Constellation Loss em Modelos de Reconhecimento Facial

Erick Ramos dos Santos [*] **Patrick Marques Ciarelli** [*]
Jorge Leonid Aching Samatelo [*]

[*] *Departamento de Engenharia Elétrica, Universidade Federal do Espírito Santo, ES, (e-mails: erick.r.santos@edu.ufes.br; patrick.ciarelli@ufes.br; jorge.samatelo@ufes.br)*

Abstract: Feature extracting models for facial recognition systems have become objects of in-depth study over the past few years. In order to find a better discrimination of objects and, consequently, a better separability of classes, loss functions, such as Triplet Loss, were designed to be used in conjunction with Siamese Neural Networks. However, networks with these functions suffer from a slow convergence by considering only two classes (positive and negative) at each learning iteration. Recently, a loss function called Constellation Loss was proposed in order to minimize these problems. In this work, a model for facial recognition using a CNN as a backbone and Constellation Loss as a loss function is proposed. To validate the model, two public databases were used and comparisons were made with different loss functions and CNNs architectures. It is also proposed in this work the use of an approach for the construction of batches, which allows network training with a reduced memory usage. The results obtained indicate that Constellation Loss is a promising technique when compared to the other loss functions evaluated, reaching average values of AUC (Area Under The Curve) equal to 99.9% in the Olivetti Faces dataset and 98.7% in the challenging Labeled Faces in the Wild (LFW) dataset. The effectiveness of the method could be certified, enabling its application to facial recognition systems.

Resumo: Modelos extratores de características para sistemas de reconhecimento facial se tornaram objetos de profundo estudo ao longo dos últimos anos. Desejando encontrar uma melhor discriminação dos objetos e, consequentemente, uma melhor separabilidade das classes, funções de perdas, como a *Triplet Loss*, foram projetadas para serem empregadas em conjunto com as Redes Neurais Siamesas. Entretanto, redes com essas funções sofrem com uma lenta convergência por considerar apenas duas classes (positiva e negativa) a cada iteração de aprendizagem. Recentemente, uma função de perda chamada *Constellation Loss* foi proposta no sentido de minimizar estes problemas. Neste trabalho é proposto um modelo para reconhecimento facial usando uma CNN como *backbone* e o *Constellation Loss* como função de perda. Para validar o modelo, foram utilizadas duas bases de dados públicas e feitas comparações com diferentes funções de perda e arquiteturas CNN. Também é proposto neste trabalho o uso de uma abordagem para construção dos *batches*, o qual permite o treinamento da rede com uso reduzido de memória. Os resultados obtidos indicam que *Constellation Loss* é uma técnica promissora quando comparadas às demais funções de perda avaliadas, alcançando valores médios de AUC (*Area Under The Curve*) iguais a 99,9% no conjunto de dados *Olivetti Faces* e 98,7% no desafiador conjunto de dados *Labeled Faces in the Wild* (LFW). A efetividade do método pôde ser certificada, viabilizando sua aplicação para sistemas de reconhecimento facial.

Keywords: Constellation Loss; Facial Recognition; Similarity Metrics; Siamese Neural Networks; Deep metric learning.

Palavras-chaves: Constellation Loss; Reconhecimento Facial; Métrica de similaridade; Redes Neurais Siamesas; Aprendizagem profunda de métricas.

1. INTRODUÇÃO

O reconhecimento facial é uma das maneiras mais utilizadas pelos seres humanos para identificação de indivíduos. Com isso, estudos na área são feitos há anos buscando alcançar um sistema completamente automatizado de reconhecimento facial utilizando visão computacional. Sistemas de reconhecimento facial se iniciam pela detecção da face, seguida pela extração das características e, por fim, a etapa de classificação, que visa verificar se uma face específica corresponde a alguma disponível em um banco de imagens de faces. Logo, ao desenvolver um projeto na área, as escolhas do detector, do extrator de características e do classificador precisam ser feitas de forma bem estruturadas, pois afetarão diretamente nos resultados do projeto.

Alguns trabalhos obtiveram bons resultados na área antes da virada do século, como Goldstein et al. (1971), Cottrell

e Fleming (1990), Turk e Pentland (1991a), Turk e Pentland (1991b) e Bromley et al. (1993), onde foram introduzidas as revolucionárias Redes Neurais Siamesas (RNS). Apresentado por Chopra et al. (2005), o *Constrastive Loss* tornou-se uma função de perda natural em modelos que utilizam o conceito de RNS no treinamento, calculando o valor de perda em torno de pares de entradas de uma mesma classe ou de classes diferentes.

Entretanto, foram as Redes Neurais Convolucionais (*Convolutional Neural Network* - CNN) que revolucionaram os modelos de reconhecimento facial. Ao contrário dos algoritmos propostos até então, que utilizavam características projetadas manualmente, as CNN's trouxeram uma nova abordagem: uma rede neural que extrai as suas próprias características, e ela mesma define quais são as mais relevantes. De forma geral, técnicas de *Deep Learning* determinam de forma autônoma as regiões da face com maior poder de discriminação dos indivíduos.

Com a popularização das CNN na classificação de imagens, o estudo em torno da aprendizagem profunda de métricas se tornou comum. O trabalho proposto por Schroff et al. (2015) foi inovador para a área, trazendo um novo conceito aos algoritmos de extração de características. Nele, os pesquisadores configuraram um modelo utilizando CNN, comparando as arquiteturas Inception (Szegedy et al., 2014) e de Zeiler e Fergus (2013), para encontrar as 128 características mais importantes de uma face. Estas características de cada face são direcionadas a uma função de perda chamada *Triplet Loss*. O conceito da função é uma extensão do *Contrastive Loss*, porém, em vez de duas, são utilizadas três imagens para determinação das distâncias euclidianas entre as mesmas. A primeira imagem é chamada de âncora e as duas imagens subsequentes são chamadas de positiva e negativa, ou seja, a imagem positiva trata-se da mesma pessoa presente na imagem âncora e a negativa, não. O objetivo do uso da função é minimizar a distância da âncora para a imagem positiva, enquanto maximiza a distância para a imagem negativa, tornando a rede capaz de aumentar as distâncias de uma imagem para as diferentes classes presentes no conjunto de dados de treinamento, enquanto minimiza a distância intraclasse. Entretanto, a convergência deste tipo de metodologia é lenta em casos de conjuntos de dados com uma grande quantidade de classes, visto que a cada atualização a função verifica apenas duas classes.

Diante de tal cenário, Sohn (2016) propôs a função de perda *Multiclass-N-Pair Loss*. O método consiste em uma generalização do *Triplet Loss*, onde a comparação da âncora é feita com todas as classes presentes no conjunto de dados, ou seja, a cada iteração, ao minimizar a distância do exemplo positivo, a distância de $N-1$ classes são maximizadas. Porém, mesmo alcançando uma convergência mais rápida, a perda de N pares persistiu com o problema de maximizar a distância das classes apenas em relação à classe da âncora.

Na busca de aproveitar o melhor dessas funções de perda, Medela e Picón (2019) propuseram a *Constellation Loss*. Agora, a cada iteração de aprendizagem, as "distâncias negativas" são encontradas não apenas em relação à âncora, mas entre as próprias imagens das classes negativas. A

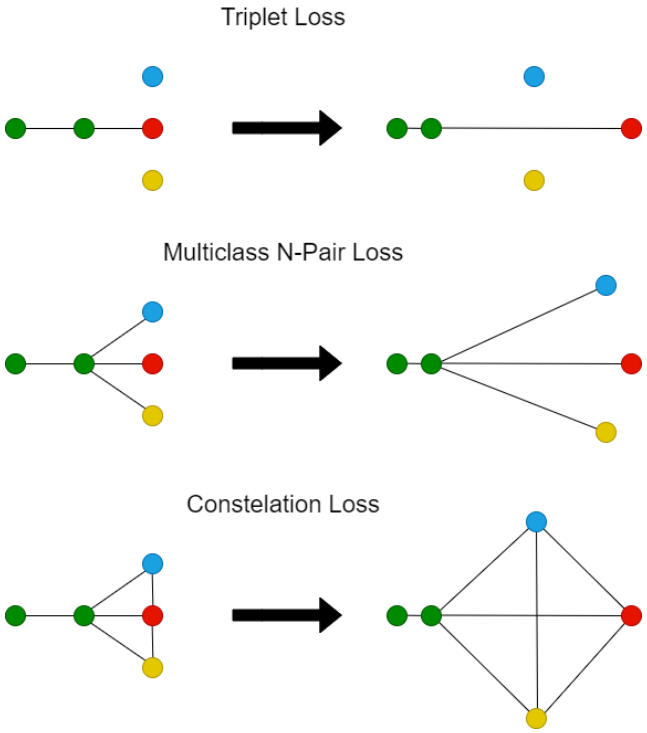

Figura 1. Diagrama Visual das Funções de Perda. Cada cor representa uma classe. Fonte: Medela e Picón (2019)

Figura 1 ilustra, de forma simplificada, o procedimento das três funções de perdas mencionadas.

Baseando-se na literatura descrita, o presente trabalho propõe um modelo extrator de características para reconhecimento facial usando uma CNN como *backbone* e o *Constellation Loss* como função de perda. Para validar o modelo, serão utilizadas duas bases de dados públicas e serão feitas comparações com diferentes funções de perda e arquiteturas CNN. Cabe indicar que, para superar problemas de escalabilidade do modelo, é apresentada uma metodologia para a construção dos *batches* de treinamento, que mostra ser uma abordagem apropriada para a arquitetura neural usada.

Este trabalho está dividido da seguinte forma: a Seção 2 descreve brevemente as RNS, enquanto a Seção 3 apresenta as funções de perdas usadas nelas. A seguir, a Seção 4 traz a proposta do trabalho, abordando a estrutura do modelo, assim como a metodologia de construção dos *batches* de treinamento. Os resultados dos testes são apresentados na Seção 5, juntamente com discussões sobre os mesmos. Por fim, são feitas as considerações finais na Seção 6.

2. REDES NEURAIS SIAMESAS

Introduzidas pela primeira vez por Bromley et al. (1993), as Redes Neurais Siamesas (RNS) foram empregadas em um sistema de verificação de assinaturas. Uma RNS é uma classe de arquiteturas de rede neural que contém duas ou mais sub-redes com mesma configuração, mesmos parâmetros e mesmos pesos. A atualização de parâmetros é espelhada em ambas as sub-redes, sendo usada para encontrar o grau de similaridade das entradas comparando seus vetores de características (que são extraídas pela rede).

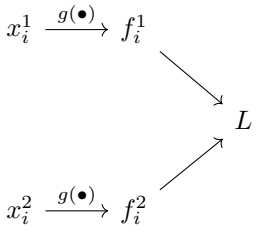

Figura 2. Estrutura da RNS ao aplicar *Contrastive Loss*

Tradicionalmente, uma rede neural aprende a prever várias classes. Isso representa um problema quando é necessário adicionar e/ou remover novas classes aos dados. Nesse caso, a rede neural tem que ser atualizada e treinada novamente com todo o conjunto de dados. A RNS, por outro lado, aprende uma função de similaridade. Assim, pode-se treiná-la verificando se duas imagens pertencem à mesma classe ou não, possibilitando classificar novas classes de dados sem treinar a rede toda novamente.

Assim, uma RNS recebe como entrada um par de entradas com o objetivo de desenvolver similaridade entre pares de uma mesma classe, e distanciar pares de diferentes classes. Com o treinamento, o modelo cria um espaço multi-dimensional igual ao número de neurônios de saída da rede neural base. A representação espacial de cada entrada então é submetida a uma função de similaridade (ou distância), normalmente sendo a distância euclidiana. Uma vez obtida a medida de similaridade, é definido se os dados de entrada pertencem ou não à uma mesma classe dado um limiar definido.

Sobre essa medida de similaridade (ou distância), são aplicadas as funções de perda, as quais são as responsáveis por informar à rede como deve ser feito o aprendizado.

3. FUNÇÕES DE PERDA

Com o sucesso da metodologia implementada por Schroff et al. (2015), os estudos se intensificaram em torno da aprendizagem de métricas de distância utilizando os *embeddings* gerados pelas CNN. Atualmente, as funções de perda baseadas na distância euclidiana possuem resultados expressivos e, consequentemente, tornaram-se comuns. Nesta seção, são detalhadas as técnicas consideradas estado da arte nesta área.

3.1 Contrastive Loss

No caso da *Contrastive Loss* a estrutura da RNS vem definido pelo diagrama ilustrado na Figura 2, onde o par de imagens de entrada (x_i^1, x_i^2), são transformados, via uma rede CNN $g(\bullet)$, nos *embeddings* (f_i^1, f_i^2). O *Contrastive Loss* calcula o somatório das perdas individuais dos pares positivos (dados de mesma classe) e pares negativos (dados de classes distintas). Um par positivo i possui um rótulo $y_i = 1$, enquanto um par negativo recebe rótulo $y_i = 0$. A equação desta função é dada por (1).

$$L = \frac{1}{2B}\sum_{i=1}^{B}\{(1-y_i)||f_i^1 - f_i^2||_2^2 + (y_i)[max(m - ||f_i^1 - f_i^2||_2^2, 0)]^2\} \quad (1)$$

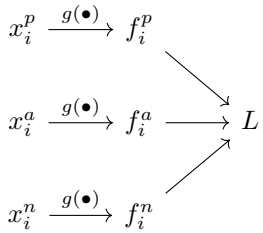

Figura 3. Estrutura da RNS ao aplicar *Triplet Loss*

onde m é a margem, normalmente setada como 1 e B é o tamanho do *batch* utilizado para o treinamento da rede.

3.2 Triplet Loss

Desde sua primeira citação, o *Triplet Loss* tornou-se a função de perda mais popular na área do aprendizado métrico. O método alcançou bons resultados em diversos trabalhos, principalmente ao usar esta perda no treinamento de RNS, podendo ser visto em (Weinberger e Saul, 2009). No caso do *Triplet Loss* a estrutura da RNS é definida pela Figura 3.

Expandindo a metodologia do *Contrastive Loss*, o *Triplet Loss* recebe três imagens de entrada (x_i^a, x_i^p, x_i^n), sendo a âncora x_i^a, uma imagem de uma das classes do conjunto de dados de treinamento, x_i^p uma imagem da mesma classe da âncora chamada de positivo e a imagem x_i^n chamada de negativo, que é um exemplo de uma classe diferente da âncora. O método tem como finalidade maximizar a distância da âncora em relação ao negativo, ao passo que minimiza a distância para o positivo, tornando o modelo extrator de características capaz de gerar uma separação melhor dos *embeddings* de classes distintas.

Deste modo, a *Triplet Loss* recebe um trio de *embeddings* (f_i^a, f_i^p, f_i^n), conformados pelos *embeddings* da âncora (f_i^a), da amostra positiva (f_i^p) e da amostra negativa (f_i^n). A equação desta função é dada em (2).

$$L = \frac{1}{B}\sum_{i=1}^{B} max(||f_i^a - f_i^p||_2^2 - ||f_i^a - f_i^n||_2^2 + \alpha, 0) \quad (2)$$

onde B é o tamanho do *batch* de treinamento e α é uma constante chamada de margem de distância, utilizada para especificar quando um trio se tornou muito "fácil" e não é desejado mais ajustar os pesos dele.

Com base nessa definição, existem três categorias de *triplets*:

- *Triplets* fáceis: *triplets* que têm uma perda igual a 0, pois $(||f_i^a - f_i^p||_2^2 + \alpha) < ||f_i^a - f_i^n||_2^2$
- *Triplets* difíceis: *triplets* onde o negativo está mais próximo que o positivo, ou seja, $||f_i^a - f_i^p||_2^2 > ||f_i^a - f_i^n||_2^2$
- *Triplets* semi-difíceis: *triplets* onde o negativo não está mais próximo que o positivo, mas ainda possui uma perda positiva, ou seja, $||f_i^a - f_i^p||_2^2 < ||f_i^a - f_i^n||_2^2 < (||f_i^a - f_i^p||_2^2 + \alpha)$

A Figura 4 ilustra as distâncias da âncora para o negativo, conforme as categorias abordadas acima.

Figura 4. Regiões de distanciamento entre a âncora e cada categoria de negativo

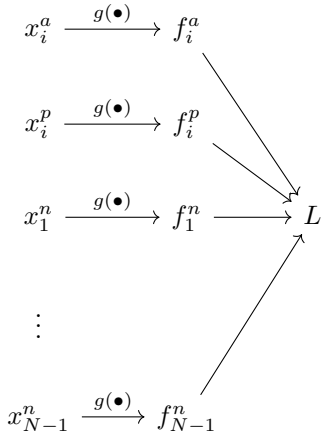

Figura 5. Estrutura da RNS ao aplicar *Multiclass-N-Pair Loss*

3.3 Multiclass-N-Pair Loss

Surgindo como uma extensão do *Triplet Loss* e com o foco de vencer a lenta convergência a cada passo de aprendizagem, em (Sohn, 2016), a função de perda *Multiclass-N-Pair Loss* utiliza o conceito de comparação com mais negativos, mais precisamente, $N-1$ negativos, sendo N o número de classes presentes no conjunto de dados de treinamento. Ou seja, a cada iteração de aprendizagem, o cálculo da perda se baseia nos *embeddings* da âncora (x_i^a), do exemplo positivo (x_i^p) e de um exemplo negativo de cada uma das demais classes (x_j^n, $j = \{1, \cdots, N-1\}$), ou seja, deve-se assegurar que $x_i^p \neq x_j^n$. Nesse contexto, a arquitetura da RNS toma a forma descrita na Figura 5.

Então, o *Multiclass-N-Pair Loss* recebe a N-tupla de *embeddings* aplicando a equação descrita a seguir:

$$L = \frac{1}{B}\sum_{i=1}^{B}\log\left(1+\sum_{j=1}^{N-1}\exp\left(f_i^{aT}f_j^n - f_i^{aT}f_i^p\right)\right) \qquad (3)$$

onde B é o tamanho do *batch* de treinamento, que no caso do *Multiclass-N-Pair Loss* é o número de diferentes classes presentes no conjunto de dados de treinamento.

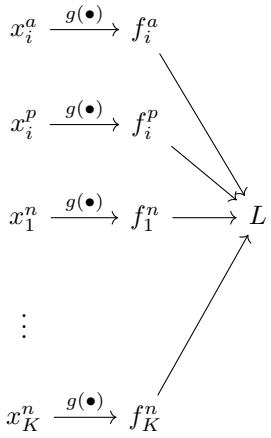

Figura 6. Estrutura da RNS ao aplicar a *Constellation Loss*

3.4 Constellation Loss

Unindo a mesma construção de *batches* de treinamento da *Triplet Loss* e uma formulação de perda semelhante à da *Multiclass-N-Pair Loss*, a função de perda *Constellation Loss* busca aproveitar o que tem de melhor nos conceitos dessas funções, como é descrito por Medela e Picón (2019). O diagrama ilustrado na Figura 6 descreve a arquitetura RNS do *Constellation Loss*, podendo notar a semelhança com a arquitetura usada para *Multiclass-N-Pair Loss*.

A principal diferença do *Constellation Loss* é que enquanto o *Multiclass-N-Pair Loss* subtrai produtos escalares de pares da mesma classe, o *Constellation Loss* faz algo semelhante ao *Triplet Loss*, onde a perda é calculada pela subtração do produto escalar dos *embeddings* de uma âncora e *embeddings* negativos; e o produto escalar dos *embeddings* de uma âncora e *embeddings* positivos. A fórmula é retratada abaixo, podendo notar sua similaridade com a Equação (3):

$$L = \frac{1}{B}\sum_{i=1}^{B}\log\left(1+\sum_{j=1}^{K}\exp\left(f_i^{aT}f_j^n - f_i^{aT}f_i^p\right)\right) \qquad (4)$$

onde K é a quantidade de *triplets* que se quer incorporar na função de perda a cada iteração.

Ao passo que a *Triplet Loss* aproxima o exemplo positivo e afasta apenas uma classe negativa, a *Multiclass-N-Pair Loss* afasta todas as classes negativas. A *Constellation Loss* tem como foco o afastamento dos negativos não apenas em relação à âncora, mas entre os próprios negativos. Visando não utilizar um alto valor no parâmetro K, devido ao significante aumento do custo computacional no treinamento do modelo, os exemplos presentes nos *batches* de treinamento são escolhidos de forma aleatória, podendo haver diversos valores negativos distintos. Assim, não existe a necessidade de um valor elevado para K para alcançar uma melhora em relação às funções de perda *Triplet Loss* e *Multiclass-N-Pair Loss*.

Devido à natureza da função considerar a relação entre todas as amostras, não se faz necessário a aplicação de máscaras para encontrar *triplets* difíceis e semi-difíceis.

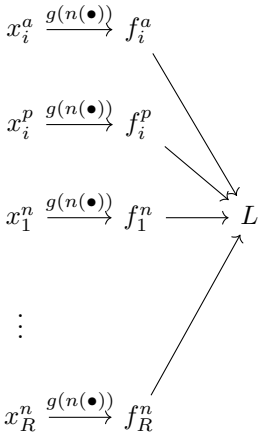

Figura 7. Estrutura da RNS implementada.

Apenas precisa calcular os produtos escalares como a função *Triplet Loss* faz, e aplicar uma máscara para encontrar os produtos escalares entre incorporações da mesma classe (âncora-positiva) e os produtos escalares entre incorporações de classes distintas (âncora-negativa).

4. MÉTODO PROPOSTO

A arquitetura proposta neste trabalho consiste de uma RNS que usa como *backbone* CNN pré-treinadas, as quais geram os *embeddings* relacionados a cada face. Na etapa de treinamento foi usada a função *Constellation Loss*, e uma metodologia particular para a conformação dos *batches* de treinamento. Tanto a metodologia de adequação das redes *backbone* CNN como a forma da composição dos *batches* serão explicados a seguir.

4.1 Backbone

No presente trabalho foram utilizadas redes CNN conceituadas na área de classificação de imagens como a *ResNet152V2* (He et al., 2016) e a *InceptionV3* (Szegedy et al., 2015). A rede *ResNet152V2* foi escolhida tendo em vista seu desempenho no desafio ImageNet, enquanto que a *InceptionV3* foi a rede utilizada em (Medela e Picón, 2019), obtendo resultados significativos na classificação de imagens ao combinar com a função de perda *Constellation Loss*.

Ao remover a última camada de cada CNN para que a arquitetura se adapte à classificação do conjunto de dados usado para treino, foi adicionada uma camada média comum de *pooling*, como sugere Schroff et al. (2015), seguida de uma camada com 128 unidades de saída, com função de ativação sigmoide. No intuito de manter o processo mais estável e mais apto para ser definida a margem entre âncoras e positivos, quando utilizadas as funções de perda *Triplet Loss* e *Constellation Loss*, os *embeddings* de saída da rede foram normalizados, via norma $L2$. Nesse contexto, o diagrama da arquitetura RNS implementada possui a forma descrita na Figura 7.

Neste caso, $g(\bullet)$ representa a CNN pré-treinada com as camadas adicionadas, $n(\bullet)$ representa a operação de normalização $L2$. Agora, $R = 1$ para a *Triplet Loss*, $R = N - 1$ para a *Multiclass-N-Pair Loss* e $R = K$ para a *Constellation Loss*.

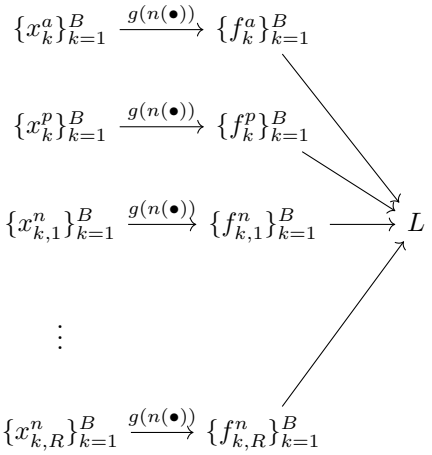

Figura 8. Aplicação de um *batch* com B tuplas, considerando que $B > 1$, na estrutura de RNS implementada.

4.2 Metodologia para Construção dos Batches

Para o caso de uma arquitetura RNS, o processo de aprendizado será resumido a seguir. Seja o conjunto de treinamento $\mathcal{T} = \{(x_i^a, x_i^p, \{x_j^n\}_{j=1}^R)\}_{i=1}^T$, conformado por T tuplas. Então, usando $\mathcal{T}$, a função de perda L, a quantidade definida de exemplos negativos R e o algoritmo de optimização SGD (*Stocastic Gradient Descent*), os parâmetros da rede $g(\bullet)$ são ajustados a partir de seus valores de pré-treinamento.

Nesse contexto, em cada iteração do SGD, as tuplas do conjunto de treinamento são passados à RNS em *batches*, ou seja, $\mathcal{T}$ é dividido em sub-conjuntos, cada um contendo B tuplas. Se cada *batch* contém uma única tupla ($B = 1$), o processo de treinamento é definido como um aprendizado *online* e se cada *batch* contém mais de uma tupla ($B > 1$), é definido como um aprendizado *offline*.

Cada tipo de aprendizado tem seus benefícios e desvantagens. Por exemplo, o aprendizado *online* permite efetuar um treinamento eficiente de um modelo neural, mas fica dependente da natureza estocástica do algoritmo SGD, enquanto que o aprendizado *offline*, ao considerar que cada *batch* contém várias tuplas, fica menos dependente de estocasticidade do SGD, mas requer que os modelos de aprendizado como as funções de perda sejam implementadas de forma vetorizada, para que, em uma única iteração do SGD se possa processar todo o *batch*, supondo que o processamento computacional será feito em uma GPU (*Graphics Processing Unit*). Então, aqui surge um inconveniente no treinamento de uma RNS, já que, tal arquitetura é uma replicação de uma única estrutura vetorizada, representada pela operação $x \xrightarrow{g(n(\bullet))} f$ da RNS mostrada no diagrama 7. Por exemplo, se for considerado o *batch* $\mathcal{X} = \{(x_k^a, x_k^p, \{x_{k,j}^n\}_{j=1}^R)\}_{k=1}^B$ constituído de B tuplas, ele será aplicado de forma vetorizada à RNS tal como ilustrado na Figura 8.

Então, ao usar um aprendizado *offline* será necessário aplicar os subconjuntos de exemplos $\{x_k^a\}_{k=1}^B$, $\{x_k^p\}_{k=1}^B$ e $\{\{x_{k,j}^n\}_{k=1}^B\}_{j=1}^R$ em cada uma das operações vetorizadas que conformam uma RNS, o que gera um problema de

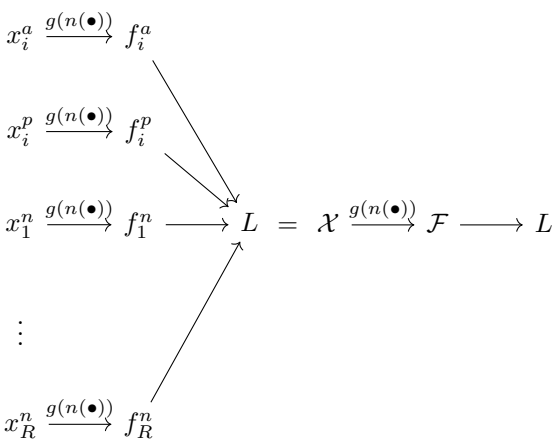

Figura 9. Relação de equivalência entre a estrutura de RNS implementada e uma única operação vetorizada, quando é aplicado um *batch* que contém só uma tupla ($B = 1$).

escalabilidade do modelo, considerando que o R é um hiperparâmetro do mesmo.

No caso de usar aprendizado *online*, o *batch* contém uma única tupla, ou seja, $\mathcal{X} = \{(x_1^a, x_1^p, \{x_{1,j}^n\}_{j=1}^R)\}$, implicando que unicamente se tenha que aplicar um único exemplo em cada operação vetorizada da RNS, o que permite a equivalência mostrada na Figura 9, onde $\mathcal{F} = \{(f_1^a, f_1^p, \{f_{1,j}^n\}_{j=1}^R)\}$ é o *batch* dos *embeddings* que serão usados pela função de perda escolhida.

Tomando em conta a equivalência indicada na Figura 9, é possível efetuar um aprendizado *online* com vários *batches*, mas para isso deve-se construir um único *batch* que contenha todos os *batches* a usar. Nesse contexto, aqui é definida a operação $concat(\bullet)$, que efetua a concatenação dos *batches* respeitando a ordem que a função de perda espera em relação aos *embeddings*. Matematicamente:

$$\begin{aligned}\mathcal{X}_{ct} &= concat(\{(x_k^a, x_k^p, \{x_{k,j}^n\}_{j=1}^R)\}_{k=1}^B) \\ &= \{(x_1^a, x_1^p, \{x_{1,j}^n\}_{j=1}^R, \cdots, x_B^a, x_B^p, \{x_{B,j}^n\}_{j=1}^R)\},\end{aligned}$$

Então, ao aplicar o $\mathcal{X}_{ct}$ à RNS, devido à relação de equivalência indicada na Figura 9, a tupla de *embeddings* calculada será: $\mathcal{F}_{ct} = \{(f_1^a, f_1^p, \{f_{1,j}^n\}_{j=1}^R, \cdots, f_B^a, f_B^p, \{f_{B,j}^n\}_{j=1}^R)\}$, que ao ser aplicada a L, deverá retornar a perda por *batch*, como indicado nas Equações (2), (3) e (4).

Com essa possibilidade de manipulação dos dados a cada *batch* disponibilizado para a rede, pode-se contornar o problema da escalabilidade das RNS quando operam com funções de perda que trabalham com três ou mais valores de entrada, como os métodos validados neste trabalho. Tal metodologia, inspirada em (Medela e Picón, 2019), foi aplicada no presente trabalho.

5. RESULTADOS E DISCUSSÕES

5.1 Conjuntos de Dados

No intuito de verificar o comportamento do modelo diante da classificação de faces, foram utilizados dois conjuntos de dados com uma importante característica que os diferem: enquanto o conjunto *Olivetti Faces* possui o mesmo número de faces por pessoa, o conjunto LFW possui números de faces que diferem de pessoa para pessoa.

O conjunto *Olivetti Faces*, utilizado em (Dhifli e Diallo, 2016), possui 400 imagens, entre os anos de 1992 e 1994, sendo 10 exemplos para cada um dos 40 indivíduos do *AT&T Laboratories Cambridge*. Para alguns sujeitos, as imagens foram tiradas em momentos diferentes, variando a iluminação, expressões faciais (olhos abertos / fechados, sorrindo / não sorrindo) e detalhes faciais (óculos / sem óculos). Todas as imagens foram tiradas contra um fundo escuro homogêneo com os indivíduos em uma posição frontal vertical (com tolerância para alguns movimentos laterais).

O conjunto LFW (*Labeled Faces in the Wild*), criado em (Huang et al., 2008), contém mais de 13.000 imagens de rostos coletados da web, extraídos com o uso do detector facial *Viola-Jones* (Bandala et al., 2013). Cada rosto foi etiquetado com o nome da pessoa retratada, levando em consideração o fato de que 1.680 pessoas retratadas possuírem duas ou mais fotos distintas. Sendo utilizado em trabalhos importantes, como em (Schroff et al., 2015), tornou-se um banco comum para validações de sistemas de reconhecimento facial.

Como a principal motivação do presente trabalho é propor um modelo extrator de características, foram escolhidos conjuntos de dados com imagens já tratadas, eliminando a necessidade de um detector facial e/ou um pré-tratamento. Assim, tornou-se necessário apenas a normalização dos dados e o redimensionamento para o tamanho desejado, sendo $224 \times 224 \times 3$ o tamanho das imagens usadas no desenvolvimento do modelo.

5.2 Resultados

Para medir a eficiência do modelo em distinguir as classes dos dados de treinamento, foi utilizado o conceito AUC-ROC. Enquanto a curva ROC (*Receiver Operating Characteristic*) consiste em uma representação gráfica do desempenho de um modelo de classificação por meio da relação da Taxa de Verdadeiro Positivo (Sensibilidade) e da Taxa de Falso Positivo (1-Especificidade), a AUC (*Area Under the Curve*) é a área abaixo da curva ROC, sendo uma medida de separabilidade entre as classes. A AUC é calculada como uma área que varia de 0 a 1, tendo sua interpretação como uma probabilidade. Assim, uma AUC próxima de 1 indica que o modelo consegue separar bem as classes, enquanto que próximo de $0,5$ indica que o modelo não tem capacidade discriminativa de distinguir entre as classes positivas e negativas. Uma vantagem de se usar a AUC-ROC neste caso é a funcionalidade de determinação do limiar (*threshold*) que realiza a separação entre as classes. Tendo em vista que: (i) um limiar igual a 0% indica que todos os valores são preditos como 1, logo todos os verdadeiros positivos são classificados corretamente e nenhum verdadeiro negativo é classificado; (ii) um limiar igual a 100% mostra que todos os valores são preditos como 0 e não há verdadeiro positivo, nem falso positivo.

As Tabelas 1 e 2 mostram o valor médio de AUC após 100 *epochs* de treinamento para as funções de perda *Constellation Loss*, *Multiclass N-Pair Loss* e *Triplet Loss*, nos con-

juntos de dados *Olivetti Faces* e LFW, respectivamente. Como pode ser observado, o *Constellation Loss* proporcionou melhores resultados mesmo nas duas diferentes arquiteturas de CNN testadas como *backbone* do modelo.

Tabela 1. AUC-ROC no Conjunto de Dados *Olivetti Faces*

Função de Perda	**ResNet152 V2**	**Inception V3**
Triplet	0,998	0,997
Multiclass N-Pair	0,985	0,987
Constellation $K = 2$	0,994	0,994
Constellation $K = 3$	0,998	0,997
Constellation $K = 4$	**0,999**	**0,999**
Constellation $K = 5$	0,997	0,998
Constellation $K = 6$	0,997	0,996
Constellation $K = 7$	0,998	0,997

Tabela 2. AUC-ROC no Conjunto de Dados LFW

Função de Perda	**ResNet152 V2**	**Inception V3**
Triplet	0,985	0,970
Multiclass N-Pair	0,970	0,955
Constellation $K = 2$	0,974	0,958
Constellation $K = 3$	0,981	0,967
Constellation $K = 4$	0,986	0,958
Constellation $K = 5$	0,983	0,962
Constellation $K = 6$	**0,987**	**0,971**
Constellation $K = 7$	0,982	0,964

Como afirmado por Medela e Picón (2019) e podendo ser comprovado pelos resultados encontrados, um aumento significativo do hiperparâmetro K não garante uma melhora nos resultados. No conjunto *Olivetti Faces*, por exemplo, o valor de $K = 4$ obteve os melhores resultados em ambas as CNN, enquanto no conjunto LFW, o valor de $K = 6$ encontrou os maiores valores de AUC-ROC.

Para via de validação do modelo treinado no conjunto *Olivetti*, adicionou-se o classificador *Nearest Neighbor* (NN) ao final da rede extratora de características. O modelo alcançou $98,33\% \pm 0,83$ de acurácia, valor próximo ao encontrado em Lu et al. (2021), onde foi aplicado aumento de dados (*data augmentation*) para ultrapassar 95% de acurácia (foi obtido 99% na métrica). A Figura 10 mostra os dois únicos exemplos na base de testes classificados erroneamente, sendo possível identificar que os pares das faces possuem um alto nível de semelhança, dificultando a diferenciação.

Como forma de realizar uma análise visual dos *embeddings* obtidos, foi utilizado o t-SNE van der Maaten e Hinton (2008). Esta técnica permite a visualização de dados de alta dimensão em espaços 2D ou 3D, preservando as relações de distância dos dados. O t-SNE foi aplicado para observar a separação das classes no modelo proposto. As Figuras 11 e 12, ilustram os *embeddings* dos conjuntos de dados testados *Olivetti Faces* e LFW, respectivamente.

Pela menor quantidade de classes existentes no conjunto de dados *Olivetti Faces* e uma padronização da quantidade de amostras por classe, fica mais nítida a separabilidade que o modelo proporcionou às classes presentes nos dados de teste, como pode ser visto na Figura 11. Isto comprova a eficácia representada pelo valor de AUC encontrado, o t-SNE mostra que o modelo conseguiu minimizar as

Figura 10. Imagens do conjunto Olivetti rotuladas de maneira equivocada pelo modelo, com a respectiva classe de saída

Figura 11. Visualização 2D t-SNE dos Vetores de *Embeddings* dos Dados de Teste do Olivetti Faces. Cada cor representa uma das classes do conjunto de dados

distâncias inter-classes e maximizar as distâncias extra-classes.

6. CONCLUSÃO

O presente trabalho teve como objetivo propor um modelo de extração de características para sistemas de reconhecimento facial, de modo a aglomerar espacialmente amostras de uma mesma pessoa e aumentar o distanciamento de aglomerações de amostras de outras pessoas. Tal representação é denominada como *embedding* e foi gerada por uma rede CNN pré-treinada, atuando como um *backbone*, que foi ajustada via uma função de perda de natureza contrastiva.

Visando reduzir o custo computacional existente no treinamento de modelos RNS, foi desenvolvida uma metodologia

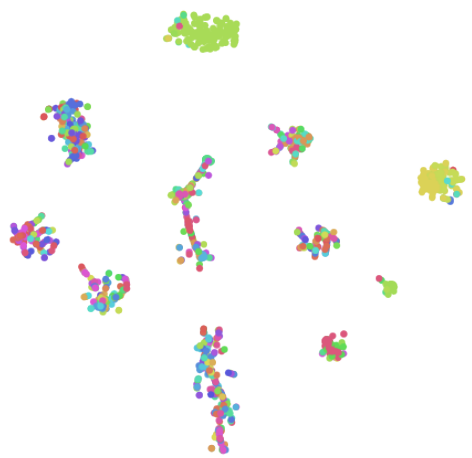

Figura 12. Visualização 2D t-SNE dos Vetores de *Embeddings* dos Dados de Teste do LFW. Cada cor representa uma das classes do conjunto de dados

específica, que, ao invés de construir uma estrutura RNS, a qual replica uma CNN para cada entrada considerada, formou-se um único *batch* de treinamento respeitando a organização das entradas esperadas pela função de perda empregada. Este procedimento eliminou a necessidade de replicar a rede para cada imagem de entrada, como é feita nas RNS, e garantiu uma maior facilidade na aplicação de funções de perda que levam em consideração 3 ou mais imagens de entrada, como as funções testadas neste trabalho.

A proposta alcançou resultados significativos ao aplicar a função de perda *Constellation Loss*, juntamente com a CNN *ResNet152V2* como *backbone*. O método proposto alcançou valores significativos de AUC-ROC e classes de *embeddings* bem separadas. No conjunto de dados *Olivetti Faces*, o modelo atingiu 99,9% de valor médio de AUC-ROC, enquanto nos dados do conjunto LFW, obteve 98,7%. Com base nos resultados encontrados, o modelo proposto provou sua eficácia, indicando a viabilidade de aplicação do mesmo em sistemas de reconhecimento facial.

REFERÊNCIAS

Bandala, A., Fernandez, M., Gob, K., Leonidas, A., Ravara, R., e Dadios, E. (2013). Simultaneous face detection and recognition using viola-jones algorithm and artificial neural networks for identity verification. doi: 10.1109/TENCONSpring.2014.6863118.

Bromley, J., Guyon, I., Lecun, Y., Säckinger, E., e Shah, R. (1993). Signature verification using a siamese time delay neural network. volume 7, 737–744.

Chopra, S., Hadsell, R., e LeCun, Y. (2005). Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, 539–546 vol. 1. doi:10.1109/CVPR.2005.202.

Cottrell, G. e Fleming, M. (1990). Face recognition using unsupervised feature extraction. *Proc. Int'l Neural Network Conf.*, 3, 322–325.

Dhifli, W. e Diallo, A. (2016). Face recognition in the wild. *Procedia Computer Science*, 96, 1571–1580. doi: 10.1016/j.procs.2016.08.204.

Goldstein, A., Harmon, L., e Lesk, A. (1971). Identification of human faces. *Proceedings of the IEEE*, 59, 748 – 760. doi:10.1109/PROC.1971.8254.

He, K., Zhang, X., Ren, S., e Sun, J. (2016). Identity mappings in deep residual networks. *CoRR*, abs/1603.05027. URL `http://arxiv.org/abs/1603.05027`.

Huang, G., Mattar, M., Berg, T., e Learned-Miller, E. (2008). Labeled faces in the wild: A database forstudying face recognition in unconstrained environments. *Tech. rep.*

Lu, P., Song, B., e Xu, L. (2021). Human face recognition based on convolutional neural network and augmented dataset. *Systems Science & Control Engineering*, 9(sup2), 29–37. doi:10.1080/21642583.2020.1836526. URL `https://doi.org/10.1080/21642583.2020.1836526`.

Medela, A. e Picón, A. (2019). Constellation loss: Improving the efficiency of deep metric learning loss functions for optimal embedding. *CoRR*, abs/1905.10675. URL `http://arxiv.org/abs/1905.10675`.

Schroff, F., Kalenichenko, D., e Philbin, J. (2015). Facenet: A unified embedding for face recognition and clustering. *Proc. CVPR*.

Sohn, K. (2016). Improved deep metric learning with multi-class n-pair loss objective. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, e R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc. URL `https://proceedings.neurips.cc/paper/2016/file/6b180037abbebea991d8b1232f8a8ca9-Paper.pdf`.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S.E., Anguelov, D., Erhan, D., Vanhoucke, V., e Rabinovich, A. (2014). Going deeper with convolutions. *CoRR*, abs/1409.4842. URL `http://arxiv.org/abs/1409.4842`.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., e Wojna, Z. (2015). Rethinking the inception architecture for computer vision. URL `http://arxiv.org/abs/1512.00567`. Cite arxiv:1512.00567.

Turk, M. e Pentland, A. (1991a). Eigenfaces for recognition. *Journal of cognitive neuroscience*, 3, 71–86. doi: 10.1162/jocn.1991.3.1.71.

Turk, M. e Pentland, A. (1991b). Face recognition using eigenfaces. 586 – 591. doi:10.1109/CVPR.1991.139758.

van der Maaten, L. e Hinton, G. (2008). Viualizing data using t-sne. *Journal of Machine Learning Research*, 9, 2579–2605.

Weinberger, K.Q. e Saul, L.K. (2009). Distance metric learning for large margin nearest neighbor classification. *J. Mach. Learn. Res.*, 10, 207–244. URL `http://dl.acm.org/citation.cfm?id=1577069.1577078`.

Zeiler, M.D. e Fergus, R. (2013). Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901. URL `http://arxiv.org/abs/1311.2901`.