UNIVERSIDADE DE SÃO PAULO

ESCOLA DE ENGENHARIA DE SÃO CARLOS

DEPARTAMENTO DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

Reconhecimento facial aplicado ao controle de presença de alunos

Autor: Thiago Francisco Martins

Orientador: Prof. Dr. Evandro Luis Linhari Rodrigues

São Carlos

2016

Thiago Francisco Martins

Reconhecimento facial aplicado ao controle de presença de alunos

Trabalho de Conclusão de Curso apresentado
à Escola de Engenharia de São Carlos, da
Universidade de São Paulo

Curso de Engenharia Elétrica

ORIENTADOR: Prof. Dr. Evandro Luis Linhari Rodrigues

São Carlos
2016

Martins, Thiago Francisco
M813r Reconhecimento facial aplicado ao controle de presença de alunos / Thiago Francisco Martins; orientador Evandro Luis Linhari Rodrigues . São Carlos, 2016.

Monografia (Graduação em Engenharia Elétrica com ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2016.

1. Reconhecimento facial. 2. controle de presença. 3. visão computacional. I. Título.

FOLHA DE APROVAÇÃO

Nome: Thiago Francisco Martins

Título: "Reconhecimento facial aplicado ao controle de presença de alunos"

Trabalho de Conclusão de Curso defendido e aprovado em 20/06/2016,

com NOTA 10,0 (DEZ, ZERO), **pela Comissão Julgadora:**

Prof. Associado Evandro Luís Linhari Rodrigues - (Orientador - SEL/EESC/USP)

Prof. Associado Adilson Gonzaga - (SEL/EESC/USP)

Mestre Alex Antonio Affonso - (Doutorando - SEL/EESC/USP)

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Dr. José Carlos de Melo Vieira Júnior

Dedicatória

Dedico este trabalho à meus pais, minha irmã e minha namorada.

Thiago Francisco Martins.

Agradecimentos

Agradeço aos meus pais pelo suporte e acompanhamento dado por toda minha trajetória acadêmica.

Agradeço à minha namorada pelas palavras de apoio e confiança quando este projeto apresentava dificuldades.

Ao meu orientador, Evandro Luís Linhari Rodrigues pela oportunidade, paciência e apoio no trabalho desenvolvido.

Aos meus amigos da USP São Carlos, pela amizade, apoio e alegrias.

Thiago Francisco Martins.

Resumo

O campo de reconhecimento facial tem evoluído cada vez mais nos últimos anos devido à necessidade crescente de identificar pessoas de forma precisa para permitir o acesso restrito a determinado recursos, a identificação de possíveis criminosos e o controle de presença em instituições. O reconhecimento facial desperta interesse em relação a outras técnicas biométricas pois apresenta várias vantagens como a utilização de equipamentos simples e relativamente baratos para sua implementação e a possibilidade de identificação de pessoas a distância. Este trabalho apresenta o estudo, desenvolvimento e avaliação de um sistema de reconhecimento biométrico aplicado ao controle de presença de alunos. O objetivo do trabalho foi desenvolver uma aplicação para dispositivos Android que possibilite ao professor realizar o controle de presença de alunos por meio de reconhecimento facial e a geração de relatório. Os resultados mostraram que o sistema apresenta uma taxa acerto de reconhecimento de identidade em torno de 80%, mas também que existem modificações que podem ser feitas para sua evolução, como por exemplo, em sua interface.

Palavras-Chave: reconhecimento facial, controle de presença, visão computacional.

Abstract

Face recognition has increasingly evolved in recent years due to the growing need of identifying people precisely to allow restricted access to certain resources, identifying of possible criminals and control presence in institutions. Face recognition is more interesting than other biometric techniques because it has several advantages such as the need of simple and relatively inexpensive equipment to implement and the possibility of identifying people at a distance. This paper presents the study, development and evaluation of a biometric recognition system applied to students attendance track. The objective was to develop an application for Android devices that enables teachers to track students attendance through face recognition and generate a report on the track. Results have shown that the system has a level of accuracy in recognition near 80% , but also that there are changes that can be made to improve it, especially in its interface.

Keywords: face recognition, attendance track, computer vision.

Lista de Figuras

Lista de Tabelas

Siglas

CPU	*Central Process Unit*
FPS	*Frames Per Second*
GB	*Gigabyte*
GPU	*Graphics Processing Unit*
IDE	*Integrated Development Environment*
IHC	Interação Humano-Computador
LBP	*Local Binary Pattern*
LBPH	*Local Binary Patterns Histograms*
MB	*Megabyte*
MB-LBP	*Multi Block - Local Binary Patterns*
RAM	*Ramdom Acess Memory*
SUS	*System Usability Scale*

Sumário

Capítulo 1

Introdução

Atualmente, instituições públicas e privadas estão cada vez mais interessadas em sistemas de segurança. Estes sistemas possibilitam o controle de acesso de pessoas a determinadas áreas restritas, reconhecimento de criminosos e até mesmo controle de presença. Uma das formas de implementar estes sistemas de segurança que vêm se tornando cada vez mais comum é a biometria.

Um sistema biométrico é essencialmente um sistema de reconhecimento de padrões que opera através de dados biométricos adquiridos de um individuo [1]. Existem diversos identificadores biométricos, tais como a impressão digital, a retina e a voz. Esse trabalho tem como foco a face. Há duas formas possíveis de utilização da biometria por reconhecimento facial, a autenticação e a identificação.

No método da autenticação o sistema valida a identidade do usuário, ou seja, compara as informações da pessoa com as informações previamente cadastradas no banco de dados. No método da identificação o sistema recebe as características do usuário e identifica se este usuário pertence ao banco de dados e a qual identidade cadastrada no banco este usuário pertence.

Uma característica biométrica, segundo [2], para ser utilizada na verificação ou identificação de uma pessoa, deve atender os seguintes requisitos:

- Universalidade: Todas as pessoas devem possuir a característica;
- Exclusividade: Singularidade, o que significa que não devem existir 2 pessoas iguais em termos de características;
- Permanência: Se a característica permanece com o passar do tempo;

- Coletividade: A característica pode ser medida quantitativamente;
- Desempenho: Refere-se a precisão de identificação de uma pessoa, os recursos utilizados para atingir a mesma e os fatores do ambiente que podem prejudicar a identificação
- Aceitabilidade: Indica a quantidade de pessoas que aceitam o sistema biométrico, quanto maior a aceitabilidade, menor a relutância das pessoas em utilizá-lo.
- Dificuldade de falsificação: Dificuldade em utilizar técnicas fraudulentas para burlar o sistema.

Uma breve comparação de diferentes técnicas de reconhecimento baseada nesses requisitos é feita na Tabela 1.1. Nela, cada técnica recebe um nível (alto, médio ou baixo) em que atende cada um dos requisitos.

Identificador Biométrico	DNA	Face	Impressão digital	Irís	Voz
Universalidade	Alto	Alto	Médio	Alto	Médio
Exclusividade	Alto	Baixo	Alto	Alto	Baixo
Permanência	Alto	Médio	Alto	Alto	Baixo
Coletividade	Baixo	Alto	Médio	Médio	Médio
Desempenho	Alto	Baixo	Alto	Alto	Baixo
Aceitabilidade	Baixo	Alto	Médio	Baixo	Alto
Dificuldade de falsificação	Baixo	Alto	Médio	Baixo	Alto

Tabela 1.1: Comparação de sistemas biométricos

[2]

A escolha de um sistema biométrico depende altamente dos requisitos que são prioritários para uma aplicação. Sistemas que utilizam reconhecimento através da impressão digital, como reconhecimento de criminosos, são conhecidos por serem extremamente confiáveis, ao contrário de aplicações que utilizam o reconhecimento por voz, onde a aceitabilidade do usuário em utilizar o sistema é seu principal fator.

1.1 Motivação

Sistema de controle de presença é uma área bastante buscada e conta com diversas soluções. Desde o uso de leitores de impressão digital integrados a sistemas de computador ou sistemas

web, como o stratustime[3], para controle de horário trabalhado de funcionários, ou sistemas embarcados, como no trabalho de [4], até aplicativos para *smartphones* para controle de presença de alunos em aulas, como o Classroom Monitor Attendance[5] e o Attendance[6], que permitem a adição manual de alunos, a partir da seleção deles em uma lista previamente cadastrada.

Existem alguns fatores que devem ser considerados no momento de decidir qual sistema é o ideal para auxiliar um usuário na tarefa de controlar presença. Deve-se levar em consideração a quantidade, tamanho e custo dos equipamentos necessários para executar a tarefa, a mobilidade dos equipamentos, a facilidade e eficiência em se utilizar o sistema e obter resultados, a segurança que ele proporciona quanto a geração e persistência dos dados, disponibilidade de acesso aos meios eletrônicos (como por exemplo base de dados e e-mails) que oferecem maiores recursos ao usuário, entre outros.

Para uma solução para controle de presença, a utilização de uma técnica biométrica é interessante pela praticidade que proporciona e a maior credibilidade dos dados obtidos. O reconhecimento facial possui vantagens quando comparado à outras técnicas biométricas de reconhecimento pois, entre outros fatores, possibilita que a identificação seja realizada sem métodos intrusivos além de não exigir equipamentos especializados, apenas um dispositivo com câmera e com uma capacidade de processamento razoável.

Dentre os dispositivos que permitem realizar o reconhecimento facial estão computadores, *notebooks*, sistemas embarcados e *smartphones*. *Smartphones* se destacam entre os outros pois oferecem maior mobilidade ao usuário, não sendo necessário um equipamento para cada ambiente onde é realizado o controle de pessoas. Além disto, é a opção de menor custo, visto que a maior parte dos usuários em potencial já possui um *smartphone*. Outro fator determinante para implementação do sistema é a velocidade de processamento de *smartphones* modernos.

Dentre os *smartphones* disponíveis no mercado, têm destaque aqueles que possuem o sistema operacional Android. Tais aparelhos detém aproximadamente 62% do mercado de *smartphones* segundo [7]. Deste modo, o desenvolvimento de um aplicativo para Android atenderia a maior parte de usuários de *smartphones*.

1.2 Objetivos

A proposta deste trabalho foi desenvolver um motor de reconhecimento facial por meio de um aplicativo Android que possa ser utilizado em *smartphones* pessoais. Esse motor foi aplicado em um sistema de controle de presença de alunos. O sistema tem como usuário esperado um professor, que pode gerar chamadas, enviar relatórios sobre chamadas e posteriormente obter informações de presença de uma aula específica.

1.3 Organização do Trabalho

Este trabalho está distribuído em 5 capítulos, incluindo esta introdução, dispostos conforme a descrição que segue:

Capítulo 1: Apresenta uma introdução sobre sistemas biométricos, a motivação em desenvolver o projeto e seus objetivos.

Capítulo 2: Nesta parte, o sistema concebido é abordado de forma teórica apresentando métodos de visão computacional, conceitos de usabilidade e de *design*.

Capítulo 3: Descreve os materiais utilizados, bem como todos os métodos utilizados para desenvolver a aplicação. Todas as telas desenvolvidas são apresentadas, bem como os métodos utilizados para desenvolver os módulos de reconhecimento facial e o banco de dados.

Capítulo 4: Os resultados do sistema desenvolvidos são apresentados. Tópicos como requerimentos, consumo de recursos, taxa de quadros, usabilidade, taxa de acerto dos algoritmos de visão computacional e a geração de relatórios do sistema são abordados e discutidos.

Capítulo 5: São apresentadas as conclusões do trabalho e sugestões de trabalhos futuros.

Capítulo 2

Embasamento Teórico

Nesta seção serão apresentados os fundamentos teóricos e práticos necessários ao entendimento do desenvolvimento do projeto. São abordados os algoritmos de visão computacional para detecção e reconhecimento facial, aspectos de IHC (interação Humano-Computador) e desenvolvimento Android.

2.1 Visão computacional

Visão computacional é um campo que desenvolve tecnologias e teorias a fim de construir sistemas autônomos capazes de realizar tarefas baseadas no reconhecimento de padrões em uma imagem ou vídeo. Para melhor discussão dos conceitos de detecção e reconhecimento facial propostos neste trabalho, é necessário o entendimento dos fundamentos da imagem digital.

2.1.1 Imagen digital

Uma imagem digital pode ser entendida como uma imagem contínua amostrada em um arranjo matricial M x N, sendo o valor de cada elemento da matriz o nível de cinza do pixel correspondente no plano de imagem (Figura 2.1).

(a)

128	128	128	128	128	128	128	128	128	128
128	128	255	255	255	128	128	128	128	128
128	128	255	0	255	128	128	128	128	128
128	128	255	0	255	128	128	128	128	128
128	128	255	0	255	128	128	128	128	128
128	128	255	0	255	128	128	128	128	128
128	128	255	0	255	255	255	255	128	128
128	128	255	0	0	0	0	255	128	128
128	128	255	255	255	255	255	255	128	128
128	128	128	128	128	128	128	128	128	128

(b)

Figura 2.1: Imagem digital.

[1] (a)Exemplo de uma imagem digital. (b) Representação matricial correspondente.

Conforme observa-se na Figura 2.1, pixels de intensidade mais clara possuem valores superiores aos de pixels de intensidade escura. A missão da visão computacional é processar estes valores para que características da imagem digital possam ser extraídas e analisadas. Um dos campos da visão computacional é desenvolver métodos para melhorar a qualidade das imagens obtidas e consequentemente obter uma melhor análise, um dos métodos mais utilizados nesta área é a equalização de histograma.

2.1.2 Equalização de histograma

Equalização de histograma é uma das técnicas mais populares para aumentar o contraste em imagens digitais. É um processo que consiste em ajustar os níveis de cinza da imagem aumentando seu contraste ocasionando em uma melhor qualidade da imagem [8].

A Figura 2.2 ilustra uma operação de equalização de histograma. Nota-se um aumento no contraste da imagem e reduzindo os efeitos de brilho.

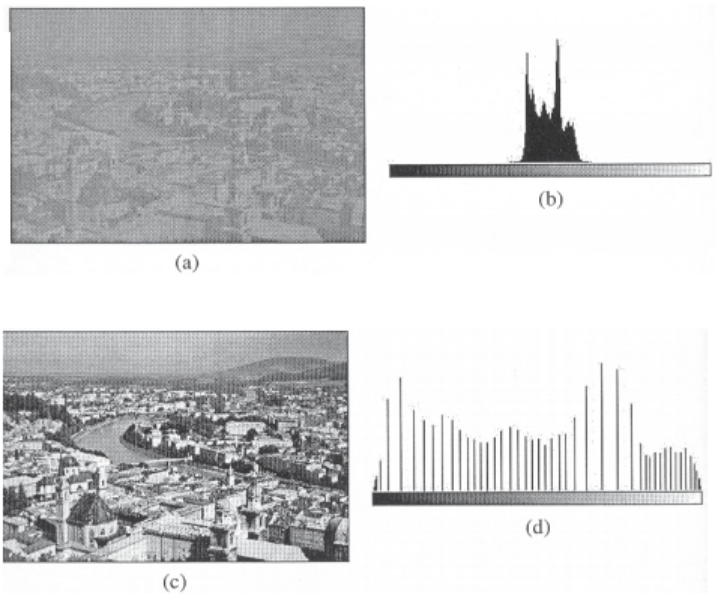

Figura 2.2: Comparação entre imagem original e equalizada.
[8] (a)Imagem original,(b) Histograma da imagem original, (c) Imagem equalizada, (d) Histograma da imagem equalizada.

2.2 Reconhecimento facial

O reconhecimento facial consiste em processamento de imagens de entrada de um sistema a fim de reconhecer a identidade de indivíduos automaticamente [9]. Embora esta seja uma tarefa fácil para seres humanos, o reconhecimento facial automático ainda é uma questão complicada, pois apresenta muitos desafios computacionais e matemáticos.

O cenário de um sistema de reconhecimento pode ser descrito da seguinte forma: dada uma imagem de entrada adquirida por meio de vídeo ou imagem estática, processa-se a imagem através de métodos de visão computacional com o objetivo de identificar se a imagem recebida apresenta a face de um dos indivíduos cadastrados previamente em um banco de dados. A solução proposta neste projeto para o problema consiste de três passos: a detecção da face do indivíduo, o pré-processamento da imagem (como por exemplo, equalização de histogramas) e por fim, o reconhecimento da identidade do individuo. A Figura 2.3 ilustra este processo.

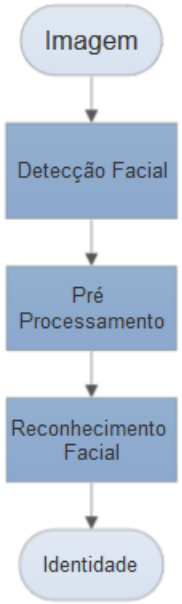

Figura 2.3: Diagrama de um módulo de reconhecimento facial.

2.2.1 Detecção facial

A detecção de face é a primeira etapa de todo o processo, e consiste na detecção da região facial de uma imagem. Esta etapa é fundamental para o funcionamento do processo inteiro pois a detecção falsa de regiões faciais pode comprometer a etapa de reconhecimento facial.Outro fator importante a se considerar nesta fase é a velocidade de detecção, pois em aplicações de vídeo, o número de quadros por segundo é sensível ao aumento do tempo de processamento.

Algoritmo de Viola-Jones

Há muitos desafios em elaborar algoritmos para tratar todos os problemas propostos pela fase de detecção, pois além dos requisitos de velocidade e confiabilidade, um algoritmo ideal ainda deve ser robusto a variações de iluminação, pose, emoção e oclusão [10]. Um dos clássicos trabalhos nesta área é o de Viola Jones [11]. Ele utiliza funções de base Haar, que são máscaras do tipo retangulares onde os valores dos pixels da região preta são subtraídos dos valores de pixels da região branca. A Figura 2.4 apresenta alguns exemplos de funções de base do tipo Haar.

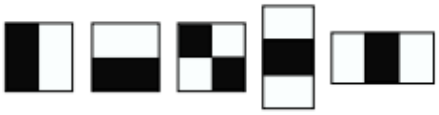

Figura 2.4: Exemplos de funções de base do tipo Haar.
[12]

Estas funções de base são utilizados para detectar padrões [11]. Ao aplicá-los sobre uma imagem, é possível identificar as regiões desta imagem onde há diferença de níveis de intensidade entre as regiões pretas e brancas da funções de base. Este método pode ser utilizado na detecção facial. Por exemplo, uma das características de presente em regiões faciais é o nível de intensidade nas regiões dos olhos ser mais escuro que o restante da face. A Figura 2.5 ilustra esta característica, onde a primeira função de base Haar mede a diferença de intensidade entre a região dos olhos e a região das bochechas e a segunda ssificador compara a intensidade da região dos olhos com a intensidade da região do nariz.

Figura 2.5: Funções de base Haar aplicado em uma imagem facial.
[11]

A diferença de intensidade de níveis de cinza entre as regiões brancas e escuras de uma funções de base resultará em um valor numérico. Para que haja correta correlação entre este valor e um possível padrão a ser reconhecido é necessário treinar classificadores.

O treinamento, próxima etapa do algoritmo de [11], consiste em treinar estas funções de base com imagens com amostras positivas e negativas de regiões faciais. O método utilizado por [11] para este treinamento é denominado *AdaBoost*, um algoritmo de aprendizado de máquina que seleciona as funções de base (chamados de classificadores fracos) que melhor

separam as amostras positivas e negativas.

A última etapa do processo então é combinar estes classificadores fracos selecionados no treinamento de modo que a detecção de padrões considerados positivos seja eficiente, formando assim classificadores fortes. Estes classificadores fortes são cascateados de modo que o algoritmo rejeite de forma rápida regiões que não satisfazem os atributos positivos treinados na etapa anterior.

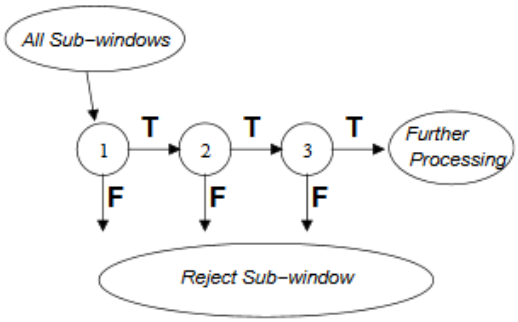

Figura 2.6: Cascateamento de classificadores.

[11] O classificador inicial (1) rejeita rapidamente amostras negativas com baixo processamento. Próximos estágios da cascata eliminam posteriores amostras negativas e exigem maior processamento.

Baseado no trabalho de [11] muitas melhorias foram propostas nos últimos anos, entre elas variações do método *boosting* como *Real Adaboost* e *Gentle Adaboost* [13]. Outra opção é melhorar as características dos funções de base Haar propostas por [11], Lienhart por exemplo propôs uma abordagem [14] implementando características mais discriminativas ou rotacionando-nas conforme a Figura 2.7.

Figura 2.7: Funções de base Haar propostos por Lienhart.

[14]

Embora a detecção utilizando funções de base do tipo Haar seja amplamente usada em detecção facial, nos últimos estágios da cascata um número elevado de características é necessário para atingir boas taxas de detecção, além disto, este método de detecção não é robusto à variações de iluminação [12]. Há estudos que mostram algoritmos de melhor desempenho, e principalmente, menor custo computacional. Entre eles o de [15], que propõe uma nova abordagem de detecção facial utilizando descritores de textura.

Operador LBP

O operador LBP (*Local Binary Pattern*) foi introduzido por [16] para classificação de texturas. A ideia do LBP é resumir a região local da imagem comparando cada pixel com seu vizinho. A Figura 2.8 ilustra esse processo. Dado o valor de um pixel central e compara-se o valor de seus vizinhos, caso o valor seja igual ou maior, atribui-se o valor 1, caso contrário valor 0. O resultado final com o valor binário para cada pixel, no caso da Figura 2.8, é o valor binário 00010011, ou em notação decimal: 19, e é chamado de código LBP. Portanto, com 8 pixels vizinhos, são possíveis 2^8 combinações, que são chamadas de *Local Binary Patterns* (Padrões binários locais), ou códigos LBP.

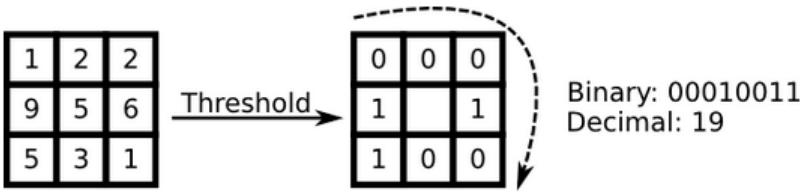

Figura 2.8: Operador LBP.

[17]

MB-LBP

[15] propõe uma nova característica distintiva de retângulos, chamadas *Multi-Block Local Binary Pattern* (MB-LBP).Diferentemente das funções de base do tipo Haar, que codifica os retângulos de uma região baseados na diferença entre as médias entre duas regiões, a idéia do MB-LBP é codificar regiões retangulares utilizando o mesmo príncipio de descritores binários locais (LBP), porém aplicando a comparação em regiões ao invés de *pixels*. O operador MB-LBP é definido a partir da comparação da intensidade de nível de cinza média na região central de um retângulo com as intensidades médias de suas regiões vizinhas, caso o valor de intensidade seja maior ou igual que a região central, atribui-se àquele vizinho o valor 1, caso contrário o valor 0. O resultado final é um número binário de 8 dígitos que descreve a região. A Figura 2.9 ilustra o processo.

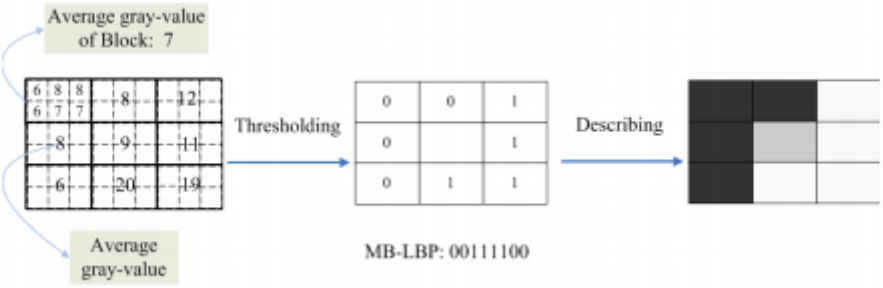

Figura 2.9: *Multi-Block Local Binary Pattern (MB-LBP).*

[15]

A última etapa do método consiste em treinar os classificadores com imagens positivas e negativas de imagens faciais. A técnica utilizada também é o *Adaboost*, que seleciona os melhores classificadores, ou seja, que melhor descrevem uma região facial como positiva e uma região não facial como negativa [15].

2.2.2 Pré-Processamento

Após a detecção da região facial de uma pessoa, a imagem ainda passa por um processo de normalização que visa minimizar os efeitos de eventos externos ao sistema e aumentar a eficiência do algoritmo [18]. Os métodos utilizados nesta etapa podem ser classificados em geométricos e fotométrica [18].

Na normalização geométrica, a posição de características individuais de cada pessoa como boca, nariz e olhos são alteradas a fim de padronizar todas as imagens com uma mesma escala e evitar erros relacionados à pose da face encontrada.

Na normalização fotométrica são aplicadas técnicas de visão computacional a fim de enaltecer características da imagem como brilho e contraste, de forma a minimizar a influência de fatores externos (como iluminação) no desempenho do reconhecimento facial.

2.2.3 Reconhecimento de identidade

O reconhecimento é a última etapa em um processo de reconhecimento facial. Essa etapa consiste em extrair as características da imagem facial encontrada e avaliar se a pessoa presente na imagem pertence à alguma identidade previamente cadastrada no banco de dados. O reconhecimento pode ser feito por meio de dois modos: autenticação ou identificação.

A autenticação é utilizada para validar a identidade de uma pessoa através da comparação

entre os dados biométricos capturados e os dados armazenados em um banco de dados [1].A vantagem da autenticação é a velocidade de processamento, desde que é necessário realizar uma comparação 1 x 1, ou seja, compara-se a imagem de entrada com outra imagem presente no banco de dados e então permitir, ou não, o acesso do usuário a uma área restrita. A autenticação é tipicamente utilizada para reconhecimentos positivos, onde o objetivo é prevenir múltiplas pessoas de utilizar a mesma identidade.

Na identificação o sistema reconhece um indivíduo avaliando modelos de todos os usuários no banco de dados a procura de uma correspondência de dados. Este sistema é bem mais lento que a autenticação, visto que desta vez é necessário realizar uma comparação 1 x N, ou seja, uma imagem de entrada pode chegar a ser comparada com todas as imagens do banco de dados.

Existem diversos algoritmos e métodos que buscam realizar esta etapa de forma eficiente, entre os mais conhecidos estão o reconhecimento facial através de análise de componentes principais [19] e por meio de análise discriminante [20] mas ultimamente descritores locais (LBP) também vem ganhando atenção nesta área principalmente pela sua robustez a alguns desafios de reconhecimento facial como variações de iluminação e variações de pose.

LBP no reconhecimento facial

Um dos métodos de reconhecimento facial que utilizam padrões binários locais (LBP) é o de Timo Ahonen et al, em [21] e em [22].

No trabalho de [21], o operador LBP é estendido de modo a usar vizinhanças circulares. A notação LBP (N,R) é utilizada para tal operadores, onde N representa o número de pixels vizinhos a serem comparados com o pixel central e R representa a distância entre o pixel central e pixel vizinho. A Figura 2.10 ilustra três destes operadores.

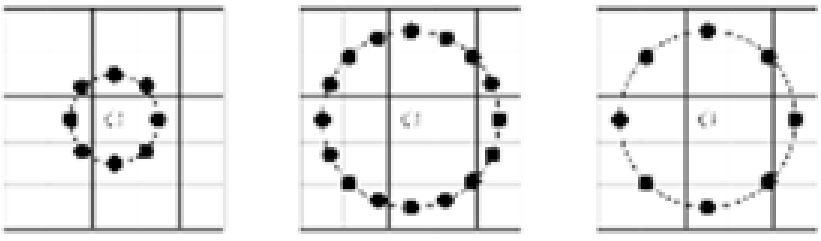

Figura 2.10: Vizinhanças circulares do operador LBP.

[22] Operadores dos tipos (8,1),(16,2) e (8,2).

Esse método consiste em dividir uma imagem de amostra em R regiões de mesmo tamanho. Para cada região então aplica-se um operador circular de forma a se obter um código LBP para cada pixel da região. Histogramas destes códigos LBP são calculados para cada região e então concatenados formando um descritor geral da imagem facial. A Figura 2.11 ilustra este processo.

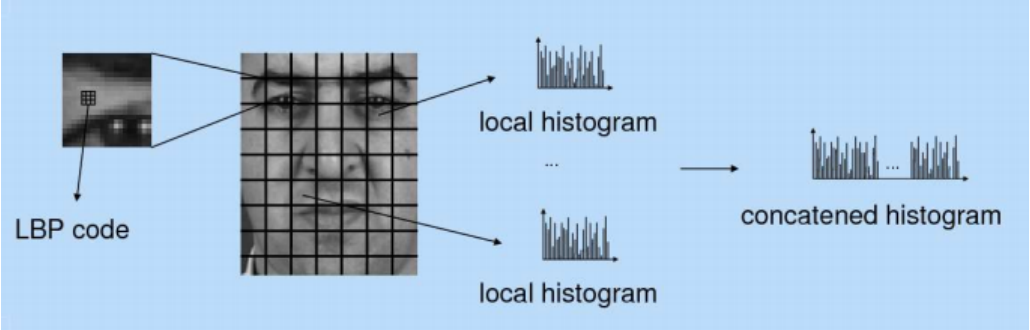

Figura 2.11: Representação facial por meio de códigos LBP.

[12]

Outra extensão do operador LBP utilizada por [21], é a utilização apenas de operadores LBP uniformes [23]. Um operador LBP é chamado uniforme nessa abordagem se o padrão binário resultante contém no máximo duas transições de 0 para 1 ou vice-versa. Nessa abordagem se atribui então um código LBP para cada operador uniforme e um apenas um código para todos padrões não uniformes.

A divisão da imagem em R regiões apresenta vantagens. Alguns locais específicos da face são mais importantes para o reconhecimento facial, entre eles a região dos olhos [10]. Portanto aplica-se pesos para cada região facial conforme sua importância. A Figura 2.12 exemplifica a divisão da face em regiões com peso, nela quanto mais próximo da cor branca, maior a importância para o reconhceimento.

Figura 2.12: Pesos em um imagem facial.
[22] (a) Imagem facial dividida em 7x7 janelas. (b)Pesos utilizados para cada região. regiões pretas representam peso 0, cinza escuro 1, cinza claro 2 e branco 4.

Na última etapa do método proposto por [21], os histogramas obtidos são utilizados para classificação de identidade. Esta classificação é realizada calculando a distância entre o histograma de uma amostra de imagem sob avaliação com os histogramas previamente cadastrados utilizando o algoritmo. A distância utilizada por [21] foi o Qui-Quadrado por apresentar melhor performance em relação à outros tipos de distância (Equação 2.1).

$$\chi^2_w(x,\varepsilon) = \sum_{i,j}(w_j \frac{(x_{(j,i)} - \varepsilon_{(i,j)})^2}{x_{(i,j)} + \varepsilon_{(i,j)}}) \tag{2.1}$$

2.3 IHC (Interação Humano-Computador)

Interação Humano-Computador é uma área de estudo que tem como objetivo projetar, desenvolver e testar sistemas que ajudem humanos a desempenhar suas atividades de forma produtiva [1]. Ela trata da camada responsável pela comunicação entre sistema e usuário. Seus estudos englobam desde estruturas de hardware, levando em consideração aspectos de ergonomia, até psicologia cognitiva. IHC é, assim, uma área interdisciplinar que busca produzir sistemas que possam ser utilizados da melhor forma possível. O conceito de usabilidade [24] está atrelado aos esforços de se melhorar a qualidade e facilidade de uso de um sistema. Ele pode ser dividido em cinco componentes de qualidade [24]:

- Intuitividade: Facilidade em utilizar o sistema desde o primeiro contato;
- Eficiência: Agilidade em que o usuário consegue finalizar tarefas no sistema;

- Memorização: Facilidade em lembrar de como o sistema é utilizado mesmo por usuários ocasionais;
- Erro: Habilidade do usuário de se recuperar de uma utilização errada do sistema;
- Satisfação: O sistema deve ser agradável para o usuário.

Uma das formas de se avaliar que o sistema segue de fato esses componentes e possui usabilidade é por meio do *System Usability Scale* (SUS). System Usability Scale SUS é uma métrica proposta por John Brooke [25] e cobre vários aspectos de um sistema como a necessidade de suporte, treinamento, complexidade, dificuldade de uso. Essa métrica baseia-se em um questionário a ser respondido por um usuário do sistema. Esse questionário consiste em dez itens que devem ser respondidos, cada qual com um valor que varia de um à cinco, onde o valor 1 siginifica "Discordo totalmente", 2 "Discordo", 3 "Indiferente", 4 "Concordo"e 5 "Concordo totalmente". Os itens são [25]:

1. Eu acho que gostaria de usar esse sistema com frequência.
2. Eu acho o sistema desnecessariamente complexo.
3. Eu achei o sistema fácil de usar.
4. Eu acho que precisaria de ajuda de uma pessoa com conhecimentos técnicos para usar o sistema.
5. Eu acho que as várias funções do sistema estão muito bem integradas.
6. Eu acho que o sistema apresenta muita inconsistência.
7. Eu imagino que as pessoas aprenderão como usar esse sistema rapidamente.
8. Eu achei o sistema atrapalhado de usar.
9. Eu me senti confiante ao usar o sistema.
10. Eu precisei aprender várias coisas novas antes de conseguir usar o sistema.

Para calcular os resultados do teste, primeiramente devem ser somados os resultados de cada item. Cada item terá um resultado final de 0 a 4. Para os itens 1,3,5,7 e 9 o resultado é a resposta do usuário menos 1, para os demais itens o resultado é 5 menos a resposta do usuário [25]. Finalmente, a soma dos resultados deve ser multiplicada por 2,5 para se obter o

valor médio de usabilidade. Esse valor pode variar entre 0 e 100 e, de acordo com [26], um valor de usabilidade maior do que 68 pode ser considerado acima da média, e valores abaixo de 68 estão abaixo da média.

2.4 *Material Design*

Material Design [27] é uma ferramenta criada pela Google cujo objetivo é sintetizar princípios clássicos de uma boa interface, possui guias e referências para estilo, animação, componentes e padrões de projeto. apresenta todas as funcionalidades e documentação de seus padrões. Os principais padrões utilizados para o desenvolvimento deste projeto foram os botões flutuantes, ícones e listas.

Botões flutuantes são utilizados para ressaltar o botão em questão do restante da interface, é recomendado utilizá-lo preferencialmente em ações positivas como criar, navegar, e compartilhar e desejável evitar o seu uso em ações destrutivas como deletar, arquivar ou ver alertas de erros. Apenas um botão flutuante é recomendado por tela e preferencialmente deve representar a principal ação da aplicação [27].A Figura 2.13 apresenta um exemplo de um botão flutuante.

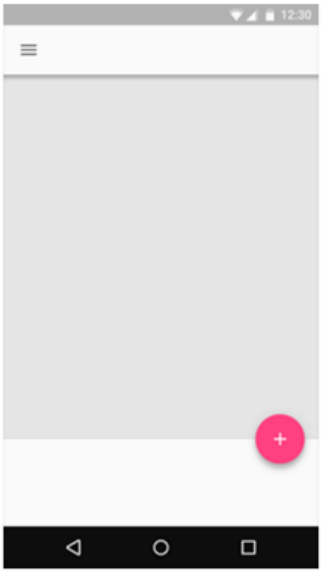

Figura 2.13: Exemplo de um botão flutuante

[28]

A representação de listas de acordo com o guia do *Material Design* deve ser utilizada sob certas condições e respeitar certos padrões. Não é recomendável utilizar listas quando

o tamanho da informação apresentada de cada elemento não for maior que três linhas ou quando o elemento principal da lista seja uma imagem. Listas devem ser subdivididas em linhas e possuir um padrão em sua posição vertical. Na Figura 2.14 observa-se o exemplo de uma lista onde as imagens das pessoas representadas possuem a mesma margem e seguem a orientação de uma linha. Listas devem ainda possui rolagem apenas na direção vertical e a informação deve ser apresentada em uma específica ordem, como por exemplo data, tamanho do arquivo e ordem alfabética.

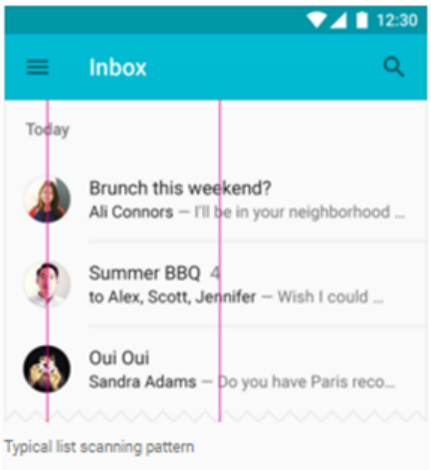

Figura 2.14: Exemplo de uma lista

[29]

Existem ícones padronizados que simbolizam comandos, arquivos ou diretórios que podem ser utilizados no desenvolvimento da aplicação. Todos os ícones foram desenhados a fim de tornar a interface simples, moderna e amigável.

Figura 2.15: Ícones disponibilizados por *Material Design*

[30]

Capítulo 3

Materiais e Métodos

O objetivo deste trabalho foi implementar um sistema de controle de presença de alunos realizado através de reconhecimento facial. O sistema completo é composto pelos subsistemas: reconhecimento facial, banco de dados, interface, e geração de relatórios. Neste capítulo são descritos os materiais e métodos utilizadas para o desenvolvimento de cada subsistema.

3.1 Materiais

A seguir são apresentados os materiais utilizados para o desenvolvimento do projeto. Para a execução desse projeto foi desenvolvido um aplicativo para dispositivos Android, utilizando a linguagem Java, bibliotecas de visão computacional do OpenCV e sistema de banco de dados SQLite. Apesar da ferramenta Appinventor [31] apresentar um método ágil de desenvolvimento, o tamanho de seu código final gerado é maior, não há configurações para acesso ao mercado Android (usuários teriam que habilitar seus *smartphones* para instalação de aplicativos não oficiais) [32], e falta de componentes necessários para algoritmos de visão computacional [33]. Para o desenvolvimento foi utilizada a plataforma Android Studio e os testes do aplicativo foram feitos em um *smartphone* modelo Nexus 5. Essa seção apresenta uma descrição desses materiais.

3.1.1 Ambiente de desenvolvimento

A plataforma de desenvolvimento escolhida para o projeto foi o Android Studio. Android Studio é a IDE (*Integrated Development Environment*) oficial para desenvolvimento de aplicativos Android. Ela conta com compilação rápida (otimizada a partir da versão 2.0), mecanismo de arrrastar componentes para a criação de interface e possibilidade de verificação do

consumo de memória no *smartphone*.

3.1.2 Java

Java é uma linguagem de programação orientada a objetos, similar ao C++, porém é uma linguagem mais robusta e simples de usar [34]. Esta linguagem é compilada em um *byte-code* por uma máquina virtual própria, a JVM (*Java Virtual Machine*). A JVM pode ser instalada em diversos sistemas operacionais, garantindo portabilidade à linguagem. Aplicativos para dispositivos Android [35]são comumente desenvolvidos na linguagem Java. Após compilados pela JVM, estes aplicativos passam por algumas traduções e compactações até se transformarem no arquivo de aplicativo com extensão ".apk", que podem ser instalados nos dispositivos.

3.1.3 OpenCV

OpenCV é uma biblioteca de algoritmos utilizados em visão computacional de uso livre [36]. É escrita em linguagem C e C++ mas possui interfaces com outras linguagens como Python, Matlab, Java entre outros. Possui suporte operacional para Windows, Linux e Mac OS. o objetivo da biblioteca é fornecer uma infraestrutura de uso simples de forma que desenvolvedores possam criar aplicações de forma rápida e eficaz. Dentre as aplicações mais utilizadas na área de visão computacional estão a inspeção de produtos de fábrica; a área médica; interface homem-máquina; calibração de câmera; robótica e segurança.

No contexto da aplicação Android desenvolvida, é necessário que o usuário ainda tenha o OpenCV Manager instalado no *smartphone*. OpenCV Manager [37] fornece a melhor versão do OpenCV para o *hardware* utilizado e recebe as atualizações de estabilidade e performance da biblioteca. Há a possibilidade de realizar um projeto sem a utilização do OpenCV Manager, entretanto o tamanho do aplicativo aumentará consideravelmente [38]. Um tutorial para desenvolvimento Android com OpenCV é apresentado em [39].

3.1.4 SQLite

SQLite [40] é um sistema de banco de dados relacional que utiliza a linguagem SQL. O SQLite tem como diferencial o fato de não precisar estar atrelado a um servidor, diferente de outros sistemas de banco de dados que ficam implementados em um servidor separado do cliente. Com o SQLite é possível que o banco de dados seja parte integrante da aplicação,

sendo apenas um arquivo que pode estar armazenado em qualquer dispositivo. Dispositivos Android já possuem a biblioteca do SQLite embutida, para utilizá-la não é necessário nenhum tipo de instalação ou administração [41].

3.1.5 *Smartphone*

O *smartphone* utilizado para o desenvolvimento do aplicativo foi o modelo Nexus 5 (Figura 3.1). Conforme observado na Tabela 3.1, este dispositivo apresenta um processamento potente com CPU Qualcomm Snapdragon 800, 2,26 GHz e 2GB RAM, que devem suportar um cenário de reconhecimento facial onde há um processamento elevado. Outros fatores importantes são a alta capacidade de armazenamento de dados, com 32GB de memória e uma câmera de alta resolução, com 8MP, suficiente para que as imagens adquiridas possuam qualidade para aplicação.

CPU	Qualcomm Snapdragon 800, 2,26GHz
GPU	Adreno 330, 450 MHz
Memória RAM	2 GB
Câmera	8 MP
Câmera frontal	1.3 MP
Bateria	2300 mAh
Sistema operacional	Android 4.4 (*Kit Kat*)

Tabela 3.1: Especificações do Nexus 5

[42]

Figura 3.1: Nexus 5.

[43]

3.2 Métodos

Neste tópico são detalhado os métodos utilizados para o desenvolvimento da aplicação. Dentre os principais aspectos do projeto descrito neste tópico estão a estrutura do sistema, modelagem de dados, interface e o módulo de reconhecimento facial.

3.2.1 Modelagem de dados

O objetivo do projeto foi implementar um sistema de reconhecimento facial aplicado a um cenário acadêmico, onde o professor realiza o controle de alunos de forma rápida, robusta e eficaz. A fim de implementar um banco de dados robusto a falhas e para um desenvolvimento mais eficaz, foi necessário, antes do desenvolvimento da aplicação, organizar e estruturar os dados a serem utilizados. O escopo do projeto foi dividido em cinco diferentes entidades, que se relacionam entre si. Tais entidades e seus atributos são descritos a seguir nas Tabelas 3.2, 3.3, 3.4, 3.5 e 3.6.

Id	Identificação utilizada como chave primária para a tabela "Professor"
E-mail (*login*)	E-mail pelo qual o professor acessa o sistema
Senha	Senha do professor para acesso ao sistema
Nome	Nome do Professor

Tabela 3.2: Tabela Professor.

Id	Identificação utilizada como chave primária para a tabela "Disciplina"
código	Código da disciplina
Nome	Nome da disciplina

Tabela 3.3: Tabela Discplina.

Id	Identificação utilizada como chave primária da tabela "Aula"
Data	Data de realização da aula
Id disciplina	Identificação de qual disciplina aquela aula pertence

Tabela 3.4: Tabela Aula.

Nome	Nome do aluno
Numero USP	Identificação do aluno perante instituição de ensino
Id	Identificação do aluno utilizada como chave primária da tabela "Aluno"

Tabela 3.5: Tabela Aluno.

Numero USP	Identificação utilizada pelo sistema para acesso à foto
Id aluno	Identificação de qual aluno a foto pertence
bitmap	Imagem no formato *bitmap* contendo a face do aluno
Id aula	Identificação de qual aula a foto pertence

Tabela 3.6: Tabela Foto.

A forma como as entidades se relacionam é representada por meio de um diagrama entidade-relacionamento, ilustrado na Figura 3.2.Nota-se que um professor pode possuir nenhuma, ou inúmeras disciplinas, entretanto uma disciplina pode ser lecionada por apenas um professor. Uma disciplina pode possuir nenhum ou vários alunos, e um aluno deve estar matriculado em ao menos uma disciplina. Uma disciplina possui várias aulas ao longo do semestre e cada aula pertence à somente uma disciplina. Uma aula aleatória durante o período

deve possuir de zero à muitos alunos, da mesma forma que um aluno pode estar presente de nenhuma à muitas aulas, e por fim, um aluno pode possuir várias fotos, entretanto uma foto é atribuída a apenas um aluno.

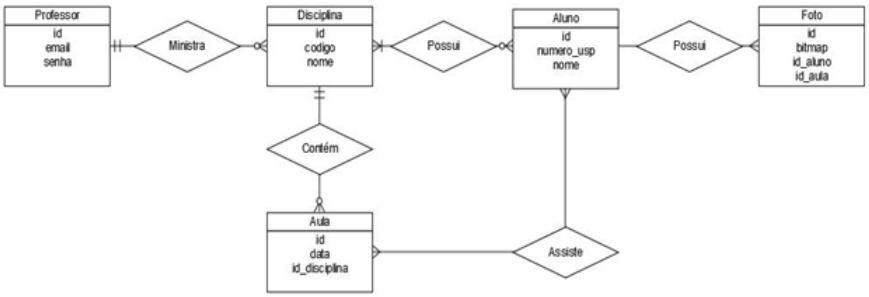

Figura 3.2: Diagrama entidade-relacioamento.

Após a definição das entidades, foi desenvolvido e estruturado o banco de dados utilizando SQLite (Anexo II). As entidades descritas anteriormente se tornaram tabelas no banco de dados, e o relacionamento entre elas definido conforme a Figura 3.2.

3.2.2 Estrutura do sistema

O sistema proposto consiste de um controle de presença de alunos, realizado por reconhecimento facial. O usuário direto do aplicativo é o professor. A premissa utilizada é de que o professor fará o cadastro de seus alunos e de suas disciplinas. Em cada disciplina adicionada é possível realizar o cadastro de alunos e realização de chamadas ao longo do semestre.

O processo de chamada foi realizado por meio de reconhecimento facial de forma que, utilizando uma câmera, captura-se a imagem da face do aluno, e por meio desta imagem, é realizado o processamento biométrico para identificar a identidade do aluno. Na posse da identidade do aluno, é possível atribuir sua presença na aula a qual a chamada se refere. Após a identificação de todos os alunos presentes em uma determinada aula, o professor deve confirmar a presença de alunos visando evitar erros de atribuição de identidade.

Finalmente, após a confirmação, os dados são armazenados no banco de dados para posterior consulta do professor e o sistema automaticamente abre o aplicativo de email definido como padrão do usuário e escreve um e-mail contendo todas as informações de alunos daquela aula.

O diagrama de fluxo de dados do sistema é descrito na Figura 3.3. No processo de cadastro de disciplinas o professor adiciona uma nova disciplina que irá ministrar no banco de dados,

assim como os alunos nela matriculados.

No processo de cadastro de aluno, informações de um aluno são gravadas no banco de dados. Para este processo, é necessário o nome, código do aluno, e imagens faciais do mesmo. O processo de reconhecimento pode ser entendido como uma nova aula, na qual o controle de presença é realizado por meio da câmera do *smartphone* de forma que, para cada nova imagem proveniente da câmera, é realizada a detecção facial e o reconhecimento da identidade do aluno. Em caso de reconhecimento de identidade, é atribuída presença ao aluno em questão. O professor possui a opção de confirmar a veracidade de todos os dados processados para que então as informações de aula e fotos dos alunos presentes sejam persistidas.

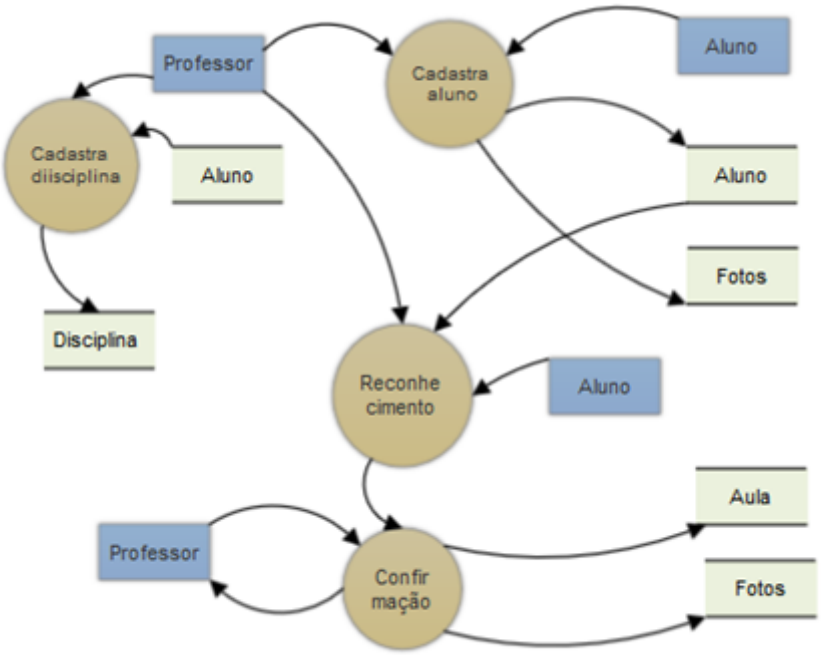

Figura 3.3: Diagrama de fluxo de dados.

3.2.3 Interface

Este módulo do projeto trata das ferramentas e métodos que foram utilizados para tornar a interface da aplicação agradável.O desenvolvimento da aplicação foi realizado utilizando o as ferramentas do Material Design. No total foram criadas dez telas para realizar todo o processo de controle de presença de alunos.

Tela de Login

Esta é a tela inicial da aplicação (Figura 3.4). É utilizada para acesso do usuário ao sistema. A tela é composta por dois campos de texto, onde é esperada a entrada de texto do usuário. No campo de texto situado ao topo da aplicação é esperado o endereço de e-mail do usuário e no campo de texto abaixo é esperada a senha, que deve possuir no mínimo seis caracteres. A tela ainda é composta por um botão retangular que realiza a função de cadastro caso o e-mail digitado pelo usuário seja novo no sistema ou a função de acesso ao sistema caso os campos preenchidos estejam de acordo com cadastro previamente realizado.

Figura 3.4: Tela de *Login*.

Tela com lista de disciplinas

Esta tela (Figura 3.5) é iniciada logo após o sucesso da fase de login, é responsável por listar todas as disciplinas cadastradas pelo professor. Possui um cabeçalho com a descrição "disciplinas", uma lista e um botão flutuante com um ícone de adição. A lista é composta pelas disciplinas cadastradas no banco de dados, cada item desta lista possui apenas duas linhas de texto com tamanhos diferentes, a linha superior com tamanho maior descreve o nome da disciplina e a linha inferior com tamanho menor descreve a sigla da disciplina. Ao clicar em um item da lista é iniciada uma nova tela com maiores informações do curso. O botão flutuante é utilizado para enfatizar a ação de adicionar uma nova disciplina e o seu

acionamento inicia uma nova tela para realizar o cadastro de disciplinas.

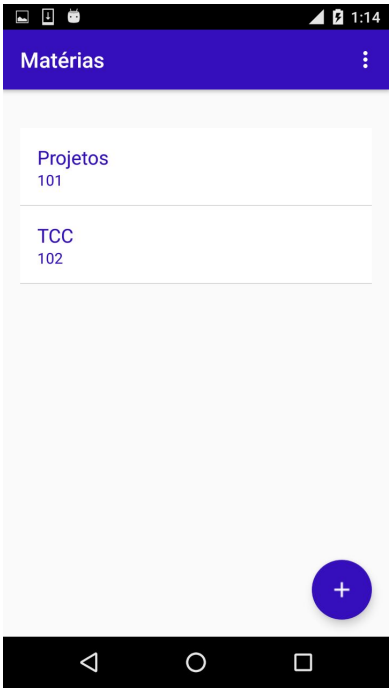

Figura 3.5: Tela com lista de disciplinas.

Tela de cadastro de disciplinas

A tela de cadastro de disciplinas (Figura 3.6) apresenta um cabeçalho com nome "Nova disciplina", um ícone com uma imagem de uma seta para voltar à tela anterior e um ícone de finalização da atividade que se acionado, salva as informações de entrada do usuário no banco de dados. A tela possui dois campos de texto, uma lista e um botão flutuante. No campo de texto superior, é esperado que o usuário digite o nome da disciplina e no campo inferior é esperado a sigla da disciplina. A tela ainda possui uma lista que é composta de alunos cadastrados no sistema, cada item desta lista apresenta uma imagem, dois campos de texto e uma opção para seleção do item. A imagem fica situada à esquerda do item e representa a face do aluno cadastrada no banco de dados. O campo de texto superior informa o nome do aluno e o campo inferior informa o número de registro do aluno. Por fim, a opção de seleção é utilizada caso o usuário queira incluir o aluno à disciplina. Por fim há ainda um botão flutuante com ícone de adição de pessoas para enfatizar a ação de adicionar o cadastro de um novo aluno. O acionamento deste botão leva à tela de cadastro de alunos.

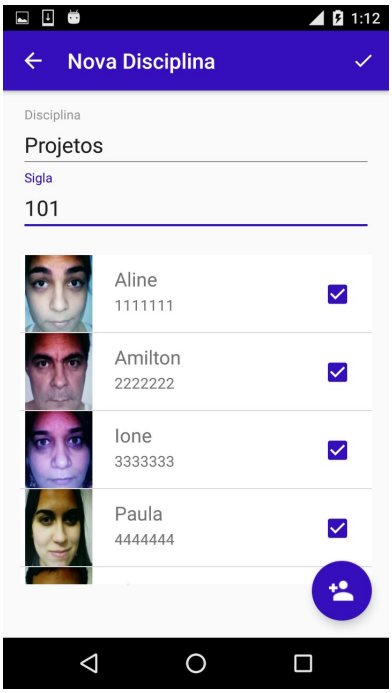

Figura 3.6: Tela de cadastro de disciplinas.

Tela de cadastro de alunos

A tela de cadastro de alunos (Figura 3.7) apresenta um cabeçalho com nome "Novo aluno", um ícone de uma seta para voltar à tela anterior e um ícone de confirmação que, se acionado, salva as informações do novo aluno no banco de dados. A tela possui dois campos de texto e nove imagens. No campo de texto superior é esperado que o usuário digite o nome do aluno e no campo de texto inferior, o número de registro do aluno na instituição. Todas as nove imagens apresentam inicialmente um ícone de adição de imagem e possuem também a função de um botão. Ao clique do usuário em qualquer uma das imagens é iniciada a tela de detecção facial, no fim da detecção todas as imagens são alteradas para mostrar as imagens de face capturadas. Ao salvar as informações do aluno, é realizado também o treinamento das imagens faciais do aluno para que posteriormente seja possível o reconhecimento automático do mesmo.

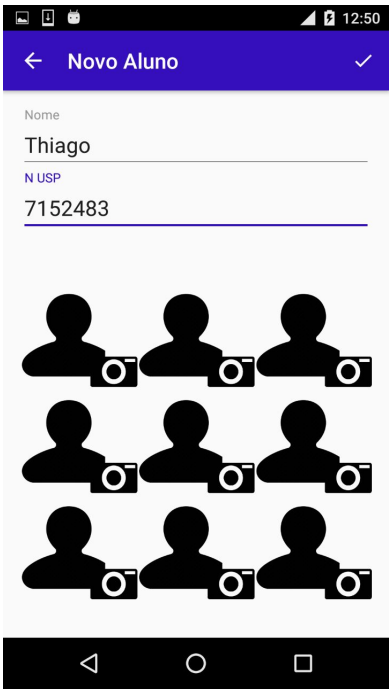

Figura 3.7: Tela de cadastro de alunos.

Tela de detecção facial

Esta tela (Figura 3.8) apresenta as imagens da câmera do *smartphone* um botão com ícone de câmera cuja finalidade é alterar a câmera utilizada (frontal ou principal) e um botão circular azul que possui a função de iniciar a captura de imagens. A função desta tela é detectar a região facial dado um quadro de imagem adquirido por meio da câmera do aplicativo. Ao fim do processo, são detectadas nove imagens da face da pessoa e estas são armazenadas em uma memória temporária da aplicação e retorna-se então à tela anterior da aplicação.

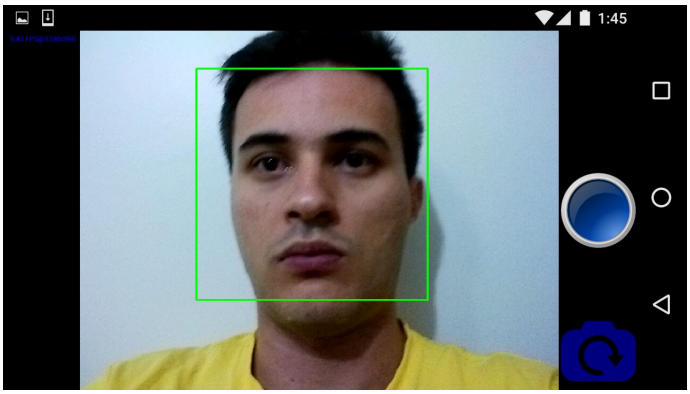

Figura 3.8: Tela de detecção facial.

Tela de informações da disciplina

Esta tela apresenta um cabeçalho com o nome da disciplina e um ícone na forma de seta cuja função é voltar à tela anterior. A tela ainda possui duas abas, cada qual com uma função.

A aba de alunos (Figura 3.9) possui um ícone no formato de pessoa e seu conteúdo é composto por um botão flutuante e uma lista composta por todos os alunos cadastrados na disciplina. Cada item da lista representa um aluno e seu conteúdo é formado por uma imagem (representa a face do aluno cadastrado) e dois campos de texto, onde o campo superior representa o nome do aluno e o campo inferior representa o número de cadastro do aluno. O botão flutuante possui um ícone que representa a ação de adicionar pessoas e seu acionamento leva à tela de cadastro de alunos.

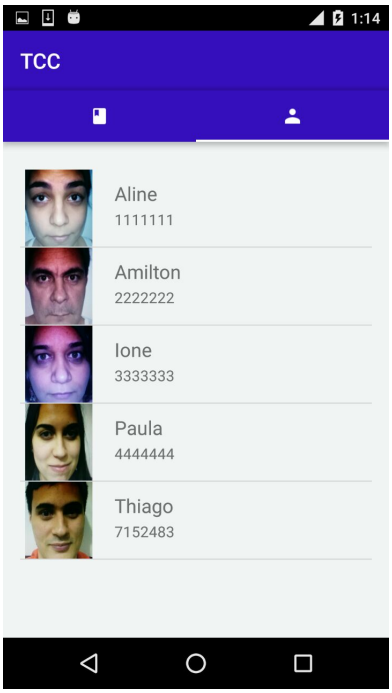

Figura 3.9: Aba de alunos na tela de informações da disciplina.

A aba de aulas (Figura 3.10) é composta por uma lista de todas as aulas existentes no semestre e um botão flutuante. Cada item da lista é formado por dois campos de texto. O campo de texto superior apresenta a informação de número da aula e o campo inferior apresenta a informação de data de realização da chamada. O botão flutuante apresenta um ícone de adição e seu acionamento leva à tela de chamada.

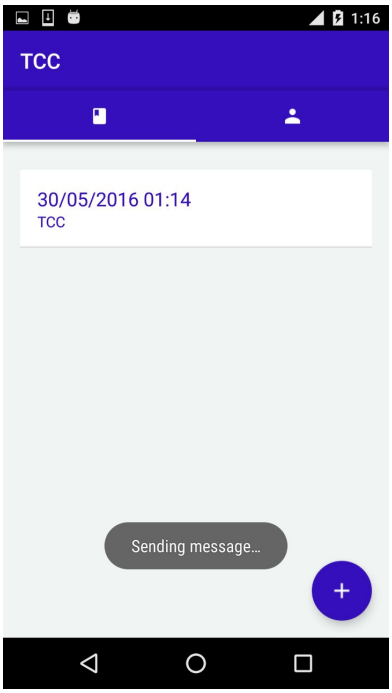

Figura 3.10: Aba de aulas na tela de informações da disciplina.

Tela de chamada

Esta é a tela (Figura 3.11) responsável pelo reconhecimento da identidade de alunos e possui um direcionamento horizontal. A tela (Anexo III)apresenta os quadros de imagens provenientes da câmera, um botão representado por ícone de câmera no canto inferior direito e um botão com a imagem de um ícone representando a ação "salvar"no canto superior direito da tela. Caso o processamento da imagem resulte em um reconhecimento de identidade, uma aba inferior é habilitada onde é apresentada informações da pessoa detectada e botões para que o usuário possa tomar ações. Esta aba inferior possui uma imagem no canto esquerdo representando a face da pessoa detectada, dois campos de texto, onde o superior apresenta o nome da identidade detectada e o campo inferior que apresenta o número de cadastro do aluno e três botões. O botão com ícone de rejeição inicia o processo de reconhecimento novamente, o botão de confirmação salva as informações do usuário e a nova foto no banco de dados e o botão de adição leva a uma nova tela para que o aluno possa ser adicionado manualmente. Finalmente, ao acionar o botão representado pelo ícone de salvar informações, os alunos presentes até o momento são adicionados à aula atual e é iniciada então uma nova tela de confirmação.

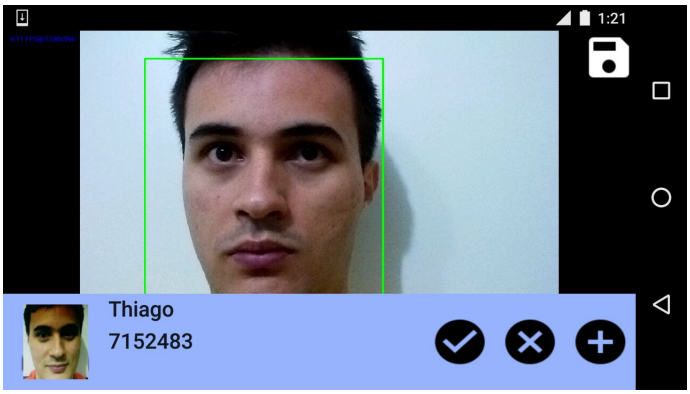

Figura 3.11: Tela de chamada.

Tela de adição manual de presença de aluno

Esta tela (Figura 3.12) é utilizada para adicionar o aluno manualmente à chamada que está sendo realizada pelo professor.Ela deve ser acionada em casos onde o reconhecimento da identidade do aluno falhe. Possui uma barra de ferramentas com a opção de voltar à tela anterior e uma lista composta por todos os alunos matriculados na disciplina. Cada item desta lista apresenta uma imagem, dois campos de texto e uma opção para seleção do item. A imagem está localizada a esquerda do item e apresenta a imagem da face do aluno, os campos de texto superior e inferior apresentam as informações de nome e código, respectivamente. Cada item da lista ainda possui a opção de clique do usuário, caso acionada, o acionamento leva à uma tela de detecção facial cuja resposta será a imagem da face do aluno. Com as informações de identidade do aluno clicado e uma nova foto é possível retornar à tela de chamada e dar prosseguimento à chamada.

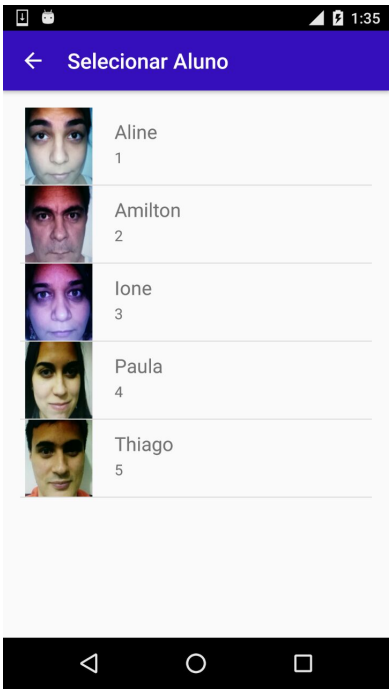

Figura 3.12: Tela de adição manual de aluno.

Tela de confirmação

Esta é a última tela (Figura 3.13) do processo de realização de chamada, apresenta uma barra de ferramentas com a opção de voltar à tela anterior e botão com ícone de finalização da atividade. A tela ainda possui uma lista representada por todos os alunos da disciplina. O principal destaque desta lista é a cor de seu plano de fundo, que é dependente da presença do aluno, onde alunos presentes são representados pela cor verde e alunos presentes ausentes à cor vermelha. Cada item da lista representa um aluno, e seu desenho é representado pela imagem do aluno (situado à esquerda da lista) e informações com o nome do aluno (campo de texto superior) e código do aluno (campo inferior).

Ao acionar o ícone de confirmação, as informações de aula são gravadas no banco de dados e é iniciado o aplicativo padrão do usuário para envio de emails, onde é criado automaticamente um e-mail com as informações dos alunos presentes ou ausentes da chamada realizada. O usuário então pode enviar o e-mail com todas as informações da chamada realizada.

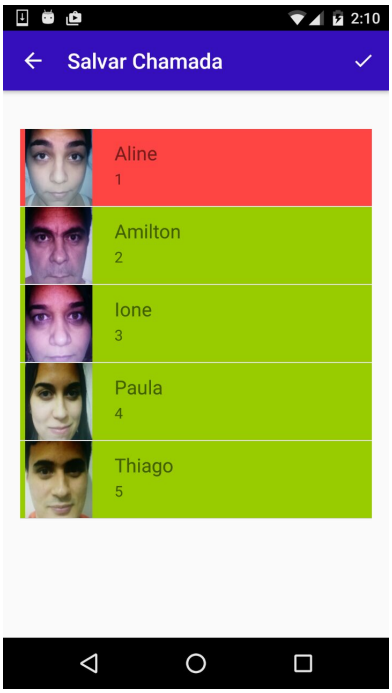

Figura 3.13: Tela de adição manual de aluno.

3.2.4 Módulo de reconhecimento

Este módulo do projeto visa descever os métodos utilizados para detecção facial e processamentos utilizados para aumentar o desempenho da detecção e do reconhecimento facial. Entre os principais fatores que afetam o reconhecimento facial estão a iluminação do ambiente, variações de pose, qualidade da imagem, e variações de humor [10]. Dentre estes, foram utilizados métodos para contornar problemas de iluminação, velocidade e eficiência de acerto através de um processamento da imagem realizado antes e após a detecção facial.

Desenvolver métodos para contornar os problemas de variações de pose e de humor não são o foco do projeto proposto. Segundo [10] a maioria dos métodos para contornar estes problemas exigem um sistema de reconhecimento de características locais eficiente, fato que contraria o método de reconhecimento facial proposto neste projeto.

Foram utilizados métodos para contornar problemas de iluminação, velocidade e eficiência de acerto realizando um pré-processamento da imagem antes e após a detecção facial. O pré-processamento visa diminuir os efeitos da iluminação e consiste em dois passos: conversão para escala de cinza e equalização de histograma. O processamento realizado após a detecção de faces é utilizado a fim de diminuir a taxa de erros de detecção.

Pré-processamento

A partir da versão 2.4, a biblioteca do openCV já possui descritores LBP que estão prontos para uso e podem ser utilizados para detecção de objetos [44], inclusive detecção frontal de faces, porém para a sua utilização de forma eficiente foi necessário realizar um pré-processamento da imagem, que consiste em dois passos: conversão para escala de cinza e equalização de histograma.

A utilização dos algoritmos de detecção facial funcionam apenas em imagens em escala de cinza, portanto foi necessário converter a imagem capturada pela câmera para tal escala. As bibliotecas do OpenCV oferecem funcionalidades que permitem fazer a conversão para escala de cinza.

Algoritmos de detecção facial e de reconhecimento facial não são confiáveis em variações elevadas de iluminação. Para diminuir os efeitos da iluminação foi realizado um processo de equalização de histograma, a fim de aumentar o brilho e contraste da imagem de entrada, consequentemente aumentando o poder de detecção e reconhecimento de faces.

Detecção facial

Este módulo do sistema recebe como entrada uma imagem capturada através da câmera e realiza o processamento a fim de retornar somente a imagem da face presente na imagem (caso exista). É possível a utilização do módulo para captura de mais de uma face para a mesma imagem, entretanto isso torna o reconhecimento mais susceptível a erros pois, neste caso, a câmera deverá estar a uma maior distância das faces para adequar todas as pessoas presentes, fato que prejudica a eficiência dos algoritmos de detecção e reconhecimento. Outro fator determinante para evitar múltiplas faces na mesma imagem é o consumo de memória e a velocidade de operação do sistema, neste caso deverá ser feito um processamento para cada região facial encontrada, o que aumenta o processamento total do sistema e diminui a velocidade de operação. Portanto, codificou-se o algoritmo de detecção facial para que este detecte apenas a maior face presente na imagem. Para definir que algoritmo de detecção utilizar levou-se em consideração que LBP é mais preciso [15] e apresenta melhor desempenho [13], além de ser mais robusto à iluminação e à oclusão e mais veloz quando comparado ao método utilizando Haar [15]. O algoritmo LBP também não possui problemas com relação a patentes, ao contrário de funções de base Haar, que possui patente registrada [45].

Tendo em vista os fatores supracitados, o algoritmo escolhido para esta etapa da aplicação

foi o LBP, devido principalmente à sua maior velocidade de processamento e por não possuir patentes registradas.

Confirmação da detecção

Após utilizar-se do operador LBP para detecção facial, utiliza-se também um detector de olhos através de funções de base Haar (a biblioteca OpenCV possui apenas este tipo de funções de base para detecção de olhos).

Este processo apresenta como entrada uma região facial detectada por meio do LBP. Aplica-se então um detector de olhos e a classificação da imagem é baseada na seguinte premissa: caso o resultado da detecção seja positivo, conclui-se que a região em questão realmente é uma face e então o sistema prossegue com o processamento. Caso o resultado seja negativo, conclui-se que a região detectada é um falso positivo e então encerra-se o processamento do quadro de imagem em questão. Este pré-processamento visou principalmente evitar a detecção falsa de imagens faciais e tornar o algoritmo menos susceptivel a erros.

Redimensionamento da imagem

O tamanho da imagem de entrada do sistema não foi um problema, a câmera do *smartphone* utilizado possui boa qualidade e o tamanho da imagem de entrada é de 960x1280. Entretanto, um dos pontos cruciais do projeto é a velocidade do sistema, a velocidade de reconhecimento de faces depende do tamanho da imagem (quanto maior a imagem menor a velocidade de processamento). Portanto é de grande importância adequar o tamanho da imagem facial processada de forma que não haja perda de performance na velocidade do sistema ou na eficiência do algoritmo.

Para tal objetivo, estipulou-se um único tamanho para todas as imagens faciais com a finalidade de padronizar as imagens do banco de dados e reduzir o custo computacional do sistema com respeito ao reconhecimento facial. Em métodos de reconhecimento facial a imagem não pode ser muito pequena, principalmente em métodos que dependem de localização precisa de características. Utilizou-se o tamanho de imagens faciais de 130x150, pois este tamanho foi utilizado em [21], apresentando performance de reconhecimento superior à 70%.

Reconhecimento de identidade

Esta é a última etapa do módulo de reconhecimento. Após a detecção facial e seus processamentos adicionais, obtém-se uma imagem facial de tamanho 130x150, equalizada e em escala de cinza. Para realizar o reconhecimento da identidade dos usuários, utilizou-se a biblioteca do OpenCV (Anexo I). O reconhecimento facial é realizado utilizando LBP (*Local Binary Patterns*). Este método se destaca em relação à outros devido à sua robustez à variações de iluminação e pose. Dada a premissa de que o aplicativo desenvolvido será utilizado em ambientes abertos, onde não haverá o controle de fatores externos, o algoritmo utilizado em [21] se torna interessante. Outro fator importante é o desempenho do algoritmo. Segundo [22] o método utilizando LBP apresenta melhores taxas de reconhecimento que o algoritmo utilizando análise de compontentes principais.

A escolha dos parâmetros do algoritmo utilizado foi baseada nos trabalhos de [22], utilizou-se operador LBP com raio de 2 pixels, 8 vizinhos e particionamento das regiões da imagem facial em 7x7 janelas resultando em 49 regiões. Estes parâmetros representam um bom casamento entre performance de reconhecimento e velocidade de processamento segundo [22].

O módulo de reconhecimento facial foi desenvolvido utilizando o *FaceRecognizer* [46], uma classe disponibilizada no OpenCV a partir da versão 2.4 que provê funções de acesso a algoritmos de reconhecimento facial. Utilizou-se desta classe em três etapas do processo: no cadastro de novos alunos, na adição manual de presença ao aluno e no reconhecimento de identidade. Segundo [10] melhores resultados de reconhecimento são atingidos com um número amplo de amostras. Entretanto a memória disponível em um dispositivo móvel é limitada e é necessário controlar o tamanho dos arquivos armazenados no banco de dados. Portanto, para o cadastro de novos alunos foi estabelecida uma quantidade de nove fotos. Nesta etapa utilizou-se a classe *FaceRecognizer* para treinamento dos dados adquiridos associados ao código de identificação do aluno.

Na etapa de adição manual de presença de aluno, a classe *FaceRecognizer* é atualizada com as informações do aluno e a nova foto facial adquirida é cadastrada e adicionada ao modelo do algoritmo de reconhecimento facial. Desta forma é possível corrigir as falhas do algoritmo e torná-lo mais robusto a variações de características pessoais (por exemplo, o crescimento de pelos faciais).

Na etapa de reconhecimento de identidade, é utilizada a função de predição da classe *FaceRecognizer*, nela é necessário fornecer a imagem a ser avaliada e o algoritmo retorna então a predição de identidade associada à imagem fornecida por meio do código de identificação

cadastrado. O algoritmo retorna ainda a confiança associada à predição. Esta confiança nada mais é que a distância descrita na Equação 2.1, e por meio dela é possível avaliar a qualidade da predição. Na seção 4.5.3 foi observado durante testes com banco de faces que a maior parte das atribuições erradas de identidade resultaram em uma distância maior que 55, portanto na aplicação desenvolvida considerou-se que predições com distâncias maiores que esta seriam consideradas inválidas.

3.2.5 Geração de relatório

Após o processo de realização de chamada, o relatório de presença é gerado automaticamente e uma aplicação do Gmail é iniciada para que o professor possa enviar os dados obtidos no processo para um recipiente adequado. O relatório gerado apresenta as informações da disciplina, data atual e alunos identificados pela aplicação. A Figura 3.14 ilustra um relatório gerado, mostrando-se uma alternativa prática se comparado com as listas preenchidas à mão, pois não há a necessidade de transcrever para um computador as informações obtidas por escrito.

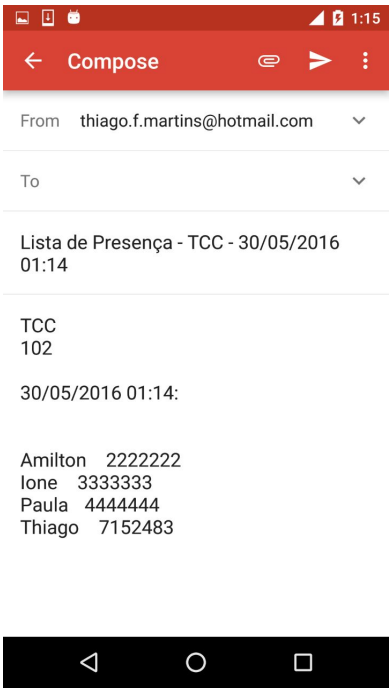

Figura 3.14: Relatório gerado ao fim do processo de chamada.

Capítulo 4

Resultados e Discussões

Este tópico apresenta as características técnicas e resultados obtidos com o desenvolvimento da aplicação. Foram verificados aspectos de requerimentos, consumo de recursos, taxa de quadros, usabilidade, taxa de acerto dos algoritmos de visão computacional e a geração de relatórios do sistema.

4.1 Requerimentos

O sistema desenvolvido possui requerimentos a serem cumpridos para funcionarem normalmente. A Tabela 4.1 apresenta os requisitos necessários. O sistema operacional mínimo deve ser o Android 4.0.3 em razão das funcionalidades do Android Studio utilizadas no desenvolvimento.

Sistema operacional mínimo	Android 4.0.3
Memória disponível	102 MB
Aplicativos	OpenCV Manager

Tabela 4.1: Requerimentos do aplicativo

4.2 Consumo de Recursos

Com a finalidade de avaliar o consumo de recursos, monitorou-se aspectos de memória do sistema nos pontos de processamento mais elevados da aplicação. Os principais pontos de processamento observados foram a ação de cadastro de um novo aluno e a realização de uma nova chamada.

4.2.1 Memória RAM

Uma das funcionalidades da plataforma Android Studio é o monitoramento de recursos consumidos pelo aplicativo. Por meio dele é possível verificar a quantidade de memória RAM (*Randon Acess Memory*)alocada para a aplicação. Procurou-se avaliar o consumo de memória RAM e CPU(*Central Process Unit*), em um *smartphone* modelo Nexus 5, durante as ações que exigem maior processamento do sistema: cadastro de novo aluno e realização de chamada. A ação de cadastro de novo aluno exige maior processamento pois realiza o processo de detecção de face do aluno, processamento da imagem e utilização do módulo de reconhecimento para extração de características utilizando LBP e posterior gravação deste modelo. A ação de nova chamada também exige maior processamento pois realiza uma nova detecção facial a cada quadro de imagem recebido pela câmera e utiliza o módulo de reconhecimento para identificação do indivíduo.

A Figura 4.1 retrata a ação de cadastro de um aluno. O primeiro aumento de alocação de memória é causado pela acionamento da tela de cadastro de aluno (em torno de 10 segundos), o segundo aumento de alocação de memória ocorre em torno de 17 segundos no gráfico e representa o momento do acionamento da tela de detecção facial. Após a detecção facial, a finalização do cadastro do usuário ocorre em torno dos 35 segundos, liberando parte da memória alocada para a aplicação.

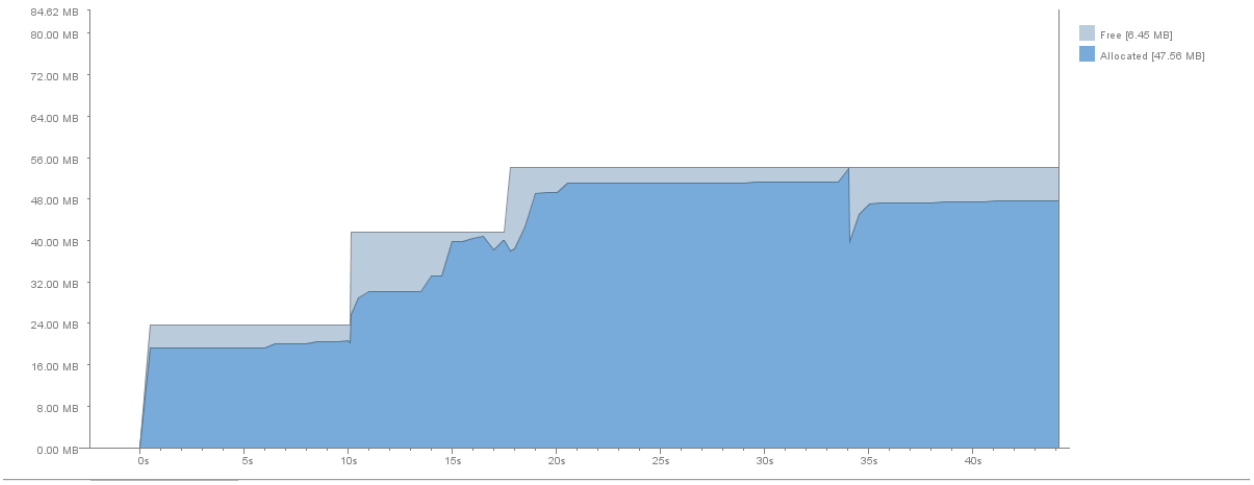

Figura 4.1: Consumo de memória na ação de cadastro (região clara:memória disponível, região escura:memória alocada

A Figura 4.2 retrata a o processo de realização de chamada onde duas pessoas foram identificadas. Nela observa-se que o primeiro aumento de memória alocada (em torno de 13

segundos) é causado pelo acionamento da tela de lista de aulas. A chamada em si é realizada entre 19 e 33 segundos. É possível observar picos sequenciais de alocação de memória nesta fase da aplicação que exige maior memória, devido principalmente ao processamento dos quadros provenientes da câmera e à utilização do módulo de reconhecimento facial. Finalmente após 33 segundos a tela de confirmação de chamada é acionada e as informações são salvas no banco de dados. Este processo exigiu maior alocação de memória, chegando a 71,51 MB de memória alocada.

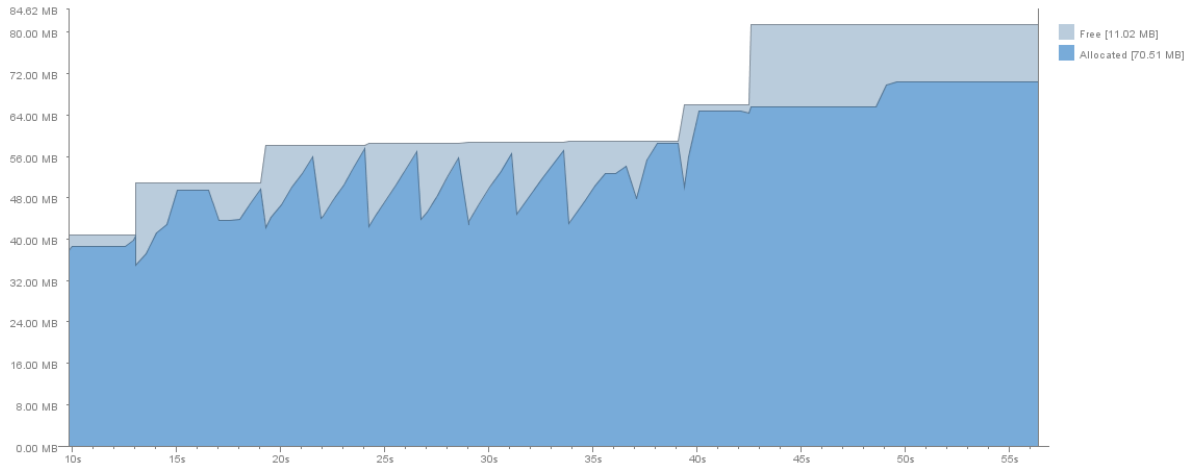

Figura 4.2: Consumo de memória durante uma chamada(região clara:memória disponível, região escura:memória alocada

Ambos processos exigem grande consumo de memória se comparado a situações onde não é necessário o processamento de quadros da câmera. Nota-se que a quantidade de memória disponível depende do consumo do aplicativo, e não houve nenhuma falha no aplicativo ou perda de desempenho durante as ações citadas.

4.2.2 CPU

Por meio da plataforma Android Studio ainda é possível monitorar o uso da CPU do aplicativo usados tanto no modo usuário e modo *kernel*. No modo usuário o código a ser executado não possui habilidade de acessar *hardware* ou memória de referência, e deve acionar API (*Application Programming Interface*) do sistema para acessar tais locais. No modo *kernel* o código tem acesso direto ao *hardware* e consegue executar qualquer instrução ou endereço de memória [47] . A seguir são apresentados o uso da CPU em um *smartphone* modelo Nexus 5 durante as ações de cadastro de novo aluno e realização de chamada. Todos os testes deste item foram realizados em condições normais de uso, ou seja, sem nenhum outro aplicativo

ativo paralelamente e sem nenhuma modificação de configuração de consumo de CPU. O consumo de CPU durante a ação de cadastro de um novo aluno é descrita na Figura 4.3, após o caminho a tela de cadastro de aluno, inicia-se o processo de detecção facial de nove imagens, ação realizada entre os intervalos de 16 e 25 segundos. O ato de salvar informações cadastrais do aluno no sistema é observada entre os intervalos de 35 e 40 segundos. Observa-se que ao longo do processo, utiliza-se no máximo 50%.

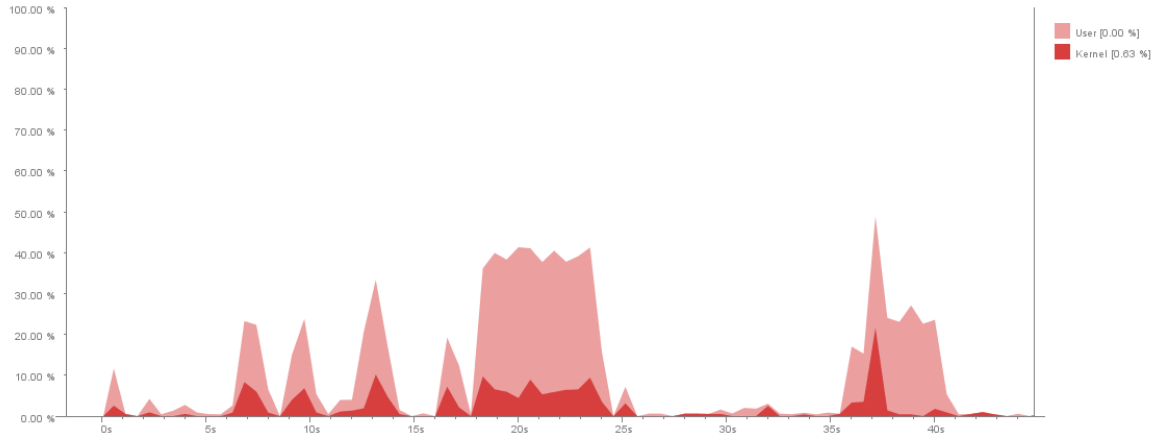

Figura 4.3: Porcentagem de utilização de CPU durante o cadastro de alunos(área clara:modo usuário, área escura: modo *kernel*

Para verificar o consumo de CPU durante a ação de realização de chamada, foi aberta uma chamada onde duas pessoas foram identificadas. Esta ação é visualizada na Figura 4.4, nota-se que o a maior consumo ocorre durante a realização de chamada.

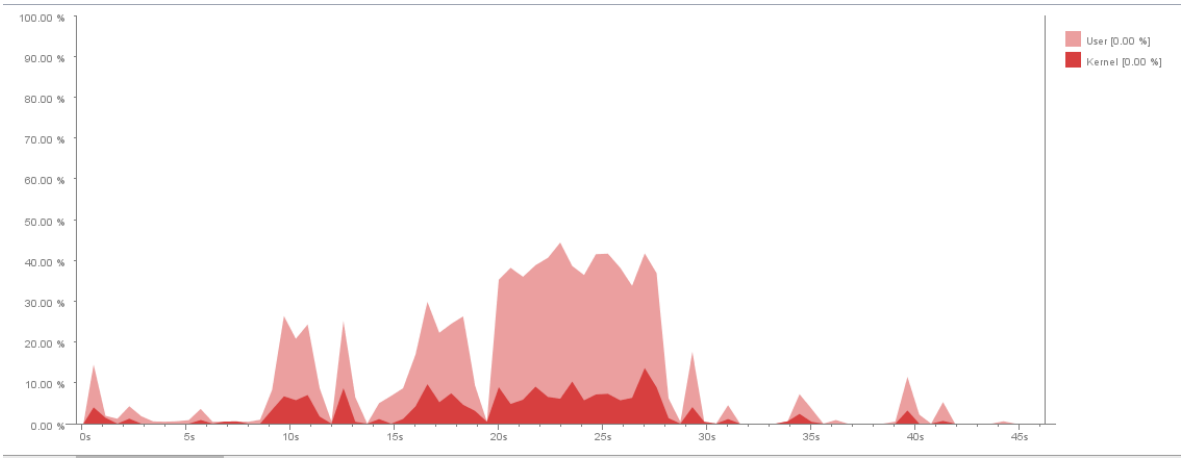

Figura 4.4: Porcentagem de utilização de CPU durante chamada(área clara:modo usuário, área escura: modo *kernel*

Em ambos casos o processamento máximo exigido da CPU foi menor que 50%, ou

seja, metade da quantidade disponível. Portanto considera-se este um valor aceitável para o *smartphone* utilizado.

4.3 Taxa de quadros

Com o intuito de avaliar a velocidade da aplicação, ou seja, o taxa de quadros por segundo visualizada pelo usuário nas telas onde há a utilização da câmera, mensurou-se a taxa de quadros na aplicação em duas telas diferentes: Tela de detecção facial na fase de cadastro, tela de detecção facial na fase de reconhecimento de identidade. Para cada tela foram realizadas 10 medidas em situações onde havia face presente na câmera, e 10 medidas onde não havia nenhum individuo. Os resultados são apresentados na Tabela 4.2.

Condição de uso	Média (FPS)
Detecção com face	6.02
Detecção sem face	14.96
Chamada com face	3.36
Chamada sem face	13.82

Tabela 4.2: Taxa de quadros média em diferentes situações

Nota-se que a tela de chamada apresenta menor velocidade em relação à de cadastro, principalmente devido a maior quantidade de tarefas exigidas, pois além da detecção de face, é necessário reconhecer a identidade da pessoa avaliada e atualizar informações de tela como nome e identidade do aluno. Observa-se ainda que a taxa de quadros por segundo visualizada pelo usuário é maior quando não há amostras de faces na câmera, isto se deve ao método de detecção facial, que utiliza classificadores em cascata de modo que a rejeição de amostras avaliadas como negativas ocorre rapidamente. As taxas obtidas foram consideradas aceitáveis, pois mesmo a uma taxa de 3,36 quadros por segundos, é possível que o professor possa visualizar os alunos identificados com rapidez.

4.4 Aspectos de IHC

O objetivo desta seção é avaliar o desempenho da aplicação desenvolvida com respeito a aspectos de IHC (Interação Humano-Computador). Embora aspectos técnicos como uso de memória e CPU sejam importantes, ainda há desafios a serem atingidos com respeito ao

desenvolvimento de um aplicativo para *smartphones* devido a aspectos como baixa resolução da tela, dificuldade de navegação e dificuldade nos métodos de entrada de informação [48]. A usabilidade se torna então um fator importante nos requisitos desejáveis de uma aplicação, pois interferem diretamente na satisfação do usuário ao utilizar o sistema. A fim de investigar a usabilidade do sistema, utilizou-se SUS (*System Usability Scale*), pois se mostra um teste confiável e de curta duração. A avaliação de usabilidade foi realizada expondo 5 pessoas à aplicação pela primeira vez durante 5 minutos. O questionário foi então realizado e por meio dele foi calculado a nota média alcançada pela aplicação para cada pessoa. Os resultados são apresentados na Tabela 4.3.

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Nota
Participante 1	3	4	3	1	2	3	2	4	4	2	50
Participante 2	4	1	2	1	3	4	3	5	4	1	60
Participante 3	3	1	4	1	3	2	4	4	4	1	72.5
Participante 4	3	3	2	1	4	4	3	4	2	2	50
Participante 5	4	2	2	1	3	4	4	4	4	1	62.5

Tabela 4.3: Resultado obtido na avaliação de usabilidade

A nota média obtida foi 59, valor um pouco abaixo da média, porém com estes resultados foi possível notar quais aspectos do sistema precisam ser evoluídos. Os itens que obtiveram piores resultados (6 e 8) estão relacionados à inconsistência e desconforto na utilização. As melhores notas obtidas (1,4 e 10) estão relacionadas à aceitabilidade e facilidade de aprendizado.

4.5 Taxa de acerto dos algoritmos de visão computacional

Este tópico tem como objetivo avaliar a taxa de acerto dos algoritmos de visão computacional utilizados.Para aplicação foi considerada aceitável uma taxa de reconhecimento de 80%. A avaliação foi feita utilizando um banco de faces e por meio de um experimento com 10 pessoas.

4.5.1 Banco de faces

Com o intuito de avaliar o algoritmo de reconhecimento facial, é desejável utilizar um banco de imagens faciais com uma quantidade elevada de pessoas em diferentes aspectos como

variação de pose, iluminação e emoção, simulando assim situações de uso cotidiano. O banco de faces utilizado foi o *Essex Faces 94* [49], banco de faces criado por Dr. Libor Spacek com 20 imagens de 153 indivíduos com mesmo plano de fundo, sem variações de tamanho da imagem facial, com pequenas variações de condições de iluminação e pose [49]. A Figura 4.5 ilustra uma amostra de imagens de um indíviduo deste banco.

Figura 4.5: Amostra do banco de faces Essex Faces 94

4.5.2 Detecção de face

Com o propósito de apurar a taxa de detecção facial do sistema proposto, aplicou-se o detector no banco de imagens [49], e logo após foi verificado então se o detector capturou corretamente a região facial da imagem. O banco proposto apresenta indivíduos com variação limitada de pose (menor que 10 graus de inclinação), útil no aferimento da taxa de detecção do método proposto pois como o treinamento dos classificadores foi treinado apenas para imagens faciais frontais, o detector não possuirá boa taxa de detecção em casos onde há elevada inclinação da posição da face (face inclinada 90 graus para a direita, por exemplo).

Utilizando um total de 3060 imagens, o detector facial falhou em apenas 38 ocasiões, todas apresentando um resultado falso negativo, ou seja, o algoritmo classificou erroneamente 38 imagens faciais como não faciais. A taxa de acerto do algoritmo resultou então em 98,75%.

Durante a utilização do aplicativo, notou-se que a detecção facial implementada possui alta taxa de detecção em ambientes controlados. Em ambientes com iluminação extrema-

mente elevada ou baixa porém, o detector não identifica faces presentes na câmera.

4.5.3 Reconhecimento de identidade

O método de avaliação proposto consiste em cadastrar determinada quantidade de imagens faciais de diferentes indivíduos no sistema utilizando o algoritmo de reconhcecimento facial da biblioteca OpenCV e posteriormente avaliar imagens (diferentes das previamente cadastradas) dos mesmos indivíduos para que se possa avaliar a taxa de acerto do algoritmo. O banco de imagens utilizado [49] possui 20 imagens contento a região facial de 152 indivíduos. Para avaliar a taxa de acerto do algoritmo, foram realizados testes em três cenários distintos.

- Primeiro Cenário: Para cada pessoa, cadastro de 5 imagens e avaliação do reconhecimento em 15. imagens;
- Segundo Cenário: Para cada pessoa, cadastro de 10 imagens e avaliação do reconhecimento em 10. imagens ;
- Terceiro Cenário: Para cada pessoa, cadastro de 15 imagens e avaliação do reconhecimento em 5. imagens.

Os resultados dos testes são apresentados na Tabela 4.4.

Cenário testado	Taxa de acerto (%)
Cenário 1	89,91
Cenário 2	93,15
Cenário 3	96,26

Tabela 4.4: Taxa de reconhecimento em diferentes cenários

Confirma-se que um maior número de amostras para o treinamento do algoritmo proporciona uma maior taxa de reconhecimento, entretanto no cenário da aplicação onde há limitação de memória foi necessário limitar o tamanho das amostras para extração de características faciais[10].

Os resultados da taxa de reconhecimento obtidos foram promissores, atingir a taxa de 100% de reconhecimento de identidade é uma tarefa árdua de se atingir, especialmente em um cenário real onde fatores como a iluminação e pose interferem no reconhecimento [10]. Sabendo disto, é necessário ressaltar que o banco de faces proposto por Dr. Libor Spacek

apresenta baixa variação de condições externas como iluminação, escala e pose. Para investigar a taxa de reconhecimento do algoritmo sob condições externas mais severas realizou-se um experimento com dez pessoas.

Cadastrou-se dez pessoas em uma disciplina do aplicativo em ambiente controlado de iluminação, e posteriormente foi feita uma chamada em condições não controladas de iluminação para verificar a taxa de acerto do algoritmo de reconhecimento. Foi constatado que 8 pessoas foram reconhecidas normalmente, resultando em uma taxa de 80% de acerto. As duas pessoas não identificadas apresentaram maior dificuldade de reconhecimento, especialmente devido à variações de pose, pois quando a pessoa testada fazia pose similar à cadastrada, o algoritmo o reconhecia novamente. A taxa de reconhecimento obtida foi um valor considerado viável para utilizar o sistema em condições reais, além disto, o método de adição manual de presença de aluno procura contornar situações onde houve erro de identificação, e mais importante, tentar corrigi-las atualizando as informações cadastrais do individuo.

4.6 Banco de dados

Com o intuito de avaliar o tamanho do banco de dados da aplicação realizou-se experimentos em três cenários diferentes citados a seguir: Cenário 1: Verificou-se o crescimento do tamanho do banco de dados conforme o cadastro de alunos. A Figura 4.6 ilustra este crescimento. Nota-se um crescimento linear do tamanho do banco de dados, principalmente ao número de fotos necessárias para o armazenamento da informação do aluno.

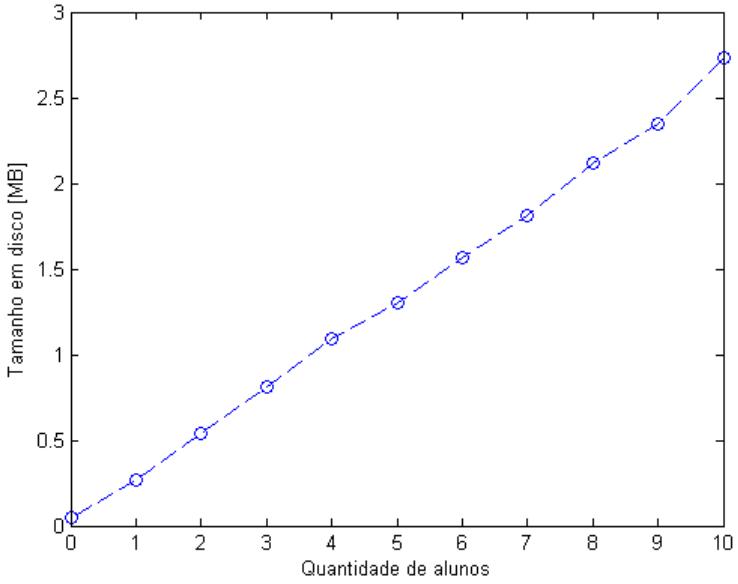

Figura 4.6: Crescimento do espaço em em função do cadastramento de alunos.

Cenário 2: Verificou-se o crescimento do tamanho do banco de dados conforme o cadastro de disciplinas conforme Figura 4.7. Desde que no processo de adição de uma nova disciplina não há informações pesadas a serem armazenadas (como uma foto), o tamanho do banco de dados permanece constante.

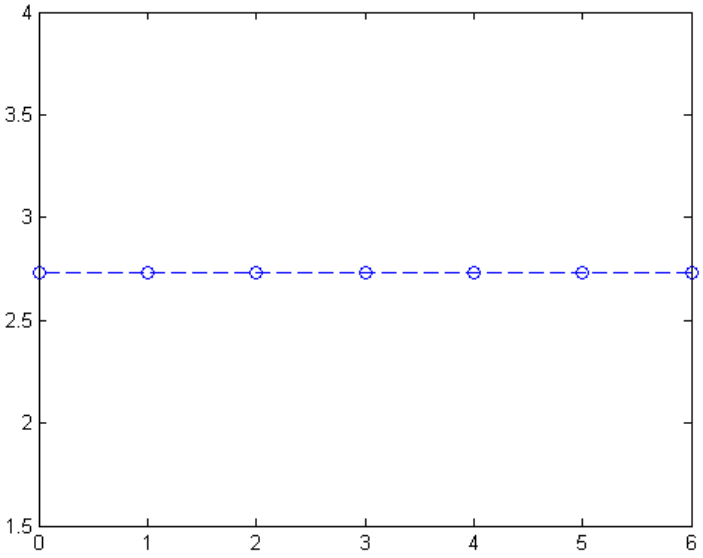

Figura 4.7: Crescimento do espaço em disco em função do número de disciplinas cadastradas.

Cenário 3: Verificou-se o crescimento do tamanho do banco de dados conforme novas

chamadas realizadas na Figura 4.8. Novamente nota-se um crescimento linear do tamanho do banco de dados ocasionado provavelmente pela quantidade de fotos de alunos presentes na chamada que foram armazenadas .

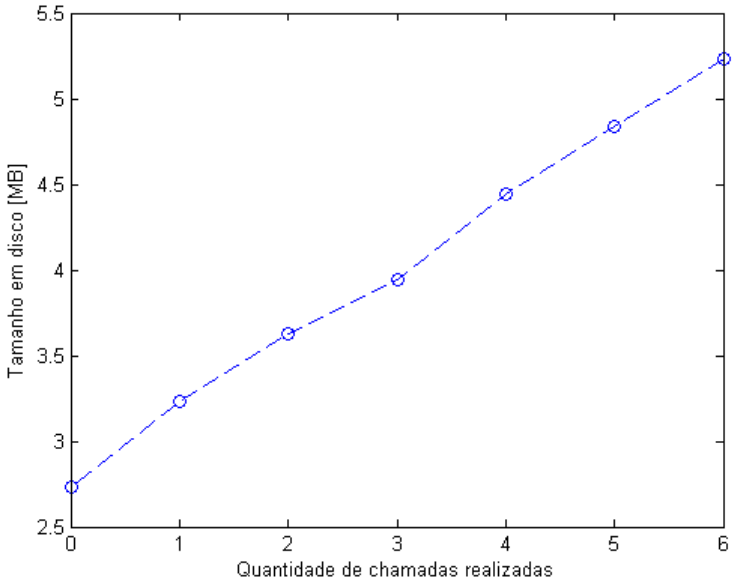

Figura 4.8: Crescimento do espaço em disco em função do número de chamadas realizadas.

Outro arquivo que ocupa espaço durante o cadastro de alunos é o modelo de reconhecimento do algoritmo. OpenCV permite o armazenamento do modelo interno do algoritmo em extensão do tipo "XML", que foi gravada na memória externa do aplicativo em razão da solução desenvolvida por meio do SQLite permitir apenas o armazenamento de dados do tipo inteiro ou alfabético. A Figura 4.9 representa o crescimento do arquivo gerado pelo algoritmo conforme o número de alunos cadastrados.

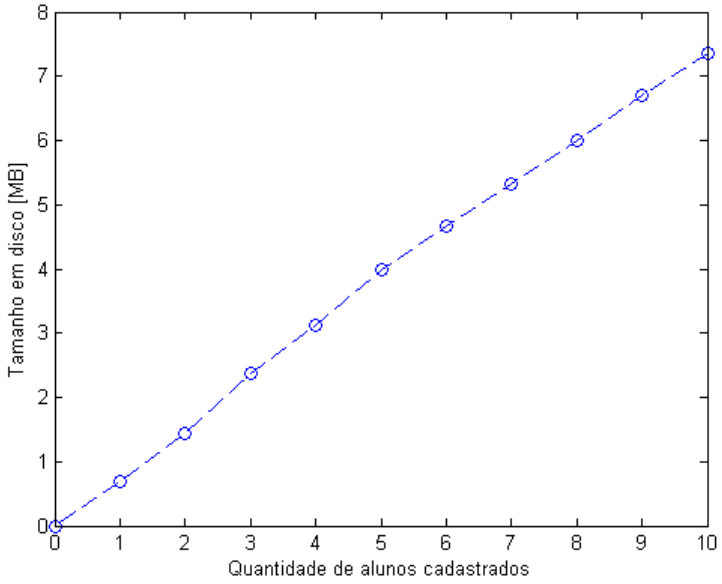

Figura 4.9: Crescimento do modelo de reconhecimento facial em função do número de alunos cadastrados

Por meio da avaliação feita, foi possível concluir que os maiores responsável pelo aumento do espaço em disco são a quantidade de imagens armazenadas pelo sistema e o arquivo gerado pelo reconhecedor facial. Uma alternativa para limitar o crescimento do tamanho do banco de dados conforme o cadastro de alunos é limitar o tamanho e o número de imagens cadastradas, entretanto isto pode interferir na performance de reconhecimento de identidade.

4.7 Relatório

Para que o professor possa ter o controle da presença de seus alunos, foi gerado um relatório com informações das aulas realizadas. A Figura 4.10 ilustra um relatório gerado, mostrando-se uma alternativa prática se comparado com as listas preenchidas à mão, pois não há a necessidade de transcrever para um computador as informações obtidas por escrito.

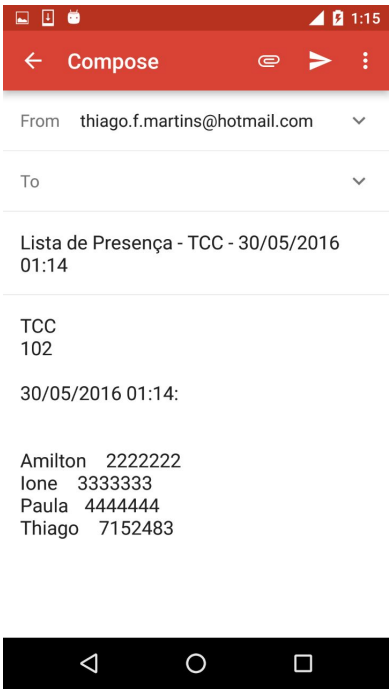

Figura 4.10: Relatório gerado ao fim do processo de chamada

Capítulo 5

Conclusão

A existência de um meio eficiente de se fazer o controle de presença com apoio de técnicas de biometria é interessante para vários tipos de usuários como empresários e professores. O aplicativo proposto neste trabalho visa sanar problemas relacionados a esta área, utilizando aspectos de reconhecimento facial e interação humano-computador em um sistema de controle de presença de alunos em aulas.

Foram utilizadas técnicas de visão computacional para realização da detecção de face e reconhecimento de identidade de alunos. Este processo acontece durante a realização da chamada após o usuário professor ter cadastrado uma disciplina e seus alunos no sistema. O sistema ainda gera relatórios a cada chamada realizada e envia e-mail contendo informações da disciplina, data e alunos presentes.

Testes com usuários mostraram que a interface desenvolvida apresenta boa aceitabilidade baseada no SUS (*System Usability Scale*), porém ainda pode ser evoluída em aspectos como inconsistência e desconforto de uso. Com relação à interface com a câmera, foi possível obter uma velocidade de aproximadamente 3 quadros por segundo durante o reconhecimento facial, taxa considerada aceitável considerando o processamento executado. Os testes realizados no *smartphone* Nexus 5 mostraram que consumo de recursos da aplicação não atingiu seu limite. Por fim, o módulo de reconhecimento facial apresentou taxas de 80%, propiciando assim as condições necessárias para o funcionamento operacional do sistema. Não obstante, métodos foram realizados a fim de atualizar o cadastro de alunos em casos de falha de reconhecimento.

Ainda há evoluções que podem ser feitas com relação à aspectos do sistema de modo a melhorar aspectos de usabilidade, performance e eficiência. O subitem seguinte apresenta sugestões de trabalhos futuros identificados ao longo deste projeto que visam aprimorar o aplicativo. Os testes realizados mostram, porém, que a ferramenta já pode ser utilizada de

forma experimental.

Trabalhos Futuros

Para evoluir a usabilidade do sistema, recomenda-se corrigir aspectos de usabilidade apurados, apesar da aplicação possuir aceitabilidade. Realizar estudos mais detalhados para investigar a razão de tais desconfortos e estudar as melhores técnicas disponíveis para que o conforto do usuário seja maior durante a utilização da aplicação.

Para melhor utilização do sistema por professores, é recomendado adicionar uma funcionalidade para organizar disciplinas por semestres, anos e turmas, tornando a visualização da informação mais agradável.

Uma alternativa para diminuir o espaço em disco utilizado no celular do usuário é utilizar a nuvem, desta forma a aplicação se torna mais atraente ao consumidor pois além de evitar o uso da memória de armazenamento de seu *smartphone*, que geralmente é limitada, a apliação ainda se torna mais eficiente pois utilizando nuvens Microsoft Azure ou Google Cloud é possível aumentar a velocidade de processamento do sistema.

Referências

[1] Luciano Xavier Medeiros. Reconhecimento facial utilizando análise de componentes e algoritmos genéticos em imagens segmentadas. 2012.

[2] Anil K Jain, Arun Ross, and Salil Prabhakar. An introduction to biometric recognition. *Circuits and Systems for Video Technology, IEEE Transactions on*, 14(1):4–20, 2004.

[3] Nettime solutions. `http://www.nettimesolutions.com/features`, Acesso em: 25 de abril de 2016.

[4] Victor Nascimento. Implementação de um sistema de identificação facial utilizando linux embarcado. 2015.

[5] Classroom monitor attendance. `https://play.google.com/store/apps/details?id=com.axonbots.app.classmonitorfree&hl=en`, Acesso em: 25 de abril de 2016.

[6] Attendance. `https://play.google.com/store/apps/details?id=com.aor.attendance&hl=en`, Acesso em: 25 de abril de 2016.

[7] Ditribuição do mercado de *smartphones*. `https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=8&qpcustomd=1`, Acesso em: 15 de abril de 2016.

[8] Rafael C Gonzalez and Richard E Woods. Digital image processing, 2002.

[9] Anil K Jain and Stan Z Li. *Handbook of face recognition*, volume 1. Springer, 2005.

[10] Wenyi Zhao, Rama Chellappa, P Jonathon Phillips, and Azriel Rosenfeld. Face recognition: A literature survey. *ACM computing surveys (CSUR)*, 35(4):399–458, 2003.

[11] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511. IEEE, 2001.

[12] Yann Rodriguez. Face detection and verification using local binary patterns. Technical report, IDIAP, 2006.

[13] Jo Chang-yeon. Face detection using lbp features. *Final Project Report*, 77, 2008.

[14] Rainer Lienhart and Jochen Maydt. An extended set of haar-like features for rapid object detection. In *Image Processing. 2002. Proceedings. 2002 International Conference on*, volume 1, pages I–900. IEEE, 2002.

[15] Lun Zhang, Rufeng Chu, Shiming Xiang, Shengcai Liao, and Stan Z Li. Face detection based on multi-block lbp representation. In *Advances in biometrics*, pages 11–18. Springer, 2007.

[16] Timo Ojala, Matti Pietikäinen, and David Harwood. A comparative study of texture measures with classification based on featured distributions. *Pattern recognition*, 29(1): 51–59, 1996.

[17] Opencv, reconhecimento facial. `http://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_tutorial.html`, Acesso em: 15 de maio de 2016.

[18] Alexandre Fieno da Silva. Reconhecimento de faces via pca: análise de desempenho. 2006.

[19] Matthew Turk and Alex Pentland. Eigenfaces for recognition. *Journal of cognitive neuroscience*, 3(1):71–86, 1991.

[20] Kamran Etemad and Rama Chellappa. Discriminant analysis for recognition of human face images. *JOSA A*, 14(8):1724–1733, 1997.

[21] Timo Ahonen, Abdenour Hadid, and Matti Pietikäinen. Face recognition with local binary patterns. In *Computer vision-eccv 2004*, pages 469–481. Springer, 2004.

[22] Timo Ahonen, Abdenour Hadid, and Matti Pietikainen. Face description with local binary patterns: Application to face recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(12):2037–2041, 2006.

[23] Timo Ojala, Matti Pietikainen, and Topi Maenpaa. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on pattern analysis and machine intelligence*, 24(7):971–987, 2002.

[24] Conceitos de usabilidade. https://www.nngroup.com/articles/usability-101-introduction-to-usability/, Acesso em: 29 de maio de 2016.

[25] John Brooke et al. Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7, 1996.

[26] Usability.gov. http://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html, Acesso em: 15 de maio de 2016.

[27] *Material design*. https://www.google.com/design/spec/material-design/introduction.html, Acesso em: 25 de abril de 2016.

[28] *Material design*: botão flutuante. https://www.google.com/design/spec/components/buttons-floating-action-button.html#, Acesso em: 25 de abril de 2016.

[29] *Material design*: lista. https://www.google.com/design/spec/components/lists.html#lists-usage, Acesso em: 25 de abril de 2016.

[30] *Material design*: ícones. https://design.google.com/icons/, Acesso em: 25 de abril de 2016.

[31] App inventor. http://appinventor.mit.edu/explore, Acesso em: 30 de maio de 2016.

[32] Limitações do app inventor. http://https://sites.google.com/site/appinventor/capabilities-limitations, Acesso em: 30 de maio de 2016.

[33] Extensões disponíveis do app inventor. http://appinventor.mit.edu/extensions/, Acesso em: 30 de maio de 2016.

[34] Conceitos básicos de java. http://www.ibm.com/developerworks/java/tutorials/j-introtojava1/, Acesso em: 17 de abril de 2016.

[35] Desenvolvimento android. https://developer.android.com/index.html, Acesso em: 17 de abril de 2016.

[36] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. "O'Reilly Media, Inc.", 2008.

[37] Opencv manager. https://play.google.com/store/apps/details?id=org.opencv.engine&hl=en, Acesso em: 15 de maio de 2016.

[38] Salil Kapur and Nisarg Thakkar. Mastering opencv android application programming. 2015.

[39] Tutorial para desenvolvimento android com opencv. http://docs.opencv.org/2.4/doc/tutorials/introduction/android_binary_package/dev_with_OCV_on_Android.html, Acesso em: 15 de maio de 2016.

[40] Sqlite. https://www.sqlite.org/about.html, Acesso em: 17 de abril de 2016.

[41] Tutorial de utilização de banco de dados sqlite em android. http://www.vogella.com/tutorials/AndroidSQLite/article.html, Acesso em: 17 de abril de 2016.

[42] Especificações do nexus 5. https://support.google.com/nexus/answer/6102470?hl=en, Acesso em: 05 de maio de 2016.

[43] Nexus 5. http://www.thenexus5.com/wp-content/uploads/2014/07/20131018T122945.png, Acesso em: 05 de maio de 2016.

[44] Opencv. http://opencv.org/, Acesso em: 03 de maio de 2016.

[45] Patente de detecção de objetos utilizando classificadores do tipo haar. http://www.google.com/patents/US20110255743, Acesso em: 27 de abril de 2016.

[46] Classe facerecognizer. http://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_api.html, Acesso em: 15 de abril de 2016.

[47] Usability.gov. https://developer.android.com/studio/profile/am-cpu.html, Acesso em: 15 de maio de 2016.

[48] Constantinos Coursaris and Dan Kim. A qualitative review of empirical mobile usability studