Reconhecimento Facial utilizando o algoritmo *Eigenface* da biblioteca Open CV

Jheime Santos da Silveira[1], Angela Abreu de Sá[2]

[1] Curso de Ciência da Computação – Centro Universitário do Triângulo (UNITRI)

[2] Faculdade de Engenharia Elétrica – Universidade Federal de Uberlândia (UFU)

[1]jheimes.silveira@gmail.com, [2]angelaabreu@gmail.com

Abstract. *This article aims to contribute to a comprehensive approach to the implementation process of a software that uses facial recognition algorithm Eigenface in an Android application, using the resources of the library OpenCV. For the software implementation, the JAVA language was used with the package Software Development Kit (SDK) Android API 21 distribution, using the Integrated Development Environment (IDE). Performance tests were carried out, reporting the potential of the library for the facial recognition; assisting all the scientific community in the use of this algorithm in the field of biometrics.*

Resumo. *Este artigo tem como objetivo contribuir com uma abordagem detalhada do processo de implementação de um* software *que utiliza o algoritmo de reconhecimento facial* Eigenface *em uma aplicação Android, utilizando os recursos da biblioteca OpenCV. Para a implementação do software, foi utilizado a linguagem JAVA com o pacote Software Development Kit (SDK) da distribuição Android API 21, utilizando a Integrated Development Environment (IDE) oficial do Android. Foram realizados testes de performance do sistema, relatando o potencial da biblioteca para reconhecimento de faces; auxiliando, assim, toda a comunidade científica na utilização deste algoritmo na área de biometria.*

1. Introdução

Nos últimos anos, os dispositivos móveis estão sendo cada vez mais utilizados devido às suas capacidades de conexão à internet, ao acesso às redes sociais, ao armazenamento de conteúdos sigilosos dentre outras funções. A maioria dos dispositivos móveis consumidos apresentam o sistema operacional *Android*, o que acarreta o aumento da popularidade desse sistema, sendo esse um dos objetos de vários estudos [Correia et al. 2014]. Além disso, devido aos consumidores guardarem cada vez mais informações

pessoais e laborais em dispositivos móveis, esses se transformaram em grandes bancos de informações que necessitam de mecanismos eficientes e precisos de segurança, pois, em geral, o conteúdo do dispositivo é mais valioso do que o próprio aparelho. Por isso, é relevante a elaboração de pesquisas na área de visão computacional para manter a segurança do acesso à esses dados [Farina 2012].

Assim, a proteção ao acesso desses equipamentos pode ser garantida por várias formas de autenticação, a biometria é uma delas. Esse tipo de autenticação se baseia em características próprias do indivíduo para permitir o acesso à uma informação ou à uma instituição. Essa, quando é feita por meio de imagem facial, é de fácil adesão, porque possui um bom desempenho, alta confiabilidade, custo reduzido e é pouco invasiva [Junior and Filho 2015].

Para a realização da biometria facial, é necessário utilizar algoritmos específicos para delimitar o formato e as proporções do rosto como, por exemplo, o algoritmo *Eigenface* [Almeida 2009; Nunes 2016]. De acordo com a pesquisa desenvolvida por [Diniz et al. 2013], a tecnologia *Eigenface* possui capacidade de oferecer características importantes da face através da sua imagem, sendo eficaz para o reconhecimento facial. Porém, essa técnica não foi desenvolvida considerando os casos de iluminação não controlada, o que é um fator importante a ser considerado quando for utilizar a técnica *Eigenface*. Apesar dessa limitação, essa técnica apresenta-se como uma boa opção de uso, já que no estudo desenvolvido por [Nunes 2016], foi observado uma baixa ocorrência de resultados falso positivo, comparando com os resultados dos testes aplicados para avaliar os algoritmos usados para a classificação de faces.

No entanto, no trabalho de [Correia et al. 2014], no qual foi feita a comparação entre as metodologias *Eigenface* e a *Local Binary Patterns* (LBP), ambas disponíveis na biblioteca *Open Source Computer Vision (Open CV)*, foi observado que o algoritmo LBP utilizou menor quantidade de memória e de tempo de processamento do que o algoritmo *Eigenface,* como também apresentou melhores resultados nos testes realizados em diferentes tipos de iluminação e em lugares comuns de usuários de dispositivos móveis. Porém, [Machado et al. 2009] ressaltaram que o algoritmo *Eigenface* é uma ótima metodologia para detecção de faces, mas é não é tão eficaz para o reconhecimento facial quando comparada às novas metodologias como, por exemplo, o algoritmo *FaceVACS.* No entanto, com o uso conjunto de técnicas atuais de pré-processamento de imagens, a ocorrência de erros falsos positivos pelo algoritmo *Eigenface* pode ser minimizada. E ainda, apesar da utilização deste algoritmo em diversos trabalhos científicos [Almeida 2009; Machado et al. 2009; Nunes 2016], o processo de desenvolvimento de sistemas de biometria facial, que utilizam algoritmo *Eigenface,* não são apresentados detalhadamente nos trabalhos científicos; o que dificulta, para muitos profissionais e estudantes, a utilização do algoritmo.

Nesse contexto, o objetivo desse artigo é contribuir com uma abordagem detalhada do processo de implementação de um *software* que utiliza o algoritmo *Eigenface* em uma aplicação Android, utilizando os recursos da biblioteca *OpenCV*. Serão apresentadas todas as etapas necessárias de configuração do ambiente e programação e detalhes da utilização da biblioteca *Eigenface*. E também, serão apresentados os testes de performance do sistema, relatando o potencial da biblioteca para reconhecimento de faces; auxiliando, assim, toda a comunidade científica na utilização deste algoritmo na área de biometria.

2. Reconhecimento facial

O sistema de reconhecimento facial é um tipo de biometria que possui a habilidade de detectar a face de uma pessoa imagem e compará-la com uma foto já existente no banco de dados. Essa comparação é realizada utilizando a detecção e o ajuste de pontos altos, baixos e dos contornos presentes no rosto [Almeida 2009; Costa et al. 2018; Nunes 2016].

O reconhecimento facial utiliza um conjunto de imagens coletadas em um período específico do tempo, o que facilita a busca de características da face e a distinção dessa em relação ao restante do ambiente. Além disso, a implementação dessa tecnologia é acessível, já que as fotografias faciais fazem parte da rotina de documentos oficiais e é uma maneira de autenticação menos invasiva e mais barata para se obter a imagem 2 D [Junior and Filho 2015; Lourenço 2009]. Contudo, essa metodologia apresenta como desvantagens a possibilidade de seu funcionamento sofrer interferência das condições ambientais, como iluminação, e pessoais tais como: a posição do indivíduo, idade e estado emocional; acarretando diversos níveis de dificuldades de implementação [Magalhães 2004].

2.1. Algoritmo *Eigenface*

O algoritmo *Eigenface* é uma técnica de reconhecimento facial que utiliza a Análise de Componentes Principais (PCA) para a decomposição de imagens em pequenos conjuntos de subimagens, denominados *eigenfaces*, com o intuito de codificar os traços importantes de um grupo de faces, auxiliando na diferenciação entre elas [Resende and Pereira 2015; Silva 2012, 2008]. O método PCA é uma análise estatística equivalente à covariância dos autovetores e autovalores da matriz de covariância de dados. Essa análise gera um grupo pequeno de componentes que sintetizam os dados originais, minimizando a dimensionalidade desses, preservando os componentes importantes. Assim, nessa técnica há a eliminação de componentes desnecessários, mantendo-se somente as informações relevantes em um espaço multidimensional, resultando na redução da dimensionalidade dos dados originais [Braga 2013; Diniz et al. 2013].

De acordo com os desenvolvedores do algoritmo *Eigenface*, em um espaço de alta dimensionalidade, as imagens das faces estão dispostas de maneira previsível, o que facilita a descrição dessas figuras em um espaço de dimensão menor com [Silva 2008]. Dessa forma, é preciso fazer uso do método PCA para encontrar vetores (*eigenfaces*) que melhor descrevam a distribuição de imagens dentro do espaço vetorial. Após encontrar as *eigenfaces*, essas definem um espaço pelo qual as imagens faciais serão projetadas, e a diferença entre os rostos conhecidos será mais bem representada [Diniz et al. 2013; Silva 2008]. Assim, o reconhecimento facial pelo algoritmo em questão é realizado projetando a nova imagem no subespaço desenvolvido pelas *eigenfaces* e logo depois é feita a comparação do posicionamento obtido por essa imagem com a posição dos sujeitos conhecidos [Nunes 2016].

Para fazer o cálculo matemático do algoritmo *Eigenface*, primeiro é necessário encontrar o vetor médio (Equação 1.0) da distribuição das imagens γ 1 , γ 2,..., γ M da base de dados, de uma matriz de dimensões N^2. Esse vetor será utilizado para encontrar o vetor diferença (Equação 1.1), equivalente à subtração entre os vetores do conjunto e o vetor médio [Turk and Pentland 1991].

Equação 1 –Vetor médio da distribuição das imagens γ 1 , γ 2,..., γ M

$$\phi = \frac{1}{M}\sum_{n=1}^{M}\gamma_n \qquad (1.0)$$

Equação 1.1 –Vetor diferença

$$n_i = \gamma_i - \phi \qquad (1.1)$$

A partir do vetor de desvio padrão do vetor diferença, será obtida a matriz A $A = \left[\phi_1 \phi_2 ... \phi_M\right]$ da qual é obtida a sua covariância $C = AA^A = \frac{1}{M}\sum_{n=1}^{M} \phi_n \phi_n^T$ para que desse modo o conjunto dos M autovetores da matriz sejam calculados. Porém, devido à matriz de covariância possuir uma dimensão N^2,é preciso calcular a matriz intermediária $L = A^T A$ de tamanho M x M para que os cálculos sejam viáveis matematicamente [Nunes 2016; Turk and Pentland 1991].

Os autovetores dessa matriz são conseguidos por meio da equação (1.2) $A^T A v_i = \mu_i v_i$ onde i= 1, 2,....., *M*, que, após a pré-multiplicação por A, é resultante na fórmula $AA^T A v_i = \mu_i A v_i$,onde i= 1, 2,..., *M* . E, substituindo Av_i por u_i e AA^T por C, é capaz de encontrar os M autovetores da matriz de covariância por meio da expressão matemática $Cu_i = \mu_i u_i$ onde i= 1, 2,....., *M* [Diniz et al. 2013]. Os M autovetores encontrados, compõe o espaço de faces do problema, sobre o qual cada figura será projetada, e essa projeção é encontrada por meio da equação $\omega_k = u_k^T \phi$ ($k = 1,2,...M'$), em que cada um dos pesos ω_k calculados compõe o vetor de pesos $\omega^T = \left[\omega_1 \omega_2 ... \omega_{M'}\right]$, os quais são as representações das respectivas imagens no espaço de face, que correspondem aos pontos analisados [Turk and Pentland 1991]. Sendo ω os pesos obtidos da figura a ser reconhecida e Ω_k corresponde à média do peso de todas as imagens pertencentes à classe *k* (ou pessoa *k*), a imagem é classificada como pertencente à classe quando resulta no menor valor de ε_k, considerando que ε esteja abaixo de um limiar θ_ε. Esse limiar refere–se à distância máxima para garantir que a imagem faça parte de tal classe. Caso o ε_k seja maior que o limiar θ_ε, a imagem é considerada desconhecida e pode, opcionalmente, ser usada para se criar uma nova classe (representando uma nova pessoa. Outro limiar determinado é a distância à qual a figura em questão localiza-se no espaço de face. Se a distância for maior do que o limiar encontrado, a imagem não será considerada de um rosto, assim não podendo ser classificada. Nesse caso, a distância da imagem para o espaço de faces é obtida pela expressão matemática $\varepsilon^2 = \left\|\left(\phi - \phi_f\right)\right\|^2$ em que $\phi_f = \sum_{i=1}^{M'} \omega_i u_i$ [Nunes 2016].

Entretanto, a iluminação, presença de ruídos, qualidade ruim ou dimensão inadequada nas imagens obtidas em ambientes comuns de usuários de dispositivos móveis dificultam o processo de reconhecimento facial. Uma alternativa para solucionar essas influências é utilizar as técnicas de pré-processamento de imagens [Jesus et al. 2015; Machado et al. 2009].

2.2 *OpenCV*

Open Source Computer Vision(OpenCV) é uma biblioteca de livre acesso que contém a implementação de várias metodologias para a elaboração de aplicativos no campo de visão computacional. Esse conjunto de ferramentas apresenta um bom desempenho nos cálculos para tratativa de imagens, pois a sua linguagem original é C/C++ [Braga 2013].

Além disso, essa biblioteca apresenta o código fonte accessível para qualquer alteração conforme a necessidade do usuário, podendo ser usada em diversas plataformas como, por exemplo, JAVA com o pacote *Software Development Kit (SDK)Android* com o *Integrated Development Environment (IDE)*[Farina 2012]. A *OpenCV* possui funções de processamento de imagens, de estrutura de dados, álgebra linear, processamento de vídeo, interface gráfica do usuário (GUI), controle de *mouse* e de teclado e mais de 2.500 algoritmos, incluindo o algoritmo *Eigenface,* que é uma técnica para reconhecimento facial e que pode ser utilizado para manter a segurança computacional em dispositivo móveis [Braga 2013], o qual será apresentado nesse trabalho.

3. Metodologia

. Nessa sessão, serão apresentadas todas as etapas de implementação de um *software,* utilizando o algoritmo *Eigenface,* baseado na detecção de padrões, utilizando a biblioteca *OpenCV* no ambiente *Android*. É importante ressaltar que todas as etapas apresentadas a seguir, foram testadas exclusivamente com as versões específicas que serão descritas. A utilização de versões diferentes, pode implicar no não funcionamento do sistema. Desta forma, a compatibilidade com outras versões e, conseqüentemente, o correto funcionamento do sistema, depende, exclusivamente, da continuidade de compatibilidade entre as versões que o fabricante/distribuidor da biblioteca fornece.

Esse trabalho foi desenvolvido no sistema operacional Linux Ubuntu 16.04. Para a implementação do *software,* foi necessário obter os *softwares JAVAJava Development Kit(JDK)*8 e IDEAndroid Studio 2.2 (ou superior com o *plug-inAndroid* para *Gradle*), pois esses equivalem, respectivamente, ao pacote de desenvolvimento e de ambiente de programação, que apresentam a capacidade de controlar as tarefas do dispositivo e a criação do aplicativo para reconhecimento facial [NDK 2018].

As etapas a serem seguidas para a implementação do *software* estão especificas no fluxograma da Figura 1.

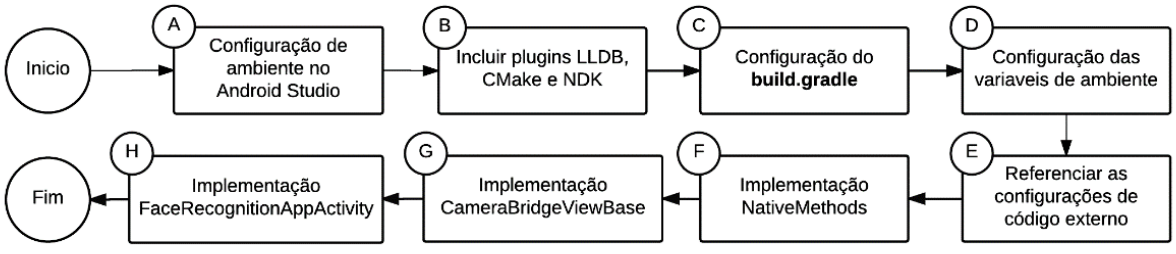

Figura 1 – Fluxograma das etapas de implementação do *software.*

A) A primeira etapa corresponde à configuração de ambiente no *Android Studio*, na qual é necessário o acesso aos seguintes componentes:

- *Native Development Kit* (NDK): é um conjunto de ferramentas utilizadas para programar uma parte da aplicação em linguagens nativas, com o intuito de manipular componentes físicos de dispositivos, como sensores, câmera e toque de entrada. No caso desse tutorial, o NDK é o código cpp.
- *Cmake:* corresponde à ferramenta que trabalha ao lado do *Gradle* para compilação externa e construção da biblioteca nativa.
- LLDB: é a ferramenta para depuração de um processo que está sendo executado em um sistema diferente do que o próprio depurador.No caso do reconhecimento facial pelo *software* com o algoritmo *Eigenface,* o LLDB estará depurando o código nativo cpp.

B) Esses componentes podem ser instalados, usando o *Software Development Kit (SDK)Manager* na seguinte ordem:

1.Para um novo projeto, selecione Ferramentas>*Android*> Gerenciador de SDK, na barra de menus.

2.Clique na guia Ferramentas do SDK.

3.Marque as opções ao lado de LLDB**,** *CMake* e NDK (Figura 2):

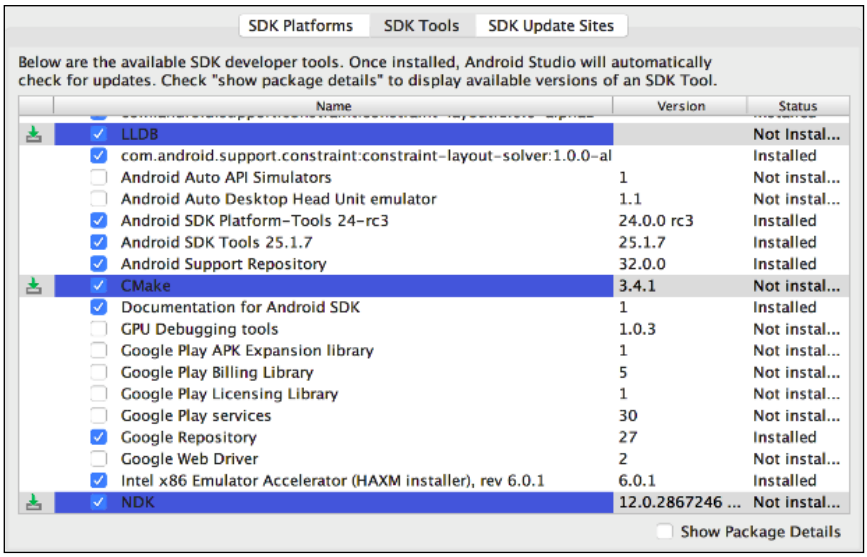

Figura 2 –Configurações do SDK Manager.

C) Logo após a montagem do ambiente do projeto, é preciso habilitar os tipos de *hardware* que podem ser executados pelo pacote NDK. Desse modo, nas configurações do *gradle* do projeto, devem ser incluídos os respectivos trechos (Figura 3).

```
vectorDrawables.useSupportLibrary = true
ndk {
    abiFilters 'armeabi-v7a', 'arm64-v8a', 'x86_64', 'x86'
    stl 'gnustl_static'
    cFlags '-std=gnu++11 -fexceptions -frtti'
}
```

Figura 3 –Trechos para habilitação dos tipos de hardware.

D) A quarta etapa a ser feita é a criação das variáveis de ambiente, na ordem apresentada na Figura 4, para que essas variáveis sejam inicialmente apontadas ao sistema operacional e em seguida para o projeto atual. O primeiro passo da criação das variáveis de ambiente é a inclusão da dependência da biblioteca *OpenCV* no projeto.

```
dependencies {
    compile 'com.android.support:appcompat-v7:24.2.1'
    compile 'com.android.support:design:24.2.1'
    compile 'jp.seesaa.android:opencv:3.1.0.0'
}
```

Figura 4 – Inclusão das dependências da biblioteca *OpenCV.*

E) Referenciar as configurações de código externo para o projeto atual com o intuito de apontar as variáveis de ambiente. Porém, é necessário, primeiramente, fazer o *download* das bibliotecas (que são as variáveis de ambiente) utilizadas para o reconhecimento facial:

- O *download* da biblioteca *OpenCV* está disponível em: https://sourceforge.net/projects/opencvlibrary/files/opencv-android/. A versão utilizada no *software* foi a versão 2.4.13.
- Para o *download* da biblioteca *Eigen*, é preciso acessar o *link*:https://eigen.tuxfamily.org/ e escolher a versão 3.3.2.

