Sistema Embarcado para reconhecimento facial utilizando métricas de similaridade

Bonifácio de Oliveira[1], **Luis Cuevas Rodriguez**[1], **Jucimar Maia Junior**[1]

[1]Laboratório LUDUS – Universidade do Estado do Amazonas (UEA)
Manaus – AM – Brazil

{bldof.eng16,lrodriguez,jjunior}@uea.edu.br

Abstract. *Considering the recent studies and interest in the area of facial recognition, this article aims to analyze the similarity metrics and test the processing speed of the Raspberry Pi 3B embedded model. Tests were made with the following available metrics: Bray-Curtis, Canberra, Chebyshev, Manhattan, Correlation, Cosine and Euclidian. After get the results, the accuracy of each metric was measured. Also was compared the processing speed using Python and C ++ using various image sizes as input to the face detection algorithm.*

Resumo. *Considerando os recentes estudos e o interesse na área de Inteligência Artificial, este artigo tem como objetivo analisar o reconhecimento facial através das métricas de similaridade e testar a velocidade de processamento do modelo embarcado no Raspberry Pi 3B. Foram feitos testes com as seguintes métricas: Bray-Curtis, Canberra, Chebyshev, Manhattan, Correlação, Cosseno e Euclidiana. Após obter os resultados, foi medida a acurácia de cada distância. Foi feita também a comparação entre a velocidade de processamento utilizando Python e C++ utilizando diversos tamanhos de imagem como entrada para o algoritmo de detecção facial.*

1. Introdução

Com o crescente interesse na área de *Machine Learning*, uma grande demanda por reconhecimento de indivíduos vem sendo gerada, implementar o Reconhecimento Facial de forma simples e que não exija muitos recursos computacionais é um desafio constante.

Há diversas aplicações que precisam obrigatoriamente ter uma boa eficácia ao fazer inferências, como por exemplo na área de segurança, onde é necessário identificar se uma pessoa é permitida ou não de frequentar um ambiente [Carneiro 2012]. Essas tarefas são de extrema importância, visto que não é viável utilizar seres humanos para analisar toda essa quantidade de dados. Por este motivo, pesquisas focadas na área de Reconhecimento Facial têm ganhado cada vez mais força.

Uma das formas de executar tal reconhecimento, consiste em fazer inferências através de comparação de imagens utilizando métricas de similaridade. Métricas de Similaridade são formas diferentes de analisar o quão dois conjuntos de dados (no caso dos testes apresentados neste artigo, um vetor de 128 posições) são parecidos, e têm diversas aplicações em Ciência da Computação.

Existem diversas métricas disponíveis para efetuar o cálculo da similaridade, porém não há um método genérico para definir qual é o mais indicado para certos problemas, no entanto, ao analisar os dados, os resultados que se deseja obter e executar

testes, é possível definir qual medida se encaixa melhor para um respectivo problema [Arora et al. 2019].

Além dos estudos em *Machine Learning*, sistemas embarcados estão presentes em diversas áreas da sociedade, com tendência a ter cada vez mais representantes. Além disto, pode-se observar também a presença de sistemas embarcados nas máquinas de uso doméstico, por exemplo: máquina de lavar roupas, micro-ondas, ou nos sistemas automotivos.

O presente artigo tem como objetivo avaliar o desempenho de um modelo de Reconhecimento Facial, comparando os resultados obtidos com sete métricas de similaridade. A sequência deste artigo está organizada da seguinte forma: a Seção 2 trata do Reconhecimento Facial por métricas de similaridade; a Seção 3 apresenta a arquitetura do modelo com detalhes sobre cada etapa realizada; na Seção 4 são apresentadas as métricas de similaridade aplicadas no modelo; a Seção 5 trata do processo de implementação seguido; e as Seções 6 e 7 apresentam respectivamente os resultados e as considerações finais.

2. Reconhecimento Facial por Métricas de Similaridade

A técnica de Reconhecimento Facial usando métricas de similaridade não exige o treino de um modelo de *Machine Learning* e consiste em utilizar distâncias para medir a diferença entre imagens. Quanto menor a diferença entre duas imagens maior a probabilidade de ser a mesma pessoa em ambas as imagens.

Visto que ao executar as devidas comparações para efetuar a inferência sobre uma imagem, dificilmente alguma comparação é exata, esse método de reconhecimento facial por métricas de similaridade consiste em comparar um conjunto de características extraídas da imagem de entrada com informações de um conjunto de características de imagens previamente selecionadas [Fujikawa 2016].

Inicialmente diversas imagens, que servirão como dataset, são selecionadas e cada uma é convertida em um vetor e este é armazenado. Ao iniciar a inferência, o algoritmo busca no dataset um vetor que seja exatamente igual ao que está sendo analisado, caso não seja encontrado, é medida a similaridade entre o vetor analisado e cada elemento do dataset, aquele elemento que possuir a menor distância é selecionado como resultado da inferência [Gawande and Agrawal 2014].

3. Arquitetura do modelo

3.1. Dataset

O Dataset utilizado no ”treino”(Figura 1), é composto por apenas uma imagem do rosto de cada pessoa que se deseja efetuar o reconhecimento. A partir da imagem, são extraídas as *embeddings* e armazenadas para serem usadas em cada consulta ao efetuar o reconhecimento.

O Dataset de treino, que é utilizado como base para as inferências, contém 10 fotos 1, sendo cada uma de uma pessoa diferente. O Dataset de teste é composto por 11622 fotos, divididas em 10 classes de pessoas conhecidas e 1 classe com pessoas desconhecidas.

Figura 1. Dataset utilizado

3.2. Detecção Facial

O detector utilizado é o *Histograms of Oriented Gradients* – HOG (Figura 2), que consiste em converter a imagem de entrada para escalas de cinza e através das variações de luminosidade entre conjuntos de pixels, definir vetores que obedecem a essa variação. Após obter o conjunto de vetores da imagem, é efetuada uma busca por padrões pré definidos, caso o padrão seja detectado, é efetuada a predição de que há um rosto [Dalal and Triggs 2005].

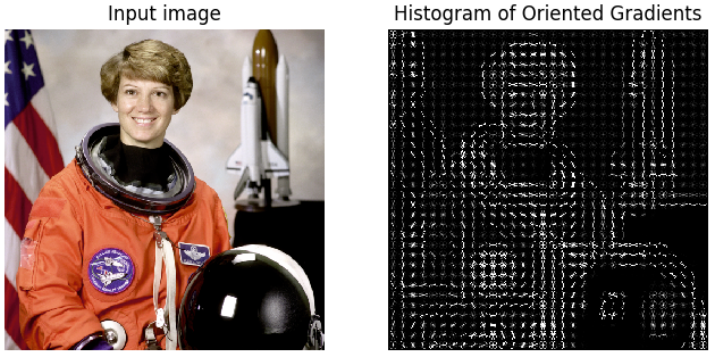

Figura 2. Detector de faces HOG
Fonte: [Chopra 2019]

3.3. Mapeamento e Alinhamento Facial

Após efetuar a detecção, com o intuito de melhorar a qualidade do reconhecimento, é necessário utilizar uma técnica de alinhamento facial. O primeiro passo é aplicar mapeamento da face, que consiste em encontrar 68 pontos do rosto, conhecidos como *landmarks* (Figura 3). Os pontos detectados são: olhos, boca, nariz, o contorno da face e acima das sobrancelhas [Kazemi and Sullivan 2014].

Figura 3. Pontos Faciais localizados
Fonte: [Geitgey 2016]

Em posse dos pontos faciais detectados, é aplicado um ajuste com o intuito de centralizar o rosto. Para não gerar distorções relevantes na face, apenas transformações básicas são aplicadas, esse conjunto de operações recebe o nome de *affine transformation* e engloba tranformações de escala, cisalhamento e rotação (Figura 4).

