf{x}) = \mathbf{o}(\mathbf{a}^{L+1}(\mathbf{x})) \tag{2.8}$$

A função $\mathbf{o}(\cdot)$ denota a função de ativação utilizada nas unidades da camada de saída. Quando o propósito de uma RNA é realizar a tarefa de classificação, normalmente seleciona-se para $\mathbf{o}(\cdot)$ a função *softmax*, uma vez que possibilita que os valores da saída sejam interpretados como probabilidades posteriores, pois geram valores em um intervalo entre 0 e 1 [Goodfellow et al., 2016]. A Equação 2.9 mostra o valor produzido pela *softmax* para o i-ésimo neurônio da camada de saída.

$$o(a_i^{L+1}) = \frac{e^{a_i^{L+1}}}{\sum_C^{j=1} e^{a_j^{L+1}}} \tag{2.9}$$

2.2.2 Treinando uma Rede Neural

Na fase de treinamento de um processo de aprendizagem supervisionada é utilizado um conjunto de dados da forma $\{(\mathbf{x}^{(t)}, \mathbf{y}^{(t)}) : 1 \leq t \leq T\}$, onde $\mathbf{x}^{(t)}$ é o vetor de características associado ao rótulo $\mathbf{y}^{(t)}$. Em redes neurais, a etapa de treinamento consiste em utilizar tal conjunto de treino com o propósito de ajustar os parâmetros da rede, de tal forma que o erro de treinamento seja minimizado [Bezerra, 2016].

Após cada entrada $\mathbf{x}^{(t)}$ ser aplicada à rede, a saída gerada é comparada com a saída desejada $\mathbf{y}^{(t)}$. O algoritmo deve ajustar os parâmetros da rede (pesos e limiares), com o objetivo de minimizar a função de custo, que mede a diferença que a rede comete relativamente ao seu rótulo $\mathbf{y}^{(t)}$. A Equação 2.10 mostra o custo relativo ao conjunto de treinamento, onde $J(f(\mathbf{x}^{(t)}; \theta), \mathbf{y}^{(t)})$ é a função de custo que se deseja minimizar e θ representa o vetor de parâmetros da RNA. Existem algumas funções de custo que podem ser usadas para medir o erro associado ao treinamento, como por exemplo a *soma dos erros quadrados* e a *entropia cruzada*.

$$\mathbf{J}(\theta) = \frac{1}{T} \sum_{t=1}^{T} J(f(\mathbf{x}^{(t)}; \theta), \mathbf{y}^{(t)}) + \frac{\lambda}{2T} \sum_k \sum_l W_{k,l}^2 \tag{2.10}$$

Na equação acima, a segunda parcela equivale ao custo de regularização contendo a soma dos quadrados de todos os pesos da RNA. O componente λ é outro hiperparâmetro da rede que controla a importância relativa entre as duas parcelas da função $\mathbf{J}(\theta)$. Considerando que o propósito do treinamento é minimizar $\mathbf{J}(\theta)$, não são desejados valores elevados de pesos, uma vez que poderá aumentar o valor final, ao invés de diminuir. Além disso, se o vetor de pesos ficar muito grande, o método de otimização que realiza pequenas alterações no vetor de pesos a cada iteração, não será capaz de alterá-lo significativamente, o que pode levar o modelo a convergir para uma solução inadequada [Bezerra, 2016].

Outro problema que uma rede neural pode sofrer no processo de treinamento é o fenômeno conhecido como sobreajuste (*overfitting*). Quando o conjunto de treino disponível não é grande o suficiente, a rede pode memorizar todos os exemplos da entrada, fazendo com que o modelo gerado não tenha uma boa capacidade de generalização. Uma forma de evitar esse tipo de problema é utilizar técnicas de regularização. Na Equação 2.10, a segunda parcela é uma das possíveis formas de evitar que o processo de otimização fique preso a uma solução sub-ótima.

2.2.2.1 Gradiente Descendente

Qualquer algoritmo de otimização pode ser usado para o treinamento de uma RNA. Entretanto, o *gradiente descendente* (*gradient descent*, GD) é um dos algoritmos mais populares para solucionar problemas de otimização e é a maneira mais comum de otimizar redes neurais. Basicamente, é uma forma de minimizar a função de custo J, atualizando os parâmetros na direção oposta do gradiente da função de custo (gradiente negativo da função) $-\nabla J$. A ideia desse algoritmo é realizar, de forma iterativa, pequenas modificações no vetor de parâmetros θ com o intuito de levar esse vetor na direção da maior descida em uma superfície multidimensional representada por J.

Considere um vetor de pesos $w^{(t)}$. A Equação 2.11 mostra a atualização desse vetor usando a informação do gradiente a cada iteração. O parâmetro $\eta > 0$ é conhecido como taxa de aprendizado (*learning rate*). Após cada atualização, o gradiente é reavaliado para o novo vetor de pesos e o processo é repetido [Bishop, 2006]. Observe que a função de erro é definida em relação ao conjunto de treinamento, e cada passo requer que todo o conjunto de treino seja processado para que o $\nabla J(w^{(t)})$ seja avaliado. Técnicas que usam todo o conjunto de dados de uma só vez são conhecidas como *batch*. No entanto, existe uma variação desse algoritmo em que a atualização do vetor de pesos é feita com base em apenas um dado, chamado de gradiente descendente estocástico (*stochastic gradient descent*) [Bishop, 2006].

$$w^{(t+1)} = w^{(t)} - \eta \nabla J(w^{(t)}) \tag{2.11}$$

2.2.2.2 Retropropagação do Erro

Um aspecto importante na Equação 2.11 é encontrar uma forma eficiente de avaliar o gradiente da função de erro $J(w)$. Isso pode ser alcançado por um esquema em que a informação calculada é enviada alternativamente para frente e para trás através da rede. Esse algoritmo, proposto por Rumelhart et al. [1986], é conhecido como retropropagação do erro (*backpropagation error*).

Nesse algoritmo, se a saída produzida pela rede para uma determinada entrada $\mathbf{x}^{(t)}$ é diferente da desejada, então é necessário determinar o grau de responsabilidade de cada parâmetro da rede nesse erro para, em seguida, alterar esses parâmetros com o objetivo de reduzir o erro produzido.

Segundo Bishop [2006], o algoritmo de retropropagação é realizado como segue.

1. Propague o vetor de entrada através de todas as camadas da rede, usando as funções de pré-ativação (Equação 2.6) e ativação (Equação 2.7), com o propósito

de obter a saída de cada unidade.

2. Calcule o erro $\delta_i^{(L+1)}$ para as unidades de saída:

$$\delta_i^{(L+1)} = \frac{\partial J_n}{\partial y_i^{(L+1)}} f'(z_i^{(L+1)}) \tag{2.12}$$

3. Determine $\delta_i^{(l)}$ para todas as camadas escondidas l usando a retropropagação do erro:

$$\delta_i^{(l)} = f'(z_i^{(l)}) \sum_{k=1}^{m^{(i+1)}} w_{i,k}^{(l+1)} \delta_k^{(l+1)} \tag{2.13}$$

4. Calcule as derivadas requeridas:

$$\frac{\partial J_n}{\partial w_{j,i}^{(l)}} = \delta_j^{(l)} y_i^{(l-1)} \tag{2.14}$$

5. Atualize os pesos (Equação 2.11).

O algoritmo de retropropagação é uma aplicação da regra da cadeia para computar os gradientes de cada camada de uma RNA de forma iterativa. Detalhes do processo de derivação das funções podem ser verificados em [Bishop, 2006].

2.2.3 Rede Neural Convolutiva

Como mencionado anteriormente, com o avanço das redes neurais profundas em representação de imagem, métodos de *hashing* a passaram utilizar aprendizado profundo no processo de geração de código binário. Os métodos que propõem abordagens de *hashing* profundo para recuperação de imagem, utilizam Redes Neurais de Convolução para fazer extração das características das imagens.

Redes neurais convolutivas (*Convolutional Neural Network, CNN*) é um caso especial da rede neural. Elas são inspiradas no funcionamento do córtex visual de mamíferos [Hijazi et al., 2015]. A Figura 2.3 apresenta a arquitetura típica de uma rede de convolução. Como mostrado nesse esquema, uma CNN, em geral, é formada por três tipos de camadas: camadas de convolução, camadas de subamostragem (*pooling*) e camada completamente conectada. Normalmente, essas camadas são organizadas em estágios. Cada estágio pode ser formado por uma ou mais camadas de convolução, seguidas por uma camada de subamostragem. No estágio final da rede, são adicionadas uma ou mais camadas completamente conectadas. A seguir, é apresentado um resumo de cada uma dessas camadas.

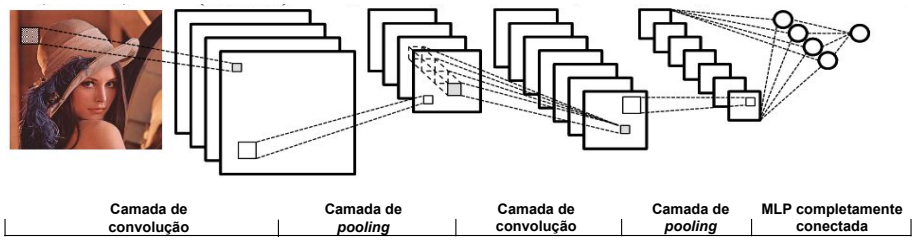


Figura 2.3. Arquitetura de uma Rede de Convolução.

2.2.3.1 Camada de Convolução

Diferente das arquiteturas de redes convencionais, que tradicionalmente são completamente conectadas e não levam em consideração a estrutura espacial que pode haver nos dados (que é o caso das imagens), as CNNs utilizam uma conectividade local. Isso quer dizer que cada neurônio de uma camada está conectado a uma quantidade restrita de unidades localizadas na camada adjacente. Além disso, as unidades das camadas de uma CNN são organizadas em conjuntos distintos, os chamados mapas de características (*feature map*). Cada mapa é responsável por uma região específica da imagem, e as unidades que formam um mapa de características compartilham os mesmos parâmetros (viés e pesos) e estão conectadas a um conjunto de unidades diferente na camada anterior. Isso permite que as unidades dentro de um mapa detectem uma mesma característica, no entanto, em regiões diferentes da imagem Bezerra [2016]. No esquema da Figura 2.3, a camada de entrada é seguida por uma camada de convolução composta por 4 mapas de características. Cada unidade em um mapa de característica realiza uma operação de convolução.

2.2.3.2 Camada de Subamostragem

A camada de subamostragem reduz a dimensionalidade do mapa de características fornecido como entrada e produz um outro mapa de característica, uma espécie de resumo do primeiro. Esse processo torna as características mais robustas contra ruídos e distorções. Existem duas maneiras de aplicar subamostragem: usando seleção por média (*average pooling*) e por valor máximo (*max pooling*). Para a subamostragem por média, calcula-se a média dos quatro valores na região e por valor máximo, o valor máximo dos quatro valores é selecionado [Hijazi et al., 2015]. Em ambos os casos, a entrada é dividida em espaços bidimensionais que não se sobrepõem. Na Figura 2.3, a camada 2 é a camada de subamostragem. Observe que cada mapa de característica dado como entrada tem seu tamanho reduzido na próxima camada.

2.2.3.3 Camada Completamente Conectada

Camadas completamente conectadas são frequentemente usadas como as camadas finais de uma CNN (na Figura 2.3 é representada pela última camada) e são responsáveis por conectar as camadas anteriores sem utilizar pesos compartilhados. Juntamente com as camadas de convolução e subamostragem, a camada completamente conectada gera descritores de características das imagens que podem ser facilmente classificadas pela camada de saída, usando uma função *softmax*, por exemplo [Stutz, 2014].

2.2.4 Arquiteturas de CNN

As arquiteturas de redes neurais convolutivas mais utilizadas pelos métodos de *hashing* profundo propostos para resolver o problema de recuperação de imagem são: AlexNet [Krizhevsky et al., 2012], GoogLeNet [Szegedy et al., 2015], VGG [Simonyan & Zisserman, 2015] e Residual Network [Li et al., 2015]. A seguir são apresentadas as principais características dessas arquiteturas.

2.2.4.1 AlexNet

AlexNet, proposta por Krizhevsky et al. [2012], foi uma das primeiras redes profundas a melhorar significativamente a acurácia de classificação na ImageNet. A arquitetura da AlexNet é formada por 5 camadas de convolução seguidas de 3 camadas completamente conectadas. Como mostra a Figura 2.4, a primeira camada dessa rede usa *kernels* de tamanho 11x11, a segunda usa *kernels* 5x5 e as restantes usam *kernels* 3x3, resultando em cerca de 65 milhões de parâmetros. Um único passo de *forward* na rede implica em cerca de 2.5 bilhões de operações [Canziani et al., 2016].

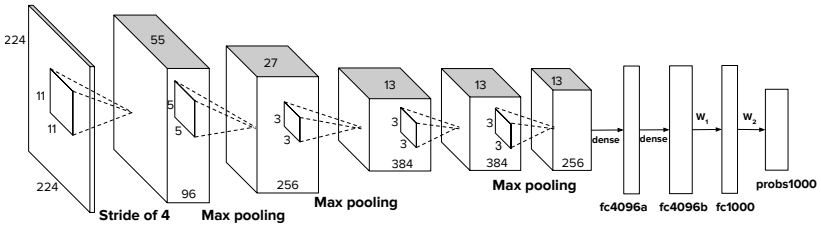


Figura 2.4. Arquitetura da AlexNet.

2.2.4.2 VGG

Com a VGGNet [Simonyan & Zisserman, 2015], os autores exploraram principalmente o impacto que o número de camadas teria sobre o desempenho da rede. Ao contrário

da AlexNet, sua arquitetura era bastante homogênea com as camadas usando sempre *kernels* 3x3 e *pooling* 2x2. Embora tenham concluído que um maior número de camadas implica em um melhor desempenho, os modelos propostos foram caracterizados por um grande número de parâmetros e operações, o que a torna particularmente cara tanto em tempo de treino quanto teste. A VGG19, por exemplo, tem cerca de 155 milhões de parâmetros, o que resulta em cerca de 40 bilhões operações no passo de *forward* [Canziani et al., 2016].

2.2.4.3 GoogLeNet

Como a VGGNet, a GoogleLeNet [Szegedy et al., 2015] e suas variantes também exploraram modelos mais profundos. Ao contrário de arquiteturas anteriores, contudo, os seus autores optaram por estruturas não estritamente lineares, com a introdução de módulos Inception, onde *kernels* com diferentes campos visuais são aplicados em paralelo. Por combinarem pilhas de pequenos *kernels* cuidadosamente arranjadas, estes módulos usam um número reduzido de parâmetros. Além disso, uma redução ainda mais significativa no número de parâmetros é alcançada pela substituição de camadas completamente conectadas por camadas de *Average Pooling*. Como resultado, a GoogleLeNet atinge uma acurácia significativamente maior que a AlexNet usando apenas 4 milhões de parâmetros, sem aumentar significativamente o número de operações em um passo de *forward* [Canziani et al., 2016]. Variantes subsequentes do modelo, como a Inception v3 [Szegedy et al., 2016], aumentaram consideravelmente o desempenho da arquitetura com números razoáveis de parâmetros e operações (30 milhões de parâmetros e 12 bilhões de operações).

2.2.4.4 Rede Residual

A principal característica das redes residuais [Li et al., 2015], mais conhecidas como ResNets, é a introdução de conexões *skip*, que permitem ao sinal saltar entre camadas. As conexões *skip* tornam a rede mais robusta ao problema de *vanishing gradients* e facilitam o aprendizado, o que resulta na possibilidade de modelos mais profundos, com centenas de camadas. Como outras arquiteturas contemporâneas, ela faz uso intenso de *batch normalization*, evita camadas completamente conectadas e prefere pilhas de pequenos *kernels*. Um modelo com 152 camadas (Resnet152) possui 65 milhões de parâmetros e usa 24 bilhões de operações em um passo de *forward* [Canziani et al., 2016]. Modernas variantes de redes residuais atingem desempenho estado-da-arte em uma variedade de problemas envolvendo redes de convolução.

2.2.5 Redes Autocodificadoras

Em aprendizado não supervisionado, dado um conjunto de treino $\{x_n : 1 \leq n \leq N\}$ sem o valor do rótulo desejado, a rede é treinada para encontrar semelhanças e regularidades dentro dos dados por si só. O treinamento não supervisionado de arquiteturas profundas pode ser realizado com base em Máquinas Restritas de *Boltzman* ou redes autocodificadoras [Bengio et al., 2009]. Este trabalho focará em redes autocodificadoras.

Redes autocodificadoras (*Autoencoder Network*), também chamadas de redes autoassociativas [Bengio et al., 2009], são uma classe de rede neural que pode ser treinada para aprender, de forma não supervisionada, características a partir de um conjunto de dados. Essas características são úteis para o uso posterior em tarefas de aprendizado, tais como reconhecimento de objetos e outras tarefas relacionadas à visão computacional, como recuperação de imagem.

A Figura 2.5 ilustra um esquema de rede autocodificadora. Nesse tipo de rede, há pelo menos uma camada intermédia, onde é computada uma representação dos dados $c(x)$ a partir de uma entrada x. A camada de saída y tenta reconstruir a entrada a partir da representação dada por $c(x)$. Observe que tanto a entrada x, quanto a saída $y \in \mathbb{R}^D$.

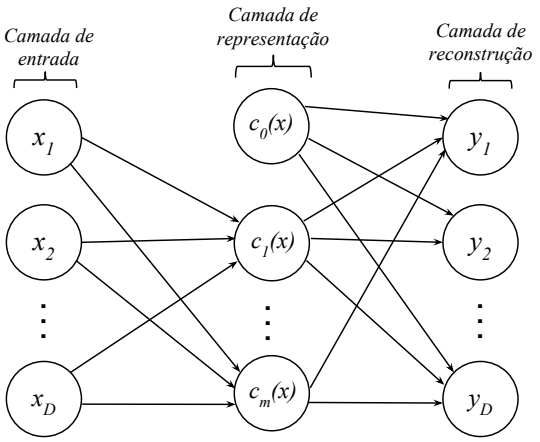


Figura 2.5. Esquema de uma rede autocodificadora. Figura adaptada de [Stutz, 2014]

Em redes autocodificadoras são realizadas duas transformações na entrada de acordo com dois tipos de função, como a seguir [Bezerra, 2016]:

- Função de extração de características, $h : \mathbb{R}^D \rightarrow \mathbb{R}^M$. Essa função conhecida como codificadora (*encoder*) mapeia as variáveis do conjunto de treino para uma representação latente.

- Função de reconstrução, $r : \mathbb{R}^M \rightarrow \mathbb{R}^D$. Essa função, conhecida como decodificadora (*decoder*), mapeia a representação produzida por h de volta para o espaço original.

Segundo Bezerra [2016], uma escolha comum para as funções h e r, principalmente quando o vetor de entrada $\mathbf{x}$ é binário, é utilizar a função sigmoide aplicada à transformação linear correspondente à multiplicação de $\mathbf{x}$ pela respectiva matriz de pesos. Durante o treinamento de uma rede autocodificadora, o objetivo é definir um conjunto de parâmetros no par de funções de h e r tal que o erro de reconstrução (i.e., a diferença entre $\mathbf{x}$ e $\mathbf{y}$), seja minimizado. Esse erro é representado pela função L apresentada na Equação 2.15. Dependendo da arquitetura da rede e da natureza do conjunto de treinamento, podem ser feitas diferentes escolhas para $\mathrm{L}(\cdot)$. Para dados binários, pode ser usada a entropia cruzada (*cross entropy*), enquanto que a média de erros quadrados (*mean squared error*) é popular para dados contínuos. Como a saída de uma autocodificadora deve ser similar a sua entrada, ela pode ser treinada conforme discutido anteriormente (usando o algoritmo de retropropagação, combinando algum procedimento de otimização), com o cuidado de substituir os valores-alvo desejados pela própria entrada $\mathbf{x}$.

$$L(x) - \sum_{x^{(i)} \in X} l(x^{(i)}, r(h(x^{(i)}))) \tag{2.15}$$

Idealmente, os parâmetros aprendidos durante o treinamento devem resultar em uma rede autocodificadora que identifique uma representação latente dos dados da entrada por meio de uma redução de dimensionalidade, ao mesmo tempo que contém o máximo de informação acerca da distribuição de entrada.

2.2.6 Transferência de Aprendizagem

Para gerar códigos compactos, métodos de *hashing*, normalmente, adicionam uma camada completamente conectada no final de uma rede convolutiva, conforme é ilustrado na Figura 2.6. O tamanho dessa camada é definido de acordo com o tamanho do código *hash* que se deseja gerar, cada bit é gerado por um neurônio dessa camada. Como esta camada é conectada normalmente a uma CNN, é possível que o treinamento do modelo como um todo inicie com a CNN pré-treinada em um problema relacionado. Esta é uma prática comum conhecida como *transferência de aprendizagem*. No aprendizado por transferência basicamente tenta-se explorar o que foi aprendido em uma tarefa para melhorar a generalização em outra, uma vez que espera-se que a representação aprendida nas primeiras camadas de um modelo deve variar pouco entre tarefas rela-

cionadas como classificação e busca. Os pesos de uma rede treinada em uma tarefa A são transferidos para uma nova tarefa B. Isso permite treinar redes neurais profundas com relativamente poucos dados, o que é muito útil, pois para a maioria dos problemas pode haver pouca disponibilidade de dados rotulados. Nos métodos de recuperação baseados em *hashing* profundo, a transferência de aprendizagem normalmente é feita inicializando o modelo com pesos pré-treinados na tarefa de classificação usando a ImageNet [Deng et al., 2009]. Uma desvantagem desta abordagem é que os pesos do modelo não são otimizados exclusivamente para a tarefa de busca, o que pode resultar em uma representação binária sub-ótima.

2.2.7 *Hashing* Profundo

Embora existam muitos métodos de *hashing* baseados em aprendizagem na literatura [Gong et al., 2013a, 2012; Lin et al., 2013; Norouzi & Blei, 2011; Wang et al., 2012], a maioria deles visa aprender uma única matriz de projeção W, que pode não capturar efetivamente as relações não lineares das amostras. Com o avanço das redes neurais profundas em representação de imagem, trabalhos nessa área passaram a utilizar tais redes no processo de extração das características. Isso ocorre porque as redes neurais aprendem naturalmente múltiplas transformações não lineares para alcançar códigos binários compactos e eficientes.

A Figura 2.6 mostra uma visão geral da arquitetura de métodos de *hashing* baseados em aprendizagem profunda. No esquema, para uma dada amostra $\mathbf{x}_n$, um vetor de códigos binário $\mathbf{y}_n$ é obtido através de uma rede que contém múltiplas camadas de transformações não lineares.

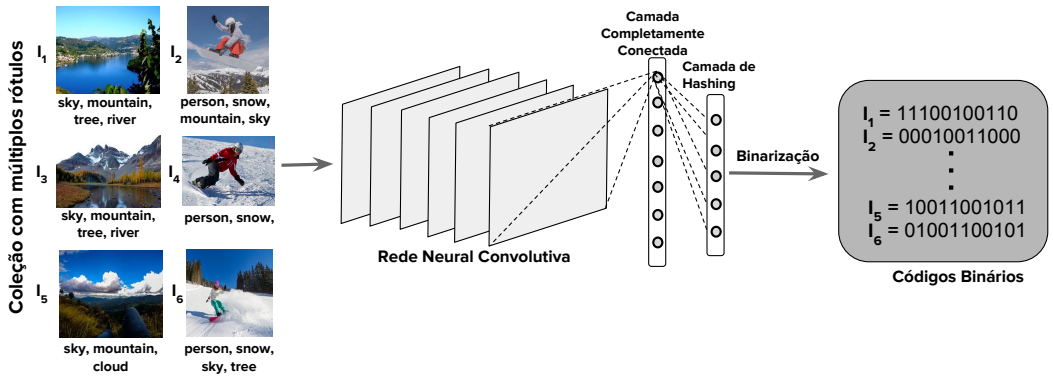


Figura 2.6. Visão geral de um modelo de *hashing* profundo. Dada uma coleção de imagens, métodos *hashing* profundo extraem características das imagens através de uma rede neural profunda. Na saída da rede algum processo de binarização é aplicado sobre as características resultantes formando assim códigos binários.

Nesse esquema, assuma que existam $M + 1$ camadas na rede profunda, e p^m unidades para a m-ésima camada, onde $m = 1, 2, 3, ..., M$. Para uma dada amostra $\mathbf{x}_n \in \mathbb{R}^d$, a saída da primeira camada é: $h_n^1 = s(W^1\mathbf{x}_n + c^1) \in \mathbb{R}^{p^1}$, onde $W^1 \in \mathbb{R}^{p^1 \times d}$ é a matriz projeção a ser aprendida na primeira camada da rede, $c^1 \in \mathbb{R}^{p^1}$ é o viés, e $s(\cdot)$ é uma função de ativação não linear. A saída da primeira camada é então considerada como entrada para a segunda camada, de modo que $h_n^2 = s(W_2\mathbf{h}_n^1 + c^2) \in \mathbb{R}^{p^2}$, onde $W^2 \in \mathbb{R}^{p^2 \times p^1}$ e $c^2 \in \mathbb{R}^{p^2}$ são a matriz de projeção e o vetor de viés da segunda camada, respectivamente. Da mesma forma, a saída da m-ésima camada é: $h_n^m = s(W_m\mathbf{h}_n^{m-1} + c^m)$, e a saída da última camada da rede é descrita como:

$$f_{DH} = \mathbf{h}^M = s(W^M\mathbf{h}_n^{M-1} + c^M) \tag{2.16}$$

onde o mapeamento $f_{DH} : \mathbb{R}^d \to \mathbb{R}^{p^M}$ é parametrizado por $\{W^m, c^m\}_{m=1}^M$, $\mathbf{h}^M$ é definida como o código compacto de valores reais aprendidos a partir de muitas transformações não lineares das características originais. Finalmente, o código binário é obtido a partir da saída h^M da última camada da rede, como segue:

$$\mathbf{y}_n = sgn(h_n^M) \tag{2.17}$$

A forma como o problema de otimização da rede será realizado dependerá de cada método. No Capítulo 3, veremos que existem diferentes categorias de método de *hashing* profundo para CBIR com múltiplos rótulos. Cada uma dessas categorias poderá adotar diferentes formatos de entrada, funções de perda e arquiteturas de redes profundas.

2.2.7.1 Função de Perda

Em *hashing* profundo, a função de perda utilizada como objetivo de otimização [Wang et al., 2018] é normalmente projetada para preservar a ordem de similaridade entre as amostras de treinamento, i.e, minimizar a diferença entre o espaço original de entrada e o espaço de codificação, que será usado na busca aproximada do vizinho mais próximo. A forma como as imagens a serem avaliadas pela função de perda são tratadas definem diferentes categorias de métodos.

Em particular, métodos denominados *pointwise* (Seção 3.2) adotam funções de perda que não comparam imagens entre si, avaliando uma imagem de acordo com como seus rótulos previstos se comparam com os rótulos reais. Métodos *pairwise* (Seção 3.3) usam funções de perda que calculam a distância entre pares de imagens, dadas como consulta e exemplo de resposta. Por fim, os métodos *tripletwise* (Seção 3.4) e *listwise*

(Seção 3.5) adotam funções que verificam em que medida são preservadas a similaridade entre múltiplas imagens (três no caso triplewise), fazendo com que a ordem entre itens computados a partir do espaço de codificação seja consistente com a ordem gerada no espaço de entrada.

Estas funções podem ser normalmente constituídas por componentes adicionais que tem o intuito de melhorar a representação do código. Em modelos de *hashing* é comum incluir a perda de quantização como um destes componentes para garantir que os números reais gerados pela rede de convolução produzam códigos binários adequados. Por exemplo, se os códigos binários são formados por $\{0, 1\}$ é desejável que os valores fiquem o mais próximo possível do intervalo $[0, 1]$, de forma que ao aplicar o processo de binarização pouca informação se perca.

2.2.7.2 Busca Baseada em *Hashing*

Considere $\mathcal{X} = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N] \in \mathbb{R}^{d \times N}$ um conjunto de N amostras (imagens), onde $\mathbf{x}_n \in \mathbb{R}^d$ $(1 \leq n \leq N)$ é a n-ésima amostra em $\mathcal{X}$. Considere ainda um conjunto de classes $\mathcal{L} = \{1, ..., C\}$, tal que cada $\mathbf{x}_n \in \mathbb{R}^d$ é associada a um subconjunto de rótulos $\mathcal{Y}_n \subseteq \mathcal{L}$. Métodos de *hashing* baseados em aprendizagem profunda buscam aprender múltiplas funções de *hashing* para mapear e quantizar cada amostra em códigos binários compactos, preservando a estrutura semântica dos dados inerente tanto aos seus conteúdos visuais quanto aos seus múltiplos rótulos. Dessa forma, podemos assumir que existem K funções de *hashing* a serem aprendidas, que mapeiam cada $\mathbf{x}_n$ com rótulos $\mathcal{Y}_n$ a um vetor de código binário de K-bits $\mathbf{y}_n = [y_{n1}, y_{n2}, ..., y_{nK}] \in \{-1, 1\}^{K \times 1}$. O k-ésimo bit $\mathbf{y}_{nK}$ de $\mathbf{x}_n$ é calculado como:

$$\mathbf{y}_{nk} = h_k(\mathbf{x}_n) = sgn(g_k(\mathbf{x}_n)) = sgn(\mathbf{w}_k^T \mathbf{x}_n) \tag{2.18}$$

onde h_k é a k-ésima função de *hashing*, $w_k \in \mathbb{R}^d$ é a projeção em h_k e $sgn(v)$ é definido como segue:

$$sgn(v) = \begin{cases} 1, & \text{se } v \geq 0 \\ -1, & \text{caso contrario} \end{cases} \tag{2.19}$$

Observe que o código *hash* não necessariamente será formado por $\{-1, 1\}$. Alguns métodos adotam códigos formados por $\{0, 1\}$ [Liu et al., 2017]. O formato dependerá da função utilizada. Dessa forma, neste trabalho, usaremos o termo códigos *hash* para referenciar tanto a forma $\{0, 1\}$ como $\{-1, 1\}$.

Finalmente, considerando $\mathbf{W} = [w_1, w_2, ..., w_K] \in \mathbb{R}^{d \times N}$ a matriz projeção, o mapeamento de $\mathbf{x}_n$ pode ser calculado como: $g(\mathbf{x}_n) = \mathbf{w}_k^T \mathbf{x}_n$, que pode ser então

binarizado para obter códigos da seguinte forma:

$$\mathbf{y}_n = sgn(\mathbf{w}_k^T \mathbf{x}_n) \tag{2.20}$$

Após obter os códigos binários, a busca dos vizinhos mais próximos a uma consulta pode ser realizada no espaço de Hamming. A distância de Hamming entre dois códigos binários $\mathbf{y}_i$ e $\mathbf{y}_j$ é definida como:

$$d_{\mathcal{H}}(\mathbf{y}_i, \mathbf{y}_j) = \mid \mathbf{y}_i - \mathbf{y}_j \mid = \sum_{k=1}^{K} \mid h_k(\mathbf{x}_i) - h_k(\mathbf{x}_j) \mid \tag{2.21}$$

onde $\mathbf{y}_i = [h_1(\mathbf{x}_i), ..., h_k(\mathbf{x}_i), ..., h_K(\mathbf{x}_i)]$ e $\mathbf{y}_j = [h_1(\mathbf{x}_j), ..., h_k(\mathbf{x}_j), ..., h_K(\mathbf{x}_j)]$. Observe que a distância de Hamming pode ser calculada de forma eficiente usando operações lógicas bit a bit. Dessa forma, realizar buscas exaustivas em um espaço de Hamming pode ser significativamente mais rápido que no espaço original [Wang et al., 2016a]. Além disso, o processo de recuperação pode ser ainda mais eficiente dependendo da complexidade da rede neural utilizada.

2.3 Coleções de Dados de Referência

Por lidarem com o problema de recuperação de imagens com vários rótulos, os métodos da literatura normalmente usar os seguintes coleções de dados de referência para avaliar seus modelos:

- MIRFLICKR-25K [Huiskes & Lew, 2008] consiste em 25.000 imagens obtidas no site Flickr, bem como as tags associadas. Essas imagens são anotadas com 24 conceitos que incluem categorias de objetos (pássaro, árvore, pessoas) e categorias de cena (interior, céu, noite). Outros 14 conceitos foram incluídos para categorização mais rigorosa. Cada imagem é anotada com um conceito apenas se for saliente. No entanto, um total de 38 imagens são usadas para anotar cada imagem e algumas delas são anotadas com vários rótulos. Cada imagem tem em média 4,7 rótulos;

- NUS-WIDE [Chua et al., 2009] contém 269.648 imagens também coletadas do site de compartilhamento de mídia social Flickr. É um conjunto de dados multi-rótulo no qual cada imagem é anotada com um ou mais rótulos de classe de um total de 81 classes, tais como: céu, pessoas, oceano, etc;

- VOC 2007 [Everingham et al., 2010] consiste em 9.963 imagens multi-rótulo coletadas do site Flickr. Neste conjunto de dados existem 20 classes de objetos

(como avião, pássaro, bicicleta, pessoa). Em média, cada imagem é anotada com 1,5 rótulos;

- VOC 2012 [Everingham et al., 2010] é um conjunto de dados amplamente utilizado para detecção e segmentação de objetos, contendo 17.125 imagens, onde cada imagem pertence a pelo menos um dos 20 rótulos semânticos;;
- MS COCO [Lin et al., 2014] é uma coleção de dados para reconhecimento de imagem, segmentação e legendagem. Consiste em um conjunto de treinamento de 83 mil imagens e um conjunto de validação de 40 mil imagens. Existem 80 categorias de anotações com várias rótulos por imagem;
- APRTC12 [Grubinger et al., 2006] é composta por 19.627 imagens com 275 rótulos para tarefa de segmentação. As imagens coletadas neste conjunto de dados incluem muitos aspectos da vida contemporânea, tais como: pessoas, natureza, cidades e paisagens.

2.4 Considerações Finais

Neste capítulo foi realizada uma síntese dos principais conceitos e técnicas que serão empregados no desenvolvimento deste trabalho. Foram abordados aspectos importantes de sistema de recuperação de imagem baseada em conteúdo, seu funcionamento e principais componentes, bem como elementos necessários para o entendimento de redes neurais profundas e técnicas de *hashing*.

A seguir, no Capítulo 3, são apresentadas as principais abordagens da literatura que investigam o problema de recuperação de imagem baseada em conteúdo.

Capítulo 3

Trabalhos Relacionados

Neste capítulo, são apresentadas soluções propostas na literatura para o problema de recuperação de imagem com múltiplos rótulos baseada em *hashing* profundo. Este Capítulo é resultado de um *survey* e por isso os métodos são descritos separadamente em subseções.

3.1 Categorização dos Métodos

Segundo Wang et al. [2016a], métodos de *hashing* podem ser categorizados considerando alguns critérios, tais como:

- Baseado na construção de funções de *hashing*, se requer ou não análise de um determinado conjunto de dados: *dependentes* e *independentes dos dados*;
- Baseado no paradigma de aprendizagem de máquina utilizado: *supervisionado*, *semi-supervisionado* ou *não-supervisionado*;
- Baseado no nível de supervisão dos métodos: *pointwise*, *pairwise*, *tripletwise* e *listwise*;
- Baseado na forma da função de *hashing* utilizada: *linear* ou *não linear*.

Os métodos relacionados à esta pesquisa utilizam aprendizado profundo no processo de geração do *hashing*, consequentemente todos são dependentes de dados por possuírem etapa de treinamento. Além disso, como mencionado na Seção 2.2.7, redes neurais aprendem naturalmente múltiplas transformações não lineares para alcançar códigos binários. E como todos os métodos são supervisionados, a classificação que faz uma melhor caracterização dos métodos é a baseada no nível de supervisão, que é a categorização adotada neste Capítulo.

3.2 Métodos *Pointwise*

Métodos dessa categoria adotam rótulos de classes das amostras de treinamento para aprender funções de classificação de objetos individuais (sem fazer nenhum tipo de comparação relativa entre imagens). Normalmente, modelam apenas a ordem relativa de amostras de dados para categorias diferentes, mas não as da mesma categoria. Na Figura 3.1, é apresentada uma visão geral dos métodos que utilizam esse tipo de solução. No esquema, uma imagem é dada como entrada para uma CNN, onde as características extraídas são aprendidas de acordo com os rótulos presentes na imagem.

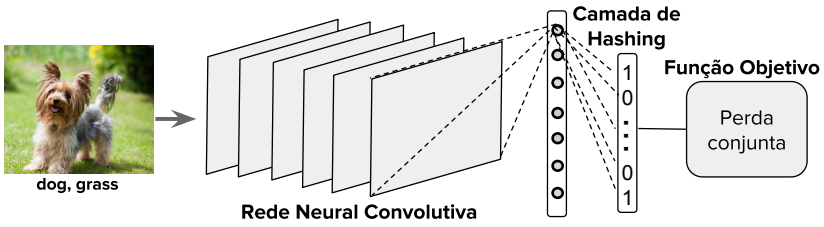


Figura 3.1. Visão geral de métodos da categoria *pointwise*.

Veremos a seguir que métodos dessa categoria costumam utilizar funções de perda combinadas para trabalhar o problema de múltiplos rótulos. Ou ainda combinar informações (além da representação da imagem gerada pela rede neural) na função de perda, para modelar similaridades entre os rótulos de diferentes imagens.

3.2.1 *Direct Binary Embedding*

O modelo *Direct Binary Embedding* (BDE) [Liu et al., 2017] treina uma ResNet50 como função de *hashing* via tarefa de classificação. Esse método visa aprender uma representação o mais próxima possível de um código binário de forma a dispensar uma técnica de quantização e, por conseguinte, evitar o erro inerente ao uso da quantização. Os autores tiram proveito da informação associada aos múltiplos rótulos ao estender a entropia cruzada de forma que ela capture as dependências de coocorrência entre os rótulos ainda que mantendo a independência de cada rótulo.

Seja $\mathbf{I} = I_{i i=1}^{N}$ o conjunto de imagens com N amostras associadas a um conjunto de rótulos $\mathbf{Y} = \{y_i\}_{i=1}^{C}$. O objetivo do modelo é minimizar a seguinte função de perda:

$$\begin{aligned}\min_{W,F} &- \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{c_+}\frac{1}{c_+}log\frac{e^{w_j^\intercal F(I_i;\Omega)}}{\sum_{p=1}^{C}e^{w_p^\intercal F(I_i;\Omega)}} \\ &- v\frac{1}{N}\sum_{i=1}^{N}\sum_{p=1}^{C}\Bigg[\rho\mathbb{1}(y_i)log\frac{1}{1+e^{w_p^\intercal F(I_i;\Omega)}} \\ &+ (1-\mathbb{1}(y_i))log\frac{e^{w_p^\intercal F(I_i;\Omega)}}{1+e^{w_p^\intercal F(I_i;\Omega)}}\Bigg],\end{aligned} \tag{3.1}$$

onde a primeira parte da função corresponde à função de classificação multiclasse; C é o número de classes, $W = [w_1, ..., w_C]$ e w_k, $k = 1, ..., C$ é o peso do classificador para a categoria k; y_i é o rótulo para a amostra de imagem I e $\mathbb{1}(y_i)$ uma função indicadora que representa a distribuição de probabilidade para o rótulo y_i. F é a combinação de $n+1$ funções de transformação não-linear parametrizada em Ω : $F(I, \Omega) = f_{DBE}(f_n(...f_2(f_1(I, ; \omega_1); \omega_2)...; \omega_n)\omega_{DBE})$, onde a composição interna das n projeções não-lineares denota as n camadas da rede neural profunda. Por fim, $f_{DBE}(\cdot; \omega_{DBE})$ é a camada de geração de código binário, anexada à rede profunda (chamada de camada Direct Binary Embedding).

A primeira parte da Equação 3.1 visa essencialmente minimizar a diferença entre a distribuição de probabilidade do rótulo real e a previsão. A segunda parte corresponde a uma combinação de uma entropia cruzada softmax com uma sigmoid binária, usada para modelar cada rótulo de forma independente. A constante c_+ indica o número de rótulos positivos para cada imagem; v é o coeficiente que controla o equilíbrio numérico entre entropia cruzada softmax e a entropia cruzada sigmóide binária; ρ é o coeficiente que penaliza a perda por predizer rótulos positivos incorretamente.

O objetivo desse método ao utilizar a função formada pela entropia cruzada softmax é capturar as dependências de coocorrência entre os rótulos, não ignorando a independência de cada rótulo individual. Por exemplo, "garfo" e "colher" geralmente coexistem em uma imagem, pois estão associados ao conceito "jantar". Entretanto, ocasionalmente, um "laptop" pode ser colocado aleatoriamente na mesa de jantar, onde também há "garfo" e "colher" na imagem.

Como pode ser observado, esse método limita o problema de recuperação com múltiplos rótulos à tarefa de classificação. Mesmo modelando as correlações existentes entre os rótulos da imagem, o modelo não considera as relações existentes entre outras imagens que compartilham rótulos da mesma categoria. Essa é uma característica importante que pode influenciar o desempenho da recuperação. Além disso, neste método são consideradas igualmente relevantes imagens recuperadas que compartilham quaisquer número de rótulos em comum com a imagem de consulta. Contudo, já vimos

que essa configuração pode ser pouco adequada para o cenário de múltiplos rótulos.

3.2.2 *Deep Multi-Label Hashing*

Zhong et al. [2017] propuseram *Deep Multi-Label Hashing* (DMLH), outra abordagem representativa desse grupo, que procura obter um código *hash* binário por combinar a representação do conteúdo visual da imagem, obtida por uma AlexNet, com aquela obtida de um grafo de similaridade entre os rótulos. Mais especificamente, o framework para recuperação de imagem com múltiplos rótulos DMLH possui dois componentes. O primeiro é uma AlexNet usada para extrair características da imagem e gerar o código *hash*. O segundo é formado por um algoritmo de partição de grafo usado para obter agrupamentos de rótulos. Com estes agrupamentos, é construído um vetor semântico para cada imagem, conforme as seguintes etapas, ilustradas na Figura 3.2:

1. Construção de um grafo de coocorrência de rótulos: nesse grafo, os rótulos são representados pelos nós e a frequência de coocorrência de rótulos como o peso das arestas conectando os nós.

2. Criação de Grafos Semânticos de Particionamento: dado o grafo de coocorrência, o método utiliza um algoritmo de particionamento para separar agrupamentos de rótulos com base na similaridade destes. Nesse caso, dois rótulos possuem similaridade semântica maior se a aresta entre eles tiver um peso maior. De forma simplificada, dado um grafo de coocorrência e K, o número de subgrafos conectados, o algoritmo de particionamento irá remover a aresta com o menor peso em cada iteração até o grafo inicial ser dividido em K sub-grafos. Ao final, cada sub-grafo representa um agrupamento de rótulo.

3. Extração do Vetor Semântico da Imagem: tendo gerado K sub-grafos, nesta fase é extraído um vetor semântico para cada imagem da base de treino. Cada imagem é representada por um vetor semântico K dimensional $I \in \{0,1\}^K$, onde cada dimensão corresponde a um agrupamento c_k, com $k = 1, 2, 3, ..., K$. Dada uma imagem com vários rótulos, um bit do vetor será definido como 1 se qualquer um de seus rótulos aparecer no agrupamento correspondente; caso contrário, este bit é 0. Desta forma, duas imagens serão representadas por vetores semânticos similares se houver alta similaridade entre seus rótulos.

Ao final, o objetivo do método é minimizar a distância euclidiana entre o código *hash* H gerado pela CNN e o vetor semântico da imagem I construído pelo algoritmo de partição de grafo, conforme Equação 3.2.

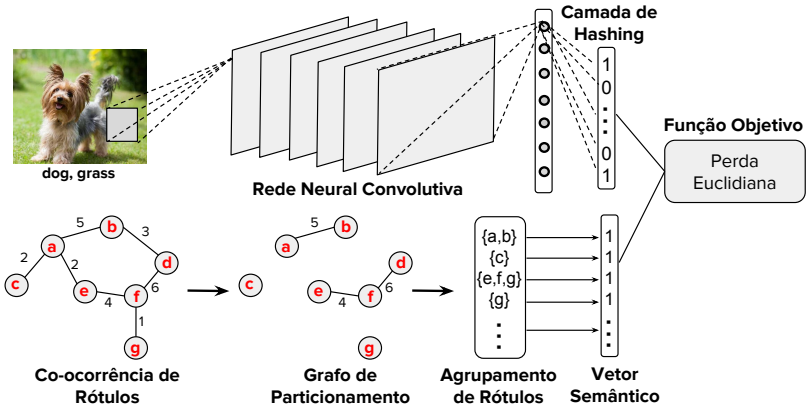


Figura 3.2. Arquitetura do modelo DMLH.

$$loss = \frac{1}{2K}||H - I||_2^2 = \frac{1}{2K}\sum_{i=1}^{K}(H_i - I)_2^2. \tag{3.2}$$

Diferente do DBE, essa abordagem mapeia as imagens em códigos binários considerando as similaridades existentes entre os rótulos das imagens de mesma categoria. Isso é feito associando à CNN informações de similaridades que são modeladas separadamente. Outra característica interessante desse método é que ao usar um grafo de particionamento para gerar vetor semântico para as imagens, ele evita funções complexas para modelar similaridade entre as amostras, facilitando a etapa de treinamento da rede neural.

3.2.3 *Multi-task Deep Hashing*

O *Multi-task Deep Hashing* (MTDH) [Liang et al., 2017] é um método de recuperação de imagem com múltiplos rótulos específico para o domínio de veículos. Nele, uma CNN (GoogLeNet) é usada para prever três categorias por instância, ou seja, a cor, o modelo e o ID de um veículo. Assim, três camadas completamente conectadas são adicionadas após a última camada de *pooling* da rede, uma para cada categoria (cf. Figura 3.3). Em cada camada é aplicada uma função softmax. A função de perda final é a soma das perdas softmax. Considerando x_{nk} a saída da k-ésima unidade da camada completamente conectada da n-ésima amostra, a função softmax é definida como:

$$\sigma(x_{nk}) = \frac{e^{x_{nk}}}{\sum_{k'=1}^{K} e^{x_{nk'}}}, \tag{3.3}$$

onde K é o número de classes. Por sua vez, a perda final é definida por:

$$loss = \sum_{n=1}^{N} log(\sigma(x_{nl_n})), \tag{3.4}$$

em que $l_n \in \{0, 1, ..., K-1\}$ é o rótulo da n-ésima amostra.

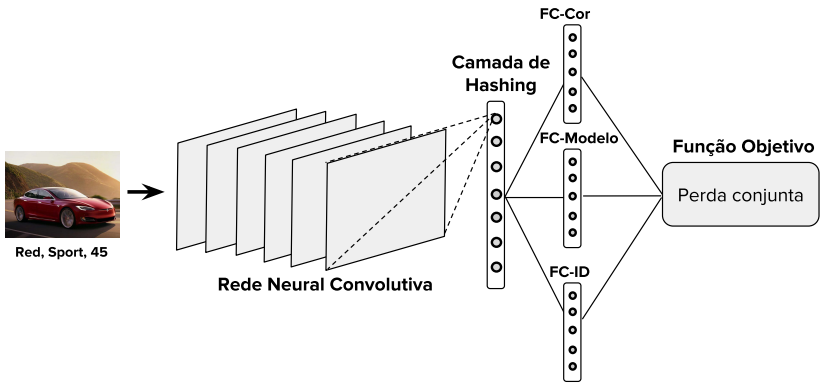


Figura 3.3. Arquitetura do modelo MTDH.

Esse método foi modelado para um domínio específico em que a coleção utilizada é formada por imagens que possuem três categorias em todo o conjunto de dados. Isso torna o método difícil de ser estendido para outros domínios, pois necessita de uma camada por categoria. Além disso, as coleções de referências não necessariamente são divididas em categorias, o que tornaria essa abordagem pouco aplicável, ainda mais se considerarmos que normalmente em coleções de imagens com múltiplos rótulos nem todos as categorias estão presentes em todas as imagens. Um resultado interessante desse trabalho foi mostrar que treinar uma rede neural usando ReLU como função de ativação da camada de *hashing* é melhor que treinar com uma sigmóide.

3.2.4 *Discriminative Cross-View Hashing*

O *Discriminative Cross-View Hashing* [Liu & Qi, 2018] (DCVH) é uma abordagem *cross-modal*[1], onde a recuperação é feita tanto com base no conteúdo da imagem quanto com a informação textual usada para descrever as classes. DCVH utiliza uma ResNet50 para mapear imagens para um espaço de Hamming de baixa dimensão e usa vetores GloVe [Pennington et al., 2014] pré-treinados para obter representação vetorial de texto. Os vetores de texto são alimentados em uma *text*-CNN [Xu et al., 2015] para gerar representação binária. A função de perda utilizada nos dois mapeamentos é a mesma utilizada pelo algoritmo DBE (Equação 3.1). Para combinar as duas representações

[1] *Cross-modal* refere-se a algoritmos que utilizam duas ou mais modalidades diferentes de representação do sinal como entrada para uma rede neural Jiang & Li [2016].

em um único código *hash*, o método adota como função de perda da rede neural a distância de Hamming, obtendo a seguinte equação:

$$\begin{aligned}\min_{\Omega^{(\mathcal{I})},\Omega^{(\mathcal{T})}} & J_{\mathcal{I},\mathcal{T}}(F^{(\mathcal{I})}(\mathcal{I};\Omega^{(\mathcal{I})}),F^{(\mathcal{T})}(\mathcal{T};\Omega^{(\mathcal{T})})) \\ &= \frac{1}{ND}\sum \mathbf{B}^{\mathcal{I}} \otimes \mathbf{B}^{\mathcal{T}} \\ &= \frac{1}{ND}\sum \mathbf{B}^{\mathcal{I}} \odot \overline{\mathbf{B}}^{\mathcal{T}} + \overline{\mathbf{B}}^{\mathcal{I}} \odot \mathbf{B}^{\mathcal{T}}\end{aligned} \tag{3.5}$$

onde N é o número de imagens e D é o tamanho do código binário. $\mathbf{B}^{\mathcal{I}}$ e $\mathbf{B}^{\mathcal{T}}$ são a representação binária da imagem $\mathcal{I}$ e do texto $\mathcal{T}$, respectivamente; F é a projeção não-linear para as duas representações; $\otimes$ é uma operação XOR bit a bit e $\odot$ é o produto *Hadamard*[2].

Devido à natureza binária dos dados, a função definida na Equação 3.5 é de difícil otimização. Por isso, o seguinte processo de relaxação é aplicado:

$$\begin{aligned}\min_{\Omega^{(\mathcal{I})},\Omega^{(\mathcal{T})}} & J_{\mathcal{I},\mathcal{T}}(F^{(\mathcal{I})}(;\Omega^{(\mathcal{I})}),F^{(\mathcal{T})}(;\Omega^{(\mathcal{T})})) \\ &\approx \frac{1}{ND}\sum_{i=1}^{N}\{F^{(\mathcal{I})}(I_i;\Omega^{(\mathcal{I})})(1-F^{(\mathcal{T})}(T_i;\Omega^{(\mathcal{I})}))^{\mathsf{T}} \\ &+ (1-F^{(\mathcal{I})}(I_i;\Omega^{(\mathcal{I})})F^{(\mathcal{T})}(T_i;\Omega^{(\mathcal{T})})^{\mathsf{T}}\}\end{aligned} \tag{3.6}$$

Assim como o DMLH, esse modelo combina duas informações ao final da rede e tenta minimizar a distância entre elas, com o intuito de incluir informações de similaridade ao código *hash*. Contudo, diferentemente do DMLH, a segunda representação da imagem (informação de texto) é também gerada por meio da construção de uma CNN, o que pode aumentar o custo de processamento tanto no treino como na etapa de recuperação. Observe que a informação de texto utilizada pelo método é a própria informação dos rótulos das imagens, por isso não há necessidade de informações adicionais aos dados.

3.3 Métodos *Pairwise*

Métodos *pairwise* utilizam pares de imagens para modelar similaridades entre as amostras de imagens. A Figura 3.4 mostra uma visão geral dos métodos dessa categoria. Para os pares de imagens, que são dados como entrada para rede, são definidos dois tipos de associação: similares e dissimilares. Essas relações são consideradas no processo

[2]O produto *Hadamard* é uma operação binária entre matrizes de mesma dimensão tal que $A = B \odot C$ implica que $A_{i,j} = B_{i,j}C_{i,j}$.

de aprendizado para preservar informação de pares de rótulos no Espaço de Hamming aprendido.

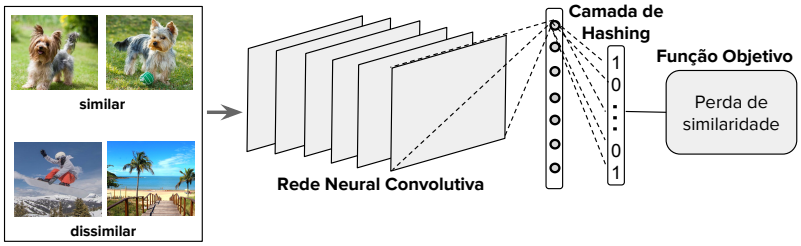


Figura 3.4. Visão geral dos métodos da categoria *pairwise*.

Além dos métodos que serão descritos nesta proposta, outros trabalhos foram propostos utilizando a abordagem *pairwise* para recuperação de imagem baseada em *deep hashing* [Li et al., 2015; Liu et al., 2016; Shen et al., 2017]. Contudo esses trabalhos não focam no problema de recuperação de imagem com múltiplos rótulos e, por isso, não são apresentados aqui.

3.3.1 *Multi-label Supervised Deep Hashing*

Lu et al. [2017] propuseram a abordagem de hashing para recuperação de imagens chamada *Multi-Label Supervised Deep Hashing* (MSDH). Diferente de métodos anteriores, eles supõem que além de (i) minimizar a perda de quantização, ou seja, a diferença entre o código de representação real aprendido e o código binário quantizado, os códigos binários produzidos também deveriam (ii) ser uniformemente distribuídos, de forma a possibilitar um uso ótimo do espaço de representação e (iii) ter bits constituintes o mais independentes possível, de forma a minimizar a redundância de representação. Os autores tiram proveito da informação de múltiplos rótulos incluindo um termo na função de perda com o intuito de minimizar a perda das amostras que pertencem à mesma classe e maximizar a perda de amostras de classes diferentes.

Para modelar as restrições (i), (ii) e (iii), os autores sugeriram a seguinte função de perda:

$$\begin{aligned}\min_{W,c} J &= \frac{1}{2}\|B - H^M\|_F^2 \\ &- \frac{\lambda_1}{2}(tr(\frac{1}{N}H^M(H^M)^\mathsf{T}) + \alpha tr(\Sigma_B - \Sigma_W)) \\ &+ \frac{\lambda_2}{2}\sum_{m=1}^{M}\|W^m(W^m)^\mathbf{T} - I\|_F^2 \\ &+ \frac{\lambda_3}{2}\sum_{m=1}^{M}(\|W^m\|_F^2\|c^m\|_F^2) \\ &= J_1 - \lambda_1 J_2 + \lambda_2 J_3 + \lambda_3 J_4\end{aligned} \tag{3.7}$$

onde $B = [b_1, ..., b_N] \in \{-1, 1\}^{K\times N}$ e $H^m = [h_1^m, h_2^m, ..., h_N^m] \in \mathbb{R}^{p^m\times N}$ são a representação matricial dos vetores de códigos binários e a saída da m-ésima camada da rede, respectivamente. λ_1, λ_2 e λ_3 são três parâmetros para ponderar o efeito dos diferentes termos. O termo J_1 visa minimizar a perda de quantização entre os vetores binários aprendidos e os vetores compactos de valor real, para que as características aprendidas, que são compactadas, possam ser bem preservadas nos códigos binários. O segundo termo, J_2, visa maximizar a variância dos vetores binários aprendidos para garantir bits balanceados. O terceiro termo, J_3, impõe uma restrição de ortogonalidade relaxada nas matrizes de projeção para que a independência de cada transformação seja maximizada. Independência nesse caso implica na quantidade de informação que cada bit carrega: quanto mais independente menos informação redundante o código final terá. O último termo, J_4, é a regularização para controlar as escalas dos parâmetros. Finalmente, Σ_B e Σ_W são calculados para selecionar aleatoriamente amostras de treino dentro das classes e entre as classes, respectivamente. São definidos como:

$$\Sigma_B = \sum_{l=1}^{L}\sum_{i=1}^{N}\delta_{il}(\mu_l - \mu)(\mu_l - \mu)^\mathsf{T} \tag{3.8}$$

$$\Sigma_W = \sum_{l=1}^{L}\sum_{i=1}^{N}\delta_{il}(h_i^M - \mu_l)(h_i^M - \mu_l)^\mathsf{T}, \tag{3.9}$$

onde μ_l é a média da saída da última camada de todas as amostras de treino pertencentes a l-ésima classe e μ é a média de todas as saídas da última camada. Se a i-ésima amostra pertence a l-ésima classe, $\delta_{il} = 1$. Se não, $\delta_{il} = 0$.

Nesse modelo, Σ_B e Σ_W exploram a informação discriminativa das amostras com múltiplos rótulos, onde Σ_B minimiza a perda das amostras que compartilham rótulos e Σ_W maximiza a perda das amostras que não tem rótulos em comum.

3.3.2 *Deep Multilevel Semantic Similarity Preserving Hashing*

Os autores em *Deep Multilevel Semantic Similarity Preserving Hashing* (DMSSPH) [Wu et al., 2017] propõem que a função de distância entre os códigos de duas imagens deveria levar em conta os graus de similaridade estabelecidos por seus rótulos. Mais especificamente, eles projetaram uma função de perda que considera diferentes graus de similaridade, ou seja, muito similares, normalmente similares e dissimilares. O objetivo do modelo é maximizar a discriminabilidade do espaço de saída codificando as informações dos rótulos dos pares de imagens de entrada e, simultaneamente, impondo uma regularização nas saídas de valor real para aproximar os valores discretos desejados.

Para esse propósito, considere $(I_1,\ I_2)$ um par de imagens associado ao seu conjunto de rótulos l_1 e l_2. A função de perda para modelar os diferentes níveis de similaridade é definida como:

$$\begin{aligned} L(I_1, I_2, n_1, n_2, y, m) &= \frac{1}{2}(1-y)D_H(h(I_1;w), h(I_2;w)) \\ &+ \frac{1}{2}y\max\left(m\left(\frac{n_1-n_2}{n_1}\right) - D_H(h(I_1;w), h(I_2;w)), 0\right) \\ &+ \alpha\||h(I_1;w)|-1\|_1 + \||h(I_2;w)|-1\|_1 \\ s.t.\ \ & h(I_j;w) \in \{+1,-1\}^K, j \in \{1,2\} \end{aligned} \tag{3.10}$$

onde $n_1 = |l_1|$ é o número de rótulos em l_1 e $n_2 = |l_1 \cap l_2|$ denota o número de rótulos compartilhados entre l_1 e l_2. Além disso, $y = 0$ se $n_1 = n_2$ e $y = 1$, caso contrário. D_H representa a distância de Hamming entre os vetores binários e $m > 0$ é um parâmetro de limiar de margem. O primeiro termo pune imagens muito similares ($l_1 \subseteq l_2$) mapeadas para códigos binários diferentes. Por sua vez, o segundo pune imagens normalmente similares ou dissimilares mapeadas para códigos binários muito próximos, quando sua distância de Hamming cai abaixo de um certo limiar de margem determinado por seus graus de similaridade. Por fim, o terceiro termo define o erro de quantização do código aprendido, que é adicionado à função de perda como regularizador para aproximar os valores reais da saída da rede para valores discretos desejados (+1/-1). 1 é um vetor onde todos os elementos são iguais a 1, $\|\cdot\|_1$ e $|\cdot|$ são a norma L1 e o módulo do vetor, respectivamente.

A intuição dessa função de perda é gerar códigos de tal forma que I_1 e I_2 sejam dissimilares se não compartilham nenhum rótulo (*i.e.* $n_2 = 0$), punindo-os quando a margem entre seus códigos binários esteja dentro da maior margem m. Imagens que compartilham pelo menos um rótulo de classe, mas nem todos os rótulos (*i.e.*, $0 < n_2 < n_1$), são tratadas como normalmente semelhantes.

A função de perda resultante será a soma das perdas para os N pares de amostras

selecionados aleatoriamente do conjunto de treinamento $\{(I_{i,1}, I_{i,2}, n_{i,1}, n_{i,2}, y_i, m_i), |i = 1, 2, ..., N\}$.

$$\mathcal{L} = \sum_{i=1}^{N} L(I_{i,1}, I_{i,2}, n_{i,1}, n_{i,2}, y_i, m_i) \tag{3.11}$$

Devido à difícil solução do problema de otimização discreta apresentada na Equação 3.10, o valor de $h(\cdot)$ é substituído pelos valores reais da saída da rede e a distância de Hamming é substituída pela distância Euclidiana.

Ao definir um limiar de margem adaptativa, esta abordagem é capaz de modelar as relações entre os rótulos dos pares de imagens, sendo o primeiro método a definir mais de dois níveis de similaridade.

3.3.3 *Deep Multi-Similarity Hashing*

Os autores em *Deep Multi-Similarity Hashing* (DMSH) [Li et al., 2017] propõem outro modelo em que a similaridade entre as imagens leva em conta seus rótulos compartilhados. Em particular, eles projetam uma função *pairwise* que induz uma CNN a aprender as similaridades entre os pares de imagens de forma que quanto mais rótulos os pares compartilham, mais similares eles são. Se considerarmos que N pares de imagens são selecionados aleatoriamente do conjunto de treinamento, a função objetivo desse modelo visa minimizar a perda entre os pares de imagens, da seguinte forma:

$$\mathcal{L} = \sum_{i=1}^{N} L(b_{i,1}, b_{i,2}, c_i), \tag{3.12}$$

onde L é a função de perda para cada par de imagem, descrita como:

$$L_r = L_p(b_1, b_2, c) + \alpha L_q(b_1, b_2), \tag{3.13}$$

em que L_p representa a perda dos pares de imagens, enquanto L_q denota a função de quantização dos códigos binários. A tupla (b_1, b_2) representa cada par de imagens e c denota o número de rótulos em comum no par de imagens. O parâmetro α determina o peso da função de quantização.

A função de perda *pairwise* L_p é definida como:

$$L_p(b_1, b_2, c) = \begin{cases} \frac{1}{2}\max(\|b_1 - b_2\|_2^2 - t(c), 0), & c > 0 \\ \frac{1}{2}\max(m - \|b_1 - b_2\|_2^2, 0), & c = 0 \end{cases} \tag{3.14}$$

onde $t(c)$ é o limiar da distância de Hamming entre pares de imagens similares, que varia de acordo com o número de rótulos comuns c. A constante m denota um limiar fixo da distância de Hamming para pares de imagens dissimilares. O primeiro termo pune imagens similares mapeadas para códigos binários distantes quando sua distância de Hamming fica acima da margem do limiar $t(c)$. O segundo termo pune imagens dissimilares mapeadas para códigos próximos quando sua distância de Hamming cai abaixo do limiar de margem m.

Para controlar quando a função deve punir pares de imagens, foram definidos intervalos onde a perda é considerada zero (*zero-loss interval*). Se o valor resultante da função de perda cair dentro desse intervalo, então a perda para esse par de imagem é zero. Neste caso, a distância de Hamming entre os pares é próximo o suficiente para imagens que são similares ou longe o suficiente para imagens que são dissimilares, de forma que não há necessidade de modificar seus códigos. Só haverá punição caso o valor resultante esteja fora do intervalo de perda zero.

Dessa forma, os intervalos são definidos com base nos valores de $t(c)$ e m:

$$\begin{aligned} m &= 2L \\ t(c) &= \lambda e^{-c} \times 4L, \quad 0.0 \leq \lambda \leq 1.0 \end{aligned} \tag{3.15}$$

onde L é o tamanho do código *hash* e λ denota um peso de decaimento.

Assim, o intervalo de perda zero para imagens similares é $[0, t(c)]$. Quando a distância da saída da rede de um par de imagens cair abaixo de $t(c)$, a perda para esse par é 0. O mesmo ocorre para pares de imagens dissimilares, no intervalo definido em $[m, 4L]$.

Por fim, considerando a função de quantização L_p (mesma utilizada em DMS-SPH), chegamos à seguinte função de perda:

$$\begin{aligned} \mathcal{L}_r = &\begin{cases} \frac{1}{2}\max(\|b_1 - b_2\|_2^2 - t(c), 0), & c > 0 \\ \frac{1}{2}\max(m - \|b_1 - b_2\|_2^2, 0), & c = 0 \end{cases} \\ &+ \alpha(\||b_1| - 1\|_1 + \||b_2| - 1\|_1) \end{aligned} \tag{3.16}$$

3.3.4 *Instance Similarity Deep Hashing*

O método *Instance Similarity Deep Hashing* (ISDH) [Zhang et al., 2018] utiliza a informação dos diversos rótulos das imagens para (a) definir a similaridade de instância com base em rótulos semânticos normalizados e (b) constrói uma função de perda *pairwise* para desenvolver, simultaneamente, o aprendizado de características e a geração de código *hash*.

Para calcular valores de similaridades, este método define a similaridade entre pares de imagens em porcentagens, da seguinte forma:

$$s_{i,j} = \begin{cases} p_{i,j}, & \text{se } I_i \text{ e } I_j \text{ compartilham rotulos} \\ 0, & \text{caso contrário} \end{cases} \tag{3.17}$$

em que $p_{i,j}$ é a distância cosseno entre pares de vetores de rótulos, que é definida como:

$$p_{i,j} = \frac{\langle l_i, l_j \rangle}{\|l_i\| \|l_j\|}, \tag{3.18}$$

onde l_i e l_j representam o vetor de rótulos semânticos da imagem I_i e I_j, respectivamente e $\langle l_i, l_j \rangle$ representa o produto interno entre os rótulos. Assim como no DMSSPH, foram definidos três níveis de similaridade: completamente similar, parcialmente similar e dissimilar. Mais especificamente, se $s_{i,j} = 1$, os códigos binários b_i e b_j devem ter uma distância de Hamming pequena; se $s_{i,j} = 0$, a distância de Hamming entre os pares de imagem é alta; caso contrário, a distância será definida de acordo com o valor de similaridade retornado por $s_{i,j}$.

Para a construção da função de perda entre pares de imagens, neste trabalho é utilizado o conceito de probabilidade a posteriori para modelar as relações de similaridades entre as imagens. Dessa forma, dado que $S = \{s_{i,j}\}$ o relacionamento de similaridade *pairwise*, a estimação Máxima a Posterior de códigos *hash* $B = \{b_{i,j}\}$ é dada por $p(B|S) \propto p(S|B)p(B) = \prod_{s_{i,j} \in S} p(s_{i,j}|B)p(B)$; onde $p(S|B)$ é a função de verossimilhança e $p(B)$ é a distribuição a priori. Para cada par de imagens, $p(s_{i,j}|B)$ é a probabilidade condicional de $s_{i,j}$ dado seus códigos *hash* b_i e b_j, definida na Equação 3.19.

$$p(s_{i,j}|B) = \begin{cases} \sigma(\Omega_{i,j}), & s_{i,j} = 1, \\ 1 - \sigma(\Omega_{i,j}), & s_{i,j} = 0, \\ 1 - (s_{i,j} - \sigma(\Omega_{i,j}))^2, & 0 < s_{i,j} < 1 \end{cases} \tag{3.19}$$

onde $\Omega i, j$ representa a distância de Hamming, que é calculada pelo produto interno escalonado $\Omega i, j = \alpha \cdot \langle b_i, b_j \rangle = \alpha \cdot b_i^T b_j$, onde α é um hiper-parâmetro positivo para controlar a restrição de largura. $\sigma(x)$ é uma função sigmóide usada para transformar a distância de Hamming em um tipo de medida de similaridade. $s_{i,j}$ é a similaridade entre pares de imagens calculada pelas Equações 3.17 e 3.18, e $(s_{i,j} - \sigma(\Omega_{i,j}))^2$ é a distância Euclidiana.

Nesse trabalho, quando os pares de imagens I_i e I_j são completamente similares ou dissimilares, a perda de similaridade é medida pela entropia cruzada: $l = s_{i,j} log(\sigma(\Omega_{i,j})) + (1 - s_{i,j}) log(\sigma(\Omega_{i,j})$. Ao substituir a função sigmóide $\sigma(\Omega_{i,j})$

por $\frac{1}{1+e^{-\Omega_{i,j}}}$, l resulta em $l = log(1 + e^{\Omega_{i,j}}) - s_{i,j}\Omega_{i,j}$. Por outro lado, se os pares de imagens são parcialmente similares, a perda de similaridade é calculada aplicando o erro médio quadrático, que neste caso equivale à distância Euclidiana. Dessa forma, a perda entre qualquer par de imagens é definido conforme apresentado na Equação 3.20.

$$l(i,j) = \begin{cases} log(1 + e^{\Omega_{ij}}) - s_{ij}\Omega_{ij}, & s_{ij} = 0 \; ou \; 1, \\ (s_{ij} - \sigma(\Omega_{i,j}))^2, & 0 < s_{ij} < 1 \end{cases} \tag{3.20}$$

Por fim, a função objetivo de otimização do modelo é definida como:

$$\begin{aligned} \min L = \sum_{i,j \in N} [\gamma \cdot M_{ij}(log(1 + e^{\Omega_{ij}}) - s_{ij}) \\ + (1 - M_{ij})(s_{ij} - \sigma(\Omega_{i,j}))^2 \\ + \lambda \| |b_1| - 1 \|_1 + \| |b_2| - 1 \|_1] \end{aligned} \tag{3.21}$$

onde M_{ij} é usado para sinalizar as duas condições definidas em $l(i,j)$ (Equação 3.20), de tal forma que $M_{ij} = 1$ denota que I_i e I_j são completamente similares e $M_{ij} = 0$ parcialmente similares. γ é um hiper-parâmetro para ponderar a importância do termo de entropia cruzada e N é o número de imagens. Assim como nos modelos DMSSPH e DMSH, a terceira parte da Equação 3.21 representa o erro de quantização e λ é um coeficiente de peso.

Note que a restrição binária de $b_i \in \{-1, 1\}^q$ torna difícil a otimização da Equação 3.21, sendo necessária aplicação de um processo de relaxação. Por isso, assim como em DMSSPH e DMSH, b_i é substituído pela saída da rede neural.

3.3.5 *Deep Uniqueness-Aware Hashing*

Diferente dos métodos anteriores dessa categoria, *Deep Uniqueness-Aware Hashing* (DUAH) [Wu et al., 2018] (uma extensão do método DMSSPH, Seção 3.3.2), além de modelar os múltiplos níveis de similaridade, também propõe incluir na função de perda a singularidade do conjunto de imagens como informação adicional para melhorar o *ranking* de resposta.

Para entender a ideia desse método, considere três imagens formadas pelos seguintes rótulos: I_1 = {"sky", "sunset", "beach", "sea"}, I_2 ={"sky", "sunset", "beach", "sea"}, I_3 = {"sky", "sunset", "beach", "sea", "pavillon"}. Para os métodos descritos anteriormente, I_2 e I_3 são igualmente similares à imagem I_1. No entanto, se considerarmos a singularidade da imagem I_3, I_1 é mais similar à imagem I_2 do que à imagem I_3. Consequentemente, tanto os múltiplos níveis de similaridade quanto a singulari-

dade do conjunto de imagem devem ser levadas em consideração. Para esse último, é proposta uma função de perda de classificação para múltiplos rótulos.

Assim como no DMSSPH, múltiplos níveis de similaridade são modelados. No entanto, ao invés de três níveis, aqui são definidos quatro níveis de similaridades, que incluem: extremamente similar, muito similar, normalmente similar e dissimilar. Esse níveis são definidos de acordo com os rótulos compartilhados, como indicado na Equação 3.22.

$$y = \begin{cases} 0\ (\textit{extremamente similar}), & n_1 = n_2 = n_3, \\ 1\ (\textit{muito similar}), & n_1 = n_2 \neq n_3 \\ 2\ (\textit{normalmente similar}), & n_1 > n_2 > n_3 \\ 3\ (\textit{dissimilar}), & n_2 = 0 \end{cases} \tag{3.22}$$

em que $n_1 = |l_1|$, $n_2 = |l_1 \cap l_2|$ e $n_3 = |l_2|$, onde l_1 e l_2 são os rótulos pertencentes a I_1 e I_2, respectivamente.

Dessa forma, para incluir um nível a mais de similaridade, a Equação 3.10 do algoritmo DMSSPH é reescrita como:

$$\begin{aligned} \mathcal{L}_{fg} = & \sum_{i=1}^{N_1} \frac{1}{2} I_{y_i=0} \max(D_H(h(I_1;w), h(I_2;w)) - m_1, 0) \\ & + \frac{1}{2} I_{y_i=1} \max(m_1 - D_H(h(I_1;w), h(I_2;w)) - m_1, 0) \\ & + \frac{1}{2} I_{y_i=2} \max(m_2(n_{i,1} - n_{i,2})/n_{i,1} - D_H(h(I_1;w), h(I_2;w)), 0) \\ & + \frac{1}{2} I_{y_i=3} \max(m_2 - D_H(h(I_1;w), h(I_2;w)) - m_1, 0) \\ & s.t.\ \ h(I_j;w) \in \{+1, -1\}^K, j \in \{1, 2\} \end{aligned} \tag{3.23}$$

onde N_1 é o número de pares selecionados aleatoriamente do conjunto de treino. I_{y_i} é uma função indicadora, em que $I_{condition} = 1$ se a condição é verdadeira; $I_{condition} = 0$ caso contrário. Para controlar os quatro níveis de similaridade, foram definidos dois parâmetros de limiar de margem ao invés de somente um, representados por m_1 e $m_2 > 0$. No entanto, diferente do DMSSPH, a função não é penalizada se a distância de Hamming entre os pares de imagens extremamente similares ficar abaixo de m_1. Além disso, para distinguir imagens muito similares de imagens extremamente similares, o método pune imagens muito similares se a distância de Hamming entre seus códigos *hash* cair abaixo de m_1. Imagens normalmente similares e dissimilares mapeadas para códigos binários próximos são punidas quando sua distância de Hamming cai abaixo de um certo limiar de margem determinado por seus graus de similaridade.

A função de classificação para múltiplos rótulos é definida como segue:

$$\mathcal{L}_{mc} = -\sum_{i=1}^{N_2}\sum_{j=1}^{c}[I_{y_j^i=1}\frac{1}{c_i}log(p_j^i) + I_{y_j^i=0}log(1-p_j^i)] \tag{3.24}$$

na qual $y^i \in \{0,1\}^c$ representa o vetor de rótulos, c é número de classes, c_i é o número de rótulos da imagem I_i. N_2 é o número de imagens de treinamento e p_j^i é a probabilidade de previsão, definida como $p_i^j = \frac{e^{W_j^T f(I_i)}}{\sum_{t=1}^{c} e^{W_j^T f(I_i)}}$, onde $W \in \mathcal{R}^{k\times c}$ denota a matriz de pesos das camadas da rede profunda.

Por último, a função objetivo (Equação 3.25) do modelo DUAH é formada pela soma das funções apresentadas nas Equações 3.22 e 3.24.

$$\min \mathcal{L} = \mathcal{L}_{fg} + \mathcal{L}_{mc} \tag{3.25}$$

3.3.6 Improved Deep Hashing Network

Zhang et al. [2019] propuseram o *Improved Deep Hashing Network* (IDHN), um modelo baseado em uma definição suave para o cálculo da similaridade entre pares de imagens, quantificando-a em uma porcentagem que corresponde aos rótulos semânticos normalizados. Este método é uma extensão do ISDH (Seção 3.3.4). Assim como o ISDH, essa abordagem usa diretamente a similaridade quantificada dos pares com o intuito de refletir uma similaridade refinada entre um par de imagens com múltiplos rótulos para aprendizado supervisionado. Entretanto, diferentemente do ISDH, para o cálculo de similaridade são consideradas apenas duas situações: "semelhança forte" e "semelhança suave". A função de perda é composta por uma função perda de entropia cruzada e perda de erro quadrático médio, que são adaptadas para aprender códigos *hash* que preservem a similaridade semântica com base na similaridade quantificada.

A função de custa desse modelo é similar a função apresentação na Eq. 3.21, com algumas alterações, como podemos ver a seguir:

$$\begin{aligned} L = \sum_{i,j\in S} [&M_{ij}(log(1+e^{\Omega_{ij}}) - s_{ij}) \\ &+ \gamma \cdot (1-M_{ij})(\frac{< b_i, b_j > + q}{2} - s_{i,j})^2] \end{aligned} \tag{3.26}$$

onde M_{ij} representa as similaridades entre os pares de imagem: $M_{ij} = 1$ denota o caso de "semelhança forte" e $M_{ij} = 0$ denota o caso de "semelhança leve".

3.3.7 Pairwise Supervised Hashing

Pairwise Supervised Hashing (PSH) [Dadaneh et al., 2020] é um modelo originalmente proposto para resolver o problema de recuperação textual e não aborda o problema de múltiplos rótulos. No entanto, ele é apresentado neste capítulo por ser o modelo que usamos como base para construção do método que propomos nesta pesquisa.

O PSH trás uma abordagem diferente das anteriores por considerar em seu desenvolvimento uma rede auto-codificadora variacional discreta com variáveis latentes de Bernoulli, para geração de códigos *hash* binários. Dessa forma, por utilizar uma distribuição de Bernoulli, este gera um espaço de mapeamento de 0s e 1s sem precisar fazer nenhum tipo de binarização dos dados de saída, como é feito nos métodos presentes neste capítulo. Além disso, as informações de rótulos dos documentos são aproveitadas para uma busca de similaridade mais direcionada, propondo, dessa forma, um *framework* de *hashing* supervisionado por pares para derivar melhores códigos *hash*, com dois objetivos principais: (1) aprender códigos binários representativos, capazes de reconstruir a representação original; (2) minimizar a distância entre o códigos *hash* dos documentos de mesma classe e maximizar essa distância para documentos de classes diferentes.

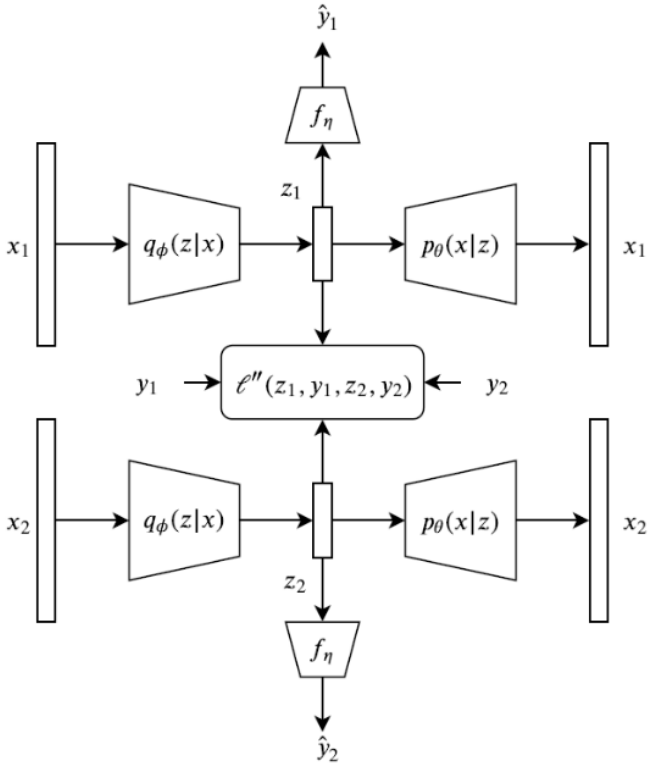


Figura 3.5. Representação gráfica do modelo PSH [Dadaneh et al., 2020].

Uma visão geral desse modelo é apresentada na Figura 3.5. Como podemos observar, o modelo é formado por duas redes autocodificadoras. Os documentos x_1 e x_2 passam cada um pela rede codificadora $q_\phi(z|x)$ para gerar códigos *hash* latentes z_1 e z_2, respectivamente. Cada código *hash* passa então por redes decodificadoras $p_\phi(x|z)$ e classificadoras (f_η) para reconstruir o documento de entrada e prever suas classes, respectivamente.

Para entender a construção do modelo, considere y o rótulo do documento de entrada x. Dada uma rede neural f_η parametrizada por η, que recebe como entrada o código *hash* latente z e prevê o rótulo do documento, a função de perda do modelo a ser minimizada pode ser expressa como:

$$-\mathcal{L}(\theta, \phi) + \alpha E_{q_\phi(z|x)}[\mathcal{L}^{'}(y; f_\eta(z))] \tag{3.27}$$

onde $\alpha > 0$ é um hiperparâmetro e $\mathcal{L}^{'}(y; f_\eta(z))$ é uma função de perda de entropia cruzada para classificação das classes.

Para melhorar a representação do código *hash*, o modelo inclui uma estrutura de treinamento de *hashing* supervisionado par a par (PSH). O principal objetivo do PSH é minimizar a distância entre códigos latentes de documentos semelhantes e simultaneamente maximizar a distância entre códigos latentes de documentos que se enquadram em diferentes categorias. Dessa forma, vamos considerar $(x^{(1)}$, $y^{(1)})$e $(x^{(2)}$, $y^{(2)})$ dois documentos amostrados aleatoriamente com seus respectivos códigos latentes $z^{(1)}$ e $z^{(2)}$, o PSH coloca uma função de perda extra como mostrado a seguir:

$$\mathcal{L}^{''}(z^{(1)}, z^{(2)}) = \mathbf{1}_{y^{(1)}=y^{(2)}} d(z^{(1)}, z^{(2)}) - \mathbf{1}_{y^{(1)}\neq y^{(2)}} d(z^{(1)}, z^{(2)}) \tag{3.28}$$

onde $d(\cdot, \cdot)$ é uma métrica de distância e $\mathbf{1}_S$ é a função indicadora sendo igual a um quando S é verdadeiro. A função de perda final para o PSH é, portanto:

$$\begin{aligned} \mathcal{L}_{PSH}(\theta, \phi) &= -[\mathcal{L}_\gamma^{(1)}(\theta, \phi) + \mathcal{L}_\gamma^{(2)}(\theta, \phi)] \\ &+ E_{\Pi_{t=1}^2 q_\phi(z^{(t)}|x^{(t)})}[\beta \mathcal{L}^{''}(z^{(1)}, z^{(2)})] \\ &+ \alpha[\mathcal{L}^{'}(y^{(1)}; f_\eta(z^{(1)})) + \mathcal{L}^{'}(y^{(2)}; f_\eta(z^{(2)}))] \end{aligned} \tag{3.29}$$

A otimização da função de perda do PSH (3.29) é difícil, pois o algoritmo de retropropagação não pode ser aplicado às camadas de amostragem discretas de Bernoulli. Dessa forma, esse modelo usa dois estimadores de gradiente amplamente utilizados para variáveis latentes discretas: o estimador de gradiente direto [Bengio et al., 2013] e o Gumbel-Softmax [Jang et al., 2016]. Em seguida, aplicada o algoritmo *augment-reinforce-merge* (ARM) [Yin & Zhou, 2019], um estimador de gradiente que pode ser

empregado para retropropagação através de camadas discretas.

3.4 Métodos *Tripletwise*

Em abordagens *tripletwise*, redes neurais profundas geram códigos *hash* explorando a comparação de proximidade entre três pontos de dados, como ilustrado na Figura 3.6. Dada uma imagem de consulta q, considere x^+ sendo mais similar a q que a imagem x^-. Dessa forma, ao aprender código *hash* é esperado que $sim(q, x^+) > sim(q, x^-)$, onde $sim(\cdot, \cdot)$ representa uma medida de similaridade semântica.

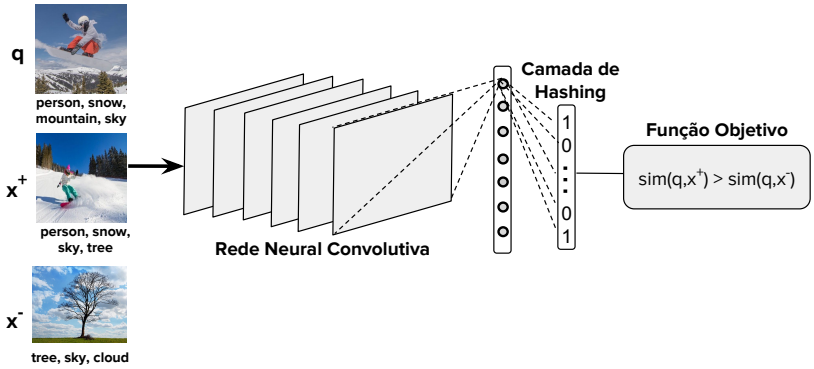


Figura 3.6. Visão geral dos métodos da categoria *tripletwise*.

3.4.1 *Instance-Aware Hashing*

O método *Instance-Aware Hashing* (IAH) [Lai et al., 2016] aprende representações para imagens com múltiplos rótulos que são organizadas em múltiplos grupos, onde cada grupo contém características de uma categoria. O método consiste de quatro módulos, descritos a seguir.

Módulo de Geração de Proposta de Região: neste módulo é utilizado o algoritmo Geodesic Object Proposal (GOP) Krähenbühl & Koltun [2014] para automaticamente gerar propostas de regiões para a imagem de entrada. O algoritmo GOP cria tanto máscaras de segmentação quanto propostas de caixa delimitadora para indicar a região a ser processada.

Módulo de Rede Neural de Convolução: neste módulo uma GoogLeNet é utilizada para fazer a extração de características das propostas de regiões retornadas no primeiro módulo. Como o número de propostas de região para uma imagem é muito grande (mais de 1000), é computacionalmente caro utilizar diretamente uma GoogLeNet para extrair características de todas as regiões. Para resolver esse problema, o IAH

utiliza um esquema de subamostragem pirâmide espacial (*Spatial Pyramid Pooling* - SPP) He et al. [2014]. Dessa forma, é calculado apenas uma vez o mapa de característica para toda a imagem e com base nesse mapa, as características são reunidas em cada proposta de região para gerar uma representação de tamanho fixo. O algoritmo SSP é adicionado após a última camada da GoogLeNet. Cada uma das N propostas de região geradas para cada imagem é codificada para um vetor 4D $L_i \in \mathbb{R}^4 (i = 1, 2, ..., N)$. Com esse vetor de quatro dimensões como entrada, a camada SPP gera um vetor de características de tamanho fixo para a proposta correspondente. Através da camada SPP, cada proposta é mapeada para um vetor intermediário de característica d-dimensional. Portanto, para cada imagem de entrada, a saída deste módulo é uma matriz $N \times d$.

Módulo de Cálculo de Probabilidade dos Rótulos: esse módulo calcula probabilidades de rótulos para cada proposta de região. Para isso, suponha que existam c classes de rótulos e um vetor de probabilidade para cada proposta. $P^i = (P_1^i, ..., P_2^i)$ indica que a probabilidade da imagem de conter a j-ésima categoria é P_j^i. Como as coleções não disponibilizam informações de rótulos por categoria, essa distribuição de probabilidade não pode ser aprendida diretamente. Para contornar esse problema, o método faz primeiro uma combinação das N propostas em uma e então usa os rótulos da imagem inteira.

Deste modo, considere D a matriz retornada no módulo anterior, em que a i-ésima linha D^i representa características intermediárias para a i-ésima região, esse módulo primeiro usa uma camada completamente conectada para comprimir D^i para um vetor de $M^i \in \mathbb{R}^c$. De forma resumida, a matriz de probabilidade P para cada proposta de região é definida como indicado na Equação 3.30.

$$P_j^i = \frac{e^{M_j^i}}{\sum_{k=1}^{c} e^{M_k^i}} (i = 1, ..., N) \tag{3.30}$$

onde P_j^i representa o j-ésimo elemento na i-ésima linha de P. P_j^i pode ser visto como a probabilidade da i-ésima proposta conter um objeto da j-ésima categoria.

Módulo de Codificação de Hashing: esse módulo converte a representação da imagem em (i) um pedaço de código *hash* semântico e (ii) múltiplos pedaços de código *hash*, um para cada categoria.

Novamente, considere a matriz D como entrada. Esse módulo primeiro usa uma camada completamente conectada para comprimir D^i para um vetor de $H^i \in \mathbb{R}^b$, onde H^i corresponde a i-ésima proposta. Com o intuito de converter H em uma representação de instância da imagem de entrada, IAH elabora uma estratégia de fusão de proposta cruzada, usando a matriz P de probabilidade calculada no módulo anterior. Dessa forma, considere H^i e P^i a i-ésima linha de H e P, respectivamente.

Com isso, a fusão de proposta cruzada consiste em:

$$f^{(j)} = \frac{1}{N}\sum_{i=1}^{N} P_j^i H_i^j \tag{3.31}$$

Como H^i representa as características da presente proposta, $f^{(j)}$ pode ser considerada como a média ponderada das características das propostas. Se a i-ésima proposta tem uma pontuação relativamente maior/menor P_j^i (o que significa que a i-ésima proposta provável/improvável pertencer a na j-ésima categoria), o vetor de características H^i (associado à i-ésima proposta) tem contribuição mais/menos para a média ponderada $f^{(j)}$.

Com o f gerado pela fusão de propostas cruzadas, podemos gerar a representação de *hashing* com reconhecimento de categoria que consiste em c partes de códigos *hash* e a representação de *hashing* semântico que consiste em uma parte do código *hash*. Esses dois processos são descritos a seguir:

1) *Representação de hashing com reconhecimento de categoria*: como f é organizada em c grupos $f^{(1)}, f^{(2)}, ..., f^{(i)}$, cada $f^{(i)}(i = 1, 2, ..., c)$ pode ser convertido em códigos binários de b-bits $b^{(i)} = sign(f^{(i)})$. Dessa forma, para uma imagem I, a representação de *hashing* com reconhecimento de categoria de I é $b(I) = b^{(1)}(I), b^{(2)}(I), ..., b^{(c)}(I)$.

Para aprender essa representação, o método define uma função de perda *triplet*, uma para cada categoria. Trios de imagens são selecionados aleatoriamente do conjunto de treino, considerando que I^+ e I pertencem a mesma categoria e I^- que não pertence. A função de perda é modelada para preservar similaridades entre os trios de imagens, de tal forma que I^+ é mais similar à I que I^-. Consequentemente, assume-se que I^+ pertence à mesma categoria de I, o que não é o caso de I^-. Então a função *triplet* associada para a j-ésima categoria é escrita como:

$$\begin{aligned} &L_T(f^{(j)}(I), f^{(j)}(I^+), f^{(j)}(I^-)) \\ &= \max(0, 1 - \|f^j(I) - f^j(I^-)\|_2^2 \\ &\quad + \|f^j(I) - f^j(I^+)\|_2^2) \end{aligned} \tag{3.32}$$

onde $f^j(I)$ representa o vetor f^j para a imagem I.

2) *Representação de hashing semântico*: Para o *hashing* semântico, assuma que os códigos são formados por q bits. Nesta fase, uma camada totalmente conectada é usada para converter o f de dimensão $c \times b$ em um vetor s de dimensão q. O vetor s pode ser convertido em um código binário de q bits usando a função $sign(s)$, em que $sign(x) = 1$ se $x > 0$ e caso contrário, $sign(x) = 0$. Em seguida, para considerar imagens com múltiplos rótulos, é usada uma função de perda *triplet* ponderada. As funções de

similaridade $sim(I_a, I_b)$ são definidas com base no número de rótulos compartilhados entre as imagens I_a e I_b. Então, para as imagens I, I^+, I^- e $sim(I, I^+) > sim(I, I^-)$, a perda da função *triplet* ponderada é definida como na Equação 3.33.

$$\begin{aligned} &L_{WT}(s(I), s(I^+), s(I^-)) \\ &\quad = (2^{sim(I,I^+)} - 2^{sim(I,I^-)}) L_T(s(I), s(I^+), s(I^-)) \end{aligned} \tag{3.33}$$

onde L_T é a mesma definida na Equação 3.32 e $s(I)$ é o vetor de dimensão q para a imagem I.

3.4.2 *Triplet-based Deep Binary Embedding*

Zhuang et al. [2016] propuseram *Triplet-based Deep Binary Embedding* (TDBE). O principal objetivo desse método é melhorar a complexidade computacional do aprendizado de código binário usando funções *triplet*, uma vez que o custo para treinar redes neurais profundas diretamente com funções *triplets* é alto, devido a enorme possibilidade de combinações de trios de imagens que podem ser selecionados do conjunto de treino. Embora o foco principal desse método seja eficiência no treino, ele foi incluído nessa revisão por também tratar do problema de recuperação de imagens com múltiplos rótulos.

TDBE emprega uma abordagem colaborativa em duas etapas para evitar treinar diretamente a rede neural com função de perda *triplet*. O primeiro estágio do TDBE usa os dados rotulados de treino para inferir um conjunto de códigos binários em que a distância Hamming entre os códigos preserva o *ranking* semântico entre os trios de dados. O segundo estágio usa uma CNN profunda para aprender o mapeamento de imagens para o espaço de código binário (ou seja, para aprender as funções de *hashing*). Para combinar as duas fases, o método propõe um esquema para intercalar o código e a aprendizagem de funções de *hashing* em grupos de bits.

Na primeira fase, uma tarefa de inferência de alta ordem é convertida para um problema de segunda ordem. Resumidamente, uma dada função de perda *triplet* pode ser decomposta em um conjunto de termos *pairwise* para cada trio de imagens. Dessa forma, considerando que as relações *triplet* podem ser codificadas em relações *pairwise*, a função de *hashing* $z_{(r)}$ pode ser modelada como segue. Seja $W \in \mathcal{R}^{n \times n}$ a matriz de pesos em que (i, j)-ésimo elemento de W, w_{ij}, representa a relação de peso entre o i-ésimo e j-ésimo ponto de treino. Assim, cada elemento de W é calculado como: $w_{ij} = \sum_{\forall(i,j)} \alpha_{ij}$, onde α_{ij} são coeficientes correspondentes dos pares (i, j). Haverá um α_{ij} para cada *triplet* em que os pontos x_i e x_j aparecem.

Logo, o problema de otimização *triplet* pode ser escrito como indicado na Equação 3.34.

$$\min_{z_{(r)} \in \{1,1\}^n} z_{(r)}^T W z_{(r)} \tag{3.34}$$

Como a função na Equação 3.34 possui termos não sub-modulares e por isso é difícil de otimizar, o método adota um esquema para criar sub-problemas (ou blocos), cada um envolvendo um subconjunto das variáveis $z_{(r)}$ nas quais as relações de pares são todas sub-modulares. Os sub-problemas são então resolvidos tratando as variáveis que não estão envolvidas no bloco atual como constantes. O problema de inferência para um bloco é escrito como:

$$\min_{z_{(r)} \in \{1,1\}^n} \sum_{i \in \mathcal{S}} u_i z_{r,i} + \sum_{i \in \mathcal{S}} \sum_{j \in \mathcal{S}} v_{ij} z_{r,i} z_{r,j}, \tag{3.35}$$

onde $u_i = 2 \sum_{j \in \mathcal{S}} w_{ij} z_{r,j}$, $v_{ij} = w_{ij}$ e $\mathcal{S}$ é um bloco a ser otimizado. Como o problema de inferência acima para um bloco é sub-modular, então pode ser resolvido eficientemente usando cortes gráficos. Os blocos são formados garantindo que $w_{ij} < 0$ para cada par x_i, x_j no bloco, o que garante sub-modularidade para todos os pares.

O objetivo da segunda fase do método é aprender funções de *hashing* $h(\cdot)$ utilizando aprendizagem profunda, para mapear os pontos de dados x_i para códigos binários. Vimos que os códigos binários são determinados independentemente da função de *hashing* gerada pela rede profunda. No entanto, o ideal é que esses estágios interajam, de modo que a escolha de códigos *hash* seja influenciada não apenas pelas relações de similaridade das imagens (modeladas pela primeira fase do método), mas também pelas características geradas pela rede. Para resolver isso, TDBE propõe um processo intercalado onde infere um grupo de bits dentro de um código, seguido por aprendizado de funções de *hashing* adequadas para esse conjunto de bits e seus predecessores, seguido por inferência do próximo grupo de bits e assim por diante. Isso proporciona um compromisso entre a aprendizagem independente dos códigos e funções de *hashing*.

Para isso, a ideia principal aqui é otimizar a estrutura de *hashing* de maneira incremental em grupo. Mais especificamente, o método assume que existem grupos de bits e cada grupo tem a bits (por exemplo, códigos de 64 bits podem ser divididos em 8 grupos de 8 bits cada). Por conveniência, considere a inferência do p-ésimo grupo de códigos, seguido pelo aprendizado das funções de *hashing* profundas, como o "p-ésimo estágio de treinamento". No estágio de treinamento, primeiro são inferidos os a bits do p-ésimo grupo e depois são treinados os parâmetros da rede de modo a minimizar a perda de entropia cruzada, conforme Equação 3.36:

$$-\sum_{\rho=1}^{r}\sum_{i=1}^{n}[\delta(z_{\rho,i}=1)logz'_{\rho,i}+\delta(z_{\rho,i}=-1)log(1-z'_{\rho,i})] \tag{3.36}$$

onde $\delta(\cdot)$ é uma função de indicação. O p-ésimo estágio requer os r primeiros bits, onde $r = pa$; $z'_{\rho,i}$ é a ρ-ésima saída da última camada sigmóide da i-ésima amostra de treino; $z_{\rho,i}$ é obtido a partir do passo de inferência que serve como o rótulo alvo do problema de classificação com vários rótulos. Observe que no p-ésimo estágio de treinamento, todos os p grupos são usados para orientar o aprendizado das funções de *hashing*. Tendo completado o treinamento das funções de *hashing*, os códigos binários são atualizados para todos os p grupos na saída das funções de *hashing* aprendidas. O efeito disso é garantir que o erro nas funções de *hashing* aprendidas influenciará a inferência do próximo grupo de bits. Essa abordagem de treinamento incremental regula adaptativamente os códigos binários de acordo com a capacidade de ajuste das funções de *hashing* e as propriedades dos dados de treinamento, melhorando continuamente a qualidade dos códigos *hash* e o desempenho final.

3.4.3 *Hashing for Multi-Labeled Data*

O *Hashing for Multi-Labeled Data* (HMLD) [Wang et al., 2017a] é uma abordagem para recuperação de imagem com múltiplos rótulos, que consiste de duas partes: aprendizado de características usando CNN e aprendizado de código *hash*. Assim como métodos anteriores, esse método considera que duas imagens são mais similares se compartilharem mais rótulos. A diferença é que o HMLD define similaridade entre dois pontos como a proporção dos rótulos comuns em relação a todos os rótulos dos pares de imagens, como indicado na Equação 3.37.

$$s_{ij} = 2 \times \frac{|l_i \cap l_j|}{|l_i \cup l_j|} - 1, \tag{3.37}$$

em que s_{ij} representa o grau de similaridade entre as imagens x_i e x_j com respectivos rótulos l_i e l_j, onde $|l|$ é o tamanho do conjunto l. Quanto maior s_{ij} mais similares são x_i e x_j.

Na etapa de extração de características, o método visa preservar a ordem de similaridade entre as imagens, com o intuito de melhorar a qualidade do *ranking* de resposta. Para isso, considere trios de imagens x_i, x_j e x_k e suas respectivas características geradas pela CNN u_i, u_j e u_k. Dessa forma, o método define que a ordem de similaridade será preservada se a seguinte relação for satisfeita: $(s_{ij}-s_{ik})(E_{ij}-E_{ik}) < 0$, onde $E_{ij} = |u_i - u_j|_F^2$ é a distância entre pares de imagens. Essa relação define que se

duas imagens tem mais rótulos em comum, elas devem estar mais próximas no espaço de características do que de outras imagens.

Consequentemente, a função de perda da rede é construída com base nessa relação de distância, como indicado na Equação 3.38.

$$J_{OF} = \frac{1}{N^3} \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{k=1}^{N} \max((s_{ij} - s_{ik})(E_{ij} - E_{ik}), 0), \tag{3.38}$$

onde N é o número de imagens no conjunto de treino. O objetivo dessa função é punir os seguintes casos: quando $s_{ij} > s_{ik}$, $E_{ij} < E_{ik}$; e quando $s_{ij} < s_{ik}$, $E_{ij} > E_{ik}$. Observe que nessas duas situações a ordem de similaridade não é preservada.

Na segunda fase, as características geradas pela primeira fase são dadas como entrada para aprender código *hash*. Nessa parte, um algoritmo de *hashing* baseado em uma função de verossimilhança máxima é construído para otimizar os códigos *hash*. Deste modo, o objetivo é aprender códigos binários $b_i \in \{-1, 1\}^d$ para cada imagem x_i, onde d é o tamanho do código.

Para a construção dos códigos binários, considere que β_{ij} é o produto interno entre os pontos b_i e b_j definido como $\beta_{ij} = b_i^T b_j$; também considere que $B = \{b_i\}_{i=1}^N$ é o conjunto de códigos *hash* de todas as imagens de treino. Dadas estas definições, a função de verossimilhança para imagens com múltiplos rótulos pode ser descrita pela Equação 3.39.

$$p(s_{ij}|B) = e^{-(\theta_{ij} - s_{ij}d)} \tag{3.39}$$

A Equação 3.39 induz que θ_{ij} seja um valor mais apropriado, ajustando ainda mais os códigos *hash*. Note que esta estimativa é significativamente diferente daquela calculada para dados com único rótulo, que tradicionalmente é definida como $p(s_{ij}|B) = a_{ij}^{s_{ij}}(1 - a_{ij})^{1-s_{ij}}$ $s.t. s_{ij} \in \{0, 1\}$ onde $a_{ij} = 1/(1 + e^{-0.5\theta_{ij}})$.

Ao aplicar a função *log* na Equação 3.39, a função de custo de *hashing* para dados com vários rótulos resulta em:

$$\begin{aligned} J_{hash} &= logp(B|S) \\ &= logp(S|B)p(B) \\ &= -\sum_{s_{ij} \in S} (\theta_{ij} - s_{ij}d)^2 + logp(B). \end{aligned} \tag{3.40}$$

onde $S = \{s_{ij}\} \in \mathbb{R}^{N \times N}$ é o conjunto de similaridades entre os pares de imagens do treino.

Uma vez que B é formado por valores discreto, a função da Equação 3.40 é de

difícil otimização. Por isso, B é substituído pelas características originais retornadas pela primeira fase.

Diferente dos métodos anteriores dessa categoria, o HMLD define similaridade como a proporção de rótulos compartilhados entre as imagens (Equação 3.37). Essa estratégia parece descrever naturalmente os vários níveis de similaridade existentes entre as imagens.

3.4.4 *Deep Supervised Hashing with Code Operation*

O *Deep Supervised Hashing with Code Operation* (DSOH) [Song & Tan, 2018] é um método de *hashing* profundo proposto para aprender múltiplos níveis de similaridades semânticas ao gerar códigos binários para imagens de múltiplos rótulos. Além disso, DSOH constrói uma rede de operação de código para unir conceitos semânticos de várias imagens de consulta, com o intuito de aumentar as possibilidades de consulta do usuário. O objetivo dos autores ao propor uma rede de operação de código é preencher a lacuna de intenção de consulta, uma vez que o usuário nem sempre consegue colocar sua intenção de consulta em uma única imagem, sendo este um problema pouco explorado pelos métodos da literatura.

O modelo é formado por duas sub-redes: uma rede de *hashing* profundo (*Deep Hashing Network* - DHN) e uma rede de operação de código (*Code Operation Network* - CON).

A DHN consiste de uma CNN para extrair características das imagens seguida de uma rede neural profunda completamente conectada (*Deep Neural Network* - DNN).

A segunda rede é construída para manipular os níveis semânticos das características geradas pela DHN. Nessa rede são definidos três tipos de operação: união, subtração e interseção, onde todos são operadores de pares que recebe um par de códigos *hash* e os transforma em um novo. Essas três operações são descritas a seguir.

Operação de União e Interseção: dados dois códigos *hash* $h(x_1)$ e $h(x_2)$, as operações de união e interseção são definidos como funções lineares da seguinte forma:

$$h_{x_{1\vee 2}} = f_u(h(x_1), h(x_2)) = W_u[h(x_1), h(x_2)] \tag{3.41}$$

$$h_{x_{1\wedge 2}} = f_t(h(x_1), h(x_2)) = W_t[h(x_1), h(x_2)] \tag{3.42}$$

onde $[h_1, h_2]$ denota uma operação de concatenação. W_u e W_t são duas matrizes de parâmetros a serem aprendidas para união e interseção, respectivamente. $h_{x_{1\vee 2}}$ e $h_{x_{1\wedge 2}}$

são códigos resultantes do processo de fusão, considerando seus respectivos rótulos l_u, definido por $l_1 \oplus l_2 = l_1 \cup l_2$; e l_t, definido por $l_1 \otimes l_2 = l_1 \cap l_2$.

Operação de Subtração: essa operação é definida com base na operação de união (Equação 3.41) da seguinte forma:

$$h_{x_{1\vee 2\setminus 2}} = f_s(h_{x_{1\vee 2}}, h(x_2)) \equiv W_s[h_{x_{1\vee 2}}, h(x_2)] \tag{3.43}$$

em que W_s é a matriz de parâmetros para o operador interseção. $h_{x_{1\vee 2\setminus 2}}$ é o código subtraído com múltiplos rótulos $l_1 \oplus l_2 \ominus l_2$.

Por meio desses operadores, DSOH consegue algumas vantagens, como por exemplo, aumentar o conjunto de características do treino, o que reduz o problema de desbalanceamento e dispersão dos dados (natural de coleção de imagens com múltiplos rótulos). Além disso, pode mais facilmente construir trios de imagens úteis usando a operação de códigos para capturar uma estrutura de dados de similaridade mais complicada. Por exemplo, pode usar a operação de união para combinar quaisquer dois conceitos (embora eles raramente apareçam juntos e, portanto, seja difícil obter suas amostras de treinamento) e gerar os códigos *hash* correspondentes; e finalmente, o nível de similaridade complexo pode ser simulado com as três operações básicas.

A função de perda desse modelo consiste de duas partes. A primeira modela os múltiplos níveis de similaridade enquanto a segunda faz a classificação ponderada com o objetivo de distinguir os códigos *hash* no espaço semântico.

Para capturar a estrutura de similaridade entre os códigos *hash*, DSOH adota o esquema de trios de imagens. Dessa forma, um determinado trio de imagens $\{I_1, I_2, I_3\}$ e seus rótulos $\{l_1, l_2, l_3\}$, passa primeiro pela CNN para obter seus respectivos códigos binários $b_1 = sign(h(f_{CNN}(I_1)))$, $b_2 = sign(h(f_{CNN}(I_2)))$ e $b_3 = sign(h(f_{CNN}(I_1)))$, e três códigos derivados pela rede CON $b_4 = f_u(b_1, b_2)$, $b_5 = f_t(b_1, b_2)$ e $b_5 = f_t(b_4, b_2)$. Esses seis códigos são divididos nos seguintes trios de imagens: $\{b_1, b_2, b_3\}$, $\{b_1, b_2, b_4\}$ e $\{b_1, b_2, b_5\}$, onde cada trio define algum aspecto da estrutura de similaridade e corresponde a um termo separado na função de perda *triplet* da final de perda final.

A função de perda *triplet* para o trio $\{b_1, b_2, b_3\}$ é definida como:

$$\begin{aligned} L_{tr1}(b_1, b_2, b_3) = &\max\{0, d_H(b_*, b_1) - d_H(b_*, b_1) + \alpha_1\} \\ &s.t \quad b \in \{-1, 1\}^k \end{aligned} \tag{3.44}$$

onde $d_H(\cdot, \cdot)$ é a distância de Hamming, b_* é a código selecionado como consulta do trio de imagem, $b_{*,1}$ e $b_{*,2}$ são os códigos de imagem mais similar e menos similar a b_*, respectivamente. α_1 é um coeficiente de margem, adaptativa calculado como: $\alpha_1 = \frac{|l_* \cap l_{*,1}| - |l_* \cap l_{*,2}|}{|l_*|} \cdot m$, em que l_*, $l_{*,1}$ e $l_{*,2}$ são os rótulos das imagens b_*, $b_{*,1}$ e $b_{*,2}$,

respectivamente; e m é um escalar cujo valor é determinado pelo número de bits do código *hash*.

De forma similar, para o grupo $\{b_1, b_2, b_4\}$ e $\{b_1, b_2, b_5\}$ (em que $l_4 = l_1 \cup l_2$ e $l_4 = l_1 \cap l_2$), suas relações de similaridade devem satisfazer respectivamente:

$$\begin{aligned} |l_1| > |l_2| &\Rightarrow d_H(b_1, b_4) < d_H(b_1, b_2), d_H(b_2, b_5) < d_H(b_1, b_2) \\ |l_1| < |l_2| &\Rightarrow d_H(b_2, b_4) > d_H(b_1, b_2), d_H(b_1, b_5) < d_H(b_1, b_2) \\ |l_1| = |l_2| &\Rightarrow d_H(b_1, b_4) = d_H(b_2, b_4), d_H(b_1, b_5) = d_H(b_2, b_5) \end{aligned} \tag{3.45}$$

Deste modo, a segunda parte da função de perda para esses dois trios de imagens pode ser definida como indicado na Equação 3.46.

$$\begin{aligned} L_{tr2} = \max\{0, &y d_h(b_1, b_4) + (1-y) d_h(b_2, b_4) \\ &- d_h(b_1, b_2) + \alpha_2\} + \max\{y d_h(b_2, b_5) \\ &+ (1-y) d_h(b_1, b_5) - d_h(b_1, b_2) + \alpha_3\} \\ s.t \quad &b \in \{-1, 1\}^k \end{aligned} \tag{3.46}$$

onde os coeficientes de margem são calculados como: $\alpha_2 = \frac{(yn_1+(1-y)n_2)^2 - n_3 n_4}{n_3(yn_1+(1-y)n_2)} \cdot m$ e $\alpha_2 = \frac{|n_1-n_2|n_4}{n_1 n_2} \cdot m$; em que $n_1 = |l_1|$, $n_2 = |l_2|$, $n_3 = |l_1 \cup l_2|$, $n_4 = |l_1 \cap l_2|$, e se $n_1 > n_2$ então $y = 1$, caso contrário $y = 0$.

Assim, a função de perda *triplet* para similaridade de múltiplos níveis é obtida combinando as funções definidas nas Equações 3.44 e 3.46.

$$L_{tr}(b_1, b_2, b_3) = L_{tr1}(b_1, b_2, b_3) + L_{tr1}(b_1, b_2, b_3) \tag{3.47}$$

A segunda parte da função de perda do modelo é a função de classificação ponderada. Para essa parte DSOH adota uma função de perda de entropia cruzada ponderada para garantir que cada código *hash* seja individualmente consistente com seu próprio conceito semântico:

$$L_{b,l} = -\sum_{j=1}^{C} (w_{pos} \cdot l_j log(\hat{l}_j) + (1 - l_j) log(1 - \hat{l}_j)) \tag{3.48}$$

onde $\hat{l}$ é o rótulo previsto do DSOH e w_{pos} é o peso das amostras positivas.

Finalmente, a função de perda final do modelo para N trios de imagens no treino $\{b_{i1}, b_{i2}, b_{i3}\}_{i=1}^{N}$ é definida como segue:

$$\begin{aligned}\mathcal{L} = \sum_{i=1}^{N}(\sum_{j=1}^{3} L_{cl}(b_{ij}, l_{ij}) + \lambda_1(L_{cl}(b_{i4}, l_{i4}) + L_{cl}(b_{i5}, l_{i5}) \\ + L_{cl}(b_{i6}, l_{i6})) + \lambda_2 L_{tr}(b_{i1}, b_{i2}, b_{i3}))\end{aligned} \tag{3.49}$$

onde λ_1 e λ_2 são parâmetros para balancear a perda de classificação do operador de código e da função *triplet*.

Para otimizar a função da Equação 3.49 dois tipos de relaxação são aplicados. Primeiro, a função de ativação tangente hiperbólica é utilizada para aproximar os códigos binários. Depois, a distância de Hamming é substituída pela distância Euclidiana, ou seja, $d(h_1, h_2) = \|h_1 - h_2\|_2^2$. Deste modo, a função final (Equação 3.49) é reescrita como:

$$\begin{aligned}\mathcal{L} = \sum_{i=1}^{N}\{\sum_{j=1}^{3} L_{cl}(h_{ij}, l_{ij}) + \lambda_1(L_{cl}(h_{i4}, l_{i4}) + L_{cl}(h_{i5}, l_{i5}) \\ + L_{cl}(h_{i6}, l_{i6})) + \lambda_2 L_{tr}(h_{i1}, h_{i2}, h_{i3}, l_{i1}, l_{i2}, l_{i3}) \\ + \lambda_3 \sum_{j=1}^{3}(\||h_{i,j}| - \mathbf{1}\|_1)\}\end{aligned} \tag{3.50}$$

onde $h_* = h(f_{CNN}(I_*))$, $h_{i,4} = f_u(h_{i1}, h_{i2})$, $h_{i,5} = f_t(h_{i1}, h_{i2})$, $h_{i,6} = f_s(h_{i4}, h_{i2})$. $\mathbf{1}$ é um vetor onde todos os elementos são 1. $\|\cdot\|_1$ denota a norma L1 e λ_3 controla o efeito da quantização.

3.5 Métodos *Listwise*

Nesta categoria, uma informação *listwise* indica a ordem de *ranking* de uma conjunto de imagens da coleção com respeito a uma consulta. Na Figura 3.7, para uma dada consulta, a rede gera uma lista de *ranking* de acordo com a ordem de similaridade em relação à consulta. Neste tipo de abordagem, também são utilizadas funções *triplet* para calcular a perda de *ranking*, com o intuito de verificar a ordem de similaridade entre as imagens da coleção e a consulta, indicando quais são as imagens mais similares e menos similares.

3.5.1 *Deep Semantic Ranking Based Hashing*

O *Deep Semantic Ranking Based Hashing* (DSRH) [Zhao et al., 2015] combina uma abordagem de *ranking* semântico e modelo de *hashing* profundo para solucionar o pro-

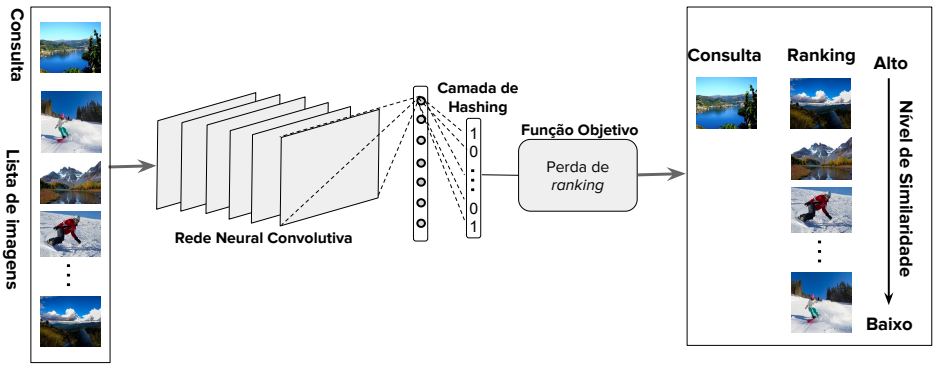


Figura 3.7. Visão geral dos métodos da categoria *listwise*.

blema de preservação de múltiplos níveis de similaridade entre imagens com múltiplos rótulos.

Para fazer extração das características das imagens de entrada, esse método usa uma AlexNet, adicionando uma camada de *hashing* depois da primeira camada completamente conectada da rede para gerar códigos binários compactos.

Para preservar a estrutura dos múltiplos níveis semânticos, DSRH adota uma abordagem em que para os pontos de dados individuais, a ordem de *ranking* dos vizinhos, calculada a partir da distância de Hamming, seja mantida de acordo com seus rótulos semânticos em termos de uma métrica de avaliação de *ranking*.

Dessa forma, considere uma consulta q, selecionada a partir da coleção de dados. Para q, um nível de similaridade semântica r de uma imagem x da base de dados em relação a q pode ser calculada com base no número de seus rótulos comuns. As imagens mais semelhantes são aquelas que compartilham todas os rótulos de q e recebem um nível $r = |\mathcal{Y}_q|$. Assim, a próxima imagem mais similar, que compartilha quaisquer $|\mathcal{Y}_q| - 1$ dos rótulos, recebe nível $r = |\mathcal{Y}_q| - 1$. Por fim, as imagens dissimilares não compartilham nenhum dos rótulos e recebem um nível $r = 0$. Assim, é possível obter uma lista de *ranking* q, ranqueando as imagens da coleção em ordem decrescente de seus níveis de similaridade. Nesse sentido, alguns critérios de avaliação podem ser usados para medir a consistência dos *rankings* preditos por funções de *hashing*, como a métrica de avaliação NDCG (Normalized Discounted Cumulative Gain), que é utilizada como função de perda desse modelo.

Dada uma consulta, o valor NDCG para p imagens recuperadas é dado por:

$$NDCG@p = \frac{1}{Z} \sum_{i=1}^{p} \frac{2^{r_i} - 1}{log(1+i)} \tag{3.51}$$

onde r_i é a similaridade entre a imagem de consulta e a imagem na i-ésima posição

do *ranking*, que é definida como o número de rótulos compartilhados entre a imagem de consulta e a i-ésima imagem recuperada, e Z é a constante de normalização que garante que o NDCG seja um valor entre 0 e 1.

Para otimização da Equação 3.51 é empregado um esquema de perda substituta. Dessa forma, dada uma consulta q e uma lista de *ranking* $\{x_i\}_{i=1}^{M}$, a função de perda desse método é definida baseada em um conjunto de trios de código *hash*, como segue:

$$\begin{aligned} &L(h(q), \{h(x_i)\}_{i=1}^{M}) = \\ &\sum_{i=1}^{M} \sum_{j:r_j<r_i} \max\{0, \delta d_H(h(q), h(x_i), h(x_j)) + \rho\}, \end{aligned} \tag{3.52}$$

onde M é o tamanho da lista de *ranking*, $\delta d_H(h(q), h(x_i), h(x_j)) = d_H(h, h_1) - d_H(h, h_2)$, $d_h(\cdot, \cdot)$ é a distância de Hamming e ρ é um parâmetro de margem que controla a margem mínima entre a distância dos pares. A partir da definição de NDCG, pode-se observar que os itens mais bem ranqueados têm maior fator de ganho para o escore. Isto reflete melhor o desempenho dos modelos de *ranking* em sistemas práticos de recuperação de imagens, pois os usuários geralmente prestam mais atenção aos primeiros resultados. Assim, deseja-se que o *ranking* desses itens possa ser previsto com maior precisão do que os dos demais. No entanto, a Equação 3.52 trata todos os trios igualmente, o que não é desejável. Dessa forma, para adicionar pesos adaptáveis relacionados aos níveis de similaridade das imagens da coleção, a Equação 3.52 é reescrita como indicado na Equação 3.53.

$$\begin{aligned} &L(h(q), \{h(x_i)\}_{i=1}^{M}) = \\ &\sum_{i=1}^{M} \sum_{j:r_j<r_i} \omega(r_i, r_j) \max\{0, \delta d_H(h(q), h(x_i), h(x_j)) + \rho\}, \end{aligned} \tag{3.53}$$

De acordo com NDCG, o peso ω na Equação 3.53 é definido como: $\omega(r_i, r_j) = \frac{2^{r_i} - 2^{r_j}}{Z}$; onde Z é uma constante de normalização na Equação 3.51. Assim, quanto maior a relevância de x_i para q em relação a x_j, maior será o declínio do valor de NDCG caso x_i seja ranqueado depois de x_j. Observe que quando $\omega(r_i, r_j) = 1$ a função de perda equivale à Equação 3.52.

Uma característica interessante desse método é que a função de perda pode ser adaptada para outras métricas de avaliação em que seja possível medir a relação de similaridade entres as amostras de treino.

3.5.2 *Order-Sensitive Deep Hashing*

Chen et al. [2018] propuseram o modelo *Order-Sensitive Deep Hashing* (OSDH) para recuperação de imagem no domínio de imagens médicas. Nesse cenário, múltiplos sintomas e doenças podem ser observados em uma imagem. Dessa forma, os autores tratam a recuperação como um problema de aprendizagem de *hashing* para imagens com vários rótulos e propõem uma função *triplet* para preservar a similaridade entre os códigos binários.

O OSDH utiliza uma AlexNet para desenvolver um modelo de *hashing* profundo para aprender conjuntamente características visuais das imagens e uma função de mapeamento para gerar códigos binários compactos.

Para preservar a similaridade entre as amostras, o método adota a estratégia descrita a seguir. Para cada imagem de consulta x_q, seu nível de similaridade semântica r em relação a uma amostra x_i é calculado pelo número de rótulos comuns compartilhados por ambas $|\mathcal{Y}_q \cap \mathcal{Y}_i|$, onde $\mathcal{Y}_q$ e $\mathcal{Y}_i$ são o conjunto de rótulos das imagens x_q e x_i, respectivamente. Ao atribuir um nível de similaridade para cada imagem de treino, uma lista de *ranking* para x_q pode ser obtida classificando as amostras na ordem decrescente de similaridade. Considerando cada consulta x_q e sua lista de *ranking* correspondente $\{x_i\}_{i=1}^{M}$, é definida a seguinte função *triplet* baseada na perda de *ranking* sobre os códigos binários:

$$\mathcal{L}_R(x_q) = \sum_{i=1}^{M} \sum_{j:r_j<r_i} \frac{2^{r_i} - 2^{r_j}}{Z} \max(0, D(b_q, b_i) - D(b_q, b_j) + \rho) \tag{3.54}$$

em que $D(b_1, b_2)$ é a distância de Hamming entre os códigos binários b_1 e b_2; ρ é um limiar para controlar a margem mínima entre a distância de Hamming entre os dois pares; r_i e r_j são os níveis de similaridade das amostras x_i e x_j com respeito a consulta x_q; e Z é uma constante relacionada ao tamanho da lista de *ranking*. O coeficiente $\frac{2^{r_i}-2^{r_j}}{Z}$ atribui peso maior para o par (x_i, x_j) quando o valor de x_i é mais relevante para x_q que x_j. Somando-se todas as amostras x_i na lista de ranking e seu par (x_i, x_j), a minimização da Equação 3.54 é capaz de incentivar a preservação da lista de *ranking* no espaço de Hamming para a consulta x_q. Para preservar a estrutura semântica de similaridade multinível, pode-se optar por otimizar a soma da Equação 3.54 em todas as amostras de treinamento, $\sum_{x_q \in X} \mathcal{L}_R(xq)$.

Enquanto a perda na Equação 3.54 está relacionada ao nível de similaridade relativo, as informações de rótulo não são totalmente exploradas para aprender funções de *hashing*. Para explorar ainda mais as informações dos múltiplos rótulos, o método inclui na saída de ativação da rede $g(x_i, \theta)$ uma função de classificação. O modelo emprega

a entropia cruzada como função de perda de classificação para múltiplos rótulos:

$$\mathcal{L}_C(y_i, \hat{y}_i) = -\sum_{c=1}^{C}(y_{ic} ln(\hat{y}_{ic}) ln(1 - \hat{y}_{ic})) \tag{3.55}$$

onde $y_{ic} \in \{0, 1\}^C$ indica se a amostra x_i contém o rótulo c. A probabilidade de x_i pertencer à c-ésima classe é inferida por um classificador linear. Ao acumular a perda de entropia cruzada de cada classe, a Equação 3.55 apresenta a perda de classificação dos vários rótulos de x_i.

Por fim, considerando a função de perda de *ranking* (Equação 3.54) e a função de perda de classificação (Equação 3.55), a função objetivo de otimização do OSDH é dada pela Equação 3.56.

$$\arg\min \mathcal{L} = \lambda_R \sum_{x_q \in X} \mathcal{L}_R(x_q) + \lambda_C \sum_{i=1}^{N} \mathcal{L}_R(y_i, \hat{y}_i) + \lambda_p \mathcal{L}_p \tag{3.56}$$

onde λ_R, λ_C e λ_p são hiperparâmetros para balancear o efeito dos três termos. O terceiro termo é um regularizador.

Para resolver o problema de otimização discreta da Equação 3.56, esse trabalho realiza o mesmo processo de otimização utilizado em [Cao et al., 2017].

3.5.3 *Object-Location-Aware Hashing*

Huang et al. [2018] propuseram o *Object-Location-Aware Hashing* (OLAH) para aprender máscaras binárias que possam identificar regiões próximas de objetos da imagem. Dada a natureza compacta dos códigos *hash*, a proposta desse modelo é incluir no código apenas informações da imagem que realmente são úteis para recuperação, eliminando informação de *background*.

O OLAH possui quatro blocos de construção, que incluem: (i) uma sub-rede de convolução para extrair representações das imagens, (ii) uma sub-rede para construir máscara binárias para identificar regiões próximas de objetos da imagem, (iii) uma operação de subamostragem para converter os mapas de características em um vetor de característica focado em objetos e (iv) uma função de perda *triplet* para preservar similaridades relativas entre as imagens e uma perda de entropia cruzada para rótulos das imagens.

O primeiro bloco é baseado na arquitetura de uma GoogLeNet. Desta, é removida a última camada de subamostragem e adicionada uma camada de convolução 3×3 com c mapas de características na saída. Ou seja, essa sub-rede converte uma imagem de

entrada em um conjunto de mapas de características, que serão usados para representar a imagem.

Na sub-rede de máscara binária, uma camada de convolução de *kernel* 1×1 mapeia os c mapas de características de tamanho $h \times w$ para um mapa de características $\mathcal{F}$ de tamanho $h \times w$. O mapa de características $\mathcal{F}$ é seguido por uma camada softmax que gera um mapa de características $\mathcal{S}$ de tamanho $h \times w$, onde os elementos em $\mathcal{S}$ formam uma distribuição de probabilidade. Mais especificamente, sejam $\mathcal{S}_{i,j}$ e $\mathcal{F}_{i,j}$ os elementos na i-ésima linha e na j-ésima coluna de $\mathcal{S}$ e $\mathcal{F}$ respectivamente, com $1 \leq i \leq h$ e $1 \leq j \leq w$. A função softmax é definida como:

$$\mathcal{S}_{i,j} = \frac{exp(\mathcal{F}_{i,j})}{\sum_{i=1} h \sum_{j=1} w}. \tag{3.57}$$

No topo do mapa de pesos $\mathcal{S}$ é adicionada uma camada *threshold* que discretiza os elementos em $\mathcal{S}$ para obter $\mathcal{T}$. Esta função *threshold* é definida como:

$$\mathcal{T}_{i,j} = Threshold(\mathcal{S}_{i,j}) \begin{cases} 1, & \mathcal{S}_{i,j} \geq \theta \\ 0, & \mathcal{S}_{i,j} < \theta \end{cases} \tag{3.58}$$

em que θ é um *threshold* predefinido. Por definição da função *threshold*, a derivada é quase zero em qualquer ponto, tornando difícil a aplicação do algoritmo de retropropagação no treinamento. Para resolver esse problema o método aplica a mesma solução proposta em [Courbariaux et al., 2016] que usa o "estimador direto" para estimar ou propagar os gradientes da função *threshold*. Após aplicação desse estimador, a saída da camada *threshold* é um mapa de "máscara" binária, onde cada um dos elementos tem valor 0 ou 1. As regiões com os elementos 1 são consideradas como locais prováveis de objetos, caso contrário, as regiões com elementos 0 são consideradas *background*.

A terceira parte do algoritmo recebe a saída dos dois primeiros blocos, mapas de características e máscara binária, e aplica um processo de subamostragem por média ponderada para gerar um vetor de característica c-dimensional, onde este vetor de característica é considerado como uma representação que presta mais atenção nos objetos em primeiro plano e ignora o *background* de imagem. Considere $\mathcal{T}$ como a máscara binária e $\mathcal{M}^{(1)}, \mathcal{M}^{(2)}, ..., \mathcal{M}^{(c)}$ como os c mapas de características retornados pela sub-rede de convolução. Note que cada $\mathcal{M}^{(1)}, \mathcal{M}^{(2)}, ..., \mathcal{M}^{(c)}$ e $\mathcal{T}$ são do mesmo tamanho $h \times w$. Dessa forma, o vetor de característica $\mathbf{x} = (x_1, x_2, ..., x_c)$ é obtido aplicando subamostragem por média ponderada, calculando cada $x_k (k = 1, 2, ..., c)$ da seguinte forma:

$$x_k = \sum_{i=1}^{h} \sum_{j=1}^{w} \mathcal{T}_{i,j} \times \mathcal{M}_{i,j}^{(k)} \tag{3.59}$$

No topo do vetor de características c-dimensional, que é a saída do processo de subamostragem, é adicionada uma camada totalmente conectada para gerar um vetor q-dimensional, seguido por uma camada de ativação que utiliza a função tangente hiperbólica para restringir cada um dos elementos de saída para o intervalo $[-1, 1]$. O vetor de saída da tangente hiperbólica é um código *hash* aproximado de valor real. Na previsão, esse código aproximado pode ser convertido em um código binário de q bits via binarização.

A função de perda desse modelo é uma combinação de duas funções. A primeira é uma perda *triplet* visando preservar a similaridades entre imagens com múltiplos rótulos, definida na Equação 3.60.

$$\begin{aligned} & L_T(B(I), B(I^+), B(I^-)) \\ & \quad = (2^{sim(I,I^+)} - 2^{sim(I,I^+)}) \\ & \quad \max(0, m_q + \|B(I) - B(I^+)\|_2^2 - \|B(I) - B(I^-)\|_2^2) \\ & \quad s.t.\ B(I), B(I^+), B(I^-) \in [-1, 1]^q \end{aligned} \tag{3.60}$$

onde no trio de imagem (I, I^+, I^-), I é considerada a consulta, I^+, I^- são as imagens mais e menos similares a I, respectivamente. $B(I), B(I^+), B(I^-)$ denotam os códigos *hash* do trio de imagens. m_q representa um parâmetro de margem ajustável dependendo do tamanho do código *hash* q. A função $sim(I_a, I_b)$ descreve a similaridade entre os pares de imagens e é definida pelo número de rótulos compartilhados entre I_a e I_b.

A segunda parte da função de perda é uma entropia cruzada para separar rótulos de classes das imagens. O OLAH considera que todas as imagens pertencem a n categorias no total. Para uma imagem de entrada, $y = (y_1, y_2, ..., y_n)$ representa seus rótulos. Para $i = 1, 2, ..., n$, $yi = 1$ se a imagem de entrada pertence à i-ésima categoria, caso contrário, $y_i = 0$. No topo do vetor c-dimensional que são saídas do da camada de subamostragem, é construído uma camada totalmente conectada que gera um vetor $x = (x_1, x_2, ..., x_n)$. Então para $i = 1, 2, ..., n$, x_i é mapeado para uma probabilidade no intervalo $[0, 1]$ pela função sigmóide $P(x_i) = \frac{1}{1+exp(-x_i)}$, onde $P(x_i)$ representa a probabilidade de uma imagem pertencer a i-ésima categoria. A entropia cruzada pode agora ser definida em termos de $P(x_i)$, como indicado na Equação 3.61.

$$L_c(y, x) = -\sum_{i=1}^{n} y_i log P(x_i) \tag{3.61}$$

Por fim, as duas funções (Equações 3.60 e 3.61) são combinadas para compor a função de perda do modelo.

$$\begin{aligned} L_{OLAH} =& \frac{1}{N_T}\sum_{i=1}^{N_T} L_T^w(B(I), B(I^+), B(I^-)) \\ & \lambda\frac{1}{N_c}\sum_{j=1}^{N_c} L_c(y(I_j), x(I_j)), \end{aligned} \tag{3.62}$$

onde (I_i, I_i^+, I_i^-) representa o i-ésimo trio $(i = 1, 2, ..., N_T)$, $x(I_j)$ e $y(I_j)$ correspondem às variáveis x e y definidos na Equação, N_T e N_C são o número de trios e imagens no treino, respectivamente. O hiper-parâmetro λ controla o equilíbrio entre as duas funções de perdas.

3.5.4 *Rank-Consistency Deep Hashing*

O principal objetivo do método *Rank-Consistency Deep Hashing* (RCDH) [Ma et al., 2018] é manter a consistência de similaridade entre duas listas de *ranking*: a do espaço original e a do espaço de Hamming. Para isso os autores propõem um modelo de *hashing* profundo que é aprendido sob duas restrições na camada superior da rede: (i) a ordem de similaridade calculada pela distância de Hamming deve ser consistente com aquela no espaço de entrada; (ii) a perda entre os códigos *hash* aprendidos e os vetores de características de valores reais deve ser minimizada. Para explorar as informações de múltiplos rótulos, é incluído um termo de classificação na função de perda, o que minimiza a perda de entropia softmax para múltiplos rótulos.

Para alcançar esse objetivo, o RCDH é modelado como segue. Seja uma imagem no espaço original, a ordem de similaridade é obtida a partir do número de rótulos comuns com respeito às imagens da coleção, como $L^O = (l_1^O, l_2^O, ..., l_n^O)$. Para vetores de códigos binários no espaço de Hamming, a ordem de similaridade pode ser obtida da mesma forma, como $L^H = (l_1^H, l_2^H, ..., l_n^H)$. Para alcançar funções de *hashing* eficientes, o método deseja manter L^O similar a L^H tanto quanto possível, ou seja, $l_i^O = l_i^H$. No espaço original, o número de rótulos entre as imagens i e j pode ser obtido por $C_{ij} = y_i^T y_j$, em que y representa os rótulos. Considere que $\{C_{ij}\}, j = 1, 2, ..., N$ tenha m_i diferentes valores, então o espaço original pode ser dividido em m_i subconjuntos, $S_{i,1}^O, S_{i,2}^O, ..., S_{i,m_i}^O$ com base em C_{ij} de forma decrescente. Como resultado, $S_{i,1}^O$ torna-se o conjunto mais próximo da imagem i e cada ponto em $S_{i,1}^O$ tem a mesma similaridade em relação a i. Da mesma forma, pode-se obter a partição do espaço de Hamming, a distância de Hamming pode ser divida em m_i intervalos, onde os limites superior e

inferior desses intervalos são denotados por $d^U_{i,j}$ e $d^U_{i,j}$ para $j = i, 2, ..., m_i$, respectivamente. Se denotamos como $S^H_{i,j}$ os pontos cuja distância de Hamming em relação à i estão entre $d^L_{i,j}$ e $d^U_{i,j}$, a proposta dessa abordagem é manter a consistência entre $S^O_{i,j}$ e $S^H_{i,j}$.

Dessa forma, a função objetivo do modelo é definida como:

$$\begin{aligned}\min \mathcal{J} = & \Bigg[\sum_{i=1}^{N} \sum_{j=1}^{m_i} \sum_{k \in S^O_{i,j}} -log\sigma(D^H_{i,k} - d^L_{i,j}) \\ & - log\sigma(d^U_{i,j} - D^H_{i,k}) \Bigg] + \Bigg[\sum_{i=1}^{N} \|b_i - u_i\|_2^2 \Bigg] \\ & - \Bigg[\sum_{i=1}^{N} log\Bigg(\frac{exp[y_i^T(Wu_i + v)]}{1^T exp(Wu_i + v)} \Bigg) \Bigg]\end{aligned} \tag{3.63}$$

A primeira parte da Equação 3.63 corresponde à perda de *ranking*, onde $D^H_{i,k} = \frac{K - u_i^T u_k}{2}$ é a distância entre u_i e u_k que são obtidos a partir da saída da rede neural. Por questões de otimização, foram utilizados valores reais da saída da rede para calcular a distância, uma vez que valores discretos tornam esse processo difícil. Por esse motivo, o termo $\sum_{i=1}^{N} \|b_i - u_i\|_2^2$ é adicionado à função para garantir a racionalidade de relaxar o código binário b por u.

Por fim, a terceira parte da Equação 3.63 explora quais rótulos as imagens têm em comum. Isso é feito por meio de uma camada de perda softmax para múltiplos rótulos, tratada como um multi-classificador para fornecer este termo de informação. Note que essa perda de entropia é diferente da perda de entropia cruzada sigmóide, como usada pela maioria das tarefas de classificação de múltiplos rótulos. Ao invés de obter os múltiplos rótulos de uma imagem, os autores consideram que é necessário apenas maximizar as respostas dos rótulos aos quais as imagens de consulta estão associadas. Por isso, estenderam a perda convencional softmax para uma versão com vários rótulos para destacar a magnitude dos múltiplos rótulos. Nesse termo da equação, $W \in \mathbb{R}^{C \times K}$ é a matriz de projeção, $v \in \mathbb{R}^{C \times 1}$ o vetor de viés e 1 é um vetor onde posições preenchidas com um indicam a presença do rótulo na imagem.

3.5.5 Resumos dos trabalhos relacionados

A Tabela 3.1 apresenta um resumo dos métodos explorados neste capítulo, incluindo informações de coleções de dados e métricas utilizadas na avaliação dos métodos. O modelo PSH (cf. Seção 3.3.7) não é apresentado nessa tabela, visto que é um método aplicado ao domínio textual.

Tabela 3.1. Coleções de dados e métricas utilizadas para avaliação dos métodos

Método	Coleção de Dados	Métricas
DBE [Liu et al., 2017]	CIFAR-10, MS-COCO	mAP
DMLH [Zhong et al., 2017]	NUSWIDE, MIRFLICKR-25K	Precision@N, Recall, mAP
MTDH [Liang et al., 2017]	1,097,649 vehicle images	mAP
DCVH [Liu & Qi, 2018]	MS-COCO MIRFLICKR-25K	mAP, Precisão-Revocação
MSDH [Lu et al., 2017]	NUSWIDE, MIRFLICKR-25K	NDCG@100 e ACG@100
DMSSPH [Wu et al., 2017]	VOC-2007, VOC-2012, MIRFLICKR-25K	NDCG@100, ACG@100, Weighted mAP, J-NDCG@100, J-ACG@100, J-mAP Ponderado
DMSH [Li et al., 2017]	NUSWIDE	mAP, mAP Ponderado
ISDH Zhang et al. [2018]	NUSWIDE, Flickr, VOC2012	ACG, NDCG, mAP
DUAH [Wu et al., 2018]	VOC 2012, NUSWIDE, MIRFLICKR-25K	NDCG, ACG e Weighted MAP
IDHN [Zhang et al., 2019]	VOC 2012, NUSWIDE, FLICKR e IAPRTC12	NDCG, ACG e MAP
IAH [Lai et al., 2016]	VOC-2007, VOC-2012, MIRFLICKR-25K	NDCG@1000, ACG@1000, mAP e mAP Ponderado
TDBE [Zhuang et al., 2016]	CIFAR-10, MIT-Indoor, NUSWIDE	mAP e Precisão Top@k
HMLD [Wang et al., 2017a]	CIFAR-100, IAPRTC-12	ACG@100, NDCG@100, mAP, mAP ponderada
DSOH [Song & Tan, 2018]	MIRFLICKR25K, VOC-2012, MS-COCO	NDCG, ACG e mAP Ponderado
DSRH Zhao et al. [2015]	NUSWIDE, MIRFLICKR-25K	NDCG@100, ACG@100 and mAP Ponderada Top@5000
OSDH Chen et al. [2018]	NIH Chest X-ray database	NDCG@100, ACG@100 and mAP
OLAH Huang et al. [2018]	NUS-WIDE/ VOC-2007/ VOC-2012/ MIRFLICKR-25K	NDCG, ACG, MAP e mAP Ponderado
RCDH Ma et al. [2018]	IAPRTC12, MIRFLICKR-25K	NDCG@100 e ACG@100

Por serem trabalhos voltados ao cenário de múltiplos rótulos, a maioria dos métodos usam coleções de dados compostas por imagens que possuem mais de um rótulo, como apresentado na Seção 2.3. Contudo, alguns estudos testaram seus modelos em coleções de dados de rótulo único, como é o caso da CIFAR-10. Isso mostra que tais estratégias são naturalmente aplicáveis nesse contexto.

As métricas mais empregadas nos métodos para avaliar a qualidade da recuperação de imagens são: mAP, ACG e NDCG com algumas variações. O método DMSSPH, por exemplo, utiliza essas métricas combinadas com o coeficiente de Jaccard[3]. Na Tabela 3.2, esta medida é representada pela letra J (J-NDCG@100, J-ACG@100, J-mAP Ponderado).

Note que as métricas utilizadas pelos métodos são todas voltadas para medir qualidade de recuperação. Embora sejam utilizados diferentes tamanhos de código *hash*, não é apresentada a relação dessa variação com os impactos da qualidade do código gerado. Além disso, nenhum dos métodos aplica alguma métrica para medir,

[3]O coeficiente de Jaccard mede a similaridade entre conjuntos de amostras e é definido como o tamanho da interseção dividido pelo tamanho da junta dos conjuntos de amostras.

Tabela 3.2. Resumo das estratégias de treino usadas pelo métodos de recuperação com múltiplos rótulos

Método	Arquitetura	Treino	Quantização vinculada ao treino	Código fonte disponível
DSRH [Zhao et al., 2015]	AlexNet	Listwise	-	-
IAH [Lai et al., 2016]	GoogLeNet	Tripletwise	-	-
TDBE [Zhuang et al., 2016]	VGG-16	Tripletwise	-	-
DBE [Liu et al., 2017]	ResNet50	Pointwise	-	-
DMLH [Zhong et al., 2017]	AlexNet	Pointwise	-	-
DMSH [Li et al., 2017]	CNN Genérica	Pairwise	sim	-
DMSSPH [Wu et al., 2017]	AlexNet	Pairwise	sim	-
HMLD [Wang et al., 2017a]	VGG	Tripletwise	-	-
MSDH [Lu et al., 2017]	CNN Genérica	Pairwise	sim	-
MTDH [Liang et al., 2017]	GoogLeNet	Pointwise	-	-
DCVH [Liu & Qi, 2018]	ResNet50	Pointwise	sim	-
DUAH [Wu et al., 2018]	ALexNet	Pairwise	-	-
DSOH [Song & Tan, 2018]	VGG e ResNet	Tripletwise	sim	-
ISDH [Zhang et al., 2018]	ALexNet	Pairwise	sim	sim
OLAH [Huang et al., 2018]	GoogLeNet	Listwise	-	-
OSDH [Chen et al., 2018]	AlexNet	Listwise	sim	-
RCDH [Ma et al., 2018]	VGG	Listwise	sim	-
IDHN [Zhang et al., 2019]	ALexNet	Pairwise	sim	sim
MPSH [esta tese]	**VAE**	**Pairwise**	**sim**	**a publicar**
MTSH [esta tese]	**VAE**	**Tripletwise**	**sim**	**a publicar**

por exemplo, as propriedades de *hashing* como independência e balanceamento de bits ou mesmo o consumo de memória e custo computacional. Não há também uma relação dessas propriedades com a qualidade de recuperação alcançada. Nesta tese, avaliamos essas propriedades.

Por fim, na Tabela 3.2, apresentamos um resumo comparativo das principais estratégias utilizadas pelos métodos, listando suas escolhas de arquitetura, forma de treino e processo de quantização diretamente vinculada ao treino ou separada do treino. Além disso, incluímos a informação se existe implementação disponível dos modelos.

Como mencionado na Seção 2.2.3, as arquiteturas de rede neural mais comumente utilizadas são: AlexNet, ResNet, GoogLeNet e VGG, todas inicializadas com pesos das redes pré-treinadas na ImageNet. No entanto, alguns métodos como o MSDH [Lu et al., 2017] e DMSH [Li et al., 2017] não especificam a arquitetura utilizada. Embora a qualidade da extração de características das imagens pareça impactar o desempenho dos modelos de recuperação, pois reflete a capacidade de representação do código *hash*, esses trabalhos não justificam a escolha das arquiteturas utilizadas. Diferentemente desses métodos, os modelos MPSH e MTSH, propostos nesta pesquisa (cf. Capítulos 4 e 5), usam uma rede autocodificadora variacional com distribuição de Bernoulli, para

gerar diretamente os códigos binários.

Note que embora os modelos DMSH, DMSSPH, MSDH, DCVH, DSOH, ISDH, OSDH, RCDH e IDHN incluam em seu aprendizado o processo de quantização, as estratégias utilizadas por eles são diferentes dos modelos MPSH e MTSH, propostos nesta tese. Como o autocodificador variacional é um modelo gerador, é possível garantir que a representação gerada é binária, pois modelamos cada bit da representação como seguindo uma distribuição de Bernoulli. Ao contrário, nos demais métodos, é adotada alguma componente na função de custo que induz uma aproximação de uma representação binária. Isto não impede, contudo, que os modelos usem diferentes números reais próximos de 0 (ou 1) para representar diferentes conceitos, ao longo da aprendizagem, que podem se perder quando a binarização final for efetivamente realizada.

Finalmente, a maioria dos métodos adotam estratégias de treino para aprender *ranking* (*pairwise*, *tripletwise* e *listwise*), visto que são mais efetivas para a tarefa de busca. Nesta pesquisa, exploramos duas destas estratégias de *ranking*, a *pairwise* e a *tripletwise*, com o intuito de verificar qual a mais vantajosa. Nós evitamos a estratégia *listwise*, dado o seu maior custo de aprendizagem.

Em nossos experimentos (cf. Capítulo 6) vamos usar o método IDHN como *baseline*, já que este representa um recente método baseado em *ranking* com resultados estado-da-arte, com quantização vinculada ao treino e implementação disponível. Para melhor compreender as diferenças entre abordagens *pairwise* e *triplewise*, nós iremos fazer comparações diretas entre nossos métodos, MPSH e MTSH.

3.6 Considerações Finais

Neste capítulo, foram apresentados os trabalhos propostos na literatura para solucionar o problema de recuperação de imagens com múltiplos rótulos, além do PSH, que também usa um auto-codificador variacional com distribuição discreta, embora para uma tarefa de busca de textos de rótulo único. Estes trabalhos foram organizados de acordo com a forma como as imagens de entrada são tratadas pelos modelos (pontos, pares de pontos, trios de pontos ou lista de pontos), incluindo: *pointwise*, *pairwise*, *tripletwise* e *listwise*. A partir dos modelos apresentados pelo métodos, podemos observar que, em geral, as abordagens estão mais preocupadas em melhorar a qualidade de busca, explorando pouco questões de eficiência da busca, como uso do espaço de endereçamento que pode ser melhorado explorando as propriedades de balanceamento e independência dos bits dos códigos binários. Ao fim, comparamos os métodos da literatura com os que propomos nesta tese, criados para explorar lacunas observadas

nas pesquisas anteriores.

Vimos ainda, no trabalho proposto Lu et al. [2017], que há algumas propriedades dos códigos *hash* que podem ser úteis em termos de eficiência de processamento e uso de memória, como a independência e balanceamento dos bits nos códigos binários gerados que tem impacto direto sobre o uso efetivo do espaço de representação. Estas propriedades, contudo, como vimos ao longo deste capítulo, são pouco exploradas pelos métodos ao gerar códigos para imagens com múltiplos rótulos. Nesta tese, exploramos e avaliamos essas propriedades nos métodos que propomos, que com intuito verificar seu impacto na qualidade e eficiência dos códigos gerados.

A seguir, nos Capítulo 4 e 5, são apresentados os modelos propostos para solucionar o problema de recuperação de imagem com múltiplos rótulos baseada em *hashing* profundo descrito nesta pesquisa.

Capítulo 4

Hashing Profundo usando Rede Pairwise Supervisonada Multilabel

Neste capítulo, apresentamos um estudo inicial, que consiste em adaptar o modelo PSH (apresentado na Seção 3.3.7) para o contexto de recuperação de imagens com múltiplos rótulos. A escolha desse modelo foi motivada principalmente pelo uso de um VAE com variáveis latentes de Bernoulli como códigos *hash*, eliminando a necessidade da etapa de binarização. Em sua forma básica, os VAEs assumem que as variáveis latentes são distribuídas de acordo com uma distribuição normal multivariada. Assim, para obter os códigos *hash* correspondentes às imagens, seria necessário binarizar as representações latentes contínuas. Como consequência, as informações contidas nestas representações poderiam ser perdidas durante o etapa de binarização. Ao contrário, um VAE com variáveis latentes de Bernoulli modela naturalmente o processo de quantização dos códigos binários. Além disso, treinamos o modelo para aprender os múltiplos rótulos das imagens.

4.1 Geração de Hashing usando VAE com Bernoulli

.

A ideia inicial de construir um VAE com variáveis latentes de Bernoulli foi proposta por Dadaneh et al. [2020], para a tarefa de busca textual. Esse modelo é apresentado a seguir para o contexto de busca de imagem.

Para entender como esse modelo funciona, considere x uma imagem de entrada e z seu código *hash* correspondente. Seja $x \in \mathbb{R}^n$ uma matriz de n características da imagem de entrada. Considerando a estrutura de um VAE, o decodificador $p_\theta(x|z)$ reconstrói a imagem de entrada a partir do código *hash* binário, enquanto o codifica-

dor $q_\phi(z|x)$ infere o código z a partir da imagem de entrada x. Considere ainda os parâmetros do modelo $\{\theta, \phi\}$ como sendo pesos da rede neural aplicados ao codificador e decodificador.

4.1.1 Estrutura do Codificador

O codificador é criado a partir de uma inferência amortizada de códigos *hash* para as imagens, construindo uma rede de inferência como $f_\phi(x)$ para aproximar a verdadeira distribuição a posteriori $p_\theta(z|x)$ por $q_\phi(z|x)$. A distribuição $q_\phi(z|x)$ aproximada para o código latente K-dimensional $z \in \{0,1\}^K$ é expressa como:

$$q_\phi(z|x) = \prod_{k=1}^{K} Bernoulli(z_k; \sigma(f_\phi(x)_k)) \tag{4.1}$$

onde $\sigma(\cdot)$ é uma função sigmóide e $f_\phi(x)_k$ é o k-ésimo elemento da saída da rede neural codificadora. Na fase de treinamento, os códigos latentes são amostrados usando as distribuições de Bernoulli em (4.1) e subsequentemente dados como entrada para a rede decodificadora. Por fim, são adicionadas probabilidades a priori de Bernoulli aos componentes dos códigos latentes, como segue:

$$p(z) = \prod_{k=1}^{K} Bernoulli(z_k; \gamma_k), \tag{4.2}$$

onde $\gamma_k \in$ [0,1]. Note que as variáveis latentes distribuídas de Bernoulli eliminam a necessidade de uma etapa de binarização separada; e, portanto, são mais eficientes para capturar a semântica das imagens de entrada.

4.1.2 Estrutura do Decodificador

Para a construção do decodificador, o modelo PSH considera uma função *softmax* como função de decodificação. Considere p_i o ith *pixel* dentro da imagem. A rede decodificadora compreende uma transformação linear do código *hash* binário latente z, seguido por uma função *softmax* que gera a probabilidade do *pixel* na reconstrução da imagem decodificada a partir de z, como mostrado a seguir:

$$p_\theta(p_i|z) = \frac{exp(z^T W + b_i)}{\sum_{j=1}^{K} exp(z_j^T W + b_j)} \tag{4.3}$$

onde W é a matriz de peso da rede decodificadora e b é o vetor de viés. Dessa forma, os parâmetros do decodificador a serem aprendidos são $\theta = \{W, b\}$.

Dadas as probabilidades individuais dos *pixels* em (4.3), a probabilidade da imagem pode ser calculada como:

$$\log p_\theta(x|z) = \sum_i p_\theta(p_i|z) \tag{4.4}$$

4.1.3 Inferência Variacional

Para estimar os parâmetros do codificador e do decodificador, o modelo utiliza o algoritmo de maximização do limite inferior de evidência (ELBO) com regularização ponderada de Kullback-Leibler (KL) [Alemi et al., 2018], conforme dado pela Equação 4.5.

$$\begin{aligned} \mathcal{L}(\theta, \phi) &:= \mathbb{E}_{\mathcal{PD}(x)}[\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z) - KL(q_\phi(z|x)||p(z))] \\ &\leq \mathbb{E}_{\mathcal{PD}(x)}[\log p_\theta(x)], \end{aligned} \tag{4.5}$$

onde KL é a divergência Kullback-Leibler e $_{\mathcal{PD}}(x)$ é a distribuição empírica da entrada. Considere que as probabilidades a priori e posteriori seguem distribuições de Bernoulli.

Na prática, para extrair representações latentes úteis e evitar o colapso de variáveis latentes, é empregada uma modificação de ELBO com o termo KL ponderado por um escalar λ:

$$\mathcal{L}_\lambda(\theta, \phi) := \mathbb{E}_{\mathcal{PD}(x)}[\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z) - \lambda KL(q_\phi(z|x)||p(z))] \tag{4.6}$$

4.2 Rede Pairwise Supervisionada Multilabel

O modelo PSH (descrito na Seção 3.3.7) foi proposto para busca textual. Além disso, ele não considera níveis de relevância entre os documentos. A seguir, adaptamos esse modelo para o contexto de imagens e inserimos uma componente para aprender os múltiplos rótulos das imagens. Para as redes codificadoras e decodificadoras, usamos uma Rede Neural Convolutiva para melhor extração das características das imagens. Chamamos esse modelo de *Multilabel Pairwise Supervised Hashing* (MPSH).

A Figura 4.1 ilustra uma visão geral da arquitetura do MPSH. O modelo é formado por duas redes autocodificadoras variacionais, formando uma arquitetura *pairwise*. As imagens x_1 e x_2 passam cada uma pela rede codificadora $q_\phi(z|x)$ para gerar códigos *hash* latentes z_1 e z_2, respectivamente. Cada código *hash* passa então por redes decodificadoras ($p_\phi(z|x)$) e classificadoras (f_η) para reconstruir a imagem

de entrada e prever suas classes, respectivamente. Note, que os rótulos das imagens (y_1, y_12) são usados apenas pelas redes classificadoras para comparar com os rótulos previstos no processo de aprendizagem.

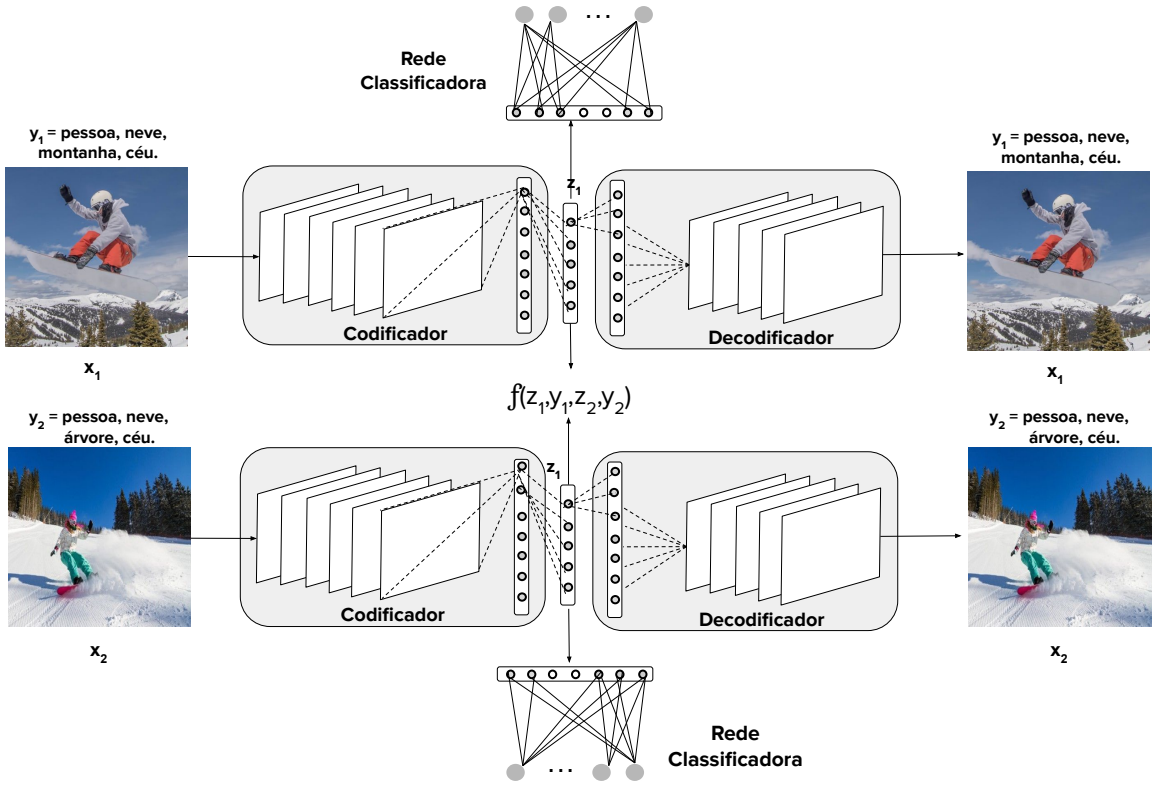


Figura 4.1. Arquitetura do modelo MPSH.

Para entender a construção do modelo, considere y o rótulo da imagem de entrada x. Dada uma rede neural f_η parametrizada por η, que recebe como entrada o código *hash* latente z e prevê o rótulo da imagem, a função de perda do modelo a ser minimizada pode ser expressa como:

$$-\mathcal{L}(\theta, \phi) + \alpha E_{q_\phi(z|x)}[\mathcal{L}^{'}(y; f_\eta(z))] \tag{4.7}$$

onde $\alpha > 0$ é um hiperparâmetro e $\mathcal{L}^{'}(y; f_\eta(z))$ é uma função de perda de entropia cruzada para classificação das classes. Note que $f_\eta(z)$ fornece probabilidades independentes para cada rótulo de forma que $\mathcal{L}^{'}(y; f_\eta(z))$ estima o custo considerando os múltiplos rótulos.

4.2.1 Múltiplos Rótulos

Para melhorar a representação do código *hash*, o modelo inclui uma estrutura de treinamento de *hashing* supervisionado par a par. O principal objetivo do MPSH é minimizar

a distância entre códigos latentes das imagens semelhantes e simultaneamente maximizar a distância entre códigos latentes de imagens que não compartilham nenhuma classe.

Dessa forma, vamos considerar $(x^{(1)}, y^{(1)})$ e $(x^{(2)}, y^{(2)})$ duas imagens amostradas aleatoriamente com seus respectivos códigos latentes $z^{(1)}$ e $z^{(2)}$. Para explorar o grau de similaridade entre as imagens, adotamos a mesma estratégia de Wang et al. [2017a], ou seja, definimos a proporção dos rótulos comuns às duas imagens em relação a todos os rótulos presentes nos pares de imagens, como descrito a seguir:

$$s_{(y^{(1)},y^{(2)})} = \begin{cases} 2 \times \frac{|y^{(1)} \cap y^{(2)}|}{|y^{(1)} \cup y^{(2)}|} - 1, \ \textit{caso compartilhem rótulos} \\ -1, \ \textit{caso contrário} \end{cases} \tag{4.8}$$

onde $y^{(1)}$ e $y^{(2)}$ são os rótulos reais das imagens $x^{(1)}$ e $x^{(2)}$, respectivamente. Assim, o MPSH calcula da seguinte forma a função de perda da distância para os pares de imagens:

$$\mathcal{L}^{''}(z^{(1)}, z^{(2)}) = s_{(y^{(1)},y^{(2)})} d(z^{(1)}, z^{(2)}) \tag{4.9}$$

onde $d(\cdot, \cdot)$ é uma métrica de distância. A Tabela 4.1 ilustra o efeito da Equação 4.9. Como podemos ver, supondo uma mesma distância, o custo é maior na medida em que os vetores compartilham mais rótulos em comum. Se o número de rótulos em comum é o mesmo que o de rótulos idênticos, o custo é zero. Se a maioria dos rótulos é distinto, os custos são negativos.

$y^{(1)}$	$y^{(2)}$	$s_{(y^{(1)},y^{(2)})}$	$d(z^{(1)}, z^{(2)})$	$\mathcal{L}^{''}(z^{(1)}, z^{(2)})$
{1,2,3}	{1,2,3}	1,00	1,00	1,00
{1,2,3}	{1,2}	0,33	1,00	0,33
{1,2}	{1}	0,00	1,00	0,00
{1,2,3}	{1}	-0,33	1,00	-0,33
{1,2,3}	{5}	-1,00	1,00	-1,00

Tabela 4.1. Custos associados com diferentes pares de exemplos, considerando uma mesma distância.

A função de perda final para o MPSH é, portanto, a composição da componente variacional, de similaridade de classes e similaridade de códigos latentes:

$$\begin{aligned} \mathcal{L}_{MPSH}(\theta, \phi) = & - [\mathcal{L}_{\lambda}^{(1)}(\theta, \phi) + \mathcal{L}_{\lambda}^{(2)}(\theta, \phi)] \\ & + \alpha[\mathcal{L}^{'}(y^{(1)}; f_{\eta}(z^{(1)})) + \mathcal{L}^{'}(y^{(2)}; f_{\eta}(z^{(2)}))] \\ & + \beta E_{\Pi_{t=1}^{2} q_{\phi}(z^{(t)}|x^{(t)})}[\mathcal{L}^{''}(z^{(1)}, z^{(2)})] \end{aligned} \tag{4.10}$$

onde α e β são hiperparâmetros usados para controlar a influência das componentes associadas com a similaridade entre as classes e entre os códigos latentes das imagens, respectivamente.

4.2.2 Estimador de Gradiente

Otimizar a função 4.10 é difícil, uma vez que o algoritmo de retropropagação não pode ser aplicado às camadas de amostragem discretas de Bernoulli. Dessa forma, Dadaneh et al. [2020] combina duas estratégias para resolver esse problema. Primeiro aplica o estimador de gradiente *straight-through*, proposto por Bengio et al. [2013], para variáveis latentes discretas, e então aplica o algoritmo ARM (*Augment-REINFORCE-Merge*) [Yin & Zhou, 2019] para fazer a retropropagação do gradiente através das camadas discretas.

O estimador *straight-through* retropropaga através de uma unidade de amostragem discreta, como se fosse a função identidade. Assim, dada a imagem de entrada x, primeiro a representação binária latente é amostrada como:

$$z \sim Bernoulli(\sigma(f_\phi(x))),$$

e então a entrada para o decodificador é calculada como:

$$z' = Stop\ Gradient(z - \sigma(f_\phi(x))) + \sigma(f_\phi(x)),$$

onde os termos dentro de operador *Stop Gradient* são consideradas as constantes no passo de retropropagação.

Para lidar com o viés de gradiente do estimador *straight-through*, utilizou-se algoritmo ARM, para estimar os gradientes de forma imparcial por meio de unidades binárias estocásticas. Esse estimador apresenta baixa variância e baixa complexidade computacional. Além disso, diferente do *straight-through*, ele é aplicável a funções de perda não diferenciáveis, adaptado para treinar VAEs discretos com a função de perda do MPSH. Uma descrição completa desse algoritmo é apresentada em [Yin & Zhou, 2019].

4.3 Considerações Finais

Neste capítulo, apresentamos o MPSH, um modelo supervisionado que usa uma rede autocodificadora variacional com distribuição de Bernoulli, para aprender códigos *hash* de imagens com múltiplos rótulos. A principal vantagem desse modelo consiste em usar

uma distribuição de Bernoulli, que elimina a necessidade da etapa de binarização, uma das propriedades desejáveis de códigos *hash*. Além disso, a estrutura *pairwise* aprende de forma eficiente a tarefa de aprendizagem supervisionada de *ranking*, efetivo para busca de imagens, aplicação de interesse desta tese.

A seguir, no Capítulo 5, apresentamos uma evolução do MPSH, usando uma estrutura *tripletwise*, que modela melhor o problema de recuperação de imagem com múltiplos rótulos baseada em *hashing* profundo.

Capítulo 5

Hashing Profundo usando Rede Tripletwise Supervisonada Multilabel

Neste capítulo, propomos um modelo para recuperação de imagem com múltiplos rótulos que utiliza uma arquitetura *tripletwise*. A proposta desse modelo foi motivada pela percepção de que a tarefa de busca de imagens deve ser melhor aprendida a partir de modelos que são treinados para a tarefa de *ranking*, onde temos uma imagem de referência para o que se deseja buscar. Dessa forma, acreditamos que um modelo com as características do MPSH combinado a uma arquitetura própria para o aprendizado específico de *ranking* baseado em trios, tem potencial para melhorar a qualidade e eficiência de códigos *hash*.

5.1 Rede Tripletwise Supervisionada Multilabel

Conforme visto no Capítulo 3 (Seção 3.4), o método *tripletwise* é um dos modelos de aprendizado de *ranking* utilizados para recuperação de imagens usando *hashing* profundo. Estes modelos, aprendem *embeddings* de imagens explorando a semelhança relativa das amostras de imagens. Dessa forma, o objetivo deste capítulo é explorar a estratégia abordada pelo MPSH, mas agora usando uma arquitetura *tripletwise* que, de agora em diante, denominamos *Multilabel Tripletwise Supervised Hashing* (MTSH).

Na Figura 5.1, apresentamos uma visão geral dessa arquitetura. O modelo é formado por três redes autocodificadoras variacionais, formando uma arquitetura *tripletwise*. As imagens x^a, x^p e x^n passam cada uma pela rede codificadora ($q_\phi(z|x)$) para

gerar códigos *hash* latentes z^a, z^p e z^n, respectivamente. Cada código *hash* passa então por redes decodificadoras ($p_\phi(z|x)$), para reconstruir a imagem de entrada, e classificadoras (f_η), para prever suas classes. O objetivo final da rede, é aprender representações binárias latentes das imagens de forma que os códigos das imagens relevantes (z^p) sejam mais próximos dos códigos das imagens de consulta (z^a) do que dos códigos das imagens não relevantes (z^n). Nesse caso, o nível de relevância é determinado pelos rótulos compartilhados entre os pares de imagens (x^a, x^p) e (x^a, x^n).

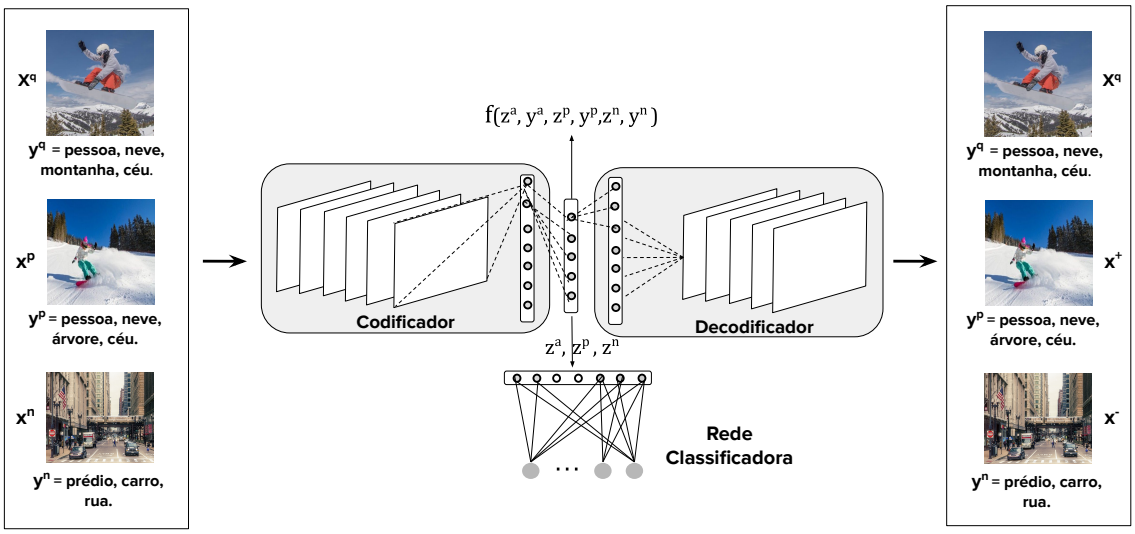


Figura 5.1. Arquitetura do modelo MTSH.

A seguir, exploramos uma abordagem de seleção de triplas para o contexto de múltiplos rótulos e ao final, apresentamos a função de perda para o modelo MTSH, considerando um VAE com distribuição de Bernoulli, conforme apresentado no capítulo anterior.

5.1.1 Função de Perda Tripla

Conforme introduzido pela primeira vez no trabalho de Schroff et al. [2015], onde foi proposto a FaceNet, uma função de perda tripla (*TripletLoss*) é uma função que treina uma rede neural profunda para minimizar a distância entre características de imagens de uma mesma classe enquanto maximiza a distância entre as características de diferentes classes. Para fazer isso, uma imagem de referência, tida como âncora (e que pode ser pensada como uma representação de uma consulta), é escolhida junto com uma amostra negativa e uma positiva.

A Figura 5.2 ilustra essa ideia. Nela, a imagem âncora (em azul) e comparada com uma imagem "relevante" (positiva, em verde) e outra "irrelevante" (negativa, em ver-

melho). Se em certo momento do treino, a imagem negativa é representada como mais próxima da âncora, à medida que o treino avança, esperamos que o modelo aprenda a representá-la de forma que ela fique mais distante da âncora que a imagem relevante.

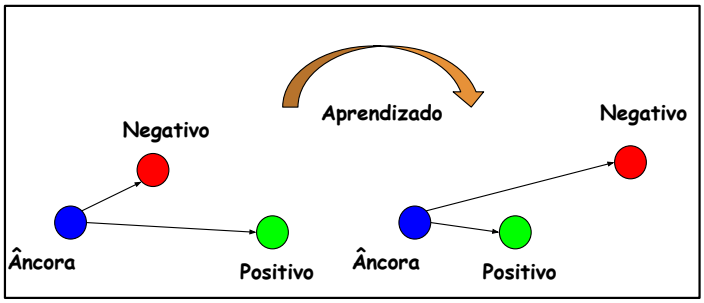


Figura 5.2. Uma função de perda tripla minimiza a distância entre a imagem âncora e a imagem positiva ao mesmo tempo que maximiza a distância em relação a imagem negativa.

Mais formalmente, conforme Schroff et al. [2015], a *TripletLoss* pode ser definida como uma função de perda que treina uma rede neural para criar *embeddings* similares para imagens similares enquanto maximiza a distância entre *embeddings* de classes distintas. Como visto anteriormente, ela opera em triplas de pontos, representando uma âncora, um exemplo positivo e um outro negativo. Esta função é descrita pela Equação 5.1.

$$\mathcal{L}(\theta) = \sum_{x_a, x_p, x_n | y_a = y_p \neq y_n} [m + D(f_\theta(x_a), f_\theta(x_p)) - D(f_\theta(x_a), f_\theta(x_n))]_+ \tag{5.1}$$

onde D é uma função de distância (por exemplo, a distância Euclidiana) tomada entre os *embeddings* obtidos pela rede neural f com pesos θ, x_a representa uma âncora amostrada do *batch*, x_p é uma amostra positiva e x_n, uma negativa. O sinal de "+" em $\mathcal{L}$ indica que esta é uma função não negativa, ou seja, $\mathcal{L}(\theta; X)$ é o máximo entre o valor estimado e 0. A constante m é uma margem usada para especificar quando uma tripla se tornou tão fácil que o modelo não deveria mais ajustar pesos para refiná-la.

5.1.2 Seleção de Triplas

Em modelos *tripletwise* o número de triplas cresce cubicamente com o número de instâncias. Como resultado, para uma grande base de dados o treino se torna praticamente inviável, não sendo possível a mesma estratégia de seleção aleatória que usamos em modelos *pairwise*. Além disso, a rede neural f_θ aprende muito rápido a mapear corretamente triplas triviais, fazendo com que grande parte das triplas disponíveis sejam de pouca utilidade, resultando em processos de treino extremamente lentos.

Para lidar com este problema é necessário definir como selecionar triplas de forma a tornar viável o treino. Uma primeira estratégia de seleção é minerar os exemplos mais difíceis. A intuição é que, por exemplo, após aprender que leões (âncora) são diferentes de elefantes, mostrar fotos de novos elefantes (exemplos negativos "fáceis") vai ajudar menos no treinamento do modelo que fotos de tigres, que não são leões mas se parecem muito mais com eles (exemplos negativos "difíceis"). Assim, as amostras mais difíceis são aqueles mais similares à âncora (mínima distância) que pertencem a classes diferentes. Esta ideia pode ser traduzida na Equação 5.2, que introduz operadores *min* e *max* na Equação 5.1:

$$\begin{aligned}\mathcal{L}(\theta; X) = \sum_{i=1}^{P}\sum_{a=1}^{K}[m + \max_{p=1..K} D(f_\theta(x_a^i), f_\theta(x_p^i)) \\ - \min_{\substack{j=1..P,\\ n=1..K,\\ j\neq i}} D(f_\theta(x_a^i), f_\theta(x_n^j))]_+\end{aligned} \tag{5.2}$$

onde X é um *batch* aleatório composto por P classes e K imagens por classe, de forma que $|X| = PK$. Para cada âncora x_a amostrada no *batch*, podemos selecionar as positivas mais difíceis ($\max D(f_\theta(x_a^i), f_\theta(x_p^i))$) e as negativas mais difíceis ($\min D(f_\theta(x_a^i), f_\theta(x_n^j))$) dentro do *batch*. Ao fazer isso, uma estratégia de treino "on line", evitamos o custo de pré-computar distâncias que não serão usadas e nos concentramos em exemplos que podem colaborar mais para a aprendizagem.

Um problema desta estratégia é que ao selecionar apenas as triplas mais difíceis, acabamos por escolher casos anômalos mais frequentemente que o normal, dificultando o aprendizado dos casos mais normais por f_θ. Por exemplo, imagine que uma imagem particularmente difícil para caracterizar o que *não* é um leão seja uma criança usando uma máscara de leão. Este é tipicamente um caso anômalo e tomá-lo como exemplo de treino preferencial pode dificultar a tarefa do modelo em aprender aspectos que melhor caracterizam um leão em contraposição a outras coisas. Para contornar este novo problema, em lugar de usar a função de distância para casos difíceis ("hard"), iremos usar uma função para os casos semi-difíceis (*semi-hard*), ou seja, *a mínima distância negativa que é pelo menos maior que a distância positiva mais a margem*. Se esta distância *semi-hard* não existe, usamos a maior distância negativa, como antes. Em outras palavras, queremos agora encontrar a menor distância negativa que satisfaz a restrição $D(f_\theta(x_a^i), f_\theta(x_p^i)) + m < D(f_\theta(x_a^i), f_\theta(x_n^i))$.

Em suma, o tipo de seleção para trios que vamos usar é a estratégia *Online Semi-Hard*, onde os trios de imagens são escolhidos de forma que a imagem negativa

esteja mais longe da âncora do que a positiva, mas ainda produza uma perda positiva. Para encontrar esses trios de imagens de forma eficiente, vamos utilizar aprendizado online e treinar apenas com os exemplos *Semi-Hard* encontrados em cada *batch* na fase de treino. Assim, as triplas serão selecionadas em tempo de treino, de acordo com as características aprendidas das imagens a cada iteração. Nos resta agora apenas adaptar esta estratégia para lidar com casos multi-rótulo.

5.1.3 Implementação

Nesta seção, descrevemos uma eficiente versão tensorial da função de perda *triplet semi-hard* adaptada para múltiplos rótulos. A notação usada neste algoritmo é descrita na Tabela 5.1. A ideia geral deste algoritmo é calcular a distância sobre todas as triplas que podem ser observadas no *batch* de treino fornecido para, então, estimar a perda final considerando prioritariamente os casos mais promissores para o treino (os *semi-hard*). Para facilitar a compreensão do algoritmo, vamos ilustrá-lo com um exemplo de execução à medida que ele for explicado.

A função MultilabelTripletSemihardLoss (cf. Algoritmo 1) tem como parâmetros de entrada (i) o *batch* de multi-rótulos $\mathbf{y}$ definido como uma matriz $\{0,1\}^{B,L}$ onde B é o tamanho do *batch* e L o número de classes possíveis[1]; (ii) o *batch* de *embeddings* $f_\theta(\mathbf{x}) \in \mathbb{R}^{B,Z}$, onde Z é o tamanho do *embedding*, $f_\theta(\mathbf{x})$ representa o *embedding* como a representação de uma entrada $\mathbf{x}$ pela rede neural f parametrizada em θ; (iii) a margem $m \in \mathbb{R}$; e (iv) a função de distância $D\colon \mathbb{R}^Z \times \mathbb{R}^Z \to \mathbb{R}$ que, dados dois *embeddings*, retorna a distância entre eles, de forma que quanto menor a distância, maior é a similaridade.

Para fins de ilustração, tomemos um *batch* de $B = 6$ *embeddings* de tamanho $Z = 2$ dado por $f_\theta(\mathbf{x})$:

$$f_\theta(\mathbf{x}) = \begin{bmatrix} -0.27348092 & 0.12087647 \\ -0.1395917 & 0.41258650 \\ -0.60139984 & -0.12218875 \\ -0.57803166 & 0.25627634 \\ -0.50771815 & 0.09456696 \\ -0.13770121 & 0.50001017 \end{bmatrix}$$

[1]Note que neste capítulo, matrizes binárias com valores 0 e 1 podem ser interpretados como tendo valores lógicos F e V. Assim, estas matrizes podem ser manipuladas tanto por operadores lógicos quanto aritméticos. O contexto determina a interpretação a ser usada.

Algorithm 1 Função de perda *triplet semi-hard* adaptada para múltiplos rótulos

function MultilabelTripletSemihardLoss($\mathbf{y}, f_\theta(\mathbf{x}), m, D$)

▷ Obtém matriz de similaridade $\mathbf{s}$ onde y_i é o conjunto de rótulos em $\mathbf{y}_i$

Compute $\mathbf{s} \in [-1, 1]^{B,B}$ tal que $\mathbf{s}_{i,j} = \begin{cases} 2\frac{|y_i \cap y_j|}{|y_i \cup y_j|} - 1, \text{ se } |y_i \cup y_j| \neq 0 \\ -1, \text{ caso contrário} \end{cases}$

▷ Calcula matriz de adjacência binária A

$A \leftarrow (\mathbf{s} = -1)$

▷ Calcula a matriz de distância $\mathcal{D}$

Compute $\mathcal{D}$ tal que $\mathcal{D}_{i,j} \leftarrow \mathbf{s}_{i,j} D(f_\theta(\mathbf{x}_i), f_\theta(\mathbf{x}_j)), \forall i, j \in [0, B)$

▷ Obtem as menores D_{an} onde $D_{an} > D_{ap}$

$\mathcal{D}_t \leftarrow tile(\mathcal{D}, [B, 1])$

mask $\leftarrow tile(\neg A, [B, 1]) \wedge (\mathcal{D}_t > (\mathcal{D}^\top)^{[\![-1,1]\!]})$

$D_{an}^{min} \leftarrow (masked(min, \mathcal{D}_t, mask)^{[\![B,B]\!]})^\top$

▷ Obtem as maiores D_{an}

$D_{an}^{max} \leftarrow tile(masked(max, \mathcal{D}, \neg A), [1, B])$

▷ Obtem as distâncias dos negativos semi-hard

$mask_f \leftarrow ((sum(mask, 1) > 0)^{[\![B,B]\!]})^\top$

$D_{SH} \leftarrow mask_f \odot D_{an}^{min} + \neg mask_f \odot D_{an}^{max}$

▷ Calcula a triplet loss para o *batch* apenas com as triplas selecionadas

$M_{loss} \leftarrow m + \mathcal{D} - D_{SH}$

$A_z \leftarrow A$ com diagonal principal zero

$loss \leftarrow \frac{sum(max(M_{loss} \odot A_z, 0))}{sum(A_z)}$

▷ Obtem triplets usadas, caso necessário

$I_{an}^{min} \leftarrow (masked(argmin, \mathcal{D}_t, mask)^{[\![B,B]\!]})^\top$

$I_{an}^{max} \leftarrow tile(masked(argmax, \mathcal{D}, \neg A), [1, B])$

$indices \leftarrow A_z \odot (mask_f \odot I_{an}^{min} + \neg mask_f \odot I_{an}^{max})$

return loss, indices

end function

Tabela 5.1. Notação

$masked(op, t, m)$	Função que retorna o resultado da operação op sobre os elementos de t para os quais a máscara m é verdadeira. Exemplo: $masked(min, [0, 1, 2], [F, V, V]) = 1$.
$sum(t, e)$	Função que retorna a soma de todos os elementos do tensor t. Se o eixo e é indicado, a soma é nos valores deste eixo. Exemplo: se $t = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$, então $sum(t) = 10$ e $sum(t, 0) = \begin{pmatrix} 4 & 6 \end{pmatrix}$.
$tile(t, m)$	Função que retorna um tensor formado por tantas cópias de t quanto definidas pelos multiplicadores m nos eixos indicados em m. Exemplo: $tile(\begin{pmatrix} 0 & 1 \end{pmatrix}, [2, 3]) = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$.
$t^{[\![n,m]\!]}$	Reformata o tensor t para n linhas e m colunas. O valor -1 indica dimensão indefinida. Exemplo: $\begin{pmatrix} 0 & 1 \end{pmatrix}^{[\![-1,1]\!]} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$.
$t_1 \wedge t_2$	Conjunção entre t_1 e t_2 (multiplicação em interpretação numérica).
$\neg t$	Negação lógica de t (inversão entre 0 e 1 em interpretação numérica).
$t_1 \odot t_2$	Produto de Hadamard entre t_1 e t_2 (multiplicação elemento-a-elemento).
$t_1 \cdot t_2$	Produto interno entre t_1 e t_2.
$t^\top$	Transposta de t.

Suponha que estes *embeddings* estão associados aos multi-rótulos (até $L = 8$ rótulos possíveis, de 0 a 7) dados por $\mathbf{y}$:

$$\mathbf{y} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Neste caso, por exemplo, o primeiro *embedding* foi rotulado como 0 e 6. A Figura 5.3 apresenta estes *embeddings* e seus rótulos.

O cálculo da perda inicia com a obtenção da matriz de similaridade entre rótulos $\mathbf{s}$ (Algoritmo 1, linha 3) dada pela Equação 5.3, proposta por Wang et al. [2017a]:

$$\mathbf{s}_{i,j} = \begin{cases} 2\frac{|y_i \cap y_j|}{|y_i \cup y_j|} - 1, & \text{caso compartilhem algum rótulo} \\ -1, & \text{caso contrário} \end{cases} \tag{5.3}$$

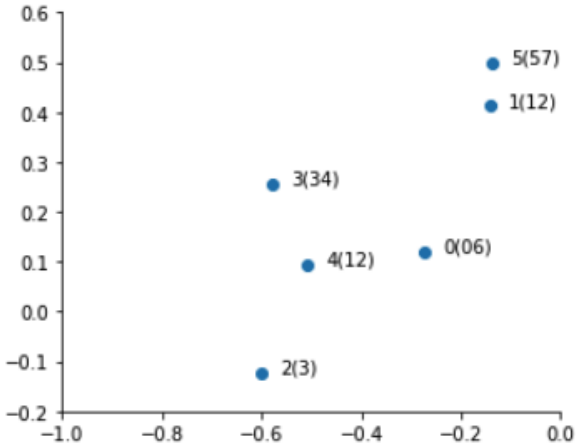


Figura 5.3. Conjuntos de *embeddings* de exemplo. As anotações ao lado de cada ponto indicam o *embedding* e as classes em que foram rotulados. Por exemplo, 0(06), indica *embedding* 0, rotulado como 0 e 6.

onde y_i e y_j representam os conjuntos de rótulos reais das instâncias x_i e x_j no *batch* X. Em seguida, linha 5, é obtida a matriz de adjacência A que indica se dois *embeddings* têm rótulos em comum. Ou seja, $A_{i,j}$ é 1 se $\mathbf{x}_i$ tem algum rótulo em comum com $\mathbf{x}_j$, de forma que basta um rótulo em comum entre imagens $\mathbf{x}_i$ e $\mathbf{x}_j$ para que uma seja vista como um caso positivo da outra. Para o nosso exemplo de execução, A é ilustrada na Figura 5.4.

	c0=[0 6]	c1=[1 2]	c2=[3]	c3=[3 4]	c4=[1 2]	c5=[5 7]
c0=[0 6]	1	0	0	0	0	0
c1=[1 2]	0	1	0	0	1	0
c2=[3]	0	0	1	1	0	0
c3=[3 4]	0	0	1	1	0	0
c4=[1 2]	0	1	0	0	1	0
c5=[5 7]	0	0	0	0	0	1

Figura 5.4. Matriz de adjacência A. Títulos nas linhas e colunas indicam o *embedding* e suas classes. Por exemplo, 0=[0 6], indica *embedding* 0, rotulado como 0 e 6. Como consideramos um cenário de múltiplos rótulos, dois *embeddings* são adjacentes se eles compartilham quaisquer rótulos.

Na linha 7, é calculada a matriz de distâncias $\mathcal{D} \in \mathbb{R}^{B,B}$, em que $\mathcal{D}_{i,j}$ representa a distância entre *embeddings* $f_\theta(\mathbf{x}_i)$ e $f_\theta(\mathbf{x}_j)$ ponderada pelo grau de similaridade $\mathbf{s}_{i,j}$ entre os múltiplos rótulos de $\mathbf{x}_i$ e $\mathbf{x}_j$ (cf. Equação 5.3). Supondo $\mathbf{s} = \mathbf{1}$, apenas para fins de ilustração, $\mathcal{D}$ seria dada pela Figura 5.5 para o nosso exemplo.

Dada a matriz de distâncias D e a matriz de adjacência A, é possível obter as menores distâncias D_{an}^{min} (D_{an}, tal que $D_{an} > D_{ap}$) e as maiores distâncias D_{an}^{max}, linhas 8-13 (cf. Figura 5.6a e Figura 5.6b). Usando a negação da matriz de adjacência, é criada uma máscara que indica onde há candidatos a negativos *semi-hard* (linha

	c0=[0 6]	c1=[1 2]	c2=[3]	c3=[3 4]	c4=[1 2]	c5=[5 7]
c0=[0 6]	0.000	0.321	0.408	0.333	0.236	0.403
c1=[1 2]	0.321	0.000	0.707	0.465	0.486	0.087
c2=[3]	0.408	0.707	0.000	0.379	0.236	0.776
c3=[3 4]	0.333	0.465	0.379	0.000	0.176	0.503
c4=[1 2]	0.236	0.486	0.236	0.176	0.000	0.549
c5=[5 7]	0.403	0.087	0.776	0.503	0.549	0.000

Figura 5.5. Matriz de Distâncias D. Títulos nas linhas e colunas indicam o vetor e suas classes. O gradiente de cores usado nesta e nas demais figuras varia de vermelho, para valores menores, até verde, para valores maiores.

	c0=[0 6]	c1=[1 2]	c2=[3]	c3=[3 4]	c4=[1 2]	c5=[5 7]
c0=[0 6]	0.236	0.333	0.000	0.403	0.321	0.408
c1=[1 2]	0.465	0.087	0.000	0.707	0.707	0.321
c2=[3]	0.707	0.776	0.236	0.408	0.408	0.000
c3=[3 4]	0.465	0.503	0.465	0.176	0.333	0.000
c4=[1 2]	0.236	0.549	0.549	0.236	0.176	0.000
c5=[5 7]	0.503	0.403	0.000	0.549	0.776	0.087

(a) D_{an}^{min}

	c0=[0 6]	c1=[1 2]	c2=[3]	c3=[3 4]	c4=[1 2]	c5=[5 7]
c0=[0 6]	0.408	0.408	0.408	0.408	0.408	0.408
c1=[1 2]	0.707	0.707	0.707	0.707	0.707	0.707
c2=[3]	0.776	0.776	0.776	0.776	0.776	0.776
c3=[3 4]	0.503	0.503	0.503	0.503	0.503	0.503
c4=[1 2]	0.549	0.549	0.549	0.549	0.549	0.549
c5=[5 7]	0.776	0.776	0.776	0.776	0.776	0.776

(b) D_{an}^{max}

	c0=[0 6]	c1=[1 2]	c2=[3]	c3=[3 4]	c4=[1 2]	c5=[5 7]
c0=[0 6]	1	1	0	1	1	1
c1=[1 2]	1	1	0	1	1	1
c2=[3]	1	1	1	1	1	0
c3=[3 4]	1	1	1	1	1	0
c4=[1 2]	1	1	1	1	1	0
c5=[5 7]	1	1	0	1	1	1

(c) $mask_f$

	c0=[0 6]	c1=[1 2]	c2=[3]	c3=[3 4]	c4=[1 2]	c5=[5 7]
c0=[0 6]	0.236	0.333	0.408	0.403	0.321	0.408
c1=[1 2]	0.465	0.087	0.707	0.707	0.707	0.321
c2=[3]	0.707	0.776	0.236	0.408	0.408	0.776
c3=[3 4]	0.465	0.503	0.465	0.176	0.333	0.503
c4=[1 2]	0.236	0.549	0.549	0.236	0.176	0.549
c5=[5 7]	0.503	0.403	0.776	0.549	0.776	0.087

(d) D_{SH}: $mask_f \odot D_{an}^{min} + \neg mask_f \odot D_{an}^{max}$

Figura 5.6. (a) As menores distâncias entre âncora e negativos, maiores que a maior distância entre o âncora e positivos; (b) As maiores distâncias entre âncora e negativos; (c) máscara que indica onde há distâncias negativas semi-hard; (d) Matriz de distâncias semi-hard já preenchida com maiores distâncias nos casos em que não há semi-hard.

15, cf. Figura 5.6c). Com a máscara e as matrizes D_{an}^{min} e D_{an}^{max}, são calculadas as distâncias dos negativos *semi-hard* (linhas 15-16, cf. Figura 5.6d).

Na linha 18, a matriz de perda é calculada considerando a margem m, a matriz de distância $\mathcal{D}$ e as distâncias *semi-hard* D_{SH}. Se considerarmos $m = 1$, para o nosso exemplo dado, a matriz de perda obtida é dada na Figura 5.7a. Em seguida, é calculada a máscara A_z que indica os casos positivos dos trios que serão selecionados (cf. Figura 5.7b). Embora isto não seja necessário para calcular a perda, com base nas máscaras criadas, é possível recuperar os índices das triplas selecionadas (linhas 22-24). Para o nosso exemplo ilustrativo (cf. Figura 5.7c), elas são (1, 4, 2), (2, 3, 0), (3, 2, 1) e (4, 1, 5), onde cada tripla corresponde a (âncora, positivo, negativo).

Finalmente, a Figura 5.8 ilustra graficamente as triplas usadas. Nesta figura,

	c0=[0 6]	c1=[1 2]	c2=[3]	c3=[3 4]	c4=[1 2]	c5=[5 7]
c0=[0 6]	0.764	0.988	1.000	0.931	0.915	0.995
c1=[1 2]	0.855	0.913	1.000	0.759	0.780	0.766
c2=[3]	0.702	0.931	0.764	0.971	0.828	1.000
c3=[3 4]	0.868	0.962	0.914	0.824	0.843	1.000
c4=[1 2]	1.000	0.938	0.687	0.941	0.824	1.000
c5=[5 7]	0.899	0.685	1.000	0.954	0.773	0.913

(a) *Matriz de perda*

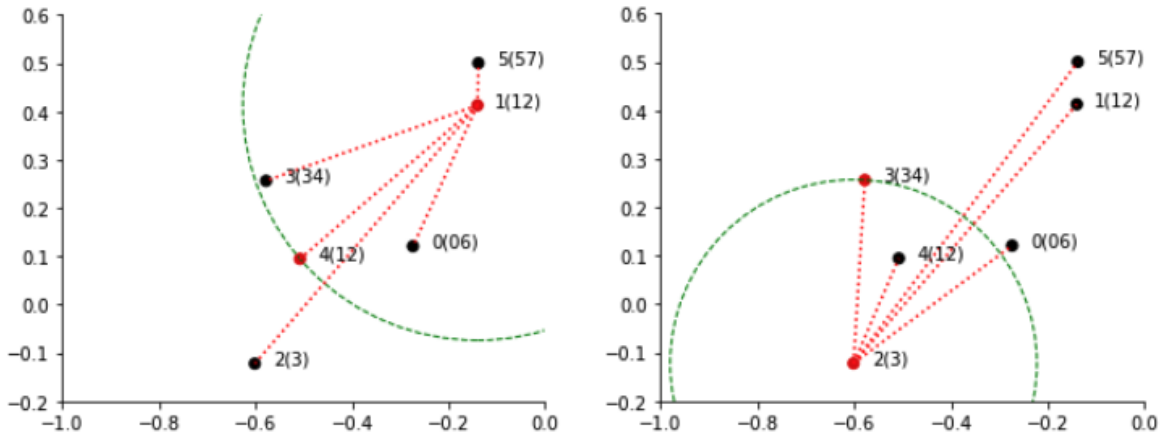
c0=[0 6]	c1=[1 2]	c2=[3]	c3=[3 4]	c4=[1 2]	c5=[5 7]
0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	1	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0

(b) *Máscara dos casos positivos*

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	2	0
2	0	0	0	0	0	0
3	0	0	1	0	0	0
4	0	5	0	0	0	0
5	0	0	0	0	0	0

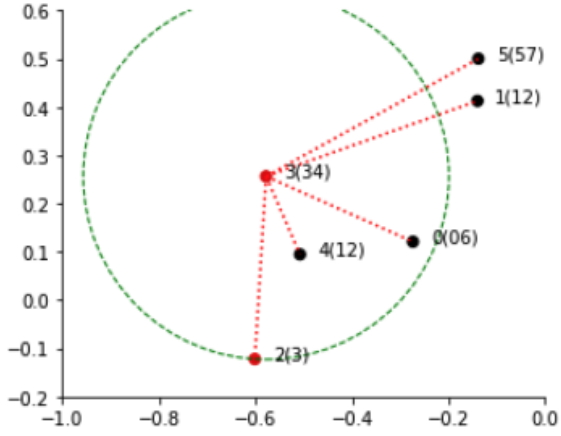
(c) *Indices*

Figura 5.7. (a) Matriz de perda para casos semi-hard; (b) máscara de casos positivos para construir trios para treino de casos semi-hard; (c) Índices dos casos negativos. Desta tabela é possível observar que as triplas a serem usadas no treino são (1, 4, 2), (2, 3, 0), (3, 2, 1) e (4, 1, 5).

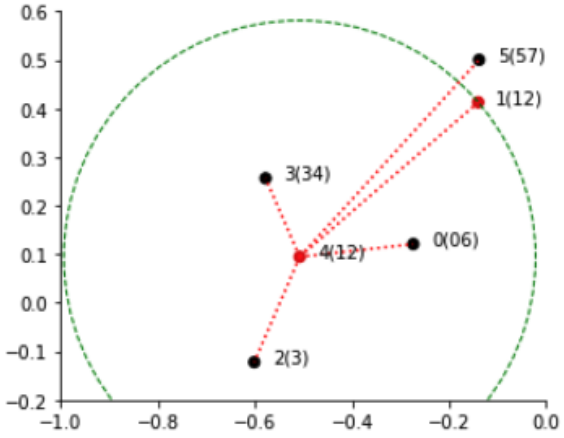


(a) *Tripla (1, 4, 2)*

(b) *Tripla (2, 3, 0)*



(c) *Tripla (3, 2, 1)*



(d) *Tripla (4, 1, 5)*

Figura 5.8. Triplas semi-hard selecionadas para cálculo da perda. Linhas pontilhadas indicam as distâncias entre o ponto âncora (em vermelho) e pontos positivos (em vermelho) e negativos (em preto). A linha tracejada verde indica o positivo mais distante.

as distâncias são dadas por linhas pontilhadas entre os pontos âncora (em vermelho) e os pontos positivos (também em vermelho) e os negativos *semi-hard* (em preto) correspondentes. A linha tracejada verde indica o positivo mais distante. Logo, o ponto mais próximo da circunferência verde é o ponto *semi-hard*. Neste exemplo, foram encontrados negativos *semi-hard* apenas para as âncoras 2 e 3. Vejamos o caso da âncora 2 (cf. Figura 5.8b). Neste caso, o *embedding* 0 é o *semi-hard* pois é o negativo mais próximo que está além do mais distante positivo, 3, mais a margem $m = 1$. Observe que os pontos 1 e 5 são os negativos mais fáceis (e, portanto, com menor potencial para auxiliar no aprendizado ao longo tempo) enquanto o ponto 4 é o mais difícil (e, portanto, o que tem maior potencial de ser uma anomalia). Considerando as quatro triplas usadas, a perda calculada para este *batch* é 0.9005449 (linha 20).

5.1.4 Função Objetivo MTSH

Dada a função de perda para o modelo *tripletwise*, baseada em uma abordagem de seleção de triplas *semi-hard* adaptada ao contexto de múltiplos rótulos, apresentamos agora a a função de perda final para o modelo MTSH, considerando um VAE com distribuição de Bernoulli. Esta função é uma composição das componentes variacional, de similaridade de classes e similaridade de códigos, descrita pela Equação 5.4.

$$\begin{aligned}\mathcal{L}_{MTSH}(\theta,\phi;X) = &-[\mathcal{L}_{\gamma}^{(a)}(\theta,\phi)+\mathcal{L}_{\gamma}^{(p)}(\theta,\phi)+\mathcal{L}_{\gamma}^{(n)}(\theta,\phi)] \\ &+\alpha[\mathcal{L}^{'}(y^{(a)};f_{\eta}(z^{(a)}))+\mathcal{L}^{'}(y^{(p)};f_{\eta}(z^{(p)}))+\mathcal{L}^{'}(y^{(n)};f_{\eta}(z^{(n)}))] \\ &+\beta\mathcal{L}(\theta;X,\mathbf{s})\end{aligned} \tag{5.4}$$

Nesta equação, $\mathcal{L}_{\gamma}$ corresponde à maximização do limite inferior de evidência (ELBO) com regularização ponderada de Kullback-Leibler (cf. Equação 4.5). Os sobrescritos (a), (p) e (n) indicam âncora, positivo e negativo. A componente $\mathcal{L}^{'}(y;f_{\eta}(z))$ captura a perda associada com a previsão das classes, calculada por meio da entropia cruzada e ponderada pelo hiperparâmetro α. Como descrito na Seção 4.2, a componente $\mathcal{L}^{'}(y;f_{\eta}(z))$ já considera os múltiplos rótulos. Finalmente, a componente relacionada com a *TripletLoss* é $\mathcal{L}(\theta;X)$, ponderada pelo hiperparâmetro β e calculada pela função MultilabelTripletSemihardLoss (cf. Algoritmo 1). Note que o cálculo das duas primeiras componentes da Equação 5.4 é feito apenas sobre as triplas selecionadas pela função MultilabelTripletSemihardLoss. Para este modelo, usou-se o mesmo estimador de gradiente que o MPSH.

5.2 Considerações Finais

Neste capítulo, apresentamos o MTSH, um modelo *tripletwise* supervisionado que usa uma rede autocodificadora variacional com distribuição de Bernoulli, para aprender códigos *hash* de imagens com múltiplos rótulos.

Este é o primeiro modelo *tripletwise* para recuperação de imagens com múltiplos rótulos que usa um uma rede autocodificadora variacional com distribuição de Bernoulli para aprender códigos *hash*. Além da vantagem de usar uma distribuição que gera diretamente os códigos binários, a arquitetura *tripletwise* modela de forma mais efetiva o tarefa de recuperação, uma vez que este tipo de aprendizado usa uma imagem como referência para aprender o conceito de relevância e não relevância, objetivo dos modelos de busca.

A seguir, no Capítulo 6, apresentamos os resultados dos experimentos realizados com os modelos MPSH e MTSH comparados ao baseline.

Capítulo 6

Experimentos e Resultados

Neste capítulo, são apresentados os resultados dos experimentos realizados para avaliar os métodos MPSH e MTSH, propostos nesta tese. Para tanto, primeiro são descritas as coleções usadas para avaliação e configurações específicas de arquiteturas e parâmetros usados. Em seguida, os resultados são apresentados para todas as coleções dadas e o *baseline* escolhido, usando métricas relacionadas com relevância e propriedades dos códigos *hash*. Ao fim, os resultados obtidos são discutidos.

6.1 Coleções de Teste

Para avaliar os métodos propostos, foram selecionadas duas coleções de imagens com diferentes tamanhos e características. A primeira é uma coleção sintética criada a partir da coleção de imagens MNIST. Como esta é uma coleção em que as imagens contém apenas um rótulo (cada imagem representa um dígito de 0 a 9), criamos uma base sintética contendo imagens que combinam os dígitos. A Figura 6.1 ilustra imagens dessa coleção. Nesta figura, temos dez imagens com seus múltiplos rótulos indicados acima das mesmas. Por exemplo, a primeira imagem combina os dígitos 3 e 4, sendo portanto rotulada como 3 e 4, o que é indicado pelo título "34" no topo da imagem. Nesta coleção, há 65.000 imagens de tamanho 28x28, cada uma representada por um (ex.: a segunda imagem), dois (ex.: a primeira imagem) ou três (ex: a terceira imagem) dos dígitos de 0 a 9. Uma imagem nunca tem o mesmo dígito ocorrendo mais de uma vez. Além disso, esta base foi criada de forma balanceada, ou seja, garantimos que quantidade de imagens fosse proporcional à quantidade de rótulos presentes em cada imagem. Como resultado, temos uma relação relativamente balanceada de imagens associadas a cada rótulo (cf. Figura 6.2) e a quantidade de imagens com 3 rótulos é maior que a quantidade de imagens com 2 rótulos que é maior que a quantidade de

imagens com 1 rótulo, conforme apresentado na Tabela 6.1. É importante notar que, da forma como esta coleção foi criada, os rótulos possíveis equivalem a objetos que podem ser vistos na imagem.

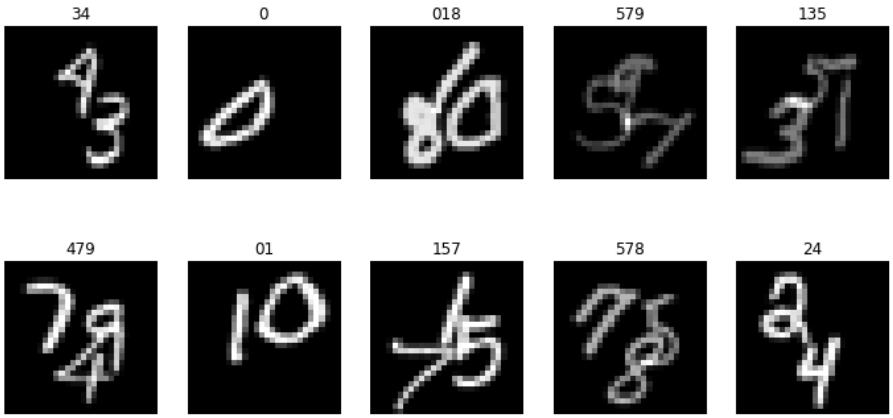


Figura 6.1. Exemplo de Imagens multi-rótulo na MNIST Multilabel

Tabela 6.1. Distribuição de imagens por rótulo

Numero de classes	Número de Imagens
1 classe	3709
2 classes	16629
3 classes	44662

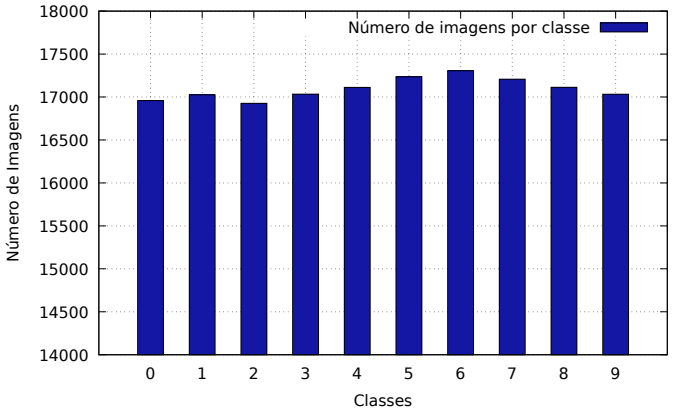


Figura 6.2. Distribuição do número de imagens por classe da coleção MNIST Multilabel.

A segunda base de dados utilizada foi a MIRFLICKR-25k. Como descrito na seção 2.3, essa coleção é composta por 25.000 imagens de tamanho 64x64, associadas a 38 classes, obtidas no site Flickr. As classes representam *tags* associadas com as imagens. Como resultado, ao contrário da primeira coleção, os rótulos nesta segunda não se restringem a objetos na imagem (ex: *snow*, *bird*, *car*, *river* etc), mas podem

descrever conceitos mais gerais que podem requerer um certo grau de interpretação (ex: *indoor*, *structures*, *urban* etc). A Figura 6.3 apresenta exemplos de imagens dessa coleção. Ao contrário da MNIST, temos uma distribuição bem mais desbalanceada de imagens por rótulo (cf. Figura 6.4) uma vez que algumas tags (ex: *people*) são muito mais usadas que outras (ex: *baby-1*). A quantidade de *tags* usadas para rotular as imagens também é bem desbalanceada com algumas imagens não rotuladas enquanto outras receberam até 16 *tags*, como mostra a Figura 6.5.

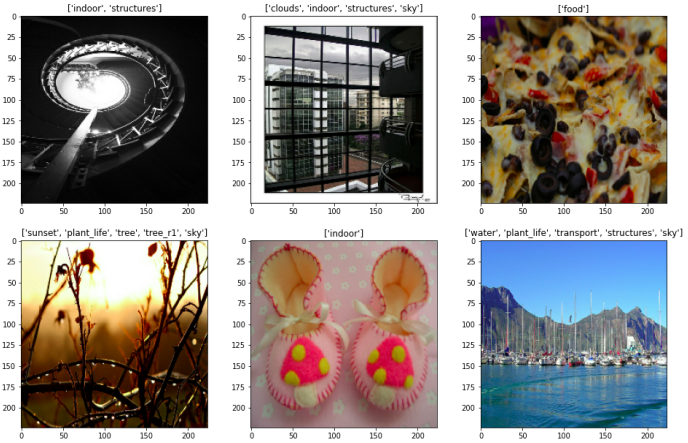


Figura 6.3. Exemplos de Imagens da Coleção MIRFLICKR-25k

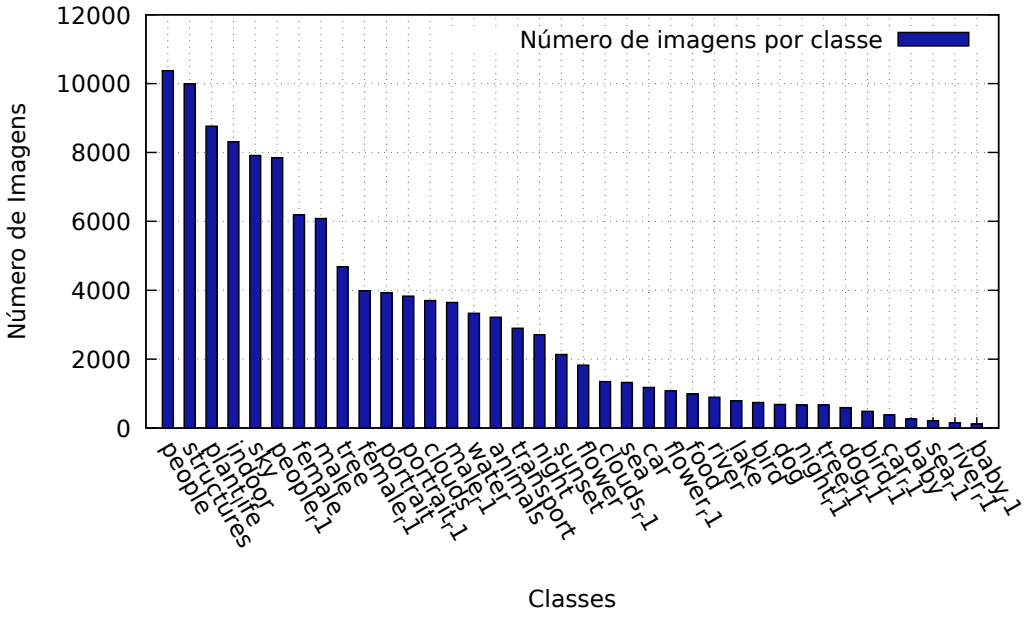


Figura 6.4. Distribuição de imagens por classe da coleção Mirflickr-25k

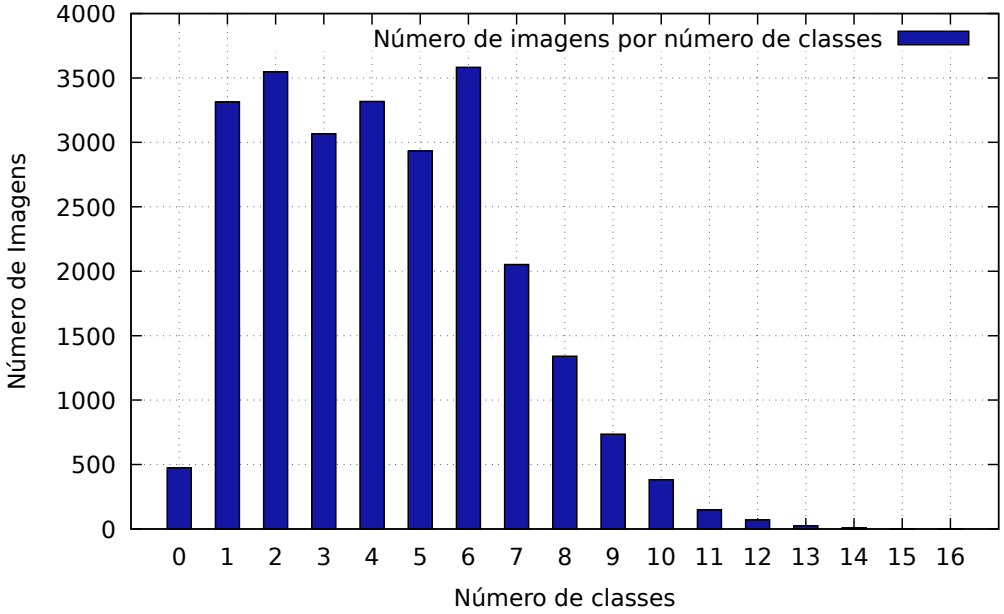


Figura 6.5. Distribuição do número de classes por imagem da coleção Mirflickr-25k

6.2 Métricas de Avaliação

Como o objetivo desta pesquisa é avaliar tanto aspectos de qualidade de representação conceitual como eficiência dos código *hash* gerados, utilizamos métricas para medir estes dois aspectos. Para avaliar qualidade da representação conceitual, utilizamos a média ponderada das precisões médias (*Weighted Mean Average Precision* - weighted MAP) [Wu et al., 2018] e a homogeneidade [Rosenberg & Hirschberg, 2007]. Para medir a eficiência, utilizamos a correlação de Pearson, o erro médio absoluto (*Mean Absolute Error*) relacionado à probabilidade de ocorrência de 0 ou 1 nos códigos, a quantidade de instâncias atribuídas por código e a cobertura do espaço de endereçamento. A seguir, apresentamos cada uma destas métricas:

- **MAP ponderada:** MAP é uma métrica de avaliação padrão usada em recuperação de informação para medir quão bom é um conjunto de respostas para uma certa consulta. Ela representa a média das precisões médias de um conjunto de consultas, que podem ser calculadas por:

$$MAP = \sum_{i=1}^{n} \frac{P@i \times pos(i)}{N_{pos}} \tag{6.1}$$

onde $pos(i)$ é uma função indicadora. Se a imagem na posição i é relevante, $pos(i)$ é 1; caso contrário é 0. N_{pos} representa o número total de imagens relevantes para

a consulta. $P@i = \frac{N_{pos(i)}}{i}$, onde $N_{pos(i)}$ é o número de imagens relevantes dentro das top i imagens retornadas. Aguns trabalhos que abordam o problema de múltiplos rótulos, utilizam uma versão ponderada dessa métrica para avaliar o nível de similaridade entre a consulta e as respostas retornadas. Esta métrica é conhecida como MAP ponderada, definida na Equação 6.2:

$$MAP\ ponderada = \sum_{i=1}^{M} s_i \times pos(i)/N_{pos} \tag{6.2}$$

onde s_i é a similaridade entre a imagem de consulta e a imagem na i-ésima posição do *ranking*, dada por $\frac{|y^{(q)} \cap y^{(i)}|}{|y^{(q)} \cup y^{(i)}|}$, onde $y^{(q)}$ é o rótulo da imagem de consulta e $y^{(i)}$ o rótulo da i-ésima posição do *ranking*.

- **Homogeneidade**: esta é uma métrica de agrupamento que mede quão homogêneo é um conjunto de instâncias em um grupo com base em suas classes. A homogeneidade é estimada através da entropia condicional[1] das classes das instâncias de um grupo [Rosenberg & Hirschberg, 2007]. Em nosso caso, o grupo é definido por todas as imagens que compartilham um mesmo código *hash*. Assim, a homogeneidade mede a concentração das classes em agrupamentos únicos de forma que, quanto menos homogêneo, mais classes o agrupamento tem (ou seja, um mesmo código foi atribuído a mais imagens de classes distintas). Para maior homogeneidade, o ideal é que cada agrupamento contenha elementos de só uma classe. No caso do cenário multi-rótulo, os agrupamentos devem conter códigos das imagens que compartilham o máximo de rótulos em comum. Para essa métrica, quanto maior o valor, melhor. Mais formalmente, a homogeneidade é dada por:

$$h = 1 - \frac{H(C|K)}{H(C)}$$

onde $H(C|K)$ é a entropia condicional das classes dados os códigos atribuídos, dada por:

$$H(C|K) = -\sum_{c=1}^{|C|}\sum_{k=1}^{|K|} \frac{n_{c,k}}{n} \cdot \log\left(\frac{n_{c,k}}{n_k}\right)$$

e $H(C)$ é a entropia das classes, definida como:

[1] A entropia condicional quantifica a quantidade de informação necessária para descrever o resultado de uma variável aleatória Y, dado que o valor de outra variável aleatória X é conhecido.

$$H(C) = -\sum_{c=1}^{|C|} \frac{n_c}{n} \cdot \log\left(\frac{n_c}{n}\right)$$

onde n é o número total de amostras, n_c é o número de amostras pertencendo à classe c, n_k é o número de amostras atribuído ao código k e $n_{c,k}$ é o número de amostras da classe c atribuída ao código k.

Note que em um problema de múltiplos rótulos, duas instâncias podem ser agrupadas se elas tem (a) o mesmo rótulo ou (b) o mesmo conjunto de rótulos. Por exemplo, considere três instâncias x_0, x_1 e x_2 com rótulos $\{0, 1\}$, $\{0\}$ e $\{0, 1\}$, respectivamente. Neste caso, se considerarmos agrupamento de único rótulo teremos os grupos $g_0 = \{x_0, x_1, x_2\}$ e $g_1 = \{x_0, x_2\}$. Por outro lado, se considerarmos agrupamento baseado em combinações de rótulos teremos os grupos $g_0 = \{x_1\}$, $g_1 = \{\}$ e $g_{01} = \{x_0, x_2\}$. Desta forma, temos duas medidas de homogeneidade, a homogeneidade de grupos baseados em rótulos tomados de forma (i) isolada e (ii) combinada.

- **Correlação de Pearson:** O coeficiente de correlação de Pearson é uma medida da força de uma associação linear entre duas variáveis [Benesty et al., 2009]. Basicamente, uma correlação de Pearson tenta traçar uma linha que melhor se ajusta aos dados de duas variáveis, e o coeficiente de correlação de Pearson, indica a distância de todos esses pontos de dados para essa linha de melhor ajuste (ou seja, quão bem, os pontos de dados se encaixam nesse novo modelo/linha de melhor ajuste). A Figura 6.6 ilustra diferentes relacionamentos e seus coeficientes de correlação, representados por r.

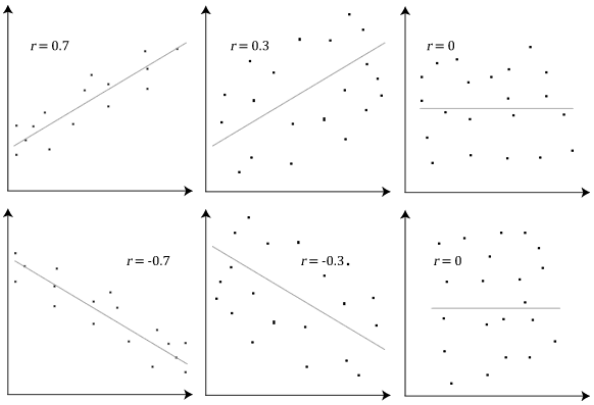


Figura 6.6. Diferentes relacionamentos e seus coeficientes de correlação

Dessa forma, quando o coeficiente de correlação r se aproxima de 1, nota-se um aumento no valor de uma variável quando a outra também aumenta, ou seja, há

uma relação linear positiva. Quando o coeficiente se aproxima de -1, também é possível dizer que as variáveis são correlacionadas, mas nesse caso quando o valor de uma variável aumenta o da outra diminui. Isso é o que é chamado de correlação negativa ou inversa.

De modo geral, um coeficiente de correlação próximo de zero indica que não há relação entre as duas variáveis, e quanto mais eles se aproximam de 1 ou -1, mais forte é a relação.

Nesta pesquisa, a correlação de Pearson é utilizada para medir a correlação entre os bits dos códigos gerados, com o intuito de avaliar o quão independentes os bits são. A ideia é que um bom método de *hashing* gera códigos que se espalham pelo espaço de maneira uniforme, de tal modo que os seus bits são independentes uns dos outros quando tomados em relação ao conjunto dos códigos gerados [Lu et al., 2017].

O coeficiente de correlação de Pearson é calculado como:

$$r = \frac{\sum_{i=1}^{i=1}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{i=1}(x_i - \overline{x})^2} \cdot \sqrt{\sum_{i=1}^{i=1}(y_i - \overline{y})^2}} \tag{6.3}$$

onde x_1, x_2, ..., x_n, y_1, y_2, ..., y_n são os valores medidos de ambas as variáveis e $\bar{x}$, $\bar{y}$ são suas médias aritméticas.

- **Erro médio absoluto da probabilidade dos bits (balanceamento de bits)**: desde que cada bit de um conjunto de códigos espalhado uniformemente por todo o espaço tem 50% de probabilidade de ser 1 [Lu et al., 2017], esta métrica indica o quanto a probabilidade de bit obtida efetivamente se aproxima de 0,5. Para isso, calculamos a média dos erros absolutos (em relação ao valor esperado 0,5) das probabilidades de ocorrência de 1s, como a seguir:

$$MAE = \frac{1}{z}\sum_{i=1}^{z}(\frac{1}{n}\sum_{j=1}^{n}|0,5 - p_i|) \tag{6.4}$$

onde z é o tamanho do código *hash*, n é o número de códigos *hash* gerados, 0,5 é a probabilidade esperada e p_i é a probabilidade observada de que o i-ésimo bit seja 1.

- **Instâncias por código**: esta métrica representa a média da quantidade de instâncias atribuídas por cada código *hash*. Quanto maior a quantidade de instâncias atribuídas a um mesmo código, maior a probabilidade de colisão.

- **Cobertura do espaço de endereçamento**: proporção dos códigos disponíveis no espaço que foram efetivamente usados para mapear imagens. Esta métrica corresponde à razão entre a quantidade de códigos efetivamente gerados e o total de códigos possíveis. Quanto mais bem espalhados são os códigos gerados, maior é a cobertura obtida no espaço de endereços possíveis.

6.3 Configuração Experimental

Como visto no Capítulo 3, o método escolhido como baseline foi o IDHN, por se tratar de um modelo recente com resultados estado-da-arte para recuperação de imagens com múltiplos rótulos.

Nos experimentos realizados, todos os modelos foram treinados por 50 épocas. A taxa de aprendizado utilizada foi de 1×10^{-4}, com taxa de decaimento de 0.96. Os tamanhos de código *hash* utilizados foram: 8, 12, 16, 24, 32, 64 e 128. Consideramos os seguintes valores para os hiperparâmetros dos modelos MPSH e MTSH: $\lambda = 0,01$, $\beta = 5 \times 10^{-2}$ e com $\alpha = 0,01$ inicial, aumentando gradualmente seu valor para $0,1$. As demais configurações dos modelos para cada coleção, são apresentadas a seguir:

- **MNIST Multilabel:** nesta coleção, consideramos 60.000 imagens para treino e 5.000 imagens para teste (consideradas consultas). Os modelos MPSH e MTSH foram treinados utilizando uma rede convolutiva composta por 3 camadas de convolução, seguidas de 2 camadas densas, no codificador. Para o decodificador, utilizou-se uma rede simétrica ao codificador. A rede de classificação foi implementada com apenas uma camada densa. O IDHN foi treinado com uma rede ResNet18 pré-treinada na Imagenet. O otimizador utilizado foi o Adam [Kingma & Ba, 2014], com *batches* de tamanho 64 para todos os modelos;

- **Mirflickr-25k:** nesta coleção, consideramos 20.000 imagens para treino e 5.000 imagens para teste (consideradas consultas). Os modelos MPSH e MTSH foram treinados usando uma rede convolutiva composta por 5 camadas de convolução, seguidas de 2 camadas completamente conectadas, no codificador. O decodificador foi construído usando uma rede simétrica ao codificador. Na rede de classificação, utilizou-se 3 camadas densas. O IDHN foi treinado como uma rede AlexNet, pré-treinada na ImageNet, conforme artigo original. O otimizador utilizado foi o RMSprop[2] [Ruder, 2016], com *batches* de tamanho 32, para todos os modelos.

[2]RMSprop é um método de taxa de aprendizado adaptativo proposto por Geoff Hinton em `http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`

Para os modelos IDHN e MPSH, os pares de imagens usados no treino foram selecionados aleatoriamente em cada iteração. Para o modelo MTSH, as triplas de imagens foram selecionadas usando a estratégia de seleção Online Semi-Hard, conforme apresentada na Seção 5.1.2. Além disso, todas os modelos foram implementados usando redes siamesas.

6.4 Resultados e Discussão

Os códigos *hash* gerados a partir dos experimentos com os modelos propostos, bem como *baseline*, foram avaliados com dois objetivos principais: qualidade e eficiência. Em termos de qualidade, avaliamos os códigos em relação à capacidade de capturar aspectos semânticos das imagens, com o intuito de melhorar precisão de recuperação e representação das imagens com múltiplos rótulos. As métricas usadas neste caso foram MAP ponderada, para Top10 e Top100, e a homogeneidade para medir a separação das classes. Em termos de eficiência, medimos as propriedades desejáveis de código *hash*. Para tanto utilizamos as métricas (i) erro médio absoluto relacionado à probabilidade dos bits, para medir balanceamento; (ii) correlação de Pearson, para avaliar independência dos bits; (iii) instâncias atribuídas por código e cobertura do espaço de endereçamento, para medir o uso da memória e colisão.

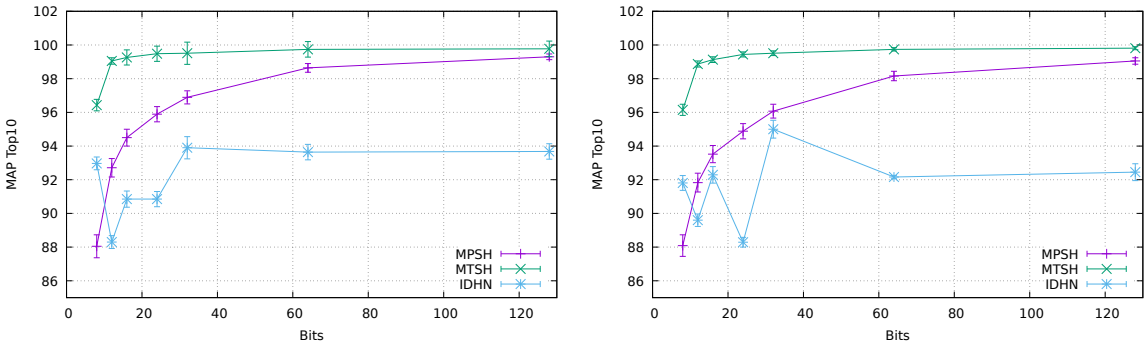
Para finalizar esta seção, apresentamos um breve resumo e discussão dos resultados empíricos alcançados por meio de nossos experimentos.

6.4.1 Resultados para MNIST Multilabel

Os gráficos apresentados nas Figuras 6.7a e 6.7b mostram, respectivamente, os valores de MAP ponderada, para as primeiras 10 (Top10) e 100 (Top100) imagens sugeridas como respostas pelos métodos MPSH, MTSH e IDHN, considerando as 5 mil imagens de consulta da coleção MNIST Multilabel. Nestas figuras, são apresentados os resultados obtidos, considerando diferentes tamanhos de código *hash*: 8, 12, 16, 24, 32, 64 a 128 bits. Os valores de MAP são exibidos junto com o intervalo de confiança considerando um nível de 95% (indicado por barras verticais), dado que estas métricas são tomadas como médias dos resultados de 5 mil consultas.

Como podemos observar, o modelo MTSH é consideravelmente melhor que os demais modelos, tanto para MAP Top10 quanto para MAP Top100[3], independente do comprimento de bits usados. As diferenças entre os métodos são estatisticamente

[3]Em geral, o Top100 é muito usado com imagens pois os usuários tendem a verificar mais respostas em busca de imagens que texto.



(a) *Top-10* **(b)** *Top-100*

Figura 6.7. MAP ponderada para diferentes comprimentos de código na coleção MNIST Mutilabel. Intervalos exibidos para nível de confiança de 95%.

significativas, considerando nível de confiança de 95%. Isso sugere que a estratégia de treino de *ranking* baseada em triplas de pontos é mais efetiva que a baseada em duplas de pontos.

De modo geral, o MPSH e MTSH produzem resultados melhores que o IDHN. A única exceção é que o IDHN é superior ao MPSH para códigos de comprimento 8. Em suma, apesar do IDHN focar mais em otimizar qualidade da representação conceitual, ele tem desempenho inferior aos demais métodos. Considerando que a principal diferença entre o IDHN e o MPSH é o uso de um processo de quantização no primeiro contra uma geração automática de códigos binários no segundo, o resultado obtido sugere que a geração direta de códigos binários é uma abordagem mais efetiva para a criação de códigos *hash* que serão usados em uma tarefa de busca.

Ao contrário do IDHN, o MPSH e o MTSH foram projetados para gerar códigos *hash* binários diretamente. Com isso esperamos produzir códigos com propriedades de *hashing* melhores que o de métodos anteriores. A seguir (nas Figuras 6.8 e 6.9), verificamos duas destas propriedades esperadas para métodos de *hashing*: o uso balanceado de todos os bits dos códigos gerados e quão correlacionados são estes bits. Se espera de métodos que geram códigos *hash* bem espalhados pelo espaço, que os bits sejam bem balanceados e pouco correlacionados.

Como antes, as figuras mostram o balanceamento de bits e o coeficiente de correlação considerando diferentes tamanhos de código *hash*: 8, 12, 16, 24, 32, 64 a 128 bits. Em particular, a Figura 6.8 mostra o balanceamento dos bits, indicando o quanto a probabilidade de uso de cada bit no código se afasta de 0.5. Quanto mais próximo de 0.5, melhor balanceado é o uso de cada bit. Como podemos ver, o MTSH é o que obtém os melhores resultados para quase todos os comprimentos de código. Ele é apenas ultrapassado pelo MPSH para códigos de comprimento 32, sendo melhor que o IDHN

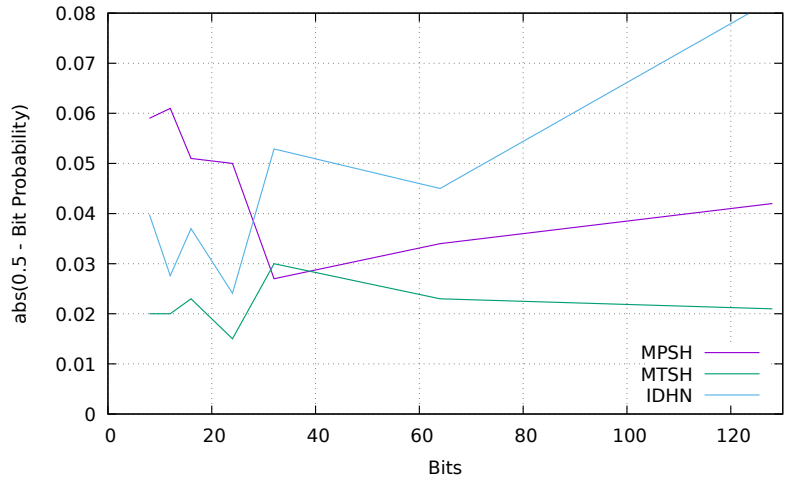


Figura 6.8. Análise do balanceamento dos bits para diferentes tamanhos de códigos, usando erro médio absoluto em relação a probabilidade dos bits - Coleção MNIST Mutilabel.

para todos os comprimentos de código. O IDHN, por sua vez, é melhor que o MPSH para comprimentos abaixo de 32, sendo pior em todos os demais casos.

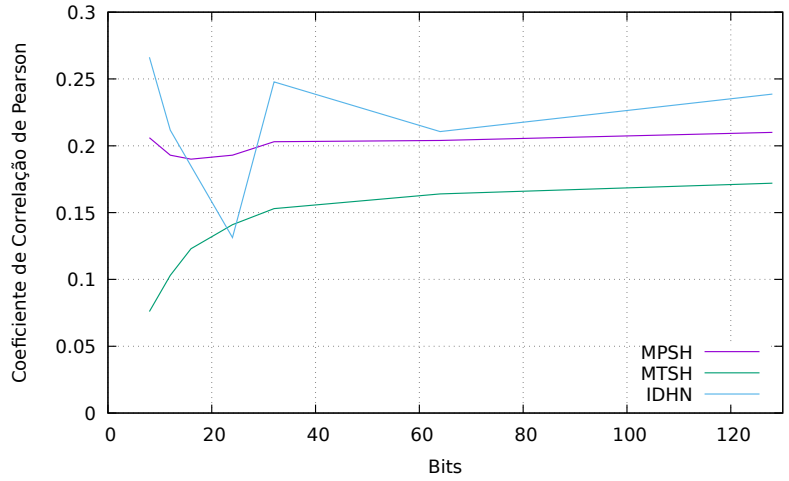


Figura 6.9. Análise da correlação entre os bits para diferentes tamanhos de códigos, usando coeficiente de correlação de Pearson - Coleção MNIST Mutilabel.

A Figura 6.9 mostra a correlação média entre os bits. Quanto menor a correlação, mais bem espalhado são os códigos gerados. De forma geral, o MTSH e MPSH são sempre melhores que o IDHN, exceto para comprimento igual a 24 bits. O MTSH é consideravelmente melhor que o MPSH em todos os comprimentos de bits.

Em termos práticos, códigos com estas propriedades devem se traduzir em métodos de *hashing* que usam melhor o espaço de endereçamento, com menos colisões por código. É isto que verificamos a seguir, no gráficos da Figura 6.10 e na Tabela 6.2, considerando diferentes comprimentos de códigos.

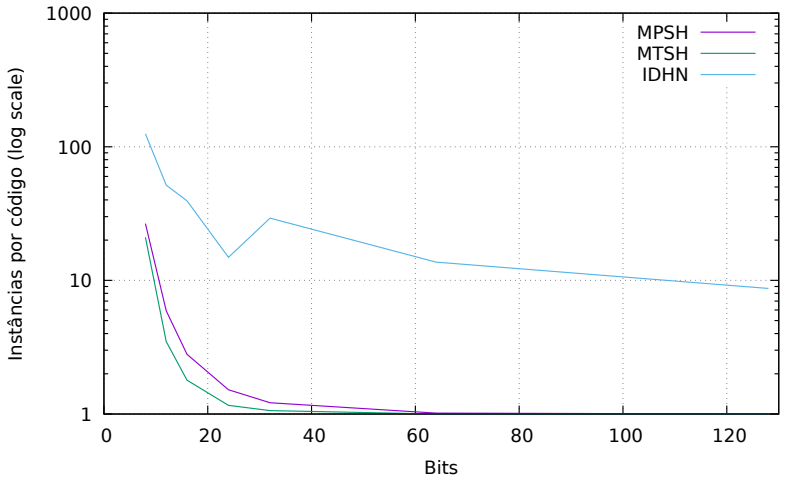


Figura 6.10. Instâncias atribuídas por código - Coleção MNIST Mutilabel.

Comparado ao IDHN, o MTSH e o MPSH compartilham muito menos códigos entre imagens, portanto, produzindo menos colisões. Isso sugere que os alvos de otimização do MTSH e MPSH, voltados à criação de uma representação binária direta pela rede neural, são realmente capazes de gerar códigos *hash* mais efetivos em termos de uso do espaço de representação. Em particular, o IDHN produz códigos com muitas colisões mesmo para comprimentos muito altos. Isso se justifica pelo fato que o IDHN não é otimizado especificamente para gerar bons códigos binários.

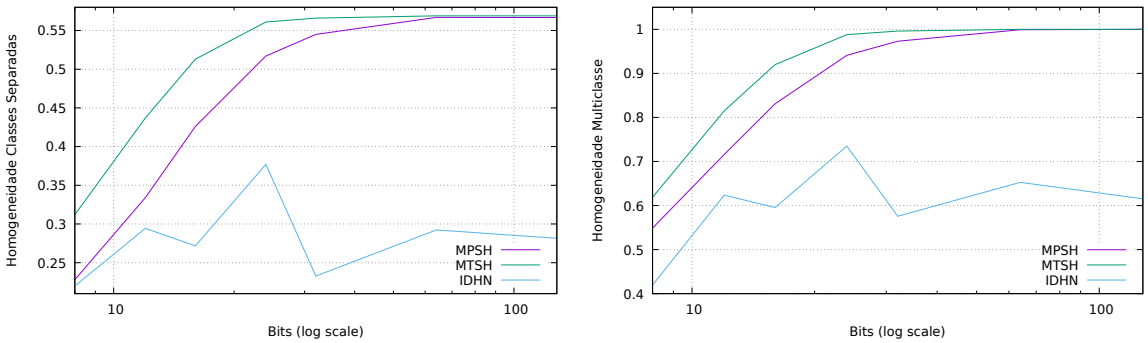
Entre o MTSH e o MPSH, o MTSH é melhor exatamente para comprimentos menores, o que implica que ele possibilita melhor uso do espaço de endereçamento para situações de menor número de recursos de memória. Isso é confirmado na Tabela 6.2, onde é mostrada a cobertura usada do espaço de *hashing* para cada método considerando diferentes comprimentos de código.

Tabela 6.2. Espaço de endereçamento - Coleção MNIST Multilabel

Modelo	8 Bits	12 Bits	16 Bits	24 Bits	32 Bits	64 Bits	128 Bits
MPSH	73.40%	20.60%	2.70%	0.00%	0.00%	0.00%	0.00%
MTSH	**93.00%**	**35.00%**	**4.20%**	0.00%	0.00%	0.00%	0.00%
IDHN	15.62%	2.36%	0.19%	0.00%	0.00%	0.00%	0.00%

Claramente o MTSH usa melhor o espaço de endereçamento que o MPSH, embora a diferença seja pequena. Ambos são bem melhores que o IDHN. Naturalmente, com o IDHN usando os mesmos códigos para tantas imagens diferentes, é de se esperar que haja mais heterogeneidade nos *clusters* de imagens gerados pelos códigos do IDHN do que do MTSH e MPSH. Isso pode ser visto nos gráficos da Figura 6.11 para diferentes tamanhos de código.

Nesses gráficos, vemos o grau de homogeneidade dos *clusters* definidos pelos códi-



(a) *Rótulos isolados* **(b)** *Rótulos combinados*

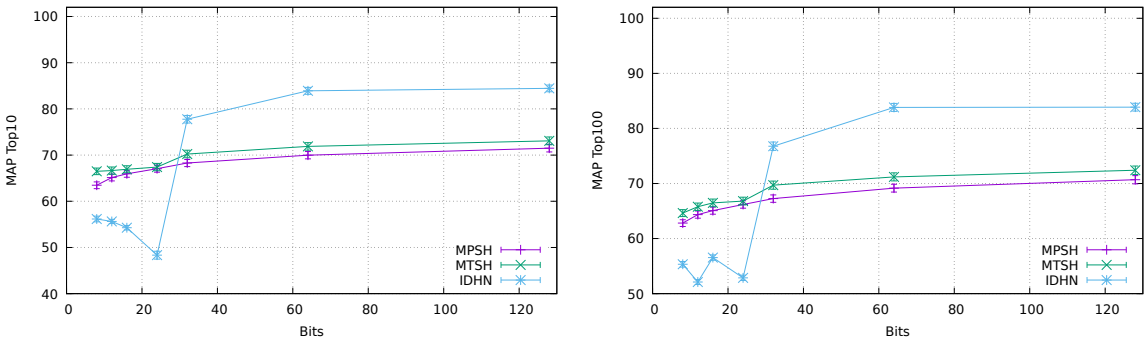
Figura 6.11. Homogeneidade por comprimento de código calculada considerando grupos definidos com base em rótulos tomados de forma (a) isolada ou (b) combinada. Tamanho dos códigos em escala logarítmica - Coleção MNIST Mutilabel.

gos *hash* para cada comprimento de código. Na Figura 6.11a, em particular, avaliamos a capacidade dos modelos de separar as 10 classes existentes na coleção (ou seja, classes de 0 a 9). Como podemos observar, o MTSH e MPSH produzem *clusters* mais homogêneos em todos os comprimentos de bits comparados ao IDHN. A Figura 6.11b mostra a formação dos *clusters* considerando todas as classes possíveis, ou seja, imagens com 1 rótulo, 2 rótulos e 3 rótulos, totalizando 175 combinações de classes. Nesse cenário, o MPSH e MTSH também se apresentam melhores em relação ao IDHN, produzindo *clusters* mais homogêneos. Em ambos os casos, os métodos MPSH e MTSH produzem *clusters* de imagens mais homogêneos enquanto que o IDHN produz *clusters* mais heterogêneos (o que indica mais colisões). O MTSH é levemente melhor que o MPSH quando consideramos códigos menores.

6.4.2 Resultados para Mirflickr-25k

Apresentamos agora os resultados dos modelos para a coleção Mirflickr-25k. Os gráficos apresentados nas Figuras 6.12a e 6.12b mostram, respectivamente, os valores de MAP ponderada, para as primeiras 10 (Top10) e 100 (Top100) imagens sugeridas como respostas pelos métodos MPSH, MTSH e IDHN, considerando as 5 mil imagens de consulta da coleção Mirflickr-25k.

Como podemos observar, o modelo MTSH é melhor que o MPSH e IDHN para códigos menores que 32 bits. Contudo, ele perde para o IDHN para comprimentos de código maiores que 24 bits, tanto para MAP Top10 quanto para MAP Top100. Esses resultados são diferentes dos obtidos na coleção MNIST Multilabel, para avaliação de MAP, onde o modelo MTSH foi melhor que os modelos MPSH e IDHN para todos



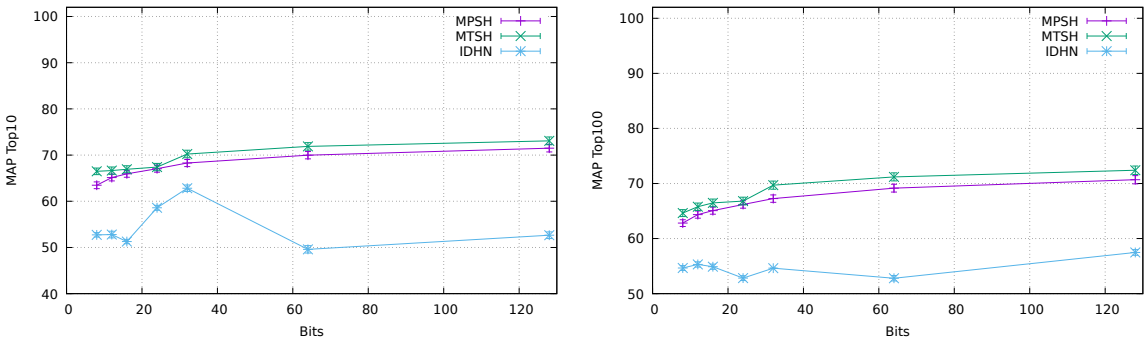
(a) *Top-10* **(b)** *Top-100*

Figura 6.12. MAP ponderada para diferentes comprimentos de código na coleção Mirflickr-25k. Intervalos exibidos para nível de confiança de 95%.

os comprimentos de código. Uma explicação para esse resultado é o uso da transferência de aprendizado. O modelo IDHN aprende características de imagens a partir de uma arquitetura profunda pré-treinada na Imagenet. Desta forma, as camadas de representação do IDHN já iniciam o seu treino com capacidade de reconhecer imagens fotográficas de cenas reais coloridas. Isso permite ao modelo focar o aprendizado na adaptação do *ranking* das imagens da Mirflickr-25k. Ao contrário, os modelos MSPH e MTSH não puderam contar com transferência de aprendizado.

Como visto na Seção 2.2.6, a transferência de aprendizado é um método de aprendizado de máquina em que reutilizamos um modelo pré-treinado como ponto de partida para um modelo em uma nova tarefa. Para simplificar, um modelo treinado em uma tarefa é reaproveitado em uma segunda tarefa relacionada como uma otimização que permite um progresso rápido ao modelar a segunda tarefa. Ao aplicar o transferência de aprendizado a uma nova tarefa, pode-se obter um desempenho significativamente maior do que o treinamento com apenas uma pequena quantidade de dados. O uso dessa estratégia é tão comum que é raro treinar um modelo para uma imagem ou tarefas relacionadas ao processamento de linguagem natural do zero (cf. Capítulo 3). Os resultados obtidos na coleção MNIST Multilabel não foram afetados pela falta de transferência de aprendizado por ser uma coleção simples, composta apenas por dígitos. Diferentemente, a coleção Mirflickr-25k é composta por imagens reais, com características mais complexas a serem aprendidas. Nos modelos MPSH e MTSH não utilizamos transferência de aprendizado por não existir na literatura uma rede autocodificadora variacional pré-treinada na Imagenet, muito menos que use uma distribuição de Bernoulli.

Para termos uma ideia do impacto da transferência de aprendizado nesses modelos, re-treinamos o IDHN completamente sem usar pesos previamente aprendidos do



(a) *Top-10* **(b)** *Top-100*

Figura 6.13. MAP ponderada para diferentes comprimentos de código na coleção Mirflickr-25k, considerando treinamento sem transferência de aprendizado para o método IDHN. Intervalos exibidos para nível de confiança de 95%.

treino com a Imagenet. Os resultados são mostrados nos gráficos das Figuras 6.13a e 6.13b. Como podemos observar, neste cenário o IDHN é pior em todos os comprimentos de bits, tanto para MAP Top10 quanto para Top100. Esse resultado nos mostra que os modelos MTSH e MPSH, mesmo sem uso de transferência de aprendizado, conseguem obter bons resultados, mesmo para comprimentos de códigos muito pequenos, como 8 bits. Note que o IDHN tem dificuldade de gerar bons resultados para códigos menores que 32 bits, mesmo usando transferência de aprendizado (cf. Figuras 6.12a e 6.12b)

Como o modelo IDHN não tem bom desempenho sem o uso de transferência de aprendizado, os próximos resultados apresentados para esse modelo são considerando essa estratégia.

A Figuras 6.14 mostra os resultados dos métodos para a propriedade de balanceamento. Como podemos observar, em geral, os modelos MTSH e MPSH geram códigos mais balanceados que o modelo IDHN. O modelo IDHN apresenta resultado superior apenas para comprimentos de código de 32 e 64 bits, e gera códigos muito desbalanceados para códigos menores que 32 bits. O MPSH é melhor nos códigos de tamanho 8 e 16 bits. E por fim, o MTSH apresenta melhor desempenho nos comprimentos de 12, 24 e 128 bits.

Na Figura 6.15, apresentamos os resultados para a propriedade de correlação entre os bits dos códigos gerados. Como visto na Seção 6.2, um coeficiente de correlação de Pearson próximo de zero indica que não há relação entre as duas variáveis, e quanto mais eles se aproximam de 1 ou -1, mais forte é a relação. Dessa forma, observamos que em geral, o MTSH e MPSH são sempre melhores que o IDHN, exceto para comprimento igual a 24 bits.

As Tabelas 6.3 e 6.4 apresentam o número de instâncias atribuídas por código e

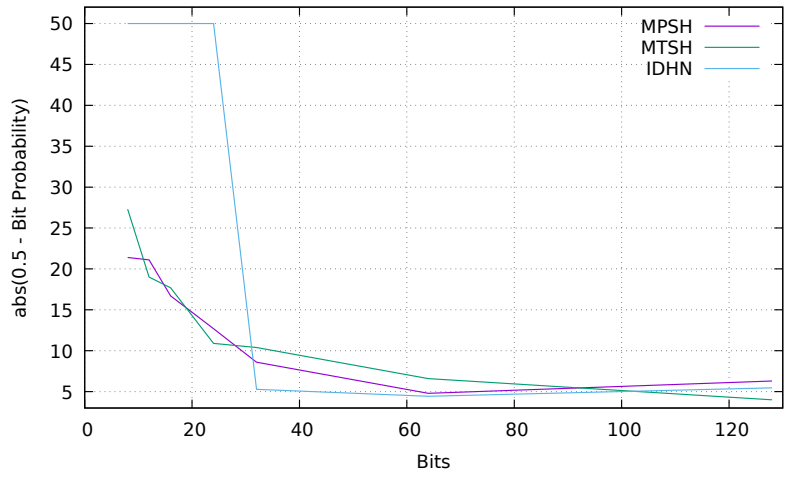


Figura 6.14. Análise do balanceamento dos bits para diferentes tamanhos de códigos, usando erro médio absoluto em relação a probabilidade dos bits - Coleção Mirflickr-25k.

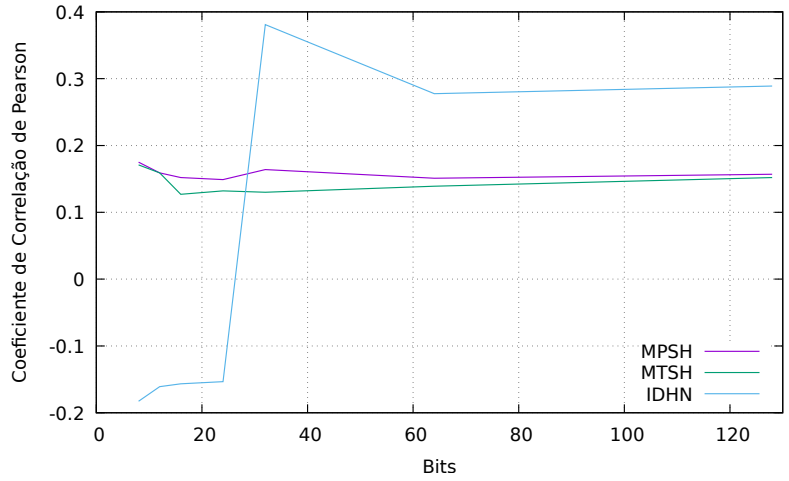


Figura 6.15. Análise da correlação entre os bits para diferentes tamanhos de códigos, usando coeficiente de correlação de Pearson - Coleção Mirflickr-25k.

cobertura do espaço de endereçamento, respectivamente. Os modelos MPSH e MTSH sempre atribuem menos instâncias por código que o IDHN, para todos os comprimentos de código (Tabela 6.3). Como consequência, eles apresentam melhor cobertura do espaço de endereçamento, como apresentado na Tabela 6.4. Note que para códigos de tamanho menores que 32 bits, o modelo IDHN atribui todo o conjunto de teste (5 mil imagens) para apenas um código. Esse efeito é consequência do processo de quantização que é feito de forma separada do processo de aprendizagem. Como esse modelo não gera códigos binários diretamente, mesmo que a rede construa diferentes representações para as imagens (composta de valores reais entre 0 e 1), se estas forem sempre próximas, ao aplicar o processo de quantização, os valores dos códigos binários serão os

mesmos para todas as imagens. Para exemplificar, imagine que este modelo gere as seguintes representações para as imagens $I_1 = [0.32, 0.87, 0.43, 0.76, 0.96, 0.23, 0.65, 0.81]$ e $I_2 = [0.48, 0.62, 0.31, 0.86, 0.75, 0.47, 0.58, 0.53]$. Embora estas sejam representações diferentes, ao aplicar o processo de quantização, ambos os códigos seriam transformados em $I_1 = I_2 = [0, 1, 0, 1, 1, 0, 1, 1]$. Como o modelo IDHN não conseguiu aprender muito bem as características das imagens para tamanhos de bits menores que 32 bits, acabou gerando representações muito próximas para todas as imagens, o que explica esse efeito.

Tabela 6.3. Instâncias atribuídas por código - Coleção Mirflickr-25k

Modelo	8 Bits	12 Bits	16 Bits	24 Bits	32 Bits	64 Bits	128 Bits
MPSH	**52.29**	11.96	**2.82**	1.16	1.04	1.00	1.00
MTSH	117.04	**10.75**	3.07	**1.10**	**1.02**	1.00	1.00
IDHN	5000.00	5000.00	5000.00	5000.00	4.11	1.58	1.21

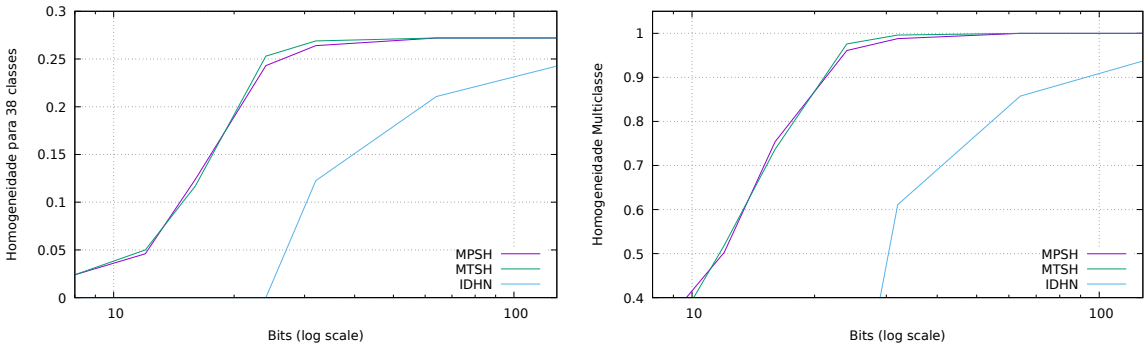
Tabela 6.4. Espaço de endereçamento - Coleção Mirflickr-25k

Modelo	8 Bits	12 Bits	16 Bits	24 Bits	32 Bits	64 Bits	128 Bits
MPSH	**36.7%**	10.00%	**2.70%**	0.00%	0.00%	0.00%	0.00%
MTSH	16,4%	**11.20%**	2.40%	0.00%	0.00%	0.00%	0.00%
IDHN	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%

Por fim, nos gráficos das Figuras 6.16a e 6.16b mostramos o nível de homogeneidade para *clusters* definidos pelos códigos *hash* para cada de comprimento de código, levando em consideração todas as classes combinadas e as 38 classes existentes na coleção, respectivamente. Como podemos observar, o MTSH e MPSH produzem *clusters* mais homogêneos em todos os comprimentos de bits comparados ao IDHN. A Figura 6.16b mostra o resultado para formação dos *clusters* considerando todas as classes possíveis, ou seja, imagens com 1 rótulo até 16 rótulos (cf. Figura 6.5), totalizando 1585 combinações de classes. Nesse cenário, o MPSH e MTSH também se apresentam melhores em relação ao IDHN, produzindo *clusters* mais homogêneos. Em ambos os casos, os métodos MPSH e MTSH produzem *clusters* de imagens mais homogêneos enquanto que o IDHN produz *clusters* mais heterogêneos. Note que o modelo IDHN obtém homogeneidade igual a zero para comprimentos de códigos menores que 32 bits, o que era esperado considerando os resultados obtidos nas métricas apresentadas anteriormente.

6.4.3 Discussão de Resultados

A partir do resultados apresentados, de forma geral, podemos concluir que o MTSH representa o melhor compromisso entre desempenho e boas características de *hashing*. Além disso, ambos métodos baseados em *ranking*, MPSH e MTSH, são quase sempre



(a) *Rótulos isolados* **(b)** *Rótulos combinados*

Figura 6.16. Homogeneidade por comprimento de código calculada considerando grupos definidos com base em rótulos tomados de forma (a) isolada ou (b) combinada. Tamanho dos códigos em escala logarítmica - Coleção Mirflickr-25k.

melhores que IDHN, tanto em termos de qualidade quanto uso do espaço de endereçamento. De fato, o IDHN não gera bons códigos *hash*. Os códigos gerados apresentam sempre muitas colisões para todos os tamanhos de código considerados. Como resultado, ele sempre apresenta a pior cobertura do espaço de endereçamento. Em geral, dado um código, há mais imagens dissimilares relacionadas com este código no IDHN que nos demais métodos.

Comparando os resultados dos três modelos nas duas coleções, observamos que o MTSH obtém um desempenho consistentemente melhor que o MPSH e o IDHN na coleção sintética MNIST. Isto não é observado na coleção de imagens reais Mirflickr-25k, principalmente, em particular, à qualidade de representação conceitual do *ranking*. Contribuem para estes resultados a inerente dificuldade de aprendizado de conceitos em imagens reais, que podem ser difíceis de caracterizar, além de ocorrerem de forma desbalanceada, o que implica em distribuições desbalanceadas de classes. Categorias em uma imagem real podem, por exemplo, ter natureza mais subjetiva ou culturalmente vinculada, como a descrição de um sentimento (“tristeza”) associado com a foto de um animal (“gato”) ou o nome de um filme (“A origem”) usado como rótulo para uma cena do mesmo (“um pião girando”). Mesmo quando as categorias descrevem objetos, estes podem ocultar uns aos outros parcialmente, dificultando a sua caracterização.

Nada disso ocorre na coleção MNIST que, além de ser balanceada, é formada sempre por combinações de imagens de dígitos. Neste cenário, as imagens correspondem a objetos de representação visual simples (dígitos 0 a 9) que são combinados de forma que há exemplos suficientes para que o modelo aprenda quaisquer combinações que ocorram (devido ao balanceamento). Além disso, tais combinações são relativamente simples, sem condições de complexidade adicional como oclusões. Como resultado, as

arquiteturas rasas de representação de imagens, usadas pelo MTSH e MPSH para a MNIST, são capazes de aprender as imagens e suas ordens relativas. Quando aplicado às imagens reais, contudo, o processo de aprendizado simultâneo das características das imagens com representação binária associado ao seu *ranking* relativo se mostra uma tarefa muito mais complexa.

O IDHN se sai melhor neste último caso, pois usa transferência de aprendizado e, deste modo, parte de um modelo pré-inicializado já capaz de reconhecer várias características complexas de imagens reais. Assim, o IDHN pode focar em uma tarefa mais simples que é adaptar o seu modelo de representação inicial para aprender um *ranking* relativo baseado em pares. Por outro lado, os modelos MPSH e MTSH, devem simultaneamente aprender a reconstruir as imagens (sem partir de um ponto inicial em que já sabem como caracterizá-las), usando uma representação binária que reflete suas ordens. E no caso do MTSH, ele deve aprender este *ranking* selecionando boas combinações de trios de imagens sobre tópicos desbalanceados, ao longo do processo. Estas são tarefas de aprendizagem claramente mais complexas para a qual o treino na Mirflickr-25k, por si só, não parece levar a um modelo robusto o suficiente para ser comparado com um pré-treinado em uma coleção muito maior, como a Imagenet. Isso fica claro quando, ao usarmos o IDHN sem transferência de aprendizagem, observamos um desempenho drasticamente inferior.

6.5 Considerações Finais

Neste capítulo, apresentamos os resultados dos experimentos realizados nas coleções MNIST Multilabel e Mirflickr-25k, para os modelos MPSH, MTSH e IDHN. Os métodos foram avaliados em ambas as coleções para medir qualidade de busca e eficiência dos códigos *hash* gerados. Para qualidade de busca, utilizamos as métricas: MAP ponderada nos Top10 e Top100 respostas mais relevantes, além das métricas de homogeneidade da formação dos *clusters* considerandos todos os rótulos. Para medir eficiência, fizemos uso das métricas de erro médio absoluto relacionado a probabilidade dos bits, para medir balanceamento; correlação de Pearson, para avaliar independência dos bits; instâncias atribuídas por código e cobertura do espaço de endereçamento, para medir o uso da memória e colisão.

Os resultados apresentados nos mostram que, de forma geral, métodos de *ranking* como o MTSH, que utilizam uma arquitetura baseada em trios de imagens, combinada ao uso de uma distribuição de Bernoulli, capaz de gerar códigos binários diretamente, apresentam um bom compromisso entre qualidade de busca e boas propriedades de

hashing, para a tarefa de recuperação de imagens com múltiplos rótulos, aplicação de interesse desta pesquisa. Contudo, vimos que modelos como o MPSH e MTSH, que utilizam redes autocodificadoras variacionais com distribuições discretas, podem ser difíceis de treinar, principalmente por não ser possível aplicar estratégias de transferência de aprendizado, diferente do modelo IDHN, que faz uso de uma rede pré-treinada na Imagenet.

Capítulo 7

Conclusões e Trabalhos Futuros

A seguir, apresentamos a conclusão deste trabalho, respondendo as questões que motivaram esta pesquisa. Além disso, apresentamos algumas limitações detectadas ao longo do seu desenvolvimento, e por fim discutimos algumas estratégias que podem ser exploradas em trabalhos futuros para melhorar os resultados apresentados nesta tese.

7.1 Conclusões

Nesta pesquisa, apresentamos duas abordagens de aprendizado profundo com o intuito de aprimorar técnicas de *hashing* para tarefas de recuperação de imagem com múltiplos rótulos. A primeira abordagem foi o MPSH, um modelo supervisionado que usa uma rede autocodificadora variacional com distribuição de Bernoulli, para aprender códigos *hashing* de imagens com múltiplos rótulos. A principal vantagem desse modelo consiste em usar uma distribuição discreta, que elimina a necessidade de uma etapa de quantização desvinculada do processo de treino, com a produção de uma representação binária da imagem de entrada, diretamente aplicável como código *hash*. Além disso, o treino baseado em uma estratégia *pairwise* possibilita a aprendizagem eficiente da tarefa supervisionada de *ranking*, adequada para busca de imagens.

A segunda estratégia apresentada foi o MTSH, um modelo *tripletwise* supervisionado que usa uma rede autodificadora variacional com distribuição de Bernoulli, para aprender códigos *hash* de imagens com múltiplos rótulos. Este foi o primeiro modelo *tripletwise* proposto para recuperação de imagens com múltiplos rótulos que usa uma rede autocodificadora variacional com distribuição de Bernoulli para aprender códigos *hash*. Além da vantagem de usar uma distribuição que gera diretamente os códigos binários, a arquitetura *tripletwise* modela de forma mais efetiva o tarefa de recuperação,

visto que este tipo de aprendizado usa uma imagem como referência para aprender o conceito de relevância e não relevância, objetivo dos modelos de busca.

Este trabalho motivou uma série de perguntas de pesquisa, que respondemos a seguir.

Como garantir representações que possibilitem hashing *eficiente em um cenário com múltiplos rótulos?*

Nossa pesquisa sugere que é melhor gerar códigos binários diretamente pelo modelo sem processo de quantização separado. Em nosso caso, fizemos isso por considerar que cada bit do código segue uma distribuição de Bernoulli e usamos um modelo capaz de lidar com esta distribuição discreta. Para incorporar o conceito de múltiplos rótulos aos modelos, incluímos nas funções de perda componentes baseados na proporção de rótulos em comum. No caso do modelo baseado em triplas, também modificamos a função de seleção de triplas para definir casos positivos e negativos em relação à imagem âncora (de consulta) com base nos múltiplos rótulos. A combinação destas ideias produziu representações que se mostraram melhores que a do *baseline* usado para comparação.

Entre os modelos de representação obtidos através de aprendizagem de ranking, seria melhor usar aqueles baseados em pares ou triplas de instâncias, considerando imagens de múltiplos rótulos?

Em nossos experimentos com modelos adaptados para imagens de múltiplos rótulos, com treinamento de *ranking* baseado em pares e triplas, observamos que os modelos baseados em triplas geralmente alcançam melhores resultados tanto em termos da relevância das imagens recuperadas quanto em termos de efetividade do código *hash* gerado. Estes métodos, contudo, são mais difíceis de treinar, uma vez que exigem o uso de técnicas específicas de seleção de exemplos para tornar viável a aprendizagem. Também ressaltamos que, embora tenhamos observado ganhos destes métodos em uma variedade de condições, eles usaram melhor o espaço de endereçamento para códigos de tamanho menor. Isso sugere que eles podem produzir soluções melhores em cenários onde o tamanho do código *hash* a ser criado é grande em relação ao espaço disponível.

Quão vantajoso é o aprendizado direto do código binário, usando um modelo gerador baseado em uma distribuição de Bernoulli em comparação com técnicas alternativas de quantização?

A geração de códigos binários diretos produz resultados claramente melhores que estratégias de quantização feitas de forma desatrelada do processo de aprendizagem do modelo. Isso sugere que processos de quantização são sub-ótimos em termos de

representação conceitual, ou seja, a quantização leva a perdas importantes na qualidade da representação feita pelo modelo.

Os modelos construídos obtém códigos binários que usam de forma mais efetiva o espaço de representação e tem propriedades de interesse como códigos com bits independentes e balanceados?

Sim. Vale ressaltar que os códigos gerados apresentaram propriedades interessantes para uso por métodos de *hashing* (quando comparados ao *baseline*) sem que eles tenham sido criados diretamente para otimizar estas propriedades. Nossa hipótese de que eles produziriam bons códigos se sustentava mais na geração direta de códigos binários, baseada na crença de que a quantização (em particular, se muito desatrelada do treino) poderia prejudicar a representação aprendida. Também observamos que o método baseado em triplas produziu os melhores códigos, especialmente para os casos onde o comprimento de código é menor (ou seja, a relação entre espaço disponível e necessário é pequena).

Como os modelos propostos se comparam com métodos estado-da-arte quanto ao seu desempenho computacional e qualidade de recuperação?

Nesta tese, comparamos nosso modelo diretamente com um *baseline*, o IDHN. Nós observamos ampla vantagem sobre o *baseline*, em particular na coleção sintética, tanto no que tange à qualidade da recuperação quanto em relação às propriedades dos códigos gerados. Este segundo aspecto nos faz acreditar que os códigos gerados por nossos modelos podem levar a métodos melhores em termos de eficiência de busca, especialmente para cenários onde o espaço de endereçamento disponível é menor em relação ao necessário, já que nossos modelos geram códigos que espalham melhor, com menos colisões e sem grande perda conceitual associada. Em termos de treino, claramente, os modelos baseados em triplas tem custo muito maior que os demais e isto deve ser considerado no treinamento de coleções muito grandes. Finalmente, em termos de qualidade de representação, observamos que para lidar com imagens reais de forma efetiva, os modelos devem ser treinados usando grandes coleções de imagens ou a partir de um processo de transferência de aprendizagem para a obtenção de resultados competitivos com o estado-da-arte.

7.2 Limitações desta Pesquisa

Entre as principais limitações desta pesquisa, citamos:

1. Modelos de aprendizagem profunda são tipicamente muito complexos, necessi-

tando de grandes volumes de dados e, por extensão, enorme quantidade de recursos computacionais para serem treinados de forma adequada. Ao longo desta pesquisa, a falta de recursos computacionais para teste com modelos estado-da-arte foi um problema que dificultou a criação de arquiteturas mais complexas e o uso de coleções de referência maiores, com imagens de melhor definição. Nesta pesquisa, tentamos mitigar este problema usando coleções pequenas, reais ou sintéticas, de forma a minimizar a necessidade de muitos recursos de hardware por muito tempo. O problema desta abordagem é que o tamanho das coleções se mostrou um obstáculo para que o treino baseado em estratégias de *ranking* levasse a resultados idênticos ao estado-da-arte em termos de representação da imagem.

2. Modelos estado-da-arte em representação de imagens tipicamente são treinados por meio de processos de transferência de aprendizagem. Assim, parte do bom resultado obtido por estes modelos se deve ao aprendizado prévio em grandes coleções como a Imagenet. Para obter resultados similares, é necessário se utilizar de processos similares, o que é complexo para novos modelos, como o que propomos, baseado em autocodificadores variacionais com distribuições discretas. Na prática, comparamos o nosso método, sem transferência de aprendizagem, com métodos cujos resultados estado-da-arte dependem de transferência.

3. Embora questões de eficiência tenham motivado esta pesquisa, não avaliamos os métodos propostos através de uma aplicação real de busca de imagens. Uma aplicação como esta nos permitira avaliar diretamente aspectos como tempo de resposta e uso efetivo de memória. Por outro lado, também iria requerer diversas outras escolhas de engenharia que teriam impacto sobre eficiência e que consideramos fora do escopo desta pesquisa como alternativas de arquiteturas, mecanismos de distribuição, *caching* etc. Assim, optamos apenas por avaliar características que induzem um bom método de *hashing*, o que seria a base para uma aplicação real.

4. Embora este trabalho tenha sido motivado pelo estudo de métodos de aprendizagem mais específicos para tarefa de busca, o que engloba técnicas supervisionadas de aprendizagem de *ranking*, não implementamos nenhuma estratégia *listwise*. Note que isto não se deveu propriamente a alguma crença sobre a não adequação ou baixa efetividade destas técnicas, mas sim ao seu alto custo de aprendizagem. Decidimos por evitar esta linha dadas as nossas restrições de tempo e *hardware*.

7.3 Trabalhos Futuros

Em um cenário ideal, seria interessante testar nossas ideias em coleções de referências maiores e com imagens de maior definição. Isso possibilitaria uma avaliação mais completa dos métodos propostos bem como uma comparação mais simples com outros métodos na literatura.

Com mais recursos e tempo, seria interessante estudar de forma mais completa como realizar a transferência de aprendizagem para os modelos variacionais propostos neste trabalho. Alternativas interessantes incluem:

- Realizar um treinamento prévio do modelo autocodificador completo na Imagenet e usar os pesos aprendidos para treinar o modelo de *ranking* em seguida;
- Realizar um treino prévio na Imagenet para posterior transferência parcial de aprendizagem para um segmento do autocodificador;
- Dividir o treino da coleção alvo em duas etapas, a saber, reconstrução e *ranking*. Na primeira etapa, o objetivo seria treinar o autodificador para reconstruir imagens da coleção alvo. A segunda etapa consistiria em treinar o modelo completo para aprendizagem de *ranking*, iniciando o autocodificador com os pesos aprendidos na primeira etapa;

Enquanto a primeira alternativa tem a vantagem de possibilitar uma comparação justa entre os diferentes métodos vistos na literatura, as demais fornecem meios mais simples de treino, dado o custo proibitivo da transferência completa.

Outras limitações descritas anteriormente podem ser endereçadas em pesquisas futuras, como o estudo de uma variante *listwise* do nosso método bem como uma avaliação por meio de uma aplicação real.

Considerando questões mais específicas dos nossos métodos, um aspecto a ser explorado em pesquisas futuras é o controle sobre a influência dos diferentes componentes das funções de custo. Em particular, a função de perda dos modelos propostos nesta pesquisa é formada por três componentes combinadas de forma linear. A primeira componente controla a perda associada à reconstrução das imagens da rede autocodificadora usando uma representação binária enquanto a segunda controla a perda relacionada à classificação dos rótulos e a terceira controla o aprendizado da ordem relativa entre os pares/trios de imagens. Para cada uma dessas componentes, são associados os hiperparâmetros λ, α e β que controlam suas influências. Nos experimentos realizados, foram definidos valores fixos para esses hiperparâmetros. Entretanto, esses pesos deveriam mudar de acordo com o aprendizado de cada componente, visto que eles

podem representar tarefas de complexidade variável ao longo do tempo. Por exemplo, para ser útil, o autocodificador deve ser capaz de reconstruir as imagens que recebe. Logo, no início do treino, este aspecto do aprendizado (controlado mais diretamente pelo hiperparâmetro λ) deveria ser reforçado em relação às outras tarefas. Contudo, uma vez que o modelo já é um reconstrutor de imagens competente, outros aspectos do problema, como a similaridade de classes e a ordem, deveriam ser enfatizados. O ideal é que este processo pudesse ser realizado de forma automática, com o modelo ajustando a sensibilidade de cada componente de acordo com sua contribuição para a perda global ao longo do treino.

Nesta tese, optamos por usar métodos baseados em aprendizagem de *ranking* para, entre outras coisas, possibilitar a busca por imagens com conteúdo cujos rótulos nunca foram vistos pelos modelos e lidar melhor com desbalanceamento de classes. Seria interessante, em um trabalho futuro, investigar melhor estes aspectos dos modelos propostos. Para o primeiro caso, seria interessante verificar como o modelo se comporta com imagens muito distintas daquelas nas coleções de treino. No segundo caso seria útil investigar como o método proposto se compara com métodos que usam técnicas especificamente projetadas para mitigar problemas de desbalanceamento, como *oversampling* e aumento de dados [Khan et al., 2018], [Buda et al., 2018].

As técnicas de busca multi-rótulo, discutidas nesta pesquisa, podem ser aplicadas em outros domínios, como busca de vídeo e recuperação de imagens baseada em objetos. De fato, embora a recuperação de imagens baseada em objetos seja diferente do problema abordado aqui, essencialmente porque requer a detecção de objetos no processo de construção do modelo e os rótulos não estão necessariamente relacionados a objetos [Rahmani et al., 2005], técnicas que aproveitam de vários rótulos ainda podem ser aplicados a imagens com vários objetos.

No domínio de vídeo, abordagens de múltiplos rótulos parecem ser uma escolha natural, considerando que temos mais objetos presentes em uma cena. Diferentemente do problema abordado nesta pesquisa, na busca de vídeo, o usuário está interessado em buscar uma cena, que neste caso é composta por um *streaming* de imagens e não por uma única imagem. Assim, o *hash* gerado deve representar essa sequência de imagens. Vale ressaltar que, para aprender a função de *hash* de um fluxo de imagem, poderíamos usar as estratégias discutidas nesta pesquisa para explorar os vários rótulos presentes.

Uma questão adicional que pode ser explorada por métodos de *hash* para recuperação de imagens é o uso de arquiteturas de redes neurais focadas em eficiência. A escolha de uma arquitetura eficiente tem impacto direto no tempo de busca devido ao processo de extração de características da imagem que implica em altos custos de processamento. Dentre as arquiteturas encontradas na literatura que focam no pro-

blema de eficiência de representação, não estudadas nesta tese, destacam-se as redes de atenção residual [Wang et al., 2017b]. Essas arquiteturas tentam produzir representações de qualidade para classificação de imagens de forma eficiente restringindo o processamento às áreas das imagens que provavelmente serão o foco da atenção dos usuários.

Por fim, vimos que abordagens não supervisionadas exploram pouco o domínio de rótulos múltiplos. Assim, as estratégias aqui apresentadas poderiam ser empregadas em modelos não supervisionados na descoberta de pseudo-rótulos.

Referências Bibliográficas

Alemi, A.; Poole, B.; Fischer, I.; Dillon, J.; Saurous, R. A. & Murphy, K. (2018). Fixing a broken elbo. Em *International Conference on Machine Learning*, pp. 159--168. PMLR.

Andoni, A. & Indyk, P. (2006). Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. Em *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pp. 459–468.

Benesty, J.; Chen, J.; Huang, Y. & Cohen, I. (2009). Pearson correlation coefficient. Em *Noise reduction in speech processing*, pp. 1--4. Springer.

Bengio, Y. et al. (2009). Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1--127.

Bengio, Y.; Léonard, N. & Courville, A. (2013). Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*.

Bezerra, E. (2016). Introdução à aprendizagem profunda. *XXXI Simposio Brasileiro de Banco de Dados*.

Bishop, C. (2006). Pattern recognition and machine learning (information science and statistics), 1st edn. 2006. corr. 2nd printing edn. *Springer, New York*.

Buda, M.; Maki, A. & Mazurowski, M. A. (2018). A systematic study of the class imbalance problem in convolutional neural networks. *Neural Networks*, 106:249--259.

Cakir, F.; He, K.; Bargal, S. A. & Sclaroff, S. (2018). Hashing with mutual information. *arXiv preprint arXiv:1803.00974*.

Canziani, A.; Paszke, A. & Culurciello, E. (2016). An analysis of deep neural network models for practical applications. cite arxiv:1605.07678Comment: 7 pages, 10 figures, legend for Figure 2 got lost :/.

Cao, Y.; Long, M.; Wang, J.; Zhu, H. & Wen, Q. (2016). Deep quantization network for efficient image retrieval. Em *AAAI*, pp. 3457--3463.

Cao, Z.; Long, M.; Wang, J. & Yu, P. (2017). Hashnet: Deep learning to hash by continuation. volume 2017-October, pp. 5609–5618. cited By 7.

Chen, Z.; Cai, R.; Lu, J.; Feng, J. & Zhou, J. (2018). Order-sensitive deep hashing for multimorbidity medical image retrieval. Em *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pp. 620--628. Springer.

Chua, T.-S.; Tang, J.; Hong, R.; Li, H.; Luo, Z. & Zheng, Y. (2009). Nus-wide: a real-world web image database from national university of singapore. Em *Proceedings of the ACM international conference on image and video retrieval*, p. 48. ACM.

Courbariaux, M.; Hubara, I.; Soudry, D.; El-Yaniv, R. & Bengio, Y. (2016). Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*.

Cruz, R.; Fernandes, K.; Cardoso, J. S. & Costa, J. F. P. (2016). Tackling class imbalance with ranking. Em *2016 International joint conference on neural networks (IJCNN)*, pp. 2182--2187. IEEE.

da S. Torres, R.; Falcão, A. X.; Zhang, B.; Fan, W.; Fox, E. A.; Gonçalves, M. A. & Calado, P. (2005). A new framework to combine descriptors for content-based image retrieval. Em *Proceedings of the 14th ACM International Conference on Information and Knowledge Management*, CIKM '05, pp. 335--336, New York, NY, USA. ACM.

da Silva Torres, R. & Falcao, A. X. (2006). Content-based image retrieval: Theory and applications. *RITA*, 13(2):161--185.

Dadaneh, S. Z.; Boluki, S.; Yin, M.; Zhou, M. & Qian, X. (2020). Pairwise supervised hashing with bernoulli variational auto-encoder and self-control gradient estimator. Em *Conference on Uncertainty in Artificial Intelligence*, pp. 540--549. PMLR.

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K. & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. Em *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248--255. Ieee.

Everingham, M.; Van Gool, L.; Williams, C. K.; Winn, J. & Zisserman, A. (2010). The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303--338.

Gong, Y.; Kumar, S.; Rowley, H. A. & Lazebnik, S. (2013a). Learning binary codes for high-dimensional data using bilinear projections. Em *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 484--491.

Gong, Y.; Kumar, S.; Verma, V. & Lazebnik, S. (2012). Angular quantization-based binary codes for fast similarity search. Em *Advances in neural information processing systems*, pp. 1196--1204.

Gong, Y.; Lazebnik, S.; Gordo, A. & Perronnin, F. (2013b). Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(12):2916--2929.

Goodfellow, I.; Bengio, Y. & Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Grubinger, M.; Clough, P.; Müller, H. & Deselaers, T. (2006). The iapr tc-12 benchmark: A new evaluation resource for visual information systems. Em *Int. Workshop OntoImage*, volume 5.

He, K.; Zhang, X.; Ren, S. & Sun, J. (2014). Spatial pyramid pooling in deep convolutional networks for visual recognition. Em *European conference on computer vision*, pp. 346--361. Springer.

Hijazi, S.; Kumar, R. & Rowen, C. (2015). Using convolutional neural networks for image recognition.

Huang, C.-Q.; Yang, S.-M.; Pan, Y. & Lai, H.-J. (2018). Object-location-aware hashing for multi-label image retrieval via automatic mask learning. *IEEE Transactions on Image Processing*, 27(9):4490--4502.

Huiskes, M. J. & Lew, M. S. (2008). The mir flickr retrieval evaluation. Em *Proceedings of the 1st ACM international conference on Multimedia information retrieval*, pp. 39--43. ACM.

Jain, P.; Kulis, B. & Grauman, K. (2008). Fast image search for learned metrics.

Jang, E.; Gu, S. & Poole, B. (2016). Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

Jiang, Q.-Y. & Li, W.-J. (2016). Deep cross-modal hashing. *CoRR*.

Khan, S. H.; Hayat, M.; Bennamoun, M.; Sohel, F. A. & Togneri, R. (2018). Cost-sensitive learning of deep feature representations from imbalanced data. *IEEE transactions on neural networks and learning systems*, 29(8):3573--3587.

Kingma, D. P. & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Krähenbühl, P. & Koltun, V. (2014). Geodesic object proposals. Em *European conference on computer vision*, pp. 725--739. Springer.

Krizhevsky, A.; Sutskever, I. & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. Em *Advances in neural information processing systems*, pp. 1097--1105.

Kulis, B. & Grauman, K. (2009). Kernelized locality-sensitive hashing for scalable image search. Em *Computer Vision, 2009 IEEE 12th International Conference on*, pp. 2130--2137. IEEE.

Kulis, B.; Jain, P. & Grauman, K. (2009). Fast similarity search for learned metrics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(12):2143--2157.

Lai, H.; Yan, P.; Shu, X.; Wei, Y. & Yan, S. (2016). Instance-aware hashing for multi-label image retrieval. *IEEE Transactions on Image Processing*, 25(6):2469--2479.

Li, T.; Gao, S. & Xu, Y. (2017). Deep multi-similarity hashing for multi-label image retrieval. Em *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pp. 2159--2162. ACM.

Li, W.-J.; Wang, S. & Kang, W.-C. (2015). Feature learning based deep supervised hashing with pairwise labels. *arXiv preprint arXiv:1511.03855*.

Liang, D.; Yan, K.; Wang, Y.; Zeng, W.; Yuan, Q.; Bao, X. & Tian, Y. (2017). Deep hashing with multi-task learning for large-scale instance-level vehicle search. Em *Multimedia & Expo Workshops (ICMEW), 2017 IEEE International Conference on*, pp. 192--197. IEEE.

Lin, G.; Shen, C.; Suter, D. & Van Den Hengel, A. (2013). A general two-step approach to learning-based hashing. Em *Proceedings of the IEEE international conference on computer vision*, pp. 2552--2559.

Lin, T.-Y.; Maire, M.; Belongie, S.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P. & Zitnick, C. L. (2014). Microsoft coco: Common objects in context. Em *European conference on computer vision*, pp. 740--755. Springer.

Liu, H.; Wang, R.; Shan, S. & Chen, X. (2016). Deep supervised hashing for fast image retrieval. Em *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2064--2072.

Liu, L. & Qi, H. (2018). Discriminative cross-view binary representation learning. Em *Applications of Computer Vision (WACV), 2018 IEEE Winter Conference on*, pp. 1736--1744. IEEE.

Liu, L.; Rahimpour, A.; Taalimi, A. & Qi, H. (2017). End-to-end binary representation learning via direct binary embedding. Em *Image Processing (ICIP), 2017 IEEE International Conference on*, pp. 1257--1261. IEEE.

Lu, J.; Liong, V. E. & Zhou, J. (2017). Deep hashing for scalable image search. *IEEE Transactions on Image Processing*, 26(5):2352--2367.

Lu, J.; Liong, V. E.; Zhou, X. & Zhou, J. (2015). Learning compact binary face descriptor for face recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(10):2041--2056.

Ma, C.; Chen, Z.; Lu, J. & Zhou, J. (2018). Rank-consistency multi-label deep hashing. Em *2018 IEEE International Conference on Multimedia and Expo (ICME)*, pp. 1--6. IEEE.

Menon, A. K. & Elkan, C. (2011). Link prediction via matrix factorization. Em *Joint european conference on machine learning and knowledge discovery in databases*, pp. 437--452. Springer.

Norouzi, M. & Blei, D. M. (2011). Minimal loss hashing for compact binary codes. Em *Proceedings of the 28th international conference on machine learning (ICML-11)*, pp. 353--360. Citeseer.

Pennington, J.; Socher, R. & Manning, C. (2014). Glove: Global vectors for word representation. Em *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532--1543.

Raginsky, M. & Lazebnik, S. (2009). Locality-sensitive binary codes from shift-invariant kernels. Em *Advances in neural information processing systems*, pp. 1509--1517.

Rahmani, R.; Goldman, S. A.; Zhang, H.; Krettek, J. & Fritts, J. E. (2005). Localized content based image retrieval. Em *Proceedings of the 7th ACM SIGMM international workshop on Multimedia information retrieval*, pp. 227--236.

Rodrigues, J.; Cristo, M. & Colonna, J. G. (2020). Deep hashing for multi-label image retrieval: a survey. *Artificial Intelligence Review*, 53(7):5261--5307.

Rosenberg, A. & Hirschberg, J. (2007). V-measure: A conditional entropy-based external cluster evaluation measure. Em *Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL)*, pp. 410--420.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

Rumelhart, D. E.; McClelland, J. L.; Group, P. R. et al. (1986). Parallel distributed processing: Explorations in the microstructures of cognition. volume 1: Foundations.

Santos, J. M. d. et al. (2016). Descritores de imagens baseados em assinatura textual.

Schroff, F.; Kalenichenko, D. & Philbin, J. (2015). Facenet: A unified embedding for face recognition and clustering. Em *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 815--823.

Shen, F.; Gao, X.; Liu, L.; Yang, Y. & Shen, H. (2017). Deep asymmetric pairwise hashing. pp. 1522–1530. cited By 0.

Silveira, D.; da Costa Carvalho, A. L.; Cristo, M. & Moens, M. (2018). Topic modeling using variational auto-encoders with gumbel-softmax and logistic-normal mixture distributions. Em *2018 International Joint Conference on Neural Networks, IJCNN 2018, Rio de Janeiro, Brazil, July 8-13, 2018*, pp. 1--8. IEEE.

Simonyan, K. & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. *ICLR*.

Song, G. & Tan, X. (2018). Learning multilevel semantic similarity for large-scale multi-label image retrieval. Em *Proceedings of the 2018 ACM on International Conference on Multimedia Retrieval*, pp. 64--72. ACM.

Stutz, D. (2014). Understanding convolutional neural networks. Em *Seminar Report, Fakultät für Mathematik, Informatik und Naturwissenschaften Lehr-und Forschungsgebiet Informatik VIII Computer Vision*.

Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V. & Rabinovich, A. (2015). Going deeper with convolutions. Em *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1--9.

Szegedy, C.; Vanhoucke, V.; Ioffe, S.; Shlens, J. & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. Em *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pp. 2818--2826.

Wang, D.; Huang, H.; Lin, H.-K. & Mao, X.-L. (2017a). Supervised hashing for multi-labeled data with order-preserving feature. Em *Chinese National Conference on Social Media Processing*, pp. 16--28. Springer.

Wang, F.; Jiang, M.; Qian, C.; Yang, S.; Li, C.; Zhang, H.; Wang, X. & Tang, X. (2017b). Residual attention network for image classification. Em *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6450–6458.

Wang, J.; Kumar, S. & Chang, S.-F. (2012). Semi-supervised hashing for large-scale search. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (12):2393--2406.

Wang, J.; Liu, W.; Kumar, S. & Chang, S.-F. (2016a). Learning to hash for indexing big data—a survey. *Proceedings of the IEEE*, 104(1):34--57.

Wang, J.; Shen, H. T.; Song, J. & Ji, J. (2014). Hashing for similarity search: A survey. *arXiv preprint arXiv:1408.2927*.

Wang, J.; Zhang, T.; j. song; Sebe, N. & Shen, H. T. (2018). A survey on learning to hash. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4):769–790.

Wang, X.; Shi, Y. & Kitani, K. M. (2016b). Deep supervised hashing with triplet labels. Em *Asian Conference on Computer Vision*, pp. 70--84. Springer.

Weiss, Y.; Torralba, A. & Fergus, R. (2009). Spectral hashing. Em *Advances in neural information processing systems*, pp. 1753--1760.

Wu, D.; Lin, Z.; Li, B.; Liu, J. & Wang, W. (2018). Deep uniqueness-aware hashing for fine-grained multi-label image retrieval. Em *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1683--1687. IEEE.

Wu, D.; Lin, Z.; Li, B.; Ye, M. & Wang, W. (2017). Deep supervised hashing for multi-label and large-scale image retrieval. Em *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval*, pp. 150--158. ACM.

Xu, J.; Wang, P.; Tian, G.; Xu, B.; Zhao, J.; Wang, F. & Hao, H. (2015). Convolutional neural networks for text hashing. Em *IJCAI*, pp. 1369--1375.

Yang, H.; Lin, K. & Chen, C. (2018). Supervised learning of semantics-preserving hash via deep convolutional neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(2):437–451.

Yin, M. & Zhou, M. (2019). Arm: Augment-reinforce-merge gradient for stochastic binary networks. *International Conference on Learning Representations*.

Zhang, Z.; Zou, Q.; Lin, Y.; Chen, L. & Wang, S. (2019). Improved deep hashing with soft pairwise similarity for multi-label image retrieval. *IEEE Transactions on Multimedia*, 22(2):540--553.

Zhang, Z.; Zou, Q.; Wang, Q.; Lin, Y. & Li, Q. (2018). Instance similarity deep hashing for multi-label image retrieval. *arXiv preprint arXiv:1803.02987*.

Zhao, F.; Huang, Y.; Wang, L. & Tan, T. (2015). Deep semantic ranking based hashing for multi-label image retrieval. Em *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1556--1564.

Zhong, C.; Yu, Y.; Tang, S.; Satoh, S. & Xing, K. (2017). Deep multi-label hashing for large-scale visual search based on semantic graph. Em *Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint Conference on Web and Big Data*, pp. 169--184. Springer.

Zhu, H.; Long, M.; Wang, J. & Cao, Y. (2016). Deep hashing network for efficient similarity retrieval. Em *AAAI*, pp. 2415--2421.

Zhuang, B.; Lin, G.; Shen, C. & Reid, I. (2016). Fast training of triplet-based deep binary embedding networks. Em *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5955--5964.