Para a inclusão das bibliotecas nas variáveis de ambiente, é preciso entrar no terminal: $ sudo gedit /etc/profile. Em seguida, editar o arquivo de configuração para a inclusão das bibliotecas baixadas e apontá-las para as variáveis de ambiente (Figura 5).

```
#OPENCV
export OPENCV_ANDROID_SDK=/home/usuario/OpenCV-android-sdk
export EIGEN3_DIR=/home/usuario/eigen-eigen-da9b4e14c255
#echo $OPENCV_ANDROID_SDK $EIGEN3_DIR
```

Figura 5 –Variáveis de ambiente no sistema.

Após as variáveis de ambiente serem devidamente referenciadas pelo sistema operacional, elas devem ser apontadas no projeto que está sendo desenvolvido (Figura 6).

```
externalNativeBuild {
    ndkBuild {
        path 'src/main/cpp/Android.mk'
    }
}
```

Figura 6 –Configuração das variáveis de ambiente no sistema operacional.

Depois do mapeamento das variáveis de ambientes no *build.gradle* e referência .mk, deve-se configurar os conteúdos presentes nessas variáveis e incluir a biblioteca *FaceRecognitionLib* por meio do *download* dessa ferramenta no *link* http://lauszus.github.io/FaceRecognitionLib.

A biblioteca *FaceRecognitionLib* apresenta a função de realizar os cálculos matemáticos dos algoritmos *Eigenface* e *Fisherface* em linguagem Cpp, isto é,utiliza como base a biblioteca *OpenCV* para a implementação do algoritmo de reconhecimento facial. Além disso, o *script Matlab* foi utilizado neste estudo para promover a capacidade de realizar cálculos matriciais em Cpp. Desse modo, as configurações estão apresentadas na Figura 7.

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

OPENCV_INSTALL_MODULES := on
include $(OPENCV_ANDROID_SDK)/sdk/native/jni/OpenCV.mk

LOCAL_MODULE := face-lib
LOCAL_SRC_FILES += $(LOCAL_PATH)/face-lib.cpp $(LOCAL_PATH)/FaceRecognitionLib/Facebase.cpp
LOCAL_SRC_FILES += $(LOCAL_PATH)/FaceRecognitionLib/Eigenfaces.cpp $(LOCAL_PATH)/FaceRecognitionLib/Fisherfaces.cpp
LOCAL_SRC_FILES += $(LOCAL_PATH)/FaceRecognitionLib/PCA.cpp $(LOCAL_PATH)/FaceRecognitionLib/LDA.cpp
LOCAL_C_INCLUDES += $(EIGEN3_DIR) $(LOCAL_PATH)/FaceRecognitionLib/RedSVD/include
LOCAL_LDLIBS += -llog -ldl
LOCAL_CPPFLAGS += -std=gnu++11 -frtti -fexceptions

include $(BUILD_SHARED_LIBRARY)
```

Figura 7 – Configurações das variáveis ambiente.

Posteriormente à definição das configurações, é necessário referenciar essas ao projeto novo,criando um arquivo *Application.mk*, e configurando-o com as instruções ilustradas na Figura 8.

```
APP_PLATFORM := android-24
APP_ABI := armeabi-v7a arm64-v8a x86_64 x86
APP_STL := gnustl_static
APP_CPPFLAGS := -std=gnu++11 -frtti -fexceptions

ifeq ($(NDK_DEBUG),0)
  APP_CPPFLAGS += -DNDEBUG
endif
```

Figura 8 –Criação de um arquivo *Application.mk*.

F) Etapas de implementação do *software*

Thread é um subsistema que executa um novo processo dentro de um programa, na qual possui como função a divisão de duas ou mais atividades para serem executadas conjuntamente [Marimoto 2005]. As principais *threads* da implementação do sistema de reconhecimento facial são: *NativeMethods, CameraBridgeViewBase*, *FaceRecognitionAppActivity*. As etapas de implementação do *software* estão representadas no fluxograma da Figura 9.

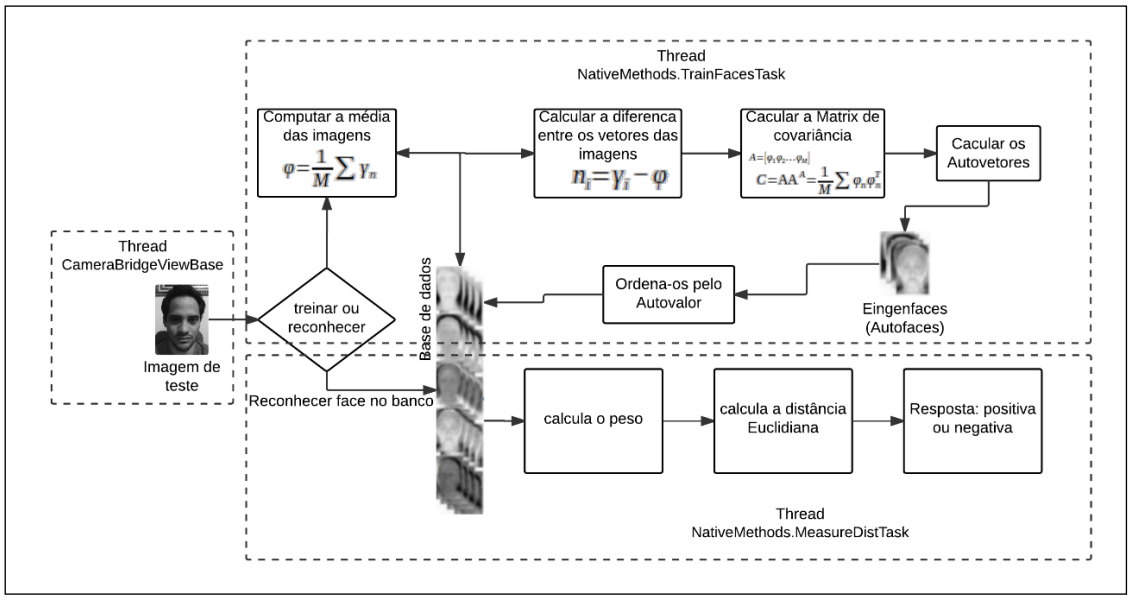

Figura 9 –Fluxograma da implementação do uso das principais *threads* *(Adaptado de* [Correia et al. 2014]*)*

Cada método que implementa a biblioteca *FaceRecognitionLib* apresenta a estrutura demonstrada na Figura 10. Dessa forma, para fazer a chamada de um determinado método, basta fazer a substituição do nome (Figura 11).

```
JNIEXPORT void JNICALL Java_{nome_pacote}_{nome_class}_{nome_metodo}(params...) {
    // Implementação de treinamento em Cpp
}
```

Figura 10 – Estrutura básica do método que implementa a biblioteca *FaceRecognitionLib.*

```
JNIEXPORT void JNICALL Java_br_exemple_facerecognition_NativeMethods_TrainFaces(
    JNIEnv, jobject, jlong addrImages, jlong addrClasses) {
    // Implementação de treinamento em Cpp

}
```

Figura 11 – Exemplo de chamada de método da biblioteca *FaceRecognitionLib.*

Logo após efetuar a nomenclatura dos métodos, deve ser criado o arquivo para mapeá-los. E a biblioteca *FaceRecognitionLib* deve carregada por meio da função apresentada na Figura 12.

```
class NativeMethods {
    static void loadNativeLibraries() {
        System.loadLibrary("face-lib");
    }
 // Conteúdo
}
```

Figura 12 –Função de carregamento da biblioteca *FaceRecognitionLib.*

A utilização do método é definida pelo nome *native*, que pode ser vista na ordem das chamadas das funções apresentadas na Figura 13. Com isso, os métodos de cálculos matemáticos podem ser usados na *activity* principal da aplicação *Android*.

```
private static native void TrainFaces(long addrImages, long addrClasses);

private static native void MeasureDist(long addrImage, float[] minDist,
                                       int[] minDistIndex, float[] faceDist,
                                       boolean useEigenfaces);
```

Figura 13 – Implementação do *native method.*

G) Uma função importante para o reconhecimento facial em dispositivo móvel é o controle constante da câmera/*hardware*. O pacote SDK do ambiente *Android* oferece a implementação nativa *SurfaceView* que possui a função de monitorar as atividades da câmera do dispositivo. A partir da codificação da *SurfaceView,* uma nova classe (*CameraBridgeViewBase,)* é criada para executar a função de monitoramento constante da câmera. Devido ao fato de utilizar a biblioteca *OpenCV* na aplicação atual, a implementação da *CameraBridgeViewBase* utiliza o *ListenerCvCameraViewListener2* da biblioteca *OpenCV*, com isso as funções de controlar a câmera ativada, de processar imagens correntes, de chamar um ouvinte para transmissão da câmera para a aplicação são também herdadas para *CvCameraViewListener2*. Desse modo, a estrutura do projeto foi arquitetada conforme Figura 14.

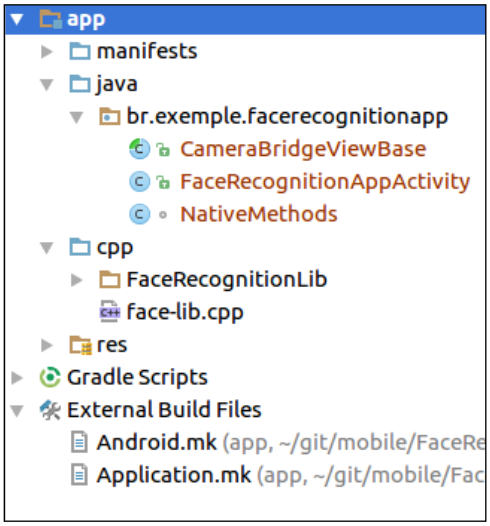

Figura 14 –Estrutura do projeto desenvolvido.

H) A outra etapa a ser seguida é a codificação da interface principal da aplicação, que consiste na utilização de variáveis globais que recebem a imagem corrente da câmera. Ao criar a *interface* principal, deve-se codificar o *implements* da *classCameraBridgeViewBase* , que possui três métodos obrigatórios de execução que são (Figura 15): 1) o método *onCameraViewStarted,* que possui o intuito de disparar o início das variáveis globais durante o a inicialização da interface do programa*; 2)* O

método *onCameraFrame,* que tem a função de monitoramento constante da câmera do dispositivo móvel, apresenta como argumento (*inputFrame*) a nova imagem para manipulação; 3) O método *onCameraViewStopped* irá encerrar as variáveis globais que estarão observando o estado da câmera.

```
public class FaceRecognitionAppActivity extends AppCompatActivity
        implements CameraBridgeViewBase.CvCameraViewListener2 {

    //Variaveis globais, imagem corrente
    private Mat mRgba, mGray;

    // metodos obrigatorios pela implementação de 'CameraBridgeViewBase.CvCameraViewListener2'
    public void onCameraViewStarted(int width, int height) {
        mGray = new Mat();
        mRgba = new Mat();
    }
    public void onCameraViewStopped() {
        mGray.release();
        mRgba.release();
    }
    public Mat onCameraFrame(CameraBridgeViewBase.CvCameraViewFrame inputFrame) {
        mGray = inputFrame.gray();
        mRgba = inputFrame.rgba();
        // logica para tratar a iamgem conforme necessicade
        return mRgba;
    }
```

Figura 15–Três métodos obrigatórios de execução para monitorar a câmera de captura de imagens no ambiente Android.

4 Resultados

A interface do programa desenvolvido apresenta as funções de "reconhecimento", de "treinamento" e "limpar base" (Figura 16). A função de "reconhecimento" corresponde à captura da imagem corrente na câmera e à comparação dessa com as imagens salvas no banco. Já a função de "treinamento" tem como objetivo inserir uma nova imagem e armazenamento dessa no banco de dados, assim preparando o *software* para reconhecimento daimagem capturada. E, por último, a função "limpar base" que é a tarefa equivalente a limpar a base de dados dos treinamentos de imagem. O *software* informa, por meio de uma mensagem escrita a partir de uma análise comparativa, se houve ou não reconhecimento. Em casos de não reconhecimento facial, a distância encontrada pelo cálculo do desvio padrão será informada.

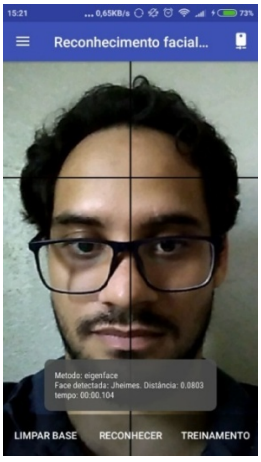

Figura 16 – Interface principal do programa desenvolvido.

Para o processo de avaliação do sistema, quinze imagens faciais, do próprio autor, foram utilizadas para o teste do *software* implementado neste estudo (Figura 17) . Foram realizados diversos testes de reconhecimento de faces, considerando a luminosidade do ambiente. No presente estudo, os testes foram realizados no dispositivo *Xiaomi redmi note3*®, com imagens apenas do rosto (frontais), digitalizadas, em dimensão 200 x 200.

Figura 17: Representação das imagens utilizadas para o teste do *software* implementado

A Tabela 1 apresenta os principais resultados, descrevendo, para cada teste, se houve reconhecimento, a distância encontrada da imagem coleta com a base de dados e o tempo necessário para a execução do algoritmo. Os testes realizados não tiveram a intenção de validar o algoritmo *Eigenface*, e sim apenas apresentar a eficácia do processo de desenvolvimento de software de biometria facial apresentada neste trabalho.

Tabela 1- Resultados do teste do algoritmo *Eigenface* implementadopara dispositivos móveis

Experimentos	Reconhecimento Facial	Luminosidade	Distância	Tempo (milissegundos)
Expressão séria	Sim	Presente	0.02	92
Expressão séria	Não	Baixa	0.45	48
Óculos escuros	Sim	Presente	0.03	97
Óculos escuros	Não	Baixa	0.26	72
Óculos de grau	Sim	Presente	0.04	70
Óculos de grau	Não	Baixa	0.27	84
Chapéu	Sim	Presente	0.04	98
Chapéu	Sim	Baixa	0.03	73
Chapéu + óculos escuros	Sim	Presente	0.06	74
Chapéu + óculos escuros	Sim	Baixa	0.02	61
Careta	Sim	Presente	0.03	11

Careta	Sim	Baixa	0.02	91
Metade esquerda da face	Não	Presente	0.30	67
Metade inferior da face (nariz, boca, queixo)	Não	Presente	0.26	116
Metade superior da face (testa, olhos, nariz)	Não	Presente	0.28	55
Média (±)Desvio Padrão)			0,14(±)	73,93 (±)

Durante os testes, foi observado que é necessário realizar primeiro o treinamento da face com a presença de luminosidade controlada para depois haver o reconhecimento facial no escuro. Também, para a detecção do rosto, foi preciso ter luminosidade suficiente para capturar o contorno desse. Isso vai ao encontro com o estudo de [Diniz et al. 2013], que afirma que a luminosidade influencia os resultados dos cálculos do algoritmo *Eigenface*. Assim, é importante controlar esse fator externo para o reconhecimento facial.

5 Conclusão

Atualmente, nos dispositivos móveis são armazenadas informações sigilosas que requerem um acesso restrito. Por isso, é preciso medidas de segurança computacional. Nesse contexto, a biblioteca *OpenCV* possui várias ferramentas cuja função é o desenvolvimento de aplicativos na área de visão computacional. Nela está disponível o algoritmo *Eigenface* que é uma opção restringir o acesso e manter a segurança de dados contidos em dispositivos móveis.

Para contribuir com esse contexto, este trabalho apresentou, em detalhes, todos os passos necessários para implementar um software de reconhecimento facial, em ambiente Android, utilizando o algoritmo *Eigenface* da biblioteca *OpenCV*. O resultado apresentado neste trabalho agrega conhecimento de desenvolvedores que necessitam implementar softwares com um padrão de segurança baseado em biometria facial, utilizando a biblioteca *OpenCV*. E ainda, o passo a passo apresentado na metodologia, pode ser, também, utilizado na área acadêmica para ensinar a desenvolver um *software* de reconhecimento facial. Os resultados encontrados com programa desenvolvido indicam que o algoritmo *Eigenface* pode sofrer influência da luminosidade e da posição do individuo, e, por isso, é preciso haver controle desses fatores para reduzir a taxa de erro positivo na implementação do algoritmo *Eigenface*.

E também, a ausência do uso de métodos de pré-processamento de imagens pode ter influenciado nos resultados encontrados, uma vez que a coleta de dados foi em ambientes comuns dos usuários de dispositivos móveis, sem haver o controle da luminosidade. Em virtude disso, sugere-se a elaboração de novos estudos que observem se a aplicabilidade do *software* será alterada com o uso conjunto de métodos de pré-processamento de imagens. E ainda, podem ser realizados estudos que testem novos

algoritmos, como o *Local Binary Patterns* (LBP), com o intuito de buscar uma porcentagem de o reconhecimento facial, visto que os resultados apresentaram apenas a possibilidade de acerto ou de erro.

Referências

Almeida, G. B. (2009). Autenticação Segura Baseada em Biometria voltada para a Dinâmica da Digitação. Universidade Federal de Goiás.

Braga, L. F. Z. (2013). Sistemas de Reconhecimento Facial. Universidade de São Carlos.

Correia, T., Piteri, A. A., Silva, F. A. and Pereira, D. R. (2014). Development of an application for security based in face recognition on Android platform. In *X Workshop de Vis˜ao Computacional*.

Costa, L., Obelheiro, R. and Fraga, J. (2018). Introdução à Biometria. http://www.advancedsourcecode.com/minicurso_biometria.pdf, [accessed on Feb 20].

Diniz, F. A., Neto, F. M., Júnior, F. das C. and Fontes, L. (2013). RedFace: um sistema de reconhecimento facial baseado em técnicas de análise de componentes principais e autofaces: comparação com diferentes classificadores. *Revista Brasileira de Computação Aplicada*, v. 5, n. 1, p. 42–54.

Farina, A. M. (2012). BioMobile: Sistema de Identificação de Usuários em Dispositivos Móveis na Plataforma Android Utilizando Reconhecimento de Faces a Partir de Vídeo. Universidade Estadual Paulista.

Jesus, L., Deivite, A., Sapucaia, F. and Frias, D. (2015). Análise de Métodos de Processamento de Imagens para Reconhecimento Facial utilizando Fisherfaces em Imagens sob Condições Desfavoráveis. In *XIII Workshop de Trabalhos de Iniciação Científica e Graduação da Escola Regional de Computação*.

Junior, A. and Filho, M. (2015). Aplicações de Processamento de Imagens a Sistemas de Segurança. https://www.ademar.org/texts/seguranca-processamento-imagens.pdf, [accessed on Feb 15].

Lourenço, G. F. da F. (2009). Reforço da Segurança das Biométricas utilizando Codificação de Fonte Distribuída. Universidade Técnica de Lisboa.

Machado, B. B., Barros, M. M., Maia, M. L. and Silva, G. P. (2009). Implementação de um Algoritmo de Reconhecimento Facial Usando EIGENFACE. *Exacta*, v. 2, n. 1.

Magalhães, P. S. T. (2004). Estudo dos padrões de digitação e sua aplicação na autenticação biométrica. Universidade do Minho.

Marimoto, C. (2005). Thread. https://www.hardware.com.br/termos/thread, [accessed on Feb 3].

NDK (2018). Primeiros passos com o NDK. https://developer.android.com/ndk/guides/index.html, [accessed on Mar 30].

Nunes, G. M. (2016). Visão computacional aplicada à detecção e ao reconhecimento facial. Universidade Federal do Rio de Janeiro.

Resende, C. and Pereira, M. (2015). Visão Computacional aplicada em reconhecimento facial na busca por pessoas desaparecidas. *Exacta*, v. 8, n. 1, p. 95–107.

Silva, R. L. (2012). Indexação de Faces em Estruturas de Dados Métricas. Universidade Federal de Itajubá.

Silva, V. A. (2008). Comparação entre técnicas de reconhecimento de faces para controle de acesso a computadores. Universidade Católica Dom Bosco.

Turk, M. and Pentland, A. (1991). Eigenfaces for recognition. *Journal of*

Cognitive Neuroscience, v. 3, n. 1, p. 71–86.