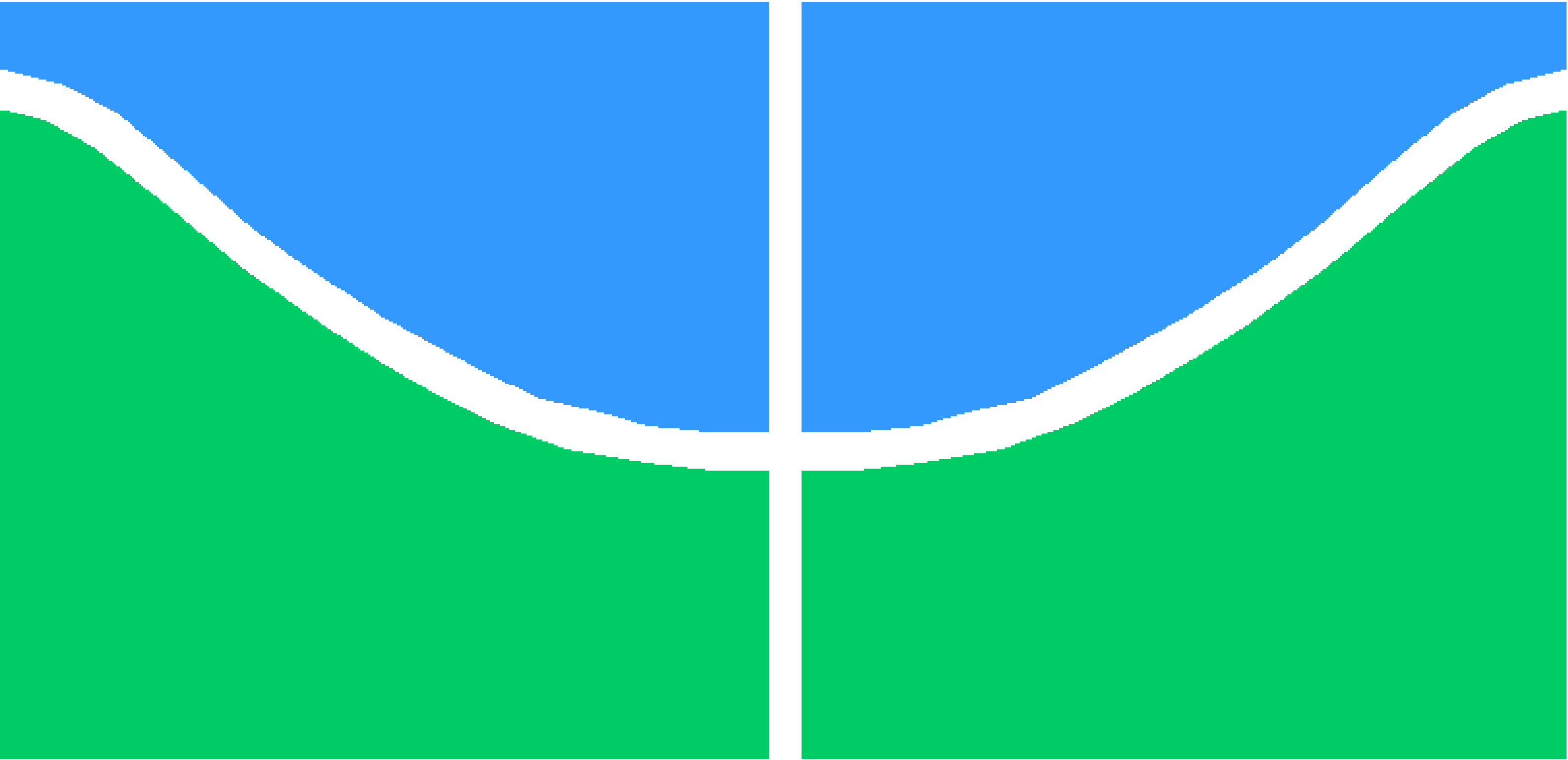

TRABALHO DE GRADUAÇÃO

ESTUDOS SOBRE VERIFICAÇÃO DE FACES COM IMAGENS CROSS-DOMAIN

Isabela Fernandes Bispo

Brasília, Dezembro de 2017

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia

TRABALHO DE GRADUAÇÃO

ESTUDOS SOBRE VERIFICAÇÃO DE FACES COM IMAGENS CROSS-DOMAIN

Isabela Fernandes Bispo

Relatório submetido ao Departamento de Engenharia Elétrica como requisito parcial para obtenção do grau de Engenheiro de Redes de Comunicação

Banca Examinadora

Prof. Flávio Elias de Deus, ENE/UnB ____________________
Orientador

Johnatan Santos de Oliveira ____________________

Prof. Georges Daniel Amvame Nze, ENE/UnB ____________________

Dedicatória

A Deus, que antes que eu respirasse soprou vida em mim.
À minha mãe, sem a qual eu nada seria.

Isabela Fernandes Bispo

Agradecimentos

Agradeço em primeiro lugar a Deus, que renova minhas forças diariamente para superar as dificuldades. À minha família, pelo amor, incentivo e apoio incondicional. Aos meus amigos de curso, que estiveram presente e me ajudaram durante toda esta trajetória, sem os quais muito do que foi feito não teria sido possível. Ao meu orientador, Prof. Dr. Flávio Elias, por todo suporte e incentivos e ao mestrando Johnatan Santos pelos ensinamentos compartilhados. E a todos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigado.

Isabela Fernandes Bispo

RESUMO

Cada vez mais tem sido possível realizar tarefas cotidianas via Internet, através de celulares e outros aparelhos tecnológicos. O que antes só podia ser feito com a presença física do usuário, hoje em dia pode ser feito de qualquer lugar pelo celular. Com esse avanço tecnológico cresce também o número de fraudes e ataques, necessitando-se mais ainda de formas de garantir a segurança do usuário.

A biometria tem sido usada como forma de identificação pessoal por diversos setores, inclusive em bancos, visto que não há como usar a identidade de outra pessoa. O reconhecimento facial tem várias vantagens em relação a outras modalidades de biometria, como a impressão digital e íris. É uma técnica onde é possível capturar uma imagem a uma certa distância do alvo e de maneira disfarçada. É possível ainda que o usuário tire sua própria foto de onde estiver, tire foto de seu documento pessoal e abra uma conta em um banco ou peça seu cartão de crédito através de um aplicativo para celular.

A verificação de faces entre imagens de domínios diferentes, como uma *selfie* e a foto de um documento pessoal, possui certas dificuldades. Há diversas diferenças de iluminação e *noise* que interferem na performance das ferramentas utilizadas para a verificação. Tendo em vista esses obstáculos, neste trabalho é realizado um estudo sobre a verificação facial em imagens *cross-domain* e uma comparação das acurácias obtidas em cada ferramenta utilizada. A ferramenta com o melhor resultado para um maior número de imagens é a *OpenFace*.

ABSTRACT

Now more than ever has been possible to do daily tasks on the internet, whether through smartphones or other devices. Tasks that before would require the physical presence of the user now can be executed from anywhere with smartphones. Because of this technologic breakthrough , online frauds and attacks were also made easier, raising the need to create mechanics that enhance and guarantee the user safety.

The biometric check has been used as a mean for personal identification in different segments, including in the corporative banking business, since different individual will never match biometric characteristics. Facial recognition has many advantages over the other kind of biometric check, such as finger prints and iris scanners. It is a technic that allows a discrete capture of an image of a subject, even from a certain distance. It also allows the user to take his picture, or a picture of his personal documents to use it to open bank accounts or to order credit and debit cards via an app.

Cross-domain face matching, i.e. selfies and pictures of personal documents, is a process with some challenges. The different lighting conditions, angle or the image noise might interfere on the overall performance of the verification tools. Hence, in this paper, is presented a study on cross-domain face matching and a comparison of the accuracy of the results obtained from different tools. OpenFace obtained the best result with a larger number of images.

SUMÁRIO

1 Introdução **1**
- 1.1 Definição do problema 1
- 1.2 Objetivos 2
- 1.3 Estrutura do Trabalho 2

2 Fundamentação Teórica **3**
- 2.1 Reconhecimento Facial 3
- 2.1.1 Fluxo de Processamento 3
- 2.2 Aprendizado de Máquina 4
- 2.3 Redes Neurais Artificiais 5
- 2.3.1 Arquitetura das RNAs 5
- 2.3.2 Algoritmo de *Backpropagation* 6
- 2.3.3 Rede Neural Convolucional 6
- 2.4 *Deep Learning* 8
- 2.5 Análise de Componentes Principais 8
- 2.6 Aprendizado Supervisionado 8
- 2.7 Aprendizado Não-Supervisionado 9
- 2.8 Algoritmo de Árvore de Decisão 9
- 2.8.1 Pseudocódigo do Algoritmo de Árvore de Decisão 10
- 2.9 SVM 10
- 2.10 *Image Augmentation* 11
- 2.11 *Cross Validation* 12
- 2.11.1 Método *holdout* 12
- 2.11.2 *K-fold* 12
- 2.11.3 *Leave-One-Out* 12

3 Método Proposto e Ferramentas Utilizadas **13**
- 3.1 Descrição do Método 13
- 3.2 Banco de Imagens 13
- 3.3 LFW 14
- 3.4 *OpenFace* 14
- 3.4.1 Reconhecimento Facial com Redes Neurais 15
- 3.4.2 Pré-processamento 16

3.4.3 Treinamento da Rede Neural de Representação Facial 17
3.4.4 Modelos 17
3.4.5 Acurácia 17
3.5 *VGG Face Descriptor* 18
3.5.1 Arquitetura 18
3.6 *Caffe* 18
3.7 *PmSVM* 19
3.7.1 Acurácia 19
3.8 *Random Forest* 20
3.8.1 Funcionamento 20

4 Análises e Resultados 21
4.1 Image Augmentation 21
4.2 *OpenFace* 25
4.2.1 Testes com base no Experimento LFW 25
4.3 Extração de Características com *VGG* 28
4.4 PmSVM 30
4.5 *Random Forest* 32
4.6 Resultados 34

5 Conclusões 36

REFERÊNCIAS BIBLIOGRÁFICAS 38

Anexos 41

I Pré-Requisitos para as análises 42
I.1 Instalação *Docker* 42
I.2 Instalação *OpenFace* via *Docker* 42
I.3 Instalação PmSVM via *Docker* 42
I.4 Instalação *Random Forest* via *Docker* 44

LISTA DE FIGURAS

2.1 Representação do fluxo de processamento de reconhecimento facial [1] 4
2.2 Arquitetura das RNAs [2] 6
2.3 Algoritmo de Classificação de Árvore de Decisão [3] 9
2.4 Classificador Linear Clássico [4] 10
2.5 Ideia básica do SVM [4] 11
2.6 Variações de uma mesma imagem através de *augmentation* [5] 11

3.1 Fluxo para uma única imagem [6] 15
3.2 Fluxo de treinamento para uma rede neural *feedforward* [7] 15
3.3 Fluxo de processamento da *OpenFace* [7] 16

4.1 Execução *script* 1 22
4.2 Resultado *script* 1 22
4.3 Execução *script* 2 23
4.4 Resultado *script* 2 23
4.5 Execução *script* 3 24
4.6 Resultado *script* Final 24
4.7 Pastas de Entrada e de Saída de cada *script* 25
4.8 Pré-processamento das imagens do Cenário 1 25
4.9 Gerando representações das imagens do Cenário 1 26
4.10 Número de imagens e número de indivíduos no Cenário 1 26
4.11 Gerando o arquivo de pares *parescenario1.txt* 26
4.12 Acurácia no Cenário 1 27
4.13 Pré-processamento das imagens do Cenário 2 27
4.14 Gerando representações das imagens do Cenário 2 27
4.15 Gerando o arquivo de pares *parescenario2.txt* 27
4.16 Número de imagens e número de indivíduos no Cenário 2 27
4.17 Acurácia no Cenário 2 28
4.18 *Download* do modelo *Caffe* [8] 28
4.19 Criando arquivo com nomes das imagens 29
4.20 Extração de características 29
4.21 Arquivos *.feat* criados após extração 29
4.22 Conversão das *features* para o formato *libsvm* 30
4.23 Conversão dos dados para o intervalo [0,1] e acurácia obtida 31

4.24 Conversão das *features* do Cenário 2 para o formato *libsvm* 31
4.25 Conversão dos dados do Cenário 2 para o intervalo [0,1] e acurácia obtida 32
4.26 Gerando o arquivo *seeds.npy* 32
4.27 Treinamento e teste no Cenário 1 33
4.28 Treinamento e teste no Cenário 2 33
4.29 Fluxo de processamento de uma imagem 34

LISTA DE TABELAS

3.1 Modelos de redes neurais e número de parâmetros [9] 17
3.2 Modelos de redes neurais e acurácia em testes no LFW [9] 18

4.1 Resultados 34
4.2 Acurácias na *OpenFace* para diferentes valores de k na validação cruzada *k-fold* 35

LISTA DE ABREVIATURAS

Acrônimos

BVLC	Berkeley Vision and Learning Center
CLAHE	Contrast Limited Adaptive Histogram Equalization
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DNN	Deep Neural Network
GPU	Graphics Processing Unit
LFW	Labeled Faces in the Wild
MLP	Multi Layer Perceptron
PCA	Principal Component Analysis
PmSVM	Power Mean Support Vector Machine
ReLU	Rectified Linear Unit
RG	Registro Geral
RNA	Redes Neurais Artificiais
RNC	Redes Neurais Convolucionais
SVM	Support Vector Machine
UnB	Universidade de Brasília
VGG	Visual Geometry Group

Capítulo 1

Introdução

O reconhecimento facial é uma tarefa realizada rotineiramente por nós humanos e tem sido usada cada vez mais em aplicações de processamento de imagens digitais [1]. Seja nas fotos de redes sociais ou nas fotos tiradas do próprio celular, seja em tarefas mais importantes, como biometria, vigilância e verificação de passaporte. Em situações restritas onde iluminação, pose, distanciamento, expressão facial, orientação da imagem e oclusão podem ser controlados, o reconhecimento facial automatizado pode superar o desempenho do reconhecimento humano [1]. Principalmente quando a base de dados é formada por um grande número de faces.

A abordagem de maior sucesso no reconhecimento facial tem sido as Redes Neurais Convolucionais. No entanto, a indústria e o governo é que dominam o estado da arte nesta área, pois possuem uma grande base de dados [7]. Os grupos que usam bases de dados de treinamento de larga escala privadas atingem uma performance alta, de até 99%, quando testados na base de imagens LFW. As redes pré treinadas de código aberto utilizam bases de dados menores, como a *CASIA-WebFace* e a *FaceScrub*, compostas por imagens coletadas da Internet.

1.1 Definição do problema

Os bancos pelo mundo estão adotando biometria como forma de identificação pessoal, visto que é impossível usar a identidade de outra pessoa, o que pode acontecer facilmente com senhas [10]. Atualmente, muito do que é feito no dia a dia é feito digitalmente. Atividades que necessitavam a presença física, como por exemplo abrir uma conta em um banco, hoje podem ser feitas pelo celular [11]. O cliente tira uma foto da sua carteira de motorista ou RG com o próprio *smartphone*, tira um autorretrato (*selfie*) e a partir dessas informações juntamente com outras, se não houver nenhum problema, a conta é aberta. No entanto, as redes e os algoritmos de classificação treinados e testados com imagens similares no quesito de iluminação e resolução não tem uma boa performance quando usadas em imagens com severas mudanças de iluminação e *noise* [12], como as fotos tiradas de imagens de documentos de identidade. A aquisição de dados para o treinamento das redes é um desafio e o número de imagens por sujeito é limitado [13].

1.2 Objetivos

O Objetivo Geral deste trabalho é realizar um estudo sobre a verificação facial em imagens *Cross-Domain* com Redes Neurais Artificiais e algoritmos de classificação de código aberto, apurando a acurácia de cada um e comparando-os. Para isso será feita uma descrição das Redes Neurais utilizadas, dos algoritmos de classificação e uma análise dos resultados obtidos com cada um.

1.3 Estrutura do Trabalho

No Capítulo 2 serão abordados os principais fundamentos utilizados neste projeto.

No Capítulo 3 serão apresentados os métodos e as ferramentas, com suas principais características.

No Capítulo 4 serão apresentados os testes e análises feitas com as ferramentas apresentadas no Capítulo 3, em dois cenários diferentes. Comparações a partir dos resultados obtidos também serão feitas.

Por último, o Capítulo 5, contém as conclusões acerca das análises e comparações feitas.

Capítulo 2

Fundamentação Teórica

Neste capítulo são apresentados os principais fundamentos teóricos nos quais este trabalho foi embasado. Conceitos sobre o que foi utilizado ao longo dos experimentos e análises são detalhados.

2.1 Reconhecimento Facial

O Reconhecimento Facial tem várias vantagens em relação a outras modalidades de biometria, como a impressão digital e íris. É uma técnica onde é possível capturar uma imagem a uma certa distância do alvo e de maneira disfarçada. Como um sistema biométrico, reconhecimento facial opera em um ou nos dois modos: autenticação (ou verificação) facial, e identificação (ou reconhecimento) facial. Verificação facial envolve a comparação de duas imagens a fim de verificar se a identidade invocada por um indivíduo é verdadeira. A identificação facial envolve a comparação de uma imagem com várias outras a fim de associar a identidade da imagem facial desconhecida com alguma da base de dados. Em algumas aplicações de identificação só é preciso achar a face mais similar. Em vídeos de vigilância, no entanto, é necessário um valor limiar de confiança e todas as faces com uma pontuação acima desse valor limiar são relatadas.

O desempenho de um sistema de reconhecimento facial depende de vários fatores, como iluminação, pose facial, expressão, faixa etária, cabelo, acessórios no rosto, e movimento. De acordo com esses fatores existe dois cenários: cenários de usuários cooperativos e cenários de usuário não cooperativos. O primeiro cenário seria o caso de quando o usuário está disposto a apresentar seu rosto da maneira correta, como por exemplo uma pose frontal com os olhos abertos e expressão neutra. O segundo cenário seria o caso de aplicações de vigilância, onde o usuário não sabe que está sendo identificado, tem uma certa distância da câmera ou quando é vídeo.

2.1.1 Fluxo de Processamento

Um sistema de reconhecimento facial consiste normalmente de quatro módulos [1], sendo eles: detecção facial, normalização, extração de características e correspondência. A Fig. 2.1 mostra esse fluxo.

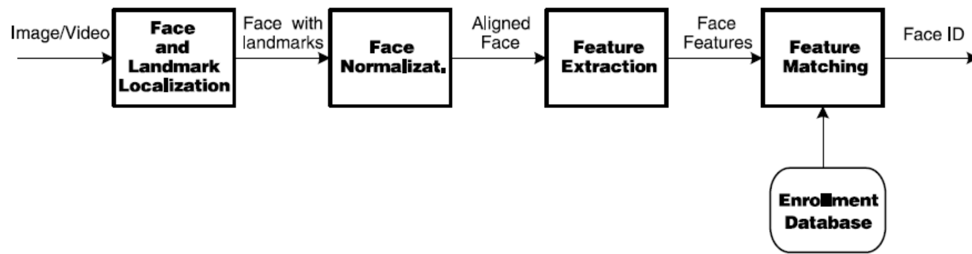

Figura 2.1: Representação do fluxo de processamento de reconhecimento facial [1]

- **Detecção facial**: separa a área da face do fundo da imagem. No caso de vídeos, as faces detectadas precisam ser localizadas em múltiplos quadros usando um componente responsável por isso. A detecção facial estima as localizações e escalas da face, mas é o *face landmarking* que determina a localização dos olhos, nariz e boca.

- **Normalização facial**: sua função é normalizar a face geometricamente e fotométricamente. Como é esperado que métodos de estado da arte de reconhecimento funcionem em imagens com diferentes iluminações e poses, a normalização facial é necessária. O processo de normalização geométrica corta a face, transformando-a em uma moldura padrão. O processo de normalização fotométrica normaliza a face baseado em propriedades de iluminação e na escala de cinza.

- **Extração de características**: realizado em faces normalizadas para extrair informações úteis para distinguir faces de pessoas diferentes. As características extraídas são usadas para correspondência facial.

- **Correspondência facial**: as características extraídas da face de entrada são comparadas com uma ou mais faces da base de dados. Em comparações 1:1, a saída é 'sim' ou 'não'; em comparações 1:N, a saída é a identidade da face de entrada quando o melhor resultado tem confiança suficiente ou desconhecido quando a pontuação do pior resultado é abaixo do valor limiar. O maior desafio é achar uma métrica de similaridade adequada para a comparação de características facial.

2.2 Aprendizado de Máquina

O Aprendizado de Máquina é uma técnica de análise de dados que ensina os computadores a fazer o que vem naturalmente aos seres humanos e aos animais: aprender com a experiência [14]. Os algoritmos de aprendizado de máquina usam métodos computacionais para "aprender"informações diretamente de dados sem depender de uma equação predeterminada como modelo. Os algoritmos melhoram de forma adaptativa seu desempenho à medida que o número de amostras disponíveis para o aprendizado aumenta.

Tornou-se a chave para a solução de problemas em áreas como: finança computacional, processamento de imagem e visão computacional, biologia computacional, produção de energia, au-

tomotiva, aeroespacial, fabricação e processamento de linguagem natural. A tomada de decisão e previsões é facilitada com os algoritmos de aprendizado de máquina, que encontram padrões naturais nos dados.

Redes neurais, árvores de decisões, hipóteses bayesianas, mapas auto-organizáveis, instâncias de aprendizado, modelos de Markov e dezenas de outras ferramentas de regressão fazem parte das técnicas de aprendizado de máquina [15].

2.3 Redes Neurais Artificiais

As Redes Neurais Artificiais tentam simular o cérebro humano e são implementadas utilizando componentes eletrônicos ou simuladas por programação em um computador digital. O tipo de organização dos neurônios está relacionado com o algoritmo de aprendizado usado para treinar a rede. A função da rede é de classificar as imagens em face e não-face [2].

Perceptron é um tipo de neurônio artificial desenvolvido nos anos de 1950 e 1960 pelo cientista Frank Rosenblatt. Atualmente, outros modelos de neurônios artificias são mais utilizados, sendo um deles o neurônio *sigmoid*. Entender o funcionamento dos perceptrons é fundamental no entendimento dos tipos que o sucederam. Um *perceptron* recebe várias entradas binárias e produz uma única saída binária. Cada entrada tem um peso e a saída é determinada se a soma ponderada é maior ou menor que algum valor limiar. Uma rede neural *perceptron* é então treinada para aprender algoritmos desejáveis. No entanto, uma pequena mudança nos pesos de qualquer *perceptron* pode causar uma saída totalmente diferente. O neurônio *sigmoid* surgiu para solucionar esse problema. Eles são similares aos perceptrons, mas modificados para que mudanças pequenas nos pesos causem pequenas mudanças nas saídas. O neurônio *sigmoid* também tem várias entradas, mas pode receber valores entre 0 e 1 e sua saída é definida pela função sigmóide representada pela Eq. 2.1:

$$f(z) = \frac{1}{1 + e^{-z}} \tag{2.1}$$

2.3.1 Arquitetura das RNAs

O que define a arquitetura de uma rede neural é a forma como os neurônios são organizados e interconectados. Há quatro tipos de arquiteturas: redes *feedfoward* unicamada, redes *feedfoward* multicamada, redes recorrentes e estrutura de *Lattice*.

Na rede da Fig. 2.2 a camada da esquerda é chamada de camada de entrada, e os neurônios dessa camada são chamados de neurônios de entrada. A camada da direita é a camada de saída, e o neurônio nessa camada é chamado de neurônio de saída. A camada intermediária é chamada de camada oculta (*hidden layer*), por causa dos seus neurônios que não são nem de entrada e nem de saída. Nessa rede há somente uma camada oculta, mas outras redes podem ter duas ou mais camadas ocultas. As redes que tem mais de três camadas são chamadas de *multilayer perceptrons*, apesar de ser composta por neurônios *sigmoids* e não *perceptrons*.

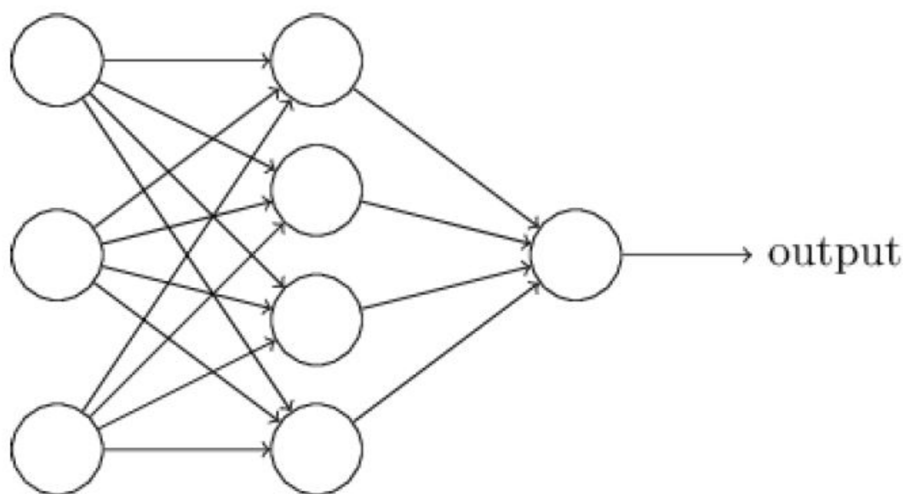

Figura 2.2: Arquitetura das RNAs [2]

As redes neurais onde a saída de uma camada se torna a entrada da camada seguinte são chamadas de redes neurais *feedforward*. Isso significa que não há *loops* na rede, a informação sempre é passada para a frente e nunca para trás. Os modelos de redes neurais artificiais onde é possível ter *loops* de *feedback* são chamados de redes neurais recorrentes. A ideia nesses modelos é ter neurônios que disparem por um certo período de tempo limitado, antes de ficarem "em repouso". Os disparos estimulam outros neurônios, que irão disparar um tempo depois, também por tempo limitado. Isso causa uma cascata de neurônios disparando. Nesses modelos os *loops* não causam problemas, pois a saída de um neurônio só irá afetar sua entrada um tempo depois e não instantaneamente. As redes neurais recorrentes são menos utilizadas do que as *feedfoward*, pois os algoritmos de aprendizagem para as redes neurais recorrentes são menos eficazes.

2.3.2 Algoritmo de *Backpropagation*

Usado para treinar redes MLP, este algoritmo é baseado na regra de aprendizado por correção de erro. A camada de entrada da rede recebe um padrão que flui através da rede até a camada de saída. A saída obtida é então comparada com a saída desejada para o padrão em particular. Se essa saída estiver errada, o erro é calculado e então propagado a partir da camada de saída até a camada de entrada. Conforme o erro é retro-propagado, os pesos das conexões das unidades das camadas internas vão sendo modificados.

2.3.3 Rede Neural Convolucional

Uma Rede Neural Convolucional é composta por uma ou mais camadas convolucionais e seguida por uma ou mais camadas totalmente conectadas como em uma rede neural multicamada padrão. Sua arquitetura é projetada para tirar proveito da estrutura 2D de uma imagem de entrada. CNNs são mais fáceis de treinar e possuem menos parâmetros do que redes totalmente conectadas com o mesmo número de unidades ocultas. O treinamento é uma calibração dos pesos, calibra-se de

forma que a rede responda ao que se é desejado. De forma que ao injetar os dados a rede vai saber processá-los. Três ideias básicas são usadas: campos receptivos locais, pesos compartilhados (*shared weights*) e *pooling*.

2.3.3.1 Arquitetura CNNs

A arquitetura dessas redes é adaptada para classificar imagens e faz com que sejam mais rápidas de treinar. A entrada para uma camada convolucional é uma imagem $\mathbf{m} \times \mathbf{m} \times \mathbf{r}$, onde m é a altura e largura da imagem e r é o número de canais (em uma imagem RGB $r = 3$). A camada convolucional terá k *kernels* com tamanho $\mathbf{n} \times \mathbf{n} \times \mathbf{q}$, onde n é menor do que a dimensão da imagem e q pode ser igual ou menor que número de canais r e pode variar para cada *kernel*. As estruturas conectadas localmente são originárias do tamanho dos *kernels* e cada uma é convoluída com a imagem para produzir mapas de característica k e de tamanho $m - n + 1$. Os mapas são então sub amostrados com *pooling* médio ou máximo sobre regiões contíguas $\mathbf{p} \times \mathbf{p}$, onde p varia entre 2 para imagens pequenas e 5 para entradas maiores. Antes ou depois da camada de sub amostra é aplicada em cada mapa de característica uma polarização aditiva e não linearidade *sigmoidal*.

- **Campos receptivos locais**: Cada neurônio da primeira camada oculta estará conectado a uma pequena região dos neurônios de entrada. Essa região na imagem de entrada é chamada de campo receptivo local para neurônios ocultos.

- **Pesos compartilhados**: Todos os neurônios da primeira camada oculta detectam exatamente a mesma característica, porém em locais diferentes da imagem de entrada. Por isso o mapa da camada de entrada para a camada oculta é chamado de mapa de características. Os pesos que definem esse mapa são os pesos compartilhados. Os pesos compartilhados e os bias definem o *kernel*. Uma camada convolucional completa consiste de vários mapas de características diferentes. Uma vantagem de compartilhar pesos e biases é a redução no número de parâmetros envolvidos em uma rede convolucional. No geral, os bias são somente para ajustes.

- **Camadas de *pooling***: São usadas imediatamente após as camadas convolucionais. Refinam as características que acabaram de ser extraídas, simplificando a informação na saída da camada convolucional. A camada de *pooling* pega a saída de cada mapa de característica da camada convolucional e prepara um mapa de características mais compacto. Cada unidade da camada de *pooling* resume uma região $\mathbf{p} \times \mathbf{p}$ da camada anterior. Um procedimento comum, mas não o único, é o *max-pooling*, onde uma unidade de *pooling* produz a ativação máxima na região $\mathbf{p} \times \mathbf{p}$. *Max-pooling* é aplicado a cada mapa de característica separadamente e é uma forma da rede perguntar se uma certa característica é encontrada em algum lugar da imagem.

2.4 *Deep Learning*

É um tipo de *Machine Learning* relacionado às Redes Neurais Artificiais. Os algoritmos dominantes de *Deep Learning* são Redes Neurais Profundas (*Deep Neural Networks*) que são redes neurais construídas a partir de muitas camadas de unidades de processamento (neurônios) lineares e não-lineares alternadas e são treinadas usando algoritmos de grande escala e grandes quantidades de treinamento de dados. Enquanto redes neurais comuns têm somente algumas camadas ocultas, uma rede neural profunda pode ter de 10 a 20 camadas ocultas. Quanto mais camadas uma rede tiver, mais características ela poderá reconhecer. No entanto, mais tempo ela demorará para calcular e mais difícil será para treiná-la.