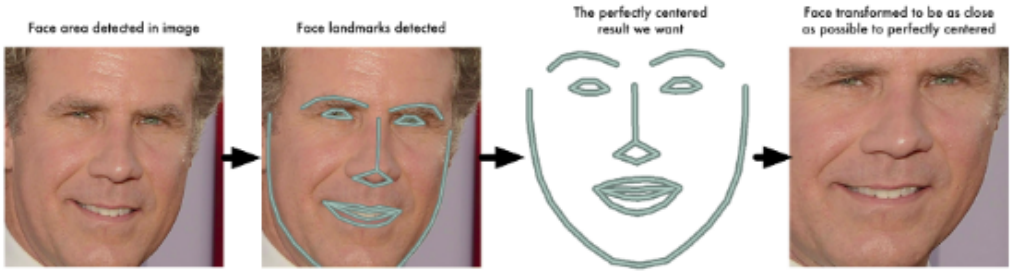

Figura 4. Aplicação de Trasnformações na face
Fonte: [Geitgey 2016]

3.4. Extração das *Embeddings*

Para gerar o vetor de 128 dimensões, que será submetido às comparações, é utilizado um modelo baseado em Redes Neurais, o *FaceNet*. Esse modelo tem como função receber uma imagem de entrada e extrair as *embeddings* (Figura 5), que são as características que melhor descrevem aquela imagem, nesse caso uma face [Schroff et al. 2015].

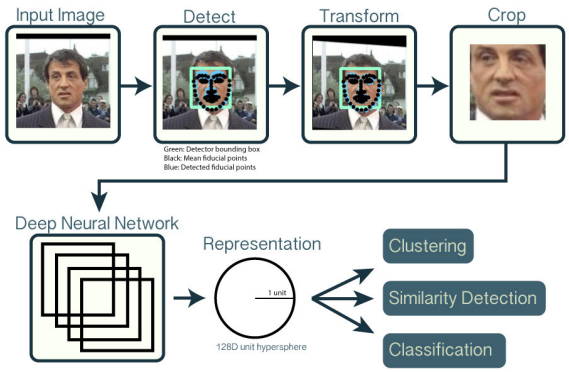

Figura 5. Extração das Embeddings
Fonte: [Amos et al. 2016]

3.5. Comparação

Após obter o vetor de 128 dimensões, é medida a similaridade entre o vetor obtido e cada um dos dados previamente armazenados como Dataset. Quanto mais os dois conjuntos de dados comparados forem parecidos, menor será a medida de similaridade, visto que menor será a distância entre eles.

Em posse dos valores das similaridades entre o vetor de entrada e os cadastrados, é verificado se o valor da medida que obteve o melhor resultado obedece a um limite, conhecido como tolerância. Se sim, o vetor do dataset com o melhor resultado é utilizado para efetuar a inferência (Figura 6), caso contrário, o sistema retorna aquele usuário como desconhecido.

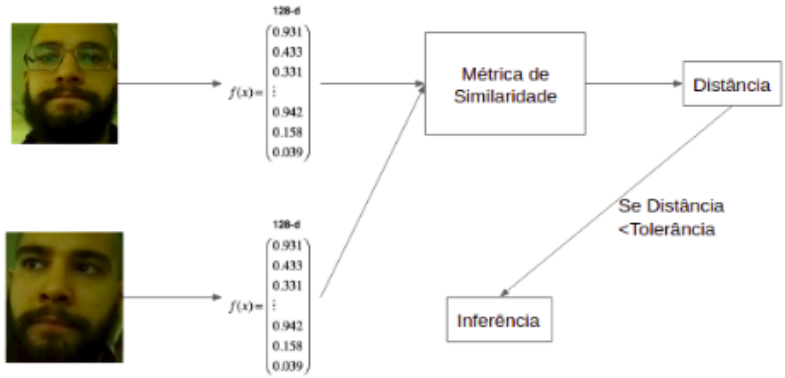

Figura 6. Modelo de Reconhecimento

3.6. Embarcar Modelo Obtido

Após obter o código fonte, o modelo de reconhecimento facial foi embarcado no Rasperry Pi 3B (Figura 7) com o intuito de analisar os resultados de velocidade, que foi medida em FPS, foi utilizado o módulo de câmera disponível para Raspberry Pi com 5MP.

Os testes foram feitos ao aplicar transformações no tamanho da imagem, que aumenta a taxa de detecção do sistema ao ser utilizado em tempo real, permitindo que o modelo seja capaz de detectar faces a uma distância maior.

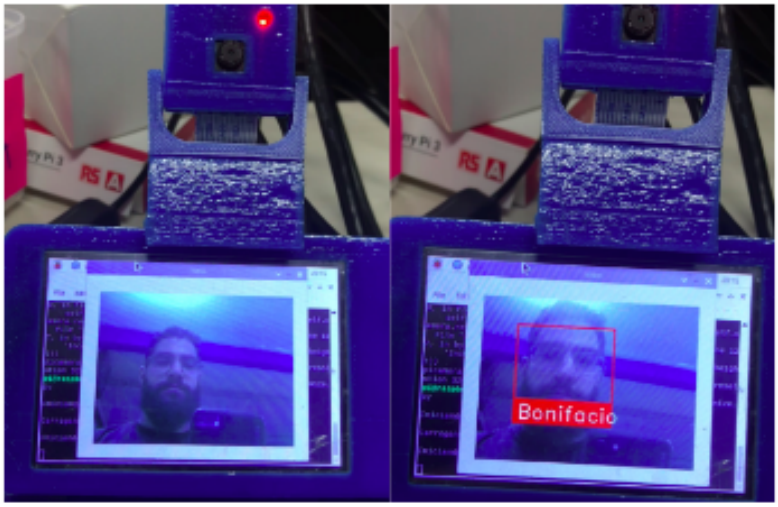

Figura 7. Modelo Embarcado

4. Métricas de Similaridade

Essa Seção apresenta as métricas de similaridade aplicadas ao modelo de Reconhecimento Facial.

4.1. Distância de Bray-Curtis

$$BC_{ij} = \sum \frac{|n_{ik} - n_{jk}|}{(n_{ik} + n_{jk})}, \tag{1}$$

A distância de Bray-Curtis é uma medida assimétrica e normalizada, muito utilizada na biologia [Thakur et al. 2019]. O resultado é compreendido entre 0 e 1, onde 0 quer dizer que os dois conjuntos de dados possuem a mesma composição e 1 que não há nenhum termo semelhante.

4.2. Distância de Canberra

$$distance(x - y) = \sum \frac{|x_i - y_i|}{|x_i| + |y_i|}, \tag{2}$$

A distância de Canberra [Lance and Williams 1966] é uma função frequentemente usada para dados espalhados em torno de uma origem.

É similar à Distância de Manhattan, a principal distinção é que a diferença absoluta entre as variáveis dos dois objetos é dividida pela soma dos valores absolutos da variável antes de cada soma.

4.3. Distância de Chebyshev

$$D_{Chebyshev}(x, y) := max(|x_i - y_i|), \tag{3}$$

É uma métrica que para definir a similaridade entre dois vetores, considera a maior diferença medida entre duas coordenadas de qualquer dimensão, ou seja, após obter a diferença entre cada dimensão correspondente, é selecionada como distância aquela de maior valor. [Linden 2009]

4.4. Distância de Manhattan

$$d = \sum_{i=0}^{n-1} |(x[i] - y[i])|, \tag{4}$$

A distância é definida como o somatório das diferenças entre cada conjunto de coordenadas correspondentes. Se feita uma comparação entre ela e a Distância Euclidiana, pode-se perceber a relação entre as duas distâncias e o Teorema de Pitágoras. [Linden 2009]

4.5. Correlação

$$1 - \frac{(u - \bar{u}) \cdot (v - \bar{v})}{||(u - \bar{u})||_2||(v - \bar{v})||_2}, \tag{5}$$

É a medida de similaridade que utiliza probabilidade, mede o relacionamento linear entre os atributos, dessa forma pode ser utilizada para medir o relacionamento. [Metz 2006]

4.6. Distância Cosseno

$$similarity = cos(\Theta) = \frac{A \cdot B}{||A||||B||} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}, \tag{6}$$

É uma medida do ângulo entre x e y que leva em consideração o ângulo entre dois elementos de um conjunto de dados. Se a similaridade é 1, ou seja, se são iguais, o ângulo entre x e y é 0, se a similaridade é 0, o ângulo é 90. [Salazar 2012]

4.7. Distância Euclidiana

$$D(a, b) = \sqrt{\sum_{i=1}^{n} (b_i - a_i)^2}, \tag{7}$$

Baseada no Teorema de Pitágoras, é uma das métricas de similaridade mais usadas. Basicamente, é calculada a "hipotenusa" e esse valor é definido como a distância entre dois vetores. No entanto, na presença de ruídos no conjunto de dados, essa métrica pode vir a perder a eficácia [Arora et al. 2019].

5. Implementação

O primeiro teste foi o de reconhecimento facial, implementado em Python 3.6 [van Rossum 1995] utilizando a biblioteca *face_recognition* [Geitgey 2017], essa biblioteca utiliza Dlib [King 2009] como subrotina para efetuar não apenas o reconhecimento, mas também a detecção, normalização e codificação de faces. Para efetuar as comparações ao executar as inferências, foram utilizadas as métricas de similaridade entre vetores disponíveis na biblioteca SciPy [Jones et al. 2001].

O hardware utilizado para embarcar e executar os testes foi um Raspberry Pi 3B, com 1GB de memória RAM e 1200 MHz. Onde foi analisado também a velocidade do reconhecimento em tempo real e o desempenho utilizando a linguaguem de Programação

Python [van Rossum 1995] e a linguagem C++ [Stroustrup 2014], o compilador utilizado para C++ foi o G++ 6.3.0.

Inicialmente foram carregadas 10 imagens, cada uma pertencendo a uma pessoa, cada imagem foi convertida em um vetor de 128 dimensões contendo as características do rosto presente. Feita a conversão, o vetor e seu rótulo foram armazenados.

O Dataset de testes foi armazenado em diversas pastas rotuladas com o respectivo nome do usuário ou "Desconhecido", cada pasta foi acessada e todas as fotos presentes naquele diretório foram classificadas, o resultado de cada inferência foi salvo em um arquivo CSV, contendo a resposta esperada e a resposta inferida.

Após gerar um arquivo CSV para cada distância e medida de tolerância testada, a acurácia de cada um foi medida e dessa forma foi possível definir qual métrica se encaixa melhor ao problema.

$$acuracia = \frac{verdadeirosPositivos + verdadeirosNegativos}{totalDeInferencias} \tag{8}$$

6. Resultados

Na primeira etapa dos testes, foi testada a operação de reconhecimento facial ao aplicar métricas de similaridades diferentes das usuais. Além das métricas aplicadas, outro parâmetro fundamental foi analisado, a tolerância, que tem como função definir uma distância máxima aceitável entre dois vetores para que uma pessoa não seja desconhecida.

Foram testadas as tolerâncias mais utilizadas e que ainda geram resultados eficientes. A Tabela 1 apresenta os resultados obtidos, a Figura 8 representa visualmente as acurácias, embora a Distância Euclidiana tenha obtido o melhor resultado utilizando 0.5 como tolerância, é possível perceber que não houve grande diferença entre as acurácias obtidas quando se varia apenas as métricas de similaridade.

No que diz respeito a eficácia do modelo ao variar as medidas de tolerância, embora a medida de tolerância usual seja 0.6 foi possível inferir 0.5 como a medida de tolerância com maior acurácia.

Tabela 1. Resultados dos testes realizados

Métricas/Tolerância	0.4	0.5	0.6
Bray-Curtis	0.784719	0.949664	0.829719
Canberra	0.784633	0.945620	0.833591
Chebyshev	0.78274	0.929014	0.828257
Manhattan	0.784719	0.949664	0.828515
Correlação	0.784719	0.949578	0.829547
Cosseno	0.784719	0.949664	0.829719
Euclidiana	0.784719	0.949750	0.827052

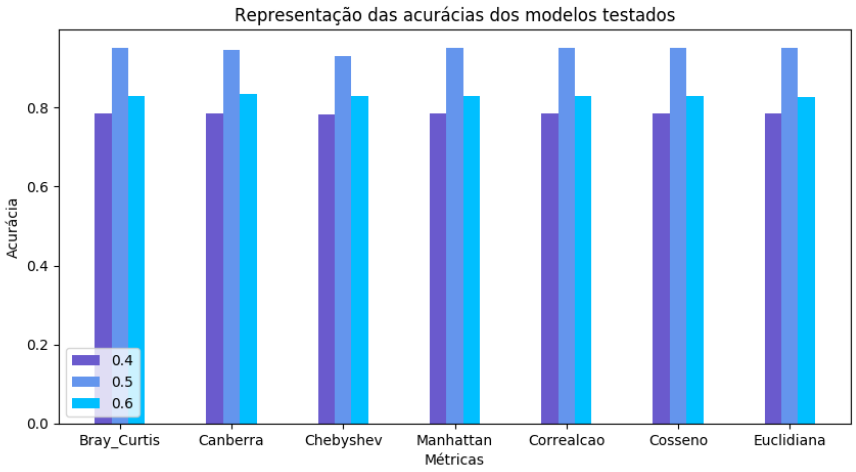

Figura 8. Taxas de Reconhecimento

Na segunda etapa dos testes, foi analisada a velocidade de processamento ao variar a linguagem utilizada para implementar e o tamanho da imagem de entrada fornecida ao detector facial (Figura: 8). A linguagem com o melhor desempenho, embora seja considerada uma baixa taxa de FPS, foi C++ que obteve 6 FPS ao dividir o tamanho da imagem pela metade. O reconhecimento permaneceu eficiente mesmo com imagens de um tamanho menor, embora o algoritmo de detecção tenha diminuído consideravelmente sua eficácia.

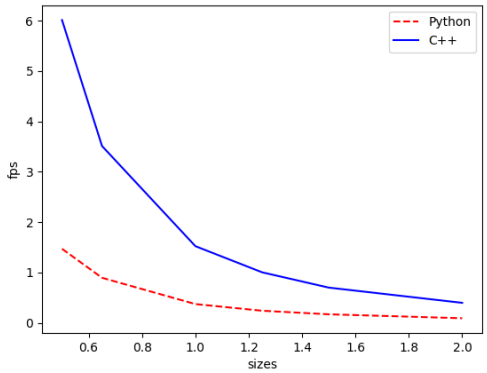

Figura 9. FPS medido para o código em C++ e Python

7. Considerações finais

Foram analisadas várias métricas de similaridade aplicadas ao reconhecimento facial, em cada métrica avaliou-se também as principais tolerâncias aplicadas. Todas as medidas, com exceção da Distância de Chebyshev, apresentaram resultados parecidos. Mesmo com a proximidade entre os resultados, a Distância Euclidiana com tolerância de 0.5 apresentou o melhor resultado, e foi possível analisar a melhor tolerância a se aplicar, visto que a tolerância igual a 0.5 apresentou uma acurácia maior independente da métrica aplicada.

O maior desafio encontrado no decorrer do projeto foi quanto à otimização e tempo de processamento do Raspberry Pi 3B, C++ apresentou a maior velocidade de processamento. Foi possível perceber a relação entre o tamanho da imagem, a acurácia da detecção facial e a velocidade de processamento, visto que a velocidade de processamento aumenta em imagens de menor tamanho, em contrapartida gera perda da eficácia ao efetuar a detecção facial.

Como trabalhos futuros, pretende-se aplicar alterações nos hiperparâmetros utilizados na codificação e reconhecimento facial e analisar algoritmos *lightweight* que apresentem menor custo computacional, dessa forma visando obter um FPS mais alto.

Agradecimentos

Os resultados apresentados nessa publicação foram obtidos por meio de atividades de Pesquisa e Desenvolvimento executadas no projeto KAM, com recursos oriundos da Lei nº 8.387/1991, devendo qualquer publicidade fazer referência a citada lei conforme Art. 48 do Decreto nº 6.008/2006, tendo a empresa Samsung Eletrônica da Amazônia LTDA financiado esse projeto nos termos da mencionada lei. A pesquisa foi realizada dentro das instalações do Laboratório de Tecnologia, Inovação e Economia Criativa – LUDUS.

Referências

Amos, B., Ludwiczuk, B., and Satyanarayanan, M. (2016). Openface: A general-purpose face recognition library with mobile applications. Technical report, CMU-CS-16-118, CMU School of Computer Science.

Arora, J., Khatter, K., and Tushir, M. (2019). Fuzzy c-means clustering strategies: A review of distance measures. In *Software Engineering*, pages 153–162. Springer.

Carneiro, L. N. d. V. (2012). Reconhecimento de face invariante a iluminação baseado em uma abordagem supervisionada.

Chopra, E. (2019). Using histogram of oriented gradients (hog) for object detection.

Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection.

Fujikawa, C. S. (2016). Reconhecimento facial utilizando descritores de textura e aprendizado não supervisionado.

Gawande, M. P. and Agrawal, D. G. (2014). Face recognition using pca and different distance classifiers. *IOSR Journal of Electronics and Communication Engineering (IOSR-JECE)*, 9(1):1–5.

Geitgey, A. (2016). Machine learning is fun! part 4: Modern face recognition with deep learning.

Geitgey, A. (2017). Welcome to face recognition's documentation!

Jones, E., Oliphant, T., Peterson, P., et al. (2001). SciPy: Open source scientific tools for Python.

Kazemi, V. and Sullivan, J. (2014). One millisecond face alignment with an ensemble of regression trees.

King, D. E. (2009). Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research*, 10:1755–1758.

Lance, G. N. and Williams, W. T. (1966). Computer Programs for Hierarchical Polythetic Classification ("Similarity Analyses"). *The Computer Journal*, 9(1):60–64.

Linden, R. (2009). Técnicas de agrupamento. 1.

Metz, J. (2006). *Interpretação de clusters gerados por algoritmos de clustering hierárquico*. PhD thesis, Universidade de São Paulo.

Salazar, F. A. S. R. (2012). *Um estudo sobre o papel de medidas de similaridade em visualização de coleções de documentos*. PhD thesis, Universidade de São Paulo.

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

Stroustrup, B. (2014). *A Tour of C++*. C++ in-depth series. Addison-Wesley.

Thakur, N., Mehrotra, D., Bansal, A., and Bala, M. (2019). *Analysis and Implementation of the Bray–Curtis Distance-Based Similarity Measure for Retrieving Information from the Medical Repository: Proceedings of ICICC 2018, Volume 2*, pages 117–125.

van Rossum, G. (1995). Python tutorial. Technical Report CS-R9526, Centrum voor Wiskunde en Informatica (CWI), Amsterdam.