Deep Learning foi a solução para o desafio da Inteligência Artificial, que era resolver tarefas que são fáceis de executar para as pessoas, mas difíceis de descrever formalmente. Problemas que são resolvidos intuitivamente, como reconhecer rostos em imagens. *Deep Learning* aprende a representar o mundo como uma hierarquia de conceitos, com cada conceito definido em relação a conceitos mais simples e representações mais abstratas computadas em termos de representações menos abstratas. A rede profunda mais usada é a Rede Convolucional Profunda.

2.5 Análise de Componentes Principais

A PCA é uma técnica usada para enfatizar a variação e traçar padrões em um conjunto de dados [16]. É usado muitas vezes para tornar os dados mais fáceis de explorar e visualizar. Sua ideia é reduzir a dimensionalidade de uma base de dados que consiste de um grade número de variáveis inter-relacionadas, mantendo o máximo possível da variação presente no conjunto de dados. Consegue-se isso transformando para um novo conjunto de variáveis, os Principais Componentes, que não estão correlacionados, e que são ordenados para que os primeiros tenham a maior parte da variação presente em todas as variáveis originais.

2.6 Aprendizado Supervisionado

A maioria das práticas de aprendizado de máquina utiliza o aprendizado supervisionado [17]. O aprendizado supervisionado é uma tarefa de mineração de dados para inferir uma função de dados de treinamento rotulados, onde os dados de treinamento são um conjunto de exemplos de treinamento [18]. Cada exemplo é um par que consiste de um objeto de entrada e o valor de saída desejado.

Um algoritmo de aprendizado supervisionado analisa os dados de treinamento e produz uma função inferida que pode ser usada para mapear novos exemplos. O algoritmo é capaz de determinar corretamente os rótulos das classes para instâncias não vistas em um cenário ideal. No entanto, isso exige que o algoritmo de aprendizado generalize a partir dos dados de treinamento para situações não vistas de forma razoável.

- **Classificação**: Quando a variável de saída é uma categoria, como "vermelho"ou "azul".
- **Regressão**: Quando a variável de saída é um valor real, como "peso", "dólares".

2.7 Aprendizado Não-Supervisionado

O aprendizado não-supervisionado é quando se tem somente o dado de entrada e nenhuma variável de saída correspondente [17]. O objetivo é modelar a estrutura ou a distribuição subjacente nos dados, a fim de aprender mais sobre os dados. Não há respostas corretas neste caso.

- ***Clustering***: Quando deseja-se descobrir os agrupamentos inerentes aos dados.
- **Associação**: Quando deseja-se descobrir regras que descrevem grandes porções de seus dados.

2.8 Algoritmo de Árvore de Decisão

O algoritmo de Árvore de Decisão está dentro do grupo de algoritmos de aprendizado supervisionado e pode ser usado para resolver problemas tanto de classificação quanto de regressão.

O motivo geral de usar Árvore de Decisão é criar um modelo de treinamento que possa ser usado para prever a classe ou o valor das variáveis alvo a partir do aprendizado de regras de decisão inferidas de dados anteriores (dados de treinamento) [3].

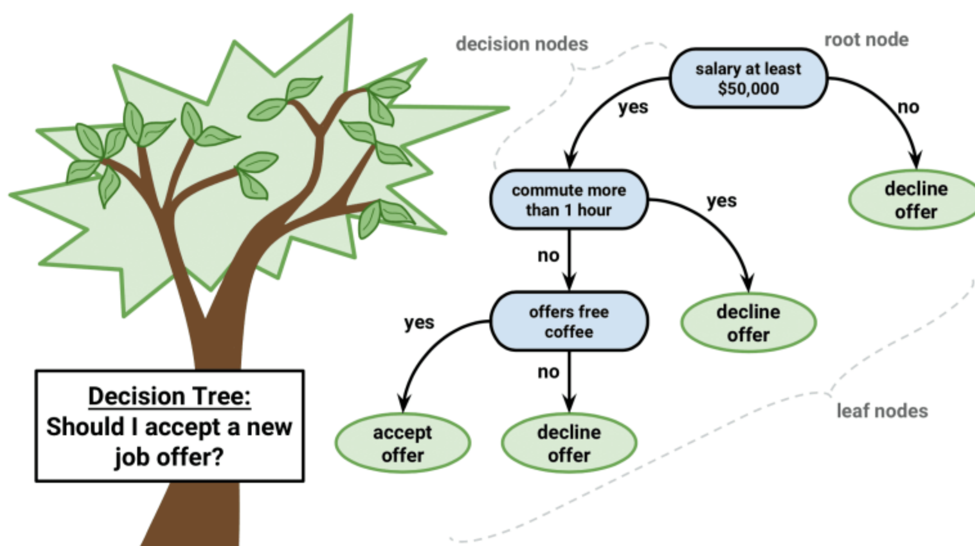

Figura 2.3: Algoritmo de Classificação de Árvore de Decisão [3]

Os problemas são resolvidos usando a representação em árvore. Na Fig. 2.3 são mostrados o nó raiz (*root node*), os nós de decisão (*decision nodes*) e os nós de folha (*leaf nodes*). Os nós internos correspondem a um atributo e cada nó de folha corresponde a um rótulo de classe.

2.8.1 Pseudocódigo do Algoritmo de Árvore de Decisão

O melhor atributo da base de dados fica na raiz da árvore (Fig. 2.3). O conjunto de treinamento é dividido em subconjuntos de forma que cada subconjunto contenha dados com o mesmo valor para um atributo. Isso é feito até que se possa encontrar nós de folha em todos os galhos da árvore.

Nas árvores de decisão, para prever um rótulo de classe para um registro, inicia-se a partir da raiz da árvore. Compara-se então os valores do atributo raiz com o atributo do registro. Com base na comparação, segue-se o ramo correspondente a esse valor e então para o próximo nó. Continua-se a comparar os valores dos atributos do registro com outros nós internos da árvore até alcançar um nó de folha com valor de classe previsto.

2.9 SVM

Uma máquina de vetores de suporte é um dos algoritmos de aprendizado de máquina mais populares. Se tornou popular por volta de 1990 quando foi desenvolvido e continua a ser o método escolhido para algoritmos de alta performance com pouca afinação [19]. É baseado no conceito de planos de decisão que definem limites de decisão. Um plano de decisão é aquele que se separa entre um conjunto de objetos com diferentes associações de classe, como na Fig. 2.4. Um clássico exemplo de classificador linear, onde os objetos verdes estão do lado direito e os objetos vermelhos do lado esquerdo. Qualquer objeto novo que caia do lado direito será classificado como verde (ou vermelho se cair do lado esquerdo).

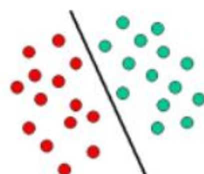

Figura 2.4: Classificador Linear Clássico [4]

Como a maioria das situações não são simples assim, o SVM é adequado para tarefas mais complexas de classificadores de hiperplano. Usando um conjunto de funções matemáticas, conhecidas como *kernels*, para rearranjar os objetos, como na Fig. 2.5. Este processo é chamado de mapeamento. Tudo o que precisa ser feito é achar a linha ideal que separa os objetos vermelhos dos verdes.

SVM é então um método classificador que executa tarefas de classificação construindo hiperplanos em um espaço multidimensional que separa casos de rótulos de classes diferentes [4]. Suporta tarefas de classificação e regressão.

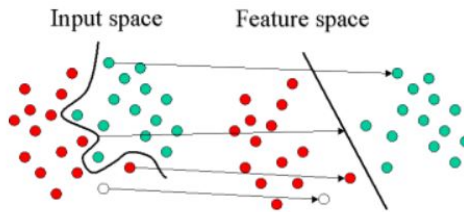

Figura 2.5: Ideia básica do SVM [4]

2.10 *Image Augmentation*

A aquisição de dados para treinamento com imagens *cross-domain* é difícil, portanto usa-se *image augmentation* para construir um conjunto de dados maior a partir do já existente. *Image Augmentation* cria artificialmente imagens de treinamento através de diferentes formas de processamento ou combinação de múltiplos processamentos, como rotação aleatória, deslocação, cortes, alteração no brilho, contraste, acréscimo de *noise*, etc.

Na Fig. 2.6 mostra-se variações de uma mesma imagem através de diferentes processos de *augmentation*. Provê mais imagens para treinamento e ajuda a expor o classificador a uma variedade maior de situações de iluminação e coloração.

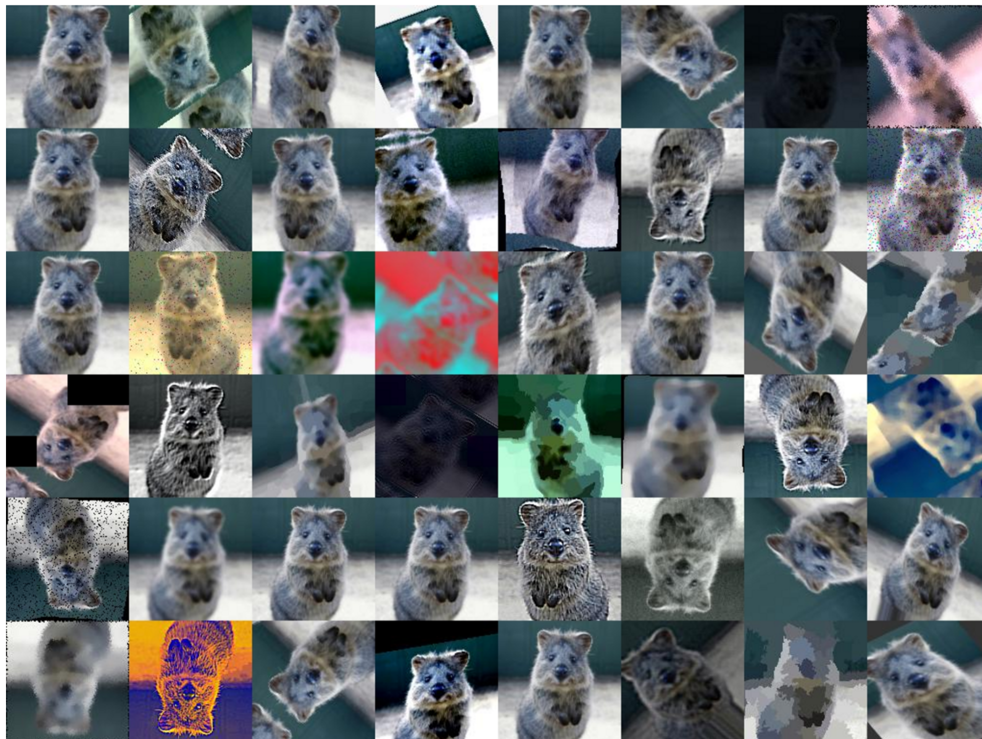

Figura 2.6: Variações de uma mesma imagem através de *augmentation* [5]

2.11 *Cross Validation*

Cross Validation é um método de avaliação modelo melhor do que o residual, tendo em vista que as avaliações residuais não dão uma indicação de quão bem o programa fará quando solicitado a fazer novas previsões de dados que ainda não tenha visto [20]. Uma maneira de superar esse problema é não usar todo o conjuntos de dados ao realizar um treinamento. Parte dos dados são removidos antes do treinamento. Parâmetros de uma função de previsão são aprendidos e testados nos dados que foram removidos, testando o desempenho do modelo aprendido nos dados "novos".

2.11.1 Método *holdout*

Este método é o mais simples de *cross validation*. Os dados são separados em dois conjuntos, de treinamento e de teste. O treinamento é realizado no conjunto de treinamento e o modelo aprendido é então testado no conjunto de testes, que é totalmente novo. Os erros são acumulados para dar o erro absoluto médio do conjunto de teste, que é usado para avaliar o modelo. No entanto, sua avaliação pode ter uma alta variação, levando em consideração que a avaliação depende de quais dados acabam no conjunto de treinamento e quais acabam no conjunto de testes. A avaliação pode ser significativamente diferente dependendo de como a divisão é feita.

2.11.2 *K-fold*

É uma melhoria do método *holdout*. Os dados são divididos em K conjuntos e o método *holdout* é repetido K vezes. A cada vez um dos K subconjuntos é usado como teste e os outros $K - 1$ subconjuntos formam juntos o conjunto de treinamento. O erro médio é computado após os K processos. A vantagem é que não importa tanto como os dados foram divididos, pois todos os dados estarão no conjunto de teste em algum ponto. Quanto maior for K menor a variação no resultado estimado.

2.11.3 *Leave-One-Out*

Esta validação cruzada é a *K-fold* levada ao extremo lógico, com $K = N$, o número de pontos de dados no conjunto. Onde N vezes separadas, o aproximador de função é treinado em todos os dados, excetor por um ponto e uma previsão é feita para esse ponto.

Capítulo 3

Método Proposto e Ferramentas Utilizadas

A partir do problema apresentado no Capítulo 1 será descrita a proposta para este trabalho, o banco de imagens e as ferramentas de código aberto aqui utilizadas para que, ao final do projeto, seja feita uma comparação e análise entre elas.

3.1 Descrição do Método

A proposta consiste em usar a *OpenFace* e os algoritmos de classificação *Random Forest* e *PmSVM* para treinar e testar a partir do banco de imagens construído para este projeto. A *OpenFace* faz desde a detecção facial até a classificação, incluindo extração de características. Para a utilização dos algoritmos *Random Forest* e *PmSVM*, primeiro foi feito o pré-processamento e a extração de características das imagens com a rede já treinada da VGG.

Todos os experimentos foram realizados em VMs. Os experimentos com *Random Forest* e *PmSVM* foram realizados na mesma VM, com Ubuntu Desktop 16.04.3 LTS e o experimento com *OpenFace* foi realizado em uma VM com Ubuntu Desktop 14.04 LTS, como indicado em [21].

O banco de imagens, os algoritmos e softwares serão descritos nas próximas seções.

3.2 Banco de Imagens

O bando de imagens teve que ser construído, visto que não há um que seja público e disponível para o problema de verificação facial com imagens de domínios diferentes. Contém 50 indivíduos, com duas imagens cada. Sendo uma imagem o autorretrato (*selfie*) e uma imagem do documento de identidade, ambas tiradas pelo próprio usuário. Dos 50 indivíduos são 28 homens e 22 mulheres. Entre as duas fotos de um mesmo indivíduo pode haver variação de pose, iluminação, presença/ausência de barba, diferença de idade, entre outros.

O processo de *augmentation* é realizado obtendo 100 novas imagens que são caracterizadas como pertencendo a 50 novos indivíduos, com duas imagens cada. O banco de imagens passa a conter 200 imagens no total e 100 indivíduos. O número de homens e mulheres simplesmente dobra, sendo então 56 homens e 44 mulheres.

3.3 LFW

O *Labeled Faces in the Wild* é um banco de fotografias de rosto projetado para estudar o problema de reconhecimento facial sem restrições [22]. O banco de dados contém mais de 13 000 imagens de faces coletadas da Internet. Cada face foi rotulada com o nome da pessoa na imagem. 1680 das pessoas retratadas têm duas ou mais fotos distintas no banco de dados.

Existem quatro conjuntos diferentes de imagens LFW que inclui o original e três de diferentes tipo de imagens "alinhadas". As imagens alinhas incluem *funneled images, LFW-a* e *deep funneled*.

3.4 *OpenFace*

OpenFace é uma implementação de reconhecimento facial com DNNs em *Python* e *Torch* com base em [23]. *Torch* permite que a rede seja executada em uma CPU ou com CUDA. Na Fig. 3.1 tem-se em uma visão geral o fluxo para uma única imagem que é descrito a seguir [6]:

1. Detectar faces com modelos pré-treinados do *dlib* ou *OpenCV*.
2. Transformar a face para a rede neural. Usando a estimativa de pose em tempo real da *dlib* e a transformação que preserva a colinearidade do *OpenCV* para tentar deixar os olhos e lábios na mesma localização em cada imagem.
3. Usar uma DNN para representar a face em 128D. Onde uma distância grande entre duas faces representadas significa que não pertencem a mesma pessoa.
4. Aplicar a técnica de agrupamento ou classificação desejada para completar o reconhecimento.

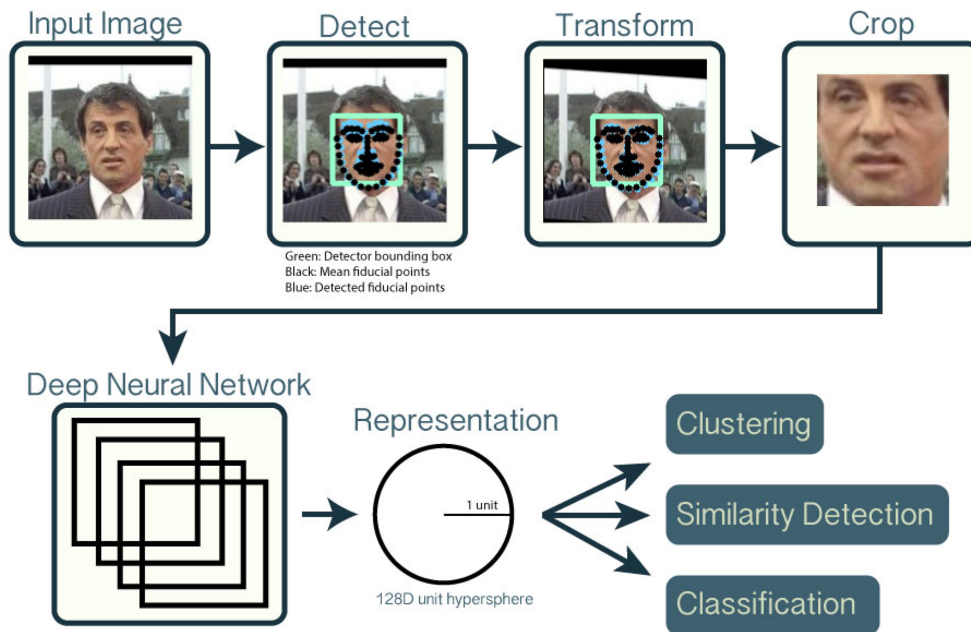

Figura 3.1: Fluxo para uma única imagem [6]

3.4.1 Reconhecimento Facial com Redes Neurais

Técnicas usadas no *DeepFace* [24] do *Facebook* e na *FaceNet* [23] da *Google* são usadas no *OpenFace*. Uma rede neural do tipo *feedforward* consiste em muitas composições de função, ou camadas, seguidas por uma função de perda L. Tal função mede o quão bem uma rede neural modela os dados, por exemplo, o quão precisa a rede neural classifica uma imagem. Cada camada i é parametrizada por θ_i, que pode ser um vetor ou matriz.

Operações de camadas comuns:

- Convoluções espaciais que deslizam um *kernel* sobre os mapas de características de entrada;
- Camadas lineares ou totalmente conectadas que levam uma soma ponderada de todas as unidades de entrada; e
- *Pooling* que tira o máximo, média ou norma Euclidiana de regiões espaciais.

O treinamento de rede neural é um problema de otimização que encontra um θ que minimiza (ou maximiza) L. Na Fig. 3.2 vemos o fluxo deste treinamento.

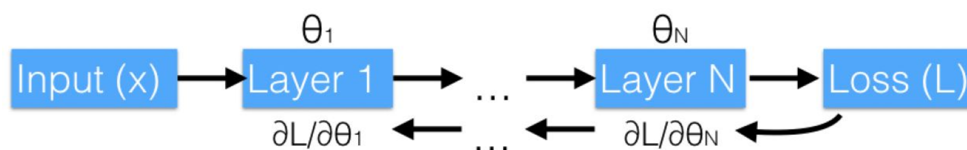

Figura 3.2: Fluxo de treinamento para uma rede neural *feedforward* [7]

Primeiro ocorre a detecção facial, então o sistema pré-processa cada face na imagem para criar uma entrada normalizada e de tamanho fixo para a rede neural. A rede neural é usada para produzir uma representação de baixa dimensão que caracterize a face de uma pessoa, o que é a chave para técnicas de classificação ou agrupamento.

A representação facial de baixa dimensão para as faces em uma imagem é obtida pelo fluxo apresentado na Fig. 3.3. O treinamento da rede neural usa *Torch*, *Lua* e *luajit*. A biblioteca *Python* usa *numpy* para arrays e operações de álgebra linear, *OpenCV* para visão computacional, e *scikit-learn* para classificação. A arquitetura de rede neural usada é a do *FaceNet* [23], que usa rede convolucional profunda. O detector facial pré-treinado do *dlib* é usado para maior acurácia ao invés do detector do *OpenCV*.

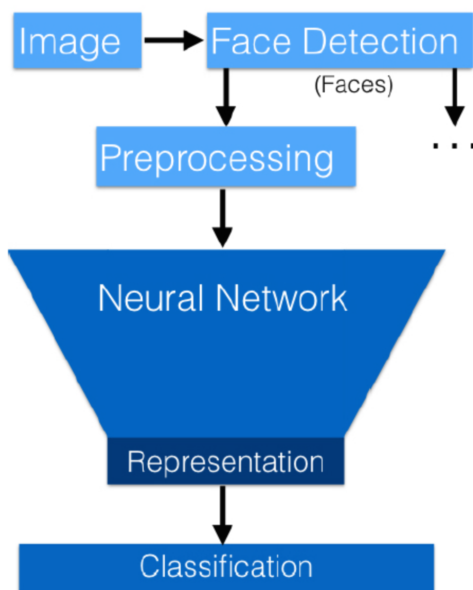

Figura 3.3: Fluxo de processamento da *OpenFace* [7]

3.4.2 Pré-processamento

A parte de detecção facial retorna uma lista de caixas de delimitação em volta das faces em uma imagem, que podem estar em condições diferentes de iluminação e pose. A normalização facial, que faz com que os olhos, nariz e boca apareçam em lugares similares em cada foto, é usada para reduzir o tamanho do espaço de entrada. Para isso é usado uma transformação 2D simples. Nela são detectados 68 pontos de referência com o detector de ponto de referência facial da *dlib*. Então, com uma imagem de entrada, os cantos dos olhos e o nariz ficam próximos dos locais médios. A imagem também é redimensionada e cortada nas bordas dos pontos de referência para que a imagem de entrada da rede neural tenha 96x96 pixels.

3.4.3 Treinamento da Rede Neural de Representação Facial

O treinamento de redes neurais requer muitos dados. No entanto, não há um grande conjunto de dados disponíveis, somente os conjuntos de dados privados contém milhões de imagens, como por exemplo o do *FaceNet* [23] que contém 100M-200M imagens e o do *DeepFace* [24] que contém 4.4M imagens. O conjunto de dados usado para treinar as redes neurais do *OpenFace* contém 500k imagens, que é a combinação dos conjuntos de dados *CASIA-WebFace* e *FaceScrub*. Uma versão modificada da rede *nn4* usada no *FaceNet* [23] é adaptada para a base de dados menor, com menos parâmetros, a *nn4.small2*. É usado a *triplet loss* (perda do trio) do *FaceNet* [23]. As redes são treinadas mapeando imagens únicas de uma rede em trios. O gradiente da *triplet loss* é então retropropagado através do mapeamento até as imagens únicas. Em cada mini lote, P imagens por pessoa de Q pessoas são amostradas na base de dados e são enviadas $M \approx PQ$ imagens pela rede. São tomados todos os pares âncora-positivos para obter o trio $N = Q\binom{P}{2}$. É calculado a perda do trio e mapeado a derivada de volta para a imagem original em uma rede *backwards*. Se uma imagem negativa não for encontrada dentro da margem α para um determinado par âncora-positivo, então não são usados.

3.4.4 Modelos

O *OpenFace* possui quatro modelos de redes neurais, mostrados na Tab. 3.1. O número de parâmetros é com incorporação de 128D e não inclui os meios de execução e as variações de normalização de lote.

Modelo	Número de Parâmetros
nn4.small2	3 733 968
nn4.small1	5 579 520
nn4	6 959 088
nn2	7 472 144

Tabela 3.1: Modelos de redes neurais e número de parâmetros [9]

Modelos podem ser treinados de diversas formas e com diferentes bases de dados. Os modelos pré-treinados possuem versões e são liberados com a definição de modelo correspondente. Os modelos atuais são treinados com uma combinação de dois bancos de dados públicos para reconhecimento facial, o *FaceScrub* e *CASIA-WebFace* [9].

3.4.5 Acurácia

Apesar dos bancos de dados públicos usados para treinar terem ordens de magnitude menos dados do que os bancos de dados da indústria privada, a precisão é notavelmente alta no *benchmark* padrão LFW. O *dlib* falhou em 58 imagens de 13233 ao detectar faces ou pontos de referência, então foi necessário voltar para as versões *deep funneled* dessas imagens. As acurácias de cada modelo estão na Tab. 3.2:

Modelo	Acurácia
nn4.small2.v1 (*Default*)	0.9292 ± 0.0134
nn4.small1.v1	0.9210 ± 0.0160
nn4.v2	0.9157 ± 0.0152
nn4.v1	0.7612 ± 0.0189
FaceNet Paper [23]	0.9963 ± 0.009

Tabela 3.2: Modelos de redes neurais e acurácia em testes no LFW [9]

3.5 *VGG Face Descriptor*

Criado pelo *Visual Geometry Group*, o *VGG Face Descriptor* é computado usando a implementação CNN baseada na arquitetura *VGG-Very-Deep-16* CNN [8]. As redes usadas são muito profundas no sentido de que elas compreendem uma longa sequência de camadas de convolução. Usa-se *triplet loss* e verificação de identidades por comparação de descritores de faces no espaço euclidiano. Com um treinamento apropriado, uma CNN profunda alcança resultados comparáveis ao estado da arte.

3.5.1 Arquitetura

A rede VGG tem 38 camadas. A arquitetura da CNN é formada por 11 blocos onde cada um contém um operador linear seguido de um ou mais não lineares, como *max pooling* ou *ReLU* [25]. Os oito primeiros blocos são convolucionais e o operador linear é um banco de filtros lineares. Os três últimos blocos são chamados de totalmente conectados, onde o tamanho dos filtros é o mesmo do dado de entrada. Todas as camadas convolucionais são seguidas de uma camada de retificação (*ReLU*). A imagem de entrada das redes é uma imagem de rosto de tamanho 224x224 com a imagem de rosto média (computada a partir do conjunto de treinamento) subtraída.

3.6 *Caffe*

Caffe é um *framework* de *deep learning* de código aberto, com expressão, velocidade e modularidade em mente [26]. Desenvolvido pela *Berkeley AI Research* (BAIR) e por contribuintes da comunidade. O código é escrito em C++ com CUDA e suporte para *Python/Numpy* e *MatLab*. Apesar de ter sido projetado para visão computacional, foi adotado e aperfeiçoado por usuários em reconhecimento de fala, robótica, neurociência e astronomia [27]. BVLC mantém e desenvolve o *Caffe* com o esforço de estudantes de pós-graduação.

3.6.0.1 Arquitetura

Os dados são armazenados em *arrays* de 4 dimensões, os chamados *blobs*. Os *blobs* fornecem uma interface de memória unificada, fixando lotes de imagens (ou outros dados), parâmetros ou

atualizações de parâmetros. Os *blobs* escondem a sobrecarga computacional e mental da operação mista de CPU / GPU, sincronizando do hospedeiro da CPU com o dispositivo GPU conforme necessário [27].

Uma camada do Caffe é a essência de uma camada de rede neural: ela leva um ou mais *blobs* como entrada e produz um ou mais *blobs* como saída. Um conjunto de tipos de camadas é fornecido, como: convolução, agrupamento, produtos internos, não linearidades como rectificação linear e logística, normalização de resposta local, operações elementares e perdas. Uma rede típica começa com uma camada de dados que é carregada a partir do disco e termina com uma camada de perda que calcula o objetivo de uma tarefa, como classificação ou reconstrução. O treinamento da rede é feito com uma camada de dados que obtém as imagens e os rótulos do disco, passa por várias camadas, como convolução, agrupamento e transformações lineares rectificadas, e alimenta a previsão final em uma camada de perda de classificação que produz a perda e os gradientes que treinam toda a rede.

3.7 *PmSVM*

O *Power Mean SVM (PmSVM)* é escrito em C++ e pode ser usado tanto em Linux quanto em Windows [28]. Este *software* visa resolver problemas de classificação em grande escala, principalmente os de visão computacional.

As imagens usadas para treinamento e teste devem estar no formato *libsvm*, como mostrado abaixo:

$$< label >< index1 >:< value1 >< index2 >:< value2 > ... \tag{3.1}$$

Cada linha contém uma instância e é encerrada por um caractere '\n'. Para classificação, <label> é um número inteiro que indica o rótulo da classe. <index> é um número inteiro a partir de 1 e <value> é um número real que deve estar entre 0 e 1. Valores de características negativos e maiores do que 1 são aceitos, mas há o risco de reduzir a acurácia. Os índices devem estar em ordem crescente. Rótulos dos arquivos de teste são usados somente para o cálculo da acurácia e erros. Se forem desconhecidos pode-se preencher a primeira coluna com qualquer número.

A conversão dos dados para o intervalo [0,1] é feita com a ferramenta *svm-scale* no pacote do *software libsvm*:

```
./svm-scale -l 0 -u 1 -s range.txt trainset > trainset.scale
./svm-scale -r range.txt testset > testset.scale
```

3.7.1 Acurácia

PmSVM reporta dois tipos de acurácia, *overall accuracy* (acurácia geral) e *average accuracy* (acurácia média). A acurácia geral é calculada da seguinte forma:

$$\frac{c_1 + c_2 + ... + c_m}{n}$$

O cálculo da acurácia média é o seguinte:

$$(\frac{c_1}{n_1} + \frac{c_2}{n_2} + ... + \frac{c_m}{n_m})/m$$

Onde m é o número de classes de uma base de dados, com $n_1, n_2, ..., n_m$ exemplos. O tamanho da base de dados é $n = n_1 + n_2 + ... + n_m$ e c_i é o número de exemplos na classe i que são corretamente previstos.

A acurácia geral é usada em aprendizado de máquina enquanto a acurácia média é mais popular em visão computacional. Ambas são reportadas pela *PmSVM*.

3.8 *Random Forest*

Random Forest é um algoritmo de classificação de aprendizado supervisionado e é o mais popular. É implementado em *python* com a biblioteca *scikit-learn*. Quanto maior o número de árvores de decisão, maior a acurácia dos resultados.

3.8.1 Funcionamento

1. Aleatoriamente seleciona-se **k** características de um total de **m** características, onde **k « m**.
2. Calcula-se o nó **d** dentre as **k** características usando o melhor ponto de divisão.
3. Divide-se o nó em nós filhos usando a abordagem de melhor divisão novamente.
4. Repete-se os passos 1-3 até que atinga l nós.
5. Constrói-se a floresta repetindo os passos 1-4 por n vezes para criar **n** árvores. Cria-se então a *random forest* (floresta aleatória).

A predição é feita usando o algoritmo *random forest* treinado. Para isso, primeiro toma-se as características de teste e usa-se as regras de cada árvore de decisão criada aleatoriamente para prever o resultado e armazena-o (alvo). Após, calcula-se os votos para cada alvo previsto e então considera-se o alvo previsto mais votado como a predição final do algoritmo *random forest*.

Capítulo 4

Análises e Resultados

Neste capítulo são apresentados os testes realizados em cada ferramenta e na seção 4.6 são apresentados os resultados.

Os testes são realizados com as três ferramentas (*OpenFace*, PmSVM e *Random Forest*) em dois cenários.

- **Cenário 1**: 50 indivíduos, 2 imagens cada, 100 imagens no total.
- **Cenário 2**: Após realizar *augmentation*, 100 indivíduos, 2 imagens cada, 200 imagens no total.

Como foi dito na Seção 3.2, não há um banco de imagens público e disponível para o problema de verificação facial com imagens de domínios diferentes. Sendo assim, o número de indivíduos utilizado no Cenário 1 deve-se simplesmente ao fato de que foi o que se conseguiu obter. No Cenário 2, a quantidade de indivíduos é o que se tinha no Cenário 1 mais 50 novos indivíduos obtidos por *augmentation*. Usa-se dois cenários para que se possa comparar o efeito que o aumento no número de imagens pode causar na acurácia das ferramentas aqui utilizadas.

4.1 Image Augmentation

A *augmentation* é feita executando três *scripts* em ordem:

1. *data_ aug_ apply_ effects_ CLAHE.py*
2. *data_ aug_ apply_ effects_ BRIGHT.py*
3. *data_ aug_ apply_ effects_ NOISE.py*

O *script* 1 aumenta o contraste local com o método CLAHE e adiciona '.C' ao nome de cada imagem. Observa-se, pelos nomes, que na Fig. 4.1 a entrada são as 100 imagens originais, onde *0001_ id.png* e *0001_ selfie.png* são as duas imagens de um mesmo indivíduo (imagem do documento

pessoal e o autorretrato), que neste caso é o indivíduo 0001. Na Fig. 4.2 tem-se o resultado do *script* 1, onde foi adicionado '.C' ao nome de todas as imagens, diferenciando-as das originais.

```
root@8fc162e4ae6f:/home/pfg/scripts# ./data_aug_apply_effects_CLAHE.py
1 OK # 0001_id.png
2 OK # 0001_selfie.png
3 OK # 0002_id.png
4 OK # 0002_selfie.png
5 OK # 0003_id.png
6 OK # 0003_selfie.png
7 OK # 0004_id.png
8 OK # 0004_selfie.png
9 OK # 0005_id.png
10 OK # 0005_selfie.png
11 OK # 0006_id.png
12 OK # 0006_selfie.png
13 OK # 0007_id.png
14 OK # 0007_selfie.png
15 OK # 0008_id.png
16 OK # 0008_selfie.png
17 OK # 0009_id.png
18 OK # 0009_selfie.png
19 OK # 0010_id.png
20 OK # 0010_selfie.png
```

Figura 4.1: Execução *script* 1

```
100 OK # 0050_selfie.png
100 Arquivos Processados
root@8fc162e4ae6f:/home/pfg/scripts# ls ../imagens_aug_CLAHE
0001.C_id.png       0011.C_id.png       0021.C_id.png       0031.C_id.png       0041.C_id.png
0001.C_selfie.png   0011.C_selfie.png   0021.C_selfie.png   0031.C_selfie.png   0041.C_selfie.png
0002.C_id.png       0012.C_id.png       0022.C_id.png       0032.C_id.png       0042.C_id.png
0002.C_selfie.png   0012.C_selfie.png   0022.C_selfie.png   0032.C_selfie.png   0042.C_selfie.png
0003.C_id.png       0013.C_id.png       0023.C_id.png       0033.C_id.png       0043.C_id.png
0003.C_selfie.png   0013.C_selfie.png   0023.C_selfie.png   0033.C_selfie.png   0043.C_selfie.png
0004.C_id.png       0014.C_id.png       0024.C_id.png       0034.C_id.png       0044.C_id.png
0004.C_selfie.png   0014.C_selfie.png   0024.C_selfie.png   0034.C_selfie.png   0044.C_selfie.png
0005.C_id.png       0015.C_id.png       0025.C_id.png       0035.C_id.png       0045.C_id.png
0005.C_selfie.png   0015.C_selfie.png   0025.C_selfie.png   0035.C_selfie.png   0045.C_selfie.png
0006.C_id.png       0016.C_id.png       0026.C_id.png       0036.C_id.png       0046.C_id.png
0006.C_selfie.png   0016.C_selfie.png   0026.C_selfie.png   0036.C_selfie.png   0046.C_selfie.png
0007.C_id.png       0017.C_id.png       0027.C_id.png       0037.C_id.png       0047.C_id.png
0007.C_selfie.png   0017.C_selfie.png   0027.C_selfie.png   0037.C_selfie.png   0047.C_selfie.png
0008.C_id.png       0018.C_id.png       0028.C_id.png       0038.C_id.png       0048.C_id.png
0008.C_selfie.png   0018.C_selfie.png   0028.C_selfie.png   0038.C_selfie.png   0048.C_selfie.png
0009.C_id.png       0019.C_id.png       0029.C_id.png       0039.C_id.png       0049.C_id.png
0009.C_selfie.png   0019.C_selfie.png   0029.C_selfie.png   0039.C_selfie.png   0049.C_selfie.png
0010.C_id.png       0020.C_id.png       0030.C_id.png       0040.C_id.png       0050.C_id.png
0010.C_selfie.png   0020.C_selfie.png   0030.C_selfie.png   0040.C_selfie.png   0050.C_selfie.png
root@8fc162e4ae6f:/home/pfg/scripts#
```

Figura 4.2: Resultado *script* 1

O *script* 2 altera os efeitos de iluminação e adiciona 'B' ao nome de cada imagem. Na Fig. 4.3, pelos nomes, observa-se que as imagens de entrada são as imagens resultantes do *script* 1, com o contraste local já aumentado. Na Fig. 4.4 tem-se o resultado do *script* 2, com o 'B' adicionado ao nome de todas as imagens, diferenciando-as das originais e das resultantes do *script* 1.

```
root@8fc162e4ae6f:/home/pfg/scripts# ./data_aug_apply_effects_BRIGHT.py
1 OK # 0001.C_id.png
2 OK # 0001.C_selfie.png
3 OK # 0002.C_id.png
4 OK # 0002.C_selfie.png
5 OK # 0003.C_id.png
6 OK # 0003.C_selfie.png
7 OK # 0004.C_id.png
8 OK # 0004.C_selfie.png
9 OK # 0005.C_id.png
10 OK # 0005.C_selfie.png
11 OK # 0006.C_id.png
12 OK # 0006.C_selfie.png
13 OK # 0007.C_id.png
14 OK # 0007.C_selfie.png
15 OK # 0008.C_id.png
16 OK # 0008.C_selfie.png
17 OK # 0009.C_id.png
18 OK # 0009.C_selfie.png
19 OK # 0010.C_id.png
20 OK # 0010.C_selfie.png
```

Figura 4.3: Execução *script* 2

```
100 OK # 0050.C_selfie.png
100 Arquivos Processados
root@8fc162e4ae6f:/home/pfg/scripts# ls ../imagens_aug_CLAHE_BRIGHT
0001.CB_id.png       0013.CB_selfie.png   0026.CB_id.png       0038.CB_selfie.png
0001.CB_selfie.png   0014.CB_id.png       0026.CB_selfie.png   0039.CB_id.png
0002.CB_id.png       0014.CB_selfie.png   0027.CB_id.png       0039.CB_selfie.png
0002.CB_selfie.png   0015.CB_id.png       0027.CB_selfie.png   0040.CB_id.png
0003.CB_id.png       0015.CB_selfie.png   0028.CB_id.png       0040.CB_selfie.png
0003.CB_selfie.png   0016.CB_id.png       0028.CB_selfie.png   0041.CB_id.png
0004.CB_id.png       0016.CB_selfie.png   0029.CB_id.png       0041.CB_selfie.png
0004.CB_selfie.png   0017.CB_id.png       0029.CB_selfie.png   0042.CB_id.png
0005.CB_id.png       0017.CB_selfie.png   0030.CB_id.png       0042.CB_selfie.png
0005.CB_selfie.png   0018.CB_id.png       0030.CB_selfie.png   0043.CB_id.png
0006.CB_id.png       0018.CB_selfie.png   0031.CB_id.png       0043.CB_selfie.png
0006.CB_selfie.png   0019.CB_id.png       0031.CB_selfie.png   0044.CB_id.png
0007.CB_id.png       0019.CB_selfie.png   0032.CB_id.png       0044.CB_selfie.png
0007.CB_selfie.png   0020.CB_id.png       0032.CB_selfie.png   0045.CB_id.png
0008.CB_id.png       0020.CB_selfie.png   0033.CB_id.png       0045.CB_selfie.png
0008.CB_selfie.png   0021.CB_id.png       0033.CB_selfie.png   0046.CB_id.png
0009.CB_id.png       0021.CB_selfie.png   0034.CB_id.png       0046.CB_selfie.png
0009.CB_selfie.png   0022.CB_id.png       0034.CB_selfie.png   0047.CB_id.png
0010.CB_id.png       0022.CB_selfie.png   0035.CB_id.png       0047.CB_selfie.png
0010.CB_selfie.png   0023.CB_id.png       0035.CB_selfie.png   0048.CB_id.png
0011.CB_id.png       0023.CB_selfie.png   0036.CB_id.png       0048.CB_selfie.png
0011.CB_selfie.png   0024.CB_id.png       0036.CB_selfie.png   0049.CB_id.png
0012.CB_id.png       0024.CB_selfie.png   0037.CB_id.png       0049.CB_selfie.png
0012.CB_selfie.png   0025.CB_id.png       0037.CB_selfie.png   0050.CB_id.png
0013.CB_id.png       0025.CB_selfie.png   0038.CB_id.png       0050.CB_selfie.png
root@8fc162e4ae6f:/home/pfg/scripts#
```

Figura 4.4: Resultado *script* 2

O *script* 3 adiciona ruído Gaussiano na altura dos olhos e adiciona 'N' ao nome de cada imagem. Na Fig. 4.5, pelos nomes, observa-se que as imagens de entrada são as imagens resultantes do *script* 2, com o contraste local já aumentado e os efeitos de iluminação já alterados. Na Fig. 4.6 tem-se o resultado final, após a execução dos três *scripts*, com o 'N' adicionado ao nome de todas as imagens, diferenciando-as das originais e das resultantes do *script* 1 e 2.

```
root@8fc162e4ae6f:/home/pfg/scripts# ./data_aug_apply_effects_NOISE.py
1 OK # 0001.CB_id.png
2 OK # 0001.CB_selfie.png
3 OK # 0002.CB_id.png
4 OK # 0002.CB_selfie.png
5 OK # 0003.CB_id.png
6 OK # 0003.CB_selfie.png
7 OK # 0004.CB_id.png
8 OK # 0004.CB_selfie.png
9 OK # 0005.CB_id.png
10 OK # 0005.CB_selfie.png
11 OK # 0006.CB_id.png
12 OK # 0006.CB_selfie.png
13 OK # 0007.CB_id.png
14 OK # 0007.CB_selfie.png
15 OK # 0008.CB_id.png
16 OK # 0008.CB_selfie.png
17 OK # 0009.CB_id.png
18 OK # 0009.CB_selfie.png
19 OK # 0010.CB_id.png
20 OK # 0010.CB_selfie.png
```

Figura 4.5: Execução *script* 3

```
100 OK # 0050.CB_selfie.png
100 Arquivos Processados
root@8fc162e4ae6f:/home/pfg/scripts# ls ../imagens_aug_CLAHE_BRIGHT_NOISE/
0001.CBN_id.png       0013.CBN_selfie.png   0026.CBN_id.png       0038.CBN_selfie.png
0001.CBN_selfie.png   0014.CBN_id.png       0026.CBN_selfie.png   0039.CBN_id.png
0002.CBN_id.png       0014.CBN_selfie.png   0027.CBN_id.png       0039.CBN_selfie.png
0002.CBN_selfie.png   0015.CBN_id.png       0027.CBN_selfie.png   0040.CBN_id.png
0003.CBN_id.png       0015.CBN_selfie.png   0028.CBN_id.png       0040.CBN_selfie.png
0003.CBN_selfie.png   0016.CBN_id.png       0028.CBN_selfie.png   0041.CBN_id.png
0004.CBN_id.png       0016.CBN_selfie.png   0029.CBN_id.png       0041.CBN_selfie.png
0004.CBN_selfie.png   0017.CBN_id.png       0029.CBN_selfie.png   0042.CBN_id.png
0005.CBN_id.png       0017.CBN_selfie.png   0030.CBN_id.png       0042.CBN_selfie.png
0005.CBN_selfie.png   0018.CBN_id.png       0030.CBN_selfie.png   0043.CBN_id.png
0006.CBN_id.png       0018.CBN_selfie.png   0031.CBN_id.png       0043.CBN_selfie.png
0006.CBN_selfie.png   0019.CBN_id.png       0031.CBN_selfie.png   0044.CBN_id.png
0007.CBN_id.png       0019.CBN_selfie.png   0032.CBN_id.png       0044.CBN_selfie.png
0007.CBN_selfie.png   0020.CBN_id.png       0032.CBN_selfie.png   0045.CBN_id.png
0008.CBN_id.png       0020.CBN_selfie.png   0033.CBN_id.png       0045.CBN_selfie.png
0008.CBN_selfie.png   0021.CBN_id.png       0033.CBN_selfie.png   0046.CBN_id.png
0009.CBN_id.png       0021.CBN_selfie.png   0034.CBN_id.png       0046.CBN_selfie.png
0009.CBN_selfie.png   0022.CBN_id.png       0034.CBN_selfie.png   0047.CBN_id.png
0010.CBN_id.png       0022.CBN_selfie.png   0035.CBN_id.png       0047.CBN_selfie.png
0010.CBN_selfie.png   0023.CBN_id.png       0035.CBN_selfie.png   0048.CBN_id.png
0011.CBN_id.png       0023.CBN_selfie.png   0036.CBN_id.png       0048.CBN_selfie.png
0011.CBN_selfie.png   0024.CBN_id.png       0036.CBN_selfie.png   0049.CBN_id.png
0012.CBN_id.png       0024.CBN_selfie.png   0037.CBN_id.png       0049.CBN_selfie.png
0012.CBN_selfie.png   0025.CBN_id.png       0037.CBN_selfie.png   0050.CBN_id.png
0013.CBN_id.png       0025.CBN_selfie.png   0038.CBN_id.png       0050.CBN_selfie.png
root@8fc162e4ae6f:/home/pfg/scripts#
```

Figura 4.6: Resultado *script* Final

As variáveis de pasta de entrada e pasta de saída de cada *script* são alteradas de forma que a pasta de saída do anterior seja a pasta de entrada do próximo (Figs. 4.7(a-c)).

No fim, após a execução dos três *scripts*, temos 50 novos indíviduos. Logo, 100 novas imagens, que farão parte do Cenário 2.

```
pastaEntrada = "../imagens/imagens_originais/"
pastaSaida = "../imagens_aug_CLAHE/"
```

(a) *Script* 1

```
pastaEntrada = "../imagens_aug_CLAHE/"
pastaSaida = "../imagens_aug_CLAHE_BRIGHT/"
```

(b) *Script* 2

```
pastaEntrada = "../imagens_aug_CLAHE_BRIGHT/"
pastaSaida = "../imagens_aug_CLAHE_BRIGHT_NOISE/"
```

(c) *Script* 3

Figura 4.7: Pastas de Entrada e de Saída de cada *script*

4.2 *OpenFace*

Os testes serão realizados com base no Experimento LFW disponível em [29]. O Experimento de verificação LFW prediz se um par de imagens é da mesma pessoa. É utilizado o modelo *nn4.small2.v1* pré-treinado e obteve acurácia de 92.92% no LFW. A vantagem de se usar um modelo já treinado é a utilização dos pesos obtidos no treinamento como valores iniciais. Não necessitando de um banco de imagens de grande escala com *selfies* e ids para obter esses valores. O treinamento feito com essas imagens será uma afinação da rede neural. As identidades do treinamento da rede neural não se sobrepõe as identidades do banco de imagens utilizado posteriormente, o LFW ou banco com *selfies* e ids.

4.2.1 Testes com base no Experimento LFW

As imagens do Cenário 1 e 2 foram colocadas no diretório *openface/data/cenario1/imagens* e *openface/data/cenario2/imagens.* Os testes são realizados a partir do diretório raiz ***openface.***

1. O pré-processamento das imagens é feito com oito processos separados (o mesmo número utilizado no Experimento LFW). Faz-se então o alinhamento e redimensiona as imagens para 96x96 *pixels* e as salva no diretório *data/cenario1/dlib-affine-sz:96* como mostra a Fig. 4.8.

```
root@232dd16c8fdf:~/openface# for N in {1..8}; do ./util/align-dlib.py data/cenario1/imagens/ alig
n outerEyesAndNose data/cenario1/dlib-affine-sz:96 --size 96 & done
```

Figura 4.8: Pré-processamento das imagens do Cenário 1

2. Gera-se as representações a partir do comando da Fig. 4.9. Para o Cenário 1 são 100 imagens, mas nota-se que apenas 93 imagens foram pré-processadas. No Experimento LFW quando a *dlib* falha para alinhar algumas imagens usa-se a versão *deep funneled* delas, mas como não há outra versão das imagens usadas neste projeto, "perde-se"essas sete imagens neste caso. No entanto, o número de indivíduos continua o mesmo (Fig. 4.10).

```
root@232dd16c8fdf:~/openface# ./batch-represent/main.lua -outDir evaluation/cenario1.nn4.small2.v1
.reps -model models/openface/nn4.small2.v1.t7 -data data/cenario1/dlib-affine-sz\:96/
{
  data : "data/cenario1/dlib-affine-sz:96/"
  imgDim : 96
  model : "models/openface/nn4.small2.v1.t7"
  device : 1
  outDir : "evaluation/cenario1.nn4.small2.v1.reps"
  cache : false
  cuda : false
  batchSize : 50
}
data/cenario1/dlib-affine-sz:96/
cache lotation:        /root/openface/data/cenario1/dlib-affine-sz:96/cache.t7
Loading metadata from cache.
If your dataset has changed, delete the cache file.
nImgs:  93
Represent: 50/93
Represent: 93/93
```

Figura 4.9: Gerando representações das imagens do Cenário 1

```
Found 93 files
Number of different people: 50
```

Figura 4.10: Número de imagens e número de indivíduos no Cenário 1

3. Na Fig. 4.11 é executado o código *gera_pares.py* no diretório *data/cenario1/dlib-affine-sz:96* que gera o arquivo *parescenario1.txt*. A validação cruzada *K-fold* é utilizada, e o arquivo gerado é responsável por indicar a quantidade de pares e de conjuntos e quais pares estão em qual conjunto. Nele serão indicados os pares positivos e os pares negativos para o treinamento e teste. O formato do arquivo deve ser: na primeira linha coloca-se o número de conjuntos seguido do número de pares positivos por conjunto (igual ao número de pares negativos por conjunto). As próximas linhas serão os pares positivos no formato:

   `nome n1 n2`

   Logo abaixo, as próximas linhas serão com os pares negativos no formato:

   `nome1 n1 nome2 n2`

   No Experimento LFW foram utilizados 300 pares positivos por conjunto e 10 conjuntos (*10-fold cross-validation*). Considerando que o banco de imagens utilizado neste projeto é muito menor, utilizou-se 13 pares positivos por conjunto e 3 conjuntos para o Cenário 1.

```
root@232dd16c8fdf:~/openface# python gera_pares.py data/cenario1/dlib-affine-sz\:96/ --nrofpairs 2
1 --output parescenario1.txt
```

Figura 4.11: Gerando o arquivo de pares *parescenario1.txt*

4. A partir do diretório *evaluation* executa-se o código *cenario1.py* que é igual ao código *lfw.py* utilizado no Experimento LFW. Este é o código que realiza treinamento e teste e reporta a acurácia. É utilizado validação cruzada, sendo necessário modificar o número total de pares e o número de conjuntos no código, que neste caso é de 78 pares e 3 conjuntos (*folds*). Os resultados são obtidos computando a distância Euclidiana ao quadrado dos pares e rotulando os pares abaixo de um valor limiar como sendo a mesma pessoa e os acima do valor limiar

como sendo pessoas diferentes. O melhor limiar nas pastas de treinamento é usado como o valor limiar na pasta restante [9]. Na Fig. 4.12 vemos a execução e a acurácia obtida de 85.90%.

```
root@232dd16c8fdf:~/openface/evaluation# ./teste1.py --lfwPairs ../teste1.txt nn4.small2.v1 teste1
.nn4.small2.v1.reps/
Loading embeddings.
  + Reading pairs.
  + Computing accuracy.
    + 0.8590
root@232dd16c8fdf:~/openface/evaluation#
```

Figura 4.12: Acurácia no Cenário 1

Os mesmos passos são realizados para o Cenário 2, como mostra as Figs. 4.13-4.17. Das 200 imagens, apenas 160 são pré-processadas e acaba "perdendo"as duas imagens de 6 indivíduos, restando apenas 94 (Fig. 4.16). Para o Cenário 2 são utilizados 21 pares positivos por conjunto e 3 conjuntos. A acurácia obtida foi de 90.83% (Fig. 4.17).

```
root@232dd16c8fdf:~/openface# for N in {1..8}; do ./util/align-dlib.py data/cenario2/imagens/ alig
n outerEyesAndNose data/cenario2/dlib-affine-sz:96 --size 96 & done
```

Figura 4.13: Pré-processamento das imagens do Cenário 2

```
root@232dd16c8fdf:~/openface# ./batch-represent/main.lua -outDir evaluation/cenario2.nn4.small2.v1
.reps -model models/openface/nn4.small2.v1.t7 -data data/cenario2/dlib-affine-sz\:96/
{
  data : "data/cenario2/dlib-affine-sz:96/"
  imgDim : 96
  model : "models/openface/nn4.small2.v1.t7"
  device : 1
  outDir : "evaluation/cenario2.nn4.small2.v1.reps"
  cache : false
  cuda : false
  batchSize : 50
}
data/cenario2/dlib-affine-sz:96/
cache lotation:         /root/openface/data/cenario2/dlib-affine-sz:96/cache.t7
Loading metadata from cache.
If your dataset has changed, delete the cache file.
nImgs:  160
Represent: 50/160
Represent: 100/160
Represent: 150/160
Represent: 160/160
```

Figura 4.14: Gerando representações das imagens do Cenário 2

```
root@232dd16c8fdf:~/openface# python gera_pares.py data/cenario2/dlib-affine-sz\:96/ --nrofpairs 3
3 --output parescenario2.txt
```

Figura 4.15: Gerando o arquivo de pares *parescenario2.txt*

```
Found 160 files
Number of different people: 94
```

Figura 4.16: Número de imagens e número de indivíduos no Cenário 2

```
root@232dd16c8fdf:~/openface/evaluation# ./teste2.py --lfwPairs ../teste2.txt nn4.small2.v1 teste2
.nn4.small2.v1.reps/
Loading embeddings.
  + Reading pairs.
  + Computing accuracy.
    + 0.9083
root@232dd16c8fdf:~/openface/evaluation#
```

Figura 4.17: Acurácia no Cenário 2

4.3 Extração de Características com *VGG*

Para utilizar os algoritmos *PmSVM* e *Random Forest* é necessário extrair as características previamente com um modelo já treinado da VGG. A extração foi feita no mesmo *docker* que os testes do *Random Forest* foram realizados, visto que ambos utilizam *Caffe*. O modelo utilizado é o *VGG_FACE.caffemodel*. O download do arquivo *vgg_face_caffe.tar.gz* é feito em [8], na seção *Downloads*, como mostra a Fig. 4.18.

Downloads

Filename	Description
vgg_face_matconvnet.tar.gz	Face detection and VGG Face descriptor source code and models (MatConvNet)
vgg_face_torch.tar.gz	VGG Face descriptor source code and models (Torch)
vgg_face_caffe.tar.gz	VGG Face descriptor source code and models (Caffe)

Figura 4.18: *Download* do modelo *Caffe* [8]

Primeiro, executa-se o *script recupera_nomes.py* que cria um arquivo *.txt*, o nome pode ser alterado no *script*. Neste caso o arquivo é *nome_imagens_originais.txt* e contém o nome de todas as imagens do Cenário 1. Na Fig. 4.19 o arquivo é gerado e uma parte dele é mostrada. A extração é feita como mostra a Fig. 4.20. Os arquivos *.feat* são criados no caminho passado na Fig. 4.20, como mostra a Fig. 4.21. Extração de características completa. O mesmo é feito posteriormente para o Cenário 2.

```
root@698d545dbeae:/home/pfg/Documents/Caffe_Extract_Features# ./recupera_nomes.py
root@698d545dbeae:/home/pfg/Documents/Caffe_Extract_Features# cat nome_imagens_originais.txt
imagens/imagens_originais/0031_selfie.png
imagens/imagens_originais/0024_selfie.png
imagens/imagens_originais/0032_id.png
imagens/imagens_originais/0017_selfie.png
imagens/imagens_originais/0040_selfie.png
imagens/imagens_originais/0034_id.png
imagens/imagens_originais/0044_id.png
imagens/imagens_originais/0037_id.png
imagens/imagens_originais/0012_id.png
imagens/imagens_originais/0037_selfie.png
imagens/imagens_originais/0033_id.png
imagens/imagens_originais/0038_selfie.png
imagens/imagens_originais/0049_id.png
imagens/imagens_originais/0009_selfie.png
imagens/imagens_originais/0016_selfie.png
imagens/imagens_originais/0007_selfie.png
imagens/imagens_originais/0006_selfie.png
imagens/imagens_originais/0017_id.png
imagens/imagens_originais/0022_selfie.png
imagens/imagens_originais/0049_selfie.png
imagens/imagens_originais/0029_id.png
imagens/imagens_originais/0025_id.png
imagens/imagens_originais/0014_selfie.png
imagens/imagens_originais/0011_selfie.png
imagens/imagens_originais/0008_id.png
imagens/imagens_originais/0020_id.png
imagens/imagens_originais/0012_selfie.png
imagens/imagens_originais/0048_selfie.png
imagens/imagens_originais/0018_id.png
imagens/imagens_originais/0036_selfie.png
imagens/imagens_originais/0015_selfie.png
imagens/imagens_originais/0003_id.png
imagens/imagens_originais/0046_selfie.png
imagens/imagens_originais/0009_id.png
imagens/imagens_originais/0035_id.png
```

Figura 4.19: Criando arquivo com nomes das imagens

```
root@698d545dbeae:/home/pfg/Documents/Caffe_Extract_Features# python2 caffe_extract_features.py -
p ../VGG_FACE_deploy.prototxt -m ../VGG_FACE.caffemodel -l nome_imagens_originais.txt -i imagens/
imagens_originais/ -o imagens/imagens_feat
```

Figura 4.20: Extração de características

```
root@698d545dbeae:/home/pfg/Documents/Caffe_Extract_Features# ls imagens/imagens_feat/imagens/ima
gens_originais/
0001_id.png.feat      0013_selfie.png.feat  0026_id.png.feat      0038_selfie.png.feat
0001_selfie.png.feat  0014_id.png.feat      0026_selfie.png.feat  0039_id.png.feat
0002_id.png.feat      0014_selfie.png.feat  0027_id.png.feat      0039_selfie.png.feat
0002_selfie.png.feat  0015_id.png.feat      0027_selfie.png.feat  0040_id.png.feat
0003_id.png.feat      0015_selfie.png.feat  0028_id.png.feat      0040_selfie.png.feat
0003_selfie.png.feat  0016_id.png.feat      0028_selfie.png.feat  0041_id.png.feat
0004_id.png.feat      0016_selfie.png.feat  0029_id.png.feat      0041_selfie.png.feat
0004_selfie.png.feat  0017_id.png.feat      0029_selfie.png.feat  0042_id.png.feat
0005_id.png.feat      0017_selfie.png.feat  0030_id.png.feat      0042_selfie.png.feat
0005_selfie.png.feat  0018_id.png.feat      0030_selfie.png.feat  0043_id.png.feat
0006_id.png.feat      0018_selfie.png.feat  0031_id.png.feat      0043_selfie.png.feat
0006_selfie.png.feat  0019_id.png.feat      0031_selfie.png.feat  0044_id.png.feat
0007_id.png.feat      0019_selfie.png.feat  0032_id.png.feat      0044_selfie.png.feat
0007_selfie.png.feat  0020_id.png.feat      0032_selfie.png.feat  0045_id.png.feat
0008_id.png.feat      0020_selfie.png.feat  0033_id.png.feat      0045_selfie.png.feat
0008_selfie.png.feat  0021_id.png.feat      0033_selfie.png.feat  0046_id.png.feat
0009_id.png.feat      0021_selfie.png.feat  0034_id.png.feat      0046_selfie.png.feat
0009_selfie.png.feat  0022_id.png.feat      0034_selfie.png.feat  0047_id.png.feat
0010_id.png.feat      0022_selfie.png.feat  0035_id.png.feat      0047_selfie.png.feat
0010_selfie.png.feat  0023_id.png.feat      0035_selfie.png.feat  0048_id.png.feat
0011_id.png.feat      0023_selfie.png.feat  0036_id.png.feat      0048_selfie.png.feat
0011_selfie.png.feat  0024_id.png.feat      0036_selfie.png.feat  0049_id.png.feat
0012_id.png.feat      0024_selfie.png.feat  0037_id.png.feat      0049_selfie.png.feat
0012_selfie.png.feat  0025_id.png.feat      0037_selfie.png.feat  0050_id.png.feat
0013_id.png.feat      0025_selfie.png.feat  0038_id.png.feat      0050_selfie.png.feat
root@698d545dbeae:/home/pfg/Documents/Caffe_Extract_Features# █
```

Figura 4.21: Arquivos *.feat* criados após extração

4.4 PmSVM

Após a extração de características descrita na seção 4.3, os dados foram salvos no diretório *imagens/imagens_feat/*. Executa-se então o *script gera_aleatorios.py* que gera o arquivo *seeds.npy*, usado para separar aleatoriamente a base de imagens em treinamento e teste. Após gerado o arquivo *seeds.npy* é necessário converter as características para o formato *libsvm*, como mostra a Fig. 4.22. Serão gerados arquivos *trainset_sample2* e *testset_sample2*, onde 80% das imagens vão aleatoriamente para o conjunto de treinamento e 20% para o de teste, no formato:

<label> <index1>:<value1> <index2>:<value2> ... <indexN>:<valueN>

A quantidade de pares possíveis é $n \times n$ com n sendo tamanho do conjunto de treinamento ou de teste. A quantidade de pares positivos é n e a quantidade de pares negativos é $n \times 5$, onde 5 é um número escolhido de forma que a quantidade de pares positivos e negativos não fique discrepante e alcance um resultado satisfatório. Outros valores foram testados, mas com 5 obteve-se melhores resultados, tanto para o Cenário 1 quanto para o Cenário 2.

```
root@8fc162e4ae6f:/home# ./gera_aleatorios.py
root@8fc162e4ae6f:/home# python convert.libsvm.balenc.py -p imagens/imagens_feat/ -s seeds.npy -o
tr trainset_sample2 -ots testset_sample2
# 2017-11-09 19:45:01 ### Main - Parse arguments!
# 2017-11-09 19:45:01  - Args: Namespace(outputTest='testset_sample2', outputTrain='trainset_samp
le2', pastaorigem='imagens/imagens_feat/', seed='seeds.npy')
# 2017-11-09 19:45:01  # Get list of files
# 2017-11-09 19:45:01  Qtd IDs: 50 - Qtd Selfies: 50
# 2017-11-09 19:45:01  # Get data matrices
# 2017-11-09 19:45:01  # > def get_data(files)
# 2017-11-09 19:45:02  # > def get_data(files)
# 2017-11-09 19:45:02  Mat IDs shape: (50, 4096) - Mat Selfies shape: (50, 4096)
# 2017-11-09 19:45:02  # Calculate folds
# 2017-11-09 19:45:02  - Train size: (40,) - Test size: (10,)
# 2017-11-09 19:45:02  # Gerando Set TRAIN
# 2017-11-09 19:45:02  - Qtd. Pares Possiveis: 1600
# 2017-11-09 19:45:02  - Qtd. Pares Positivos A GERAR: 40
# 2017-11-09 19:45:02  - Qtd. Pares Negativos A GERAR: 200
# 2017-11-09 19:45:04  # Gerando Set TEST
# 2017-11-09 19:45:04  - Qtd. Pares Possiveis: 100
# 2017-11-09 19:45:04  - Qtd. Pares Positivos A GERAR: 10
# 2017-11-09 19:45:04  - Qtd. Pares Negativos A GERAR: 50
Fim...
```

Figura 4.22: Conversão das *features* para o formato *libsvm*

Utilizando a ferramenta *svm-scale* na *libsvm*, converte-se os dados para o intervalo [0,1] (Fig. 4.23). Primeiro os dados do *trainset_sample2* e depois os dados do *testset_sample2*. São criados os *trainset_sample2.scale* e *testset_sample2.scale*, já no intervalo especificado. Como mencionado na seção 3.7, o PmSVM reporta duas acurácias, mas a analisada neste projeto é a acurácia geral, que neste teste foi 88.33%.

```
root@8fc162e4ae6f:/home# ./libsvm-3.22/svm-scale -l -1 -u 1 -s range100.txt trainset_sample2 > tr
ainset_sample2.scale
WARNING: original #nonzeros 345125
       > new      #nonzeros 692880
If feature values are non-negative and sparse, use -l 0 rather than the default -l -1
root@8fc162e4ae6f:/home# ./libsvm-3.22/svm-scale -r range100.txt testset_sample2 > testset_sample
2.scale
WARNING: feature index 1630 appeared in file testset_sample2 was not seen in the scaling factor f
ile range100.txt.
WARNING: feature index 3372 appeared in file testset_sample2 was not seen in the scaling factor f
ile range100.txt.
WARNING: feature index 3471 appeared in file testset_sample2 was not seen in the scaling factor f
ile range100.txt.
WARNING: original #nonzeros 89422
       > new      #nonzeros 173400
If feature values are non-negative and sparse, use -l 0 rather than the default -l -1
root@8fc162e4ae6f:/home# ./pmsvm trainset_sample2.scale testset_sample2.scale
Dataset loaded in 216 msec.
.*.
optimization finished, #iter = 12
nSV = 178
Finished in 186 msec.
Dataset loaded in 92 msec.
Finished in 39 msec.
4/10=40%
49/50=98%
Overall accuracy = 88.3333
Average accuracy = 69
```

Figura 4.23: Conversão dos dados para o intervalo [0,1] e acurácia obtida

Extraiu-se as características das imagens do Cenário 2, da mesma forma que as do Cenário 1, descrito na seção 4.3. Os dados foram salvos no diretório *imagens_aug_feat/*. Os passos para a realização do teste no Cenário 2 são os mesmos que para o Cenário 1 como pode-se observar na Fig. 4.24 e Fig. 4.25. A acurácia geral foi 90.00%.

```
root@8fc162e4ae6f:/home# ./gera_aleatorios.py
root@8fc162e4ae6f:/home# python convert.libsvm.balenc.py -p imagens/imagens_aug_feat/ -s seeds.np
y -otr trainset_sampleaug2 -ots testset_sampleaug2
# 2017-11-09 19:45:52 ### Main - Parse arguments!
# 2017-11-09 19:45:52  - Args: Namespace(outputTest='testset_sampleaug2', outputTrain='trainset_s
ampleaug2', pastaorigem='imagens/imagens_aug_feat/', seed='seeds.npy')
# 2017-11-09 19:45:52  # Get list of files
# 2017-11-09 19:45:52  Qtd IDs: 100 - Qtd Selfies: 100
# 2017-11-09 19:45:52  # Get data matrices
# 2017-11-09 19:45:52  # > def get_data(files)
# 2017-11-09 19:45:54  # > def get_data(files)
# 2017-11-09 19:45:55  Mat IDs shape: (100, 4096) - Mat Selfies shape: (100, 4096)
# 2017-11-09 19:45:55  # Calculate folds
# 2017-11-09 19:45:55  - Train size: (80,) - Test size: (20,)
# 2017-11-09 19:45:55  # Gerando Set TRAIN
# 2017-11-09 19:45:55  - Qtd. Pares Possiveis: 6400
# 2017-11-09 19:45:55  - Qtd. Pares Positivos A GERAR: 80
# 2017-11-09 19:45:55  - Qtd. Pares Negativos A GERAR: 400
# 2017-11-09 19:45:58  # Gerando Set TEST
# 2017-11-09 19:45:58  - Qtd. Pares Possiveis: 400
# 2017-11-09 19:45:58  - Qtd. Pares Positivos A GERAR: 20
# 2017-11-09 19:45:58  - Qtd. Pares Negativos A GERAR: 100
Fim...
```

Figura 4.24: Conversão das *features* do Cenário 2 para o formato *libsvm*

```
root@8fc162e4ae6f:/home# ./libsvm-3.22/svm-scale -l -1 -u 1 -s range100.txt trainset_sampleaug2 >
 trainset_sampleaug2.scale
WARNING: original #nonzeros 744404
       > new      #nonzeros 1417440
If feature values are non-negative and sparse, use -l 0 rather than the default -l -1
root@8fc162e4ae6f:/home# ./libsvm-3.22/svm-scale -r range100.txt testset_sampleaug2 > testset_sam
pleaug2.scale
WARNING: feature index 1192 appeared in file testset_sampleaug2 was not seen in the scaling facto
r file range100.txt.
WARNING: feature index 3471 appeared in file testset_sampleaug2 was not seen in the scaling facto
r file range100.txt.
WARNING: original #nonzeros 190571
       > new      #nonzeros 354600
If feature values are non-negative and sparse, use -l 0 rather than the default -l -1
root@8fc162e4ae6f:/home# ./pmsvm trainset_sampleaug2.scale testset_sampleaug2.scale
Dataset loaded in 596 msec.
.*.
optimization finished, #iter = 11
nSV = 348
Finished in 285 msec.
Dataset loaded in 162 msec.
Finished in 73 msec.
11/20=55%
97/100=97%
Overall accuracy = 90
Average accuracy = 76
```

Figura 4.25: Conversão dos dados do Cenário 2 para o intervalo [0,1] e acurácia obtida

4.5 *Random Forest*

Similarmente aos testes realizados na seção 4.4, após a extração de características descrita na seção 4.3 os dados foram salvos no diretório *imagens/imagens_feat/*. Executa-se então o *script gera_aleatorios.py* que gera o arquivo *seeds.npy*, usado para separar aleatoriamente a base de imagens em treinamento e teste, como mostra a Fig. 4.26.

```
root@d7e9f8721694:/workspace# ls
gera_aleatorios.py  imagens  logs  random.forest.v2.py
root@d7e9f8721694:/workspace# ./gera_aleatorios.py
root@d7e9f8721694:/workspace# ls
gera_aleatorios.py  imagens  logs  random.forest.v2.py  seeds.npy
root@d7e9f8721694:/workspace#
```

Figura 4.26: Gerando o arquivo *seeds.npy*

O treinamento e teste no Cenário 1 é realizado de acordo com a Fig. 4.27 e o treinamento e teste no Cenário 2 de acordo com a Fig. 4.28. Executa-se o *script random.forest.v2.py* nas características extraídas com o arquivo *seeds.npy* gerado antes de cada teste. Da mesma forma que no PmSVM, a quantidade de pares possíveis é $n \times n$ com n sendo tamanho do conjunto de treinamento ou de teste. A quantidade de pares positivos é n e a quantidade de pares negativos é $n \times 5$, onde 5 é um número escolhido de forma que a quantidade de pares positivos e negativos não fique discrepante e alcance um resultado satisfatório. Outros valores foram testados, mas com 5 obteve-se melhores resultados, tanto para o Cenário 1 quanto para o Cenário 2. A acurácia no Cenário 1 foi 83.33% e no Cenário 2 foi 84.17%.

```
root@d7e9f8721694:/workspace# ./gera_aleatorios.py
root@d7e9f8721694:/workspace# python2 random.forest.v2.py imagens/imagens_feat/ seeds.npy -c 2
# 2017-11-09 19:55:34 ### Main - Parse arguments!
# 2017-11-09 19:55:34  - Args: Namespace(cores=2, data='imagens/imagens_feat/', rand='seeds.npy')
# 2017-11-09 19:55:34  # > def get_files(data_dir)
# 2017-11-09 19:55:34  Qtd IDs: 50
# 2017-11-09 19:55:34  Qtd Selfies: 50
# 2017-11-09 19:55:34  # Get data matrices
# 2017-11-09 19:55:35  Mat IDs shape: (50, 4096)
# 2017-11-09 19:55:35  Mat Selfies shape: (50, 4096)
# 2017-11-09 19:55:35  # Shuffle indices
# 2017-11-09 19:55:35  # Calculate folds
# 2017-11-09 19:55:35  - Train size: (40,) - Test size: (10,)
# 2017-11-09 19:55:35  # RandomForestClassifier
# 2017-11-09 19:55:35  # Gen Samples DataTrain
# 2017-11-09 19:55:35  - Gerando Pares 40 Positivos e 200 Negativos / TOTAL: 240
# 2017-11-09 19:55:35      > Generated data shape: (240, 4096)
# 2017-11-09 19:55:35      > Generated labels shape: (240,)
# 2017-11-09 19:55:35  - Train shape: (240, 4096) - Train labels shape: (240,)
# 2017-11-09 19:55:35  # Train with whole training data
building tree 1 of 10
building tree 2 of 10
building tree 3 of 10
building tree 4 of 10
 building tree 5 of 10
building tree 6 of 10
building tree 7 of 10
building tree 8 of 10
building tree 9 of 10
building tree 10 of 10
[Parallel(n_jobs=2)]: Done  10 out of  10 | elapsed:    0.1s finished
# 2017-11-09 19:55:35  # Gen Samples DataTest
# 2017-11-09 19:55:35  - Gerando Pares 10 Positivos e 50 Negativos / TOTAL: 60
# 2017-11-09 19:55:35      > Generated data shape: (60, 4096)
# 2017-11-09 19:55:35      > Generated labels shape: (60,)
# 2017-11-09 19:55:35  - Test data shape: (60, 4096) / Test labels shape: (60,)
[Parallel(n_jobs=2)]: Done  10 out of  10 | elapsed:    0.0s finished
[Parallel(n_jobs=2)]: Done  10 out of  10 | elapsed:    0.0s finished
# 2017-11-09 19:55:36  Train Accuracy :: 0.983333333333
# 2017-11-09 19:55:36  Test Accuracy :: 0.833333333333
# 2017-11-09 19:55:36 ...Done!
```

Figura 4.27: Treinamento e teste no Cenário 1

```
root@d7e9f8721694:/workspace# ./gera_aleatorios.py
root@d7e9f8721694:/workspace# python2 random.forest.v2.py imagens/imagens_aug_feat/ seeds.npy -c
2
# 2017-11-09 19:55:44 ### Main - Parse arguments!
# 2017-11-09 19:55:44  - Args: Namespace(cores=2, data='imagens/imagens_aug_feat/', rand='seeds.n
py')
# 2017-11-09 19:55:44  # > def get_files(data_dir)
# 2017-11-09 19:55:44  Qtd IDs: 100
# 2017-11-09 19:55:44  Qtd Selfies: 100
# 2017-11-09 19:55:44  # Get data matrices
# 2017-11-09 19:55:47  Mat IDs shape: (100, 4096)
# 2017-11-09 19:55:47  Mat Selfies shape: (100, 4096)
# 2017-11-09 19:55:47  # Shuffle indices
# 2017-11-09 19:55:47  # Calculate folds
# 2017-11-09 19:55:47  - Train size: (80,) - Test size: (20,)
# 2017-11-09 19:55:47  # RandomForestClassifier
# 2017-11-09 19:55:47  # Gen Samples DataTrain
# 2017-11-09 19:55:47  - Gerando Pares 80 Positivos e 400 Negativos / TOTAL: 480
# 2017-11-09 19:55:47      > Generated data shape: (480, 4096)
# 2017-11-09 19:55:47      > Generated labels shape: (480,)
# 2017-11-09 19:55:47  - Train shape: (480, 4096) - Train labels shape: (480,)
# 2017-11-09 19:55:47  # Train with whole training data
building tree 1 of 10
building tree 2 of 10
building tree 3 of 10
building tree 4 of 10
building tree 5 of 10
building tree 6 of 10
building tree 7 of 10
building tree 8 of 10
building tree 9 of 10
building tree 10 of 10
[Parallel(n_jobs=2)]: Done  10 out of  10 | elapsed:    0.2s finished
# 2017-11-09 19:55:47  # Gen Samples DataTest
# 2017-11-09 19:55:47  - Gerando Pares 20 Positivos e 100 Negativos / TOTAL: 120
# 2017-11-09 19:55:47      > Generated data shape: (120, 4096)
# 2017-11-09 19:55:47      > Generated labels shape: (120,)
# 2017-11-09 19:55:47  - Test data shape: (120, 4096) / Test labels shape: (120,)
[Parallel(n_jobs=2)]: Done  10 out of  10 | elapsed:    0.0s finished
[Parallel(n_jobs=2)]: Done  10 out of  10 | elapsed:    0.0s finished
# 2017-11-09 19:55:48  Train Accuracy :: 0.975
# 2017-11-09 19:55:48  Test Accuracy :: 0.841666666667
# 2017-11-09 19:55:48 ...Done!
```

Figura 4.28: Treinamento e teste no Cenário 2

4.6 Resultados

É importante ressaltar que a *OpenFace* realiza todo o fluxo de processamento apresentado na Fig. 4.29. A VGG realiza o que está na caixa vermelha e o seu resultado é usado como entrada pelos algoritmos PmSVM e *Random Forest*, que realizam somente a parte de classificação, fora da caixa de delimitação vermelha.

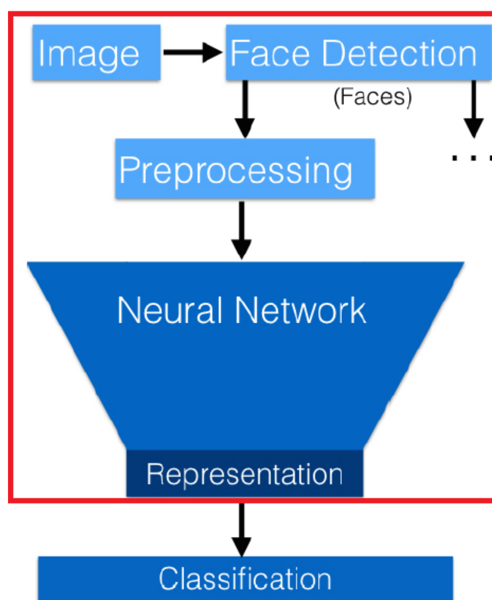

Figura 4.29: Fluxo de processamento de uma imagem

Os resultados dos testes nas três ferramentas foram organizados na Tab. 4.1. PmSVM foi a ferramenta que obteve a melhor acurácia no Cenário 1 e a *OpenFace* foi a que obteve a melhor acurácia no Cenário 2. Em relação a diferença de acurácia entre cenários, nota-se que nas três ferramentas o Cenário 2 obteve uma acurácia maior do que no Cenário 1. Um aumento no tamanho do banco de imagens resultou em uma acurácia maior.

	Acurácia Cenário 1	**Acurácia Cenário 2**
OpenFace	85.90%	90.83%
PmSVM	88.33%	90.00%
Random Forest	83.33%	84.17%

Tabela 4.1: Resultados

As acurácias da *OpenFace* apresentadas na Tab. 4.1 foram obtidas com o método *k-fold* de validação cruzada. Apesar de o método *10-fold* (10-conjuntos) ser o usado no Experimento LFW e de ser o sugerido por vários autores, as acurácias obtidas usando 3 conjuntos foram as melhores comparando-as com as acurácias obtidas com 10 e 5 conjuntos. A comparação pode ser observada na Tab. 4.2. Vale ressaltar que como o banco de imagens é pequeno e quanto maior o número de conjuntos menor o número de pares por conjunto, com 10 conjuntos tem-se 3 pares positivos

e 3 pares negativos por conjunto para o Cenário 1 e 5 pares positivos e 5 pares negativos por conjunto para o Cenário 2. Então, apesar dos testes serem realizados em todos os conjuntos, eles são realizados em poucas imagens por vez.

	Acurácia Cenário 1	**Acurácia Cenário 2**
10 conjuntos	85.67%	83.33%
5 conjuntos	73.75%	84.55%
3 conjuntos	85.90%	90.83%

Tabela 4.2: Acurácias na *OpenFace* para diferentes valores de k na validação cruzada *k-fold*

Diferentemente da *OpenFace*, nas ferramentas *Random Forest* e PmSVM é utilizado o método *holdout*, onde a base de dados é dividida em 80% de treinamento e 20% de teste.

Capítulo 5

Conclusões

Neste projeto foi feito um estudo sobre a verificação facial de imagens *cross-domain* utilizando três ferramentas de código aberto distintas. Os testes e comparações foram feitos em dois cenários, onde mudou o número de imagens de um para o outro, com o objetivo de observar o que o aumento no número de imagens, mesmo que pequeno, pode causar na acurácia das ferramentas utilizadas. Tabelas com os resultados obtidos foram criadas a fim de comparação. Conclusões acerca das análises feitas serão apresentadas neste capítulo.

Atualmente atinge-se alta performance na verificação facial de imagens de mesmo domínio. Contudo, ao usar imagens *cross-domain*, onde há severas mudanças de iluminação e *noise*, a performance cai. Treinamento e testes foram realizados com imagens *cross-domain*, a fim de analisar como essas diferenças impactam a acurácia das ferramentas utilizadas.

Para a extração de características com a VGG e na *OpenFace* foram usados modelos de redes neurais pré-treinados, onde foi necessário somente calibrar os pesos, de forma que a rede responda ao desejado, ao treiná-los com as imagens usadas neste projeto.

Dentre as ferramentas utilizadas, a *OpenFace* se difere dos algoritmos clássicos de ML de classificação, o *Random Forest* e o PmSVM, em alguns pontos. Na *OpenFace* é feito desde a detecção facial até a verificação de faces, incluindo a extração de características. Os algoritmos de classificação somente classificam as faces como pertencendo ou não a mesma pessoa. A parte de detecção facial, redimensionamento da imagem e a extração de características foi feita com um modelo de rede neural pré-treinado da VGG. O método de validação cruzada utilizado na *OpenFace* é diferente do utilizado nos algoritmos de classificação aqui apresentados. Na *OpenFace* foi usado o método *k-fold* e no *Random Forest* e PmSVM foi usado o método *holdout*.

Infere-se das acurácias apresentadas no Capítulo 4 que a PmSVM obteve o melhor resultado para o menor número de imagens enquanto a *OpenFace* obteve o melhor resultado para um maior número de imagens, o que se aplica para as condições deste trabalho. Os resultados do *Random Forest* não foram satisfatórios considerando que as acurácias foram as menores em ambos os cenários. Por a *OpenFace* realizar todo o processo de verificação facial e os algoritmos *Random Forest* e PmSVM realizarem somente a classificação, constata-se que as comparações de resultados não foram feitas com todas as três ferramentas na mesma condição.

Como observado, todas as ferramentas obtiveram acurácias maiores quando mais imagens foram usadas no treinamento, caso do Cenário 2. Sendo assim, percebe-se que pequenas variações no número de imagens causam diferenças nas acurácias obtidas. Logo, para um local pequeno, como por exemplo o Departameno de Egenharia Elétrica da UnB, os resultados aqui obtidos podem ser usados como base para a escolha de uma ferramenta. No entanto, para um aeroporto, por exemplo, seria necessário a realização de novos experimentos, com um banco de imagens muito maior.

Observou-se a dificuldade na verificação de imagens *cross-domain* ao utilizar a *OpenFace*, onde várias imagens foram perdidas por não conseguir detectar uma face, principalmente nas imagens de documentos pessoais. Todavia, com a VGG foi possível extrair características de todas as imagens, sendo assim não houve perda ao utilizar os algoritmos de classificação *Random Forest* e PmSVM.

Comparado as acurácias obtidas aqui com as obtidas em testes realizados em base de dados como a LFW, a discrepância não é muita. No caso da *OpenFace*, no Cenário 2 obteve 90.82% de acurácia e na LFW obteve 92.92%. Podendo-se dizer que os resultados obtidos na *OpenFace* e na PmSVM foram satisfatórios.

Como o banco de imagens utilizado é pequeno, para sugestão de trabalhos futuros seria a realização de novos testes e comparações utilizando uma base de dados com um maior número de imagens, na casa dos milhares. Assim, com mais imagens para treinamento das ferramentas, certamente melhores resultados serão obtidos. Podendo inclusive mudar a ferramenta com a melhor performance e sendo possível aplicar os resultados a cenários reais muito maiores.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] LI, S. Z.; JAIN, A. K. (Ed.). *Handbook of Face Recognition*. 2nd. ed. [S.l.]: Springer, 2011. 1-12 p.

[2] NIELSEN, M. *Neural Networks and Deep Learning*. 2017. [Online; acessado 15-Maio-2017]. Disponível em: <http://neuralnetworksanddeeplearning.com/index.html>.

[3] SAXENA, R. *How Decision Tree Algorithm Works*. 2017. [Online; acessado 28-Outubro-2017]. Disponível em: <https://dataaspirant.com/2017/01/30/how-decision-tree-algorithm-works/>.

[4] SUPPORT Vector Machines. [Online; acessado 08-Novembro-2017]. Disponível em: <http://www.statsoft.com/textbook/support-vector-machines>.

[5] ALLRED, R. *Image Augmentation for Deep Learning using Keras and Histogram Equalization*. 2017. [Online; acessado 07-Novembro-2017]. Disponível em: <https://towardsdatascience.com/image-augmentation-for-deep-learning-using-keras-and-histogram-equalization-9329f6ae5085>.

[6] AMOS, B. *OpenFace*. [Online; acessado 15-Maio-2017]. Disponível em: <https://cmusatyalab.github.io/openface/>.

[7] AMOS, B.; LUDWICZUK, B.; SATYANARAYANAN, M. Openface: A general-purpose face recognition library with mobile applications. 2016.

[8] PARKHI, O. M.; VEDALDI, A.; ZISSERMAN, A. *VGG Face Descriptor*. [Online; acessado 15-Maio-2017]. Disponível em: <http://www.robots.ox.ac.uk/~vgg/software/vgg_face>.

[9] AMOS, B. *OpenFace Models and Accuracies*. [Online; acessado 15-Maio-2017]. Disponível em: <https://cmusatyalab.github.io/openface/models-and-accuracies/>.

[10] THAKKAR, D. *Adoption of Biometrics in Banking and Financial Service Industry*. [Online; acessado 18-Outubro-2017]. Disponível em: <https://www.bayometric.com/biometrics-in-banking-and-finance/>.

[11] GRAHAM, L. *HSBC customers can open new bank accounts using a selfie*. 2016. [Online; acessado 23-Maio-2017]. Disponível em: <https://www.cnbc.com/2016/09/05/hsbc-customers-can-open-new-bank-accounts-using-a-selfie.html>.

[12] HO, H. T.; GOPALAN, R. Model-driven domain adaptation on product manifolds for unconstrained face recognition. 2013.

[13] FOLEGO, G. et al. Cross-domain face verification: Matching id document and self-portrait photographs. 2016.

[14] MACHINE Learning. [Online; acessado 08-Novembro-2017]. Disponível em: <https://www.mathworks.com/discovery/machine-learning.html>.

[15] QUAL é a diferença entre IA, Machine Learning, Deep Learning e computação cognitiva? 2016. [Online; acessado 08-Novembro-2017]. Disponível em: <http://cio.com.br/tecnologia/2016/07/01/qual-e-a-diferenca-entre-ia-machine-learning-deep-learning-e-computacao-cognitiva/>.

[16] LEHE, L. *Principal Component Analysis*. [Online; acessado 08-Novembro-2017]. Disponível em: <http://setosa.io/ev/principal-component-analysis/>.

[17] BROWNLEE, J. *Supervised and Unsupervised Machine Learning Algorithms*. 2016. [Online; acessado 28-Outubro-2017]. Disponível em: <https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/>.

[18] POLAMURI, S. *Supervised and Unsupervised Learning*. 2014. [Online; acessado 28-Outubro-2017]. Disponível em: <https://dataaspirant.com/2014/09/19/supervised-and-unsupervised-learning/>.

[19] BROWNLEE, J. *Support Vector Machines for Machine Learning*. 2016. [Online; acessado 08-Novembro-2017]. Disponível em: <https://machinelearningmastery.com/support-vector-machines-for-machine-learning/>.

[20] CROSS Validation. [Online; acessado 14-Novembro-2017]. Disponível em: <https://www.cs.cmu.edu/ schneide/tut5/node42.html>.

[21] AMOS, B. *OpenFace Setup*. [Online; acessado 15-Maio-2017]. Disponível em: <https://cmusatyalab.github.io/openface/setup/>.

[22] LABELED Faces in the Wild. [Online; acessado 14-Novembro-2017]. Disponível em: <http://vis-www.cs.umass.edu/lfw/>.

[23] SCHROFF, F.; KALENICHENKO, D.; PHILBIN, J. Facenet: A unified embedding for face recognition and clustering. In: *IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2015. p. 815–823.

[24] TAIGMAN, Y. et al. Deepface: Closing the gap to human-level performance in face verification. In: *IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2014. p. 1701–1708.

[25] PARKHI, O. M.; VEDALDI, A.; ZISSERMAN, A. Deep face recognition. 2015.

[26] CAFFE. [Online; acessado 08-Novembro-2017]. Disponível em: <http://caffe.berkeleyvision.org/>.

[27] JIA, Y. et al. Caffe: Convolutional architecture for fast feature embedding. 2014.

[28] WU, J. Power mean svm for large scale visual classification. In: *IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2012.

[29] AMOS, B. *Running The LFW Experiment*. [Online; acessado 08-Novembro-2017]. Disponível em: <https://cmusatyalab.github.io/openface/models-and-accuracies/#running-the-lfw-experiment>.

ANEXOS

I. PRÉ-REQUISITOS PARA AS ANÁLISES

I.1 Instalação *Docker*

```
sudo apt-get update
sudo apt-get install docker.io
```

I.2 Instalação *OpenFace* via *Docker*

Baixe a imagem do *OpenFace* (*bamos/openface*) que já está pronta para uso:

```
docker pull bamos/openface
```

Crie um container baseado nessa imagem e execute o *shell* para conectar:

```
docker run -p 9000:9000 -p 8000:8000 -t -i bamos/openface /bin/bash
```

I.3 Instalação PmSVM via *Docker*

Baixe a imagem *ubuntu*:

```
docker pull ubuntu
```

Crie o container baseado nessa imagem com um nome de sua preferência e o tamanho de memória compartilhada (*–shm-size*) de acordo com o necessário. Depois executa-se o *shell* para conectar:

```
docker run -dit -restart always -privileged -name "name--shm-size 300g ubuntu
docker exec -it "name"/bin/bash
```

DNS para o comando *apt-get update*:

```
echo "nameserver 8.8.8.8 tee /etc/resolv.conf > /dev/null
```

Depois, faça um *update* no *cache* usando:

```
apt-get update
```

Como PmSVM é escrito em C++, instale *gcc/gcc++*:

```
apt-get install gcc g++ build-essential
```

É necessário também a instalação de outras ferramentas:

```
apt-get install cmake strace wget unzip

apt-get install python-dev python-numpy python-pip python-scipy

apt-get install gnuplot python-netlib

pip install -U scikit-learn numpy
```

No diretório *home* crie dois novos diretórios:

```
cd home/

mkdir hter; mkdir logs;
```

Crie o diretório PmSVM, faça o *download* do arquivo *PmSVM.zip* e descompacte-o:

PmSVM (https://sites.google.com/site/wujx2001/home/power-mean-svm#TOC-Technical-papers)

```
mkdir PmSVM

wget https://sites.google.com/site/wujx2001/home/power-mean-svm/PmSVM.zip

unzip PmSVM.zip
```

Execute o seguinte comando para gerar um executável com o nome *pmsvm*:

```
g++ -O3 PmSVM/PmSVM.cpp -o pmsvm
```

Faça o *download* da *libsvm*, descompacte o arquivo e compile:

```
wget https://www.csie.ntu.edu.tw/~cjlin/libsvm/oldfiles/libsvm-3.22.zip

unzip libsvm-3.22.zip

cd libsvm-3.22

make
```

O mesmo é feito para a *liblinear*:

LIBLINEAR (https://www.csie.ntu.edu.tw/~cjlin/liblinear/)

```
wget https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/multicore-liblinear/
liblinear-multicore-2.11-2.zip

unzip liblinear-multicore-2.11-2.zip

cd liblinear-multicore-2.11-2

make
```

I.4 Instalação *Random Forest* via *Docker*

Baixe a imagem da *Caffe* (*bvlc/caffe:cpu*):

```
docker pull bvlc/caffe:cpu
```

Execute o seguinte comando para visualizar a *id* da imagem *Caffe* que será usada no próximo comando:

```
docker images
```

Crie o container baseado nessa imagem com um nome de sua preferência. Depois executa-se o *shell* para conectar:

```
docker run -dit -restart always -privileged -name ‘‘name’”‘id-imagem:  bvlc/caffe’’
docker exec -it "name"/bin/bash
```

Depois, faça um *update* no *cache* usando:

```
apt-get update
```

Instale a *scikit-learn* que é uma biblioteca de aprendizado de máquina *open source* para *Python*:

```
pip install -U scikit-learn
```

Instale também algumas bibliotecas e dependências necessárias:

```
apt-get install libboost-all-dev libblitz0-dev cmake libhdf5-serial-dev libtiff5
libtiff5-dev libtiff-tools giflib-dbg strace
pip install numpy bob.extension bob.blitz bob.core bob.io.base bob.io.image
apt-get install libopenblas-dev libcppnetlib-dev python-netlib libfreetype6-dev
pip install bob.measure
```

Por fim crie o diretório de *logs*:

```
mkdir logs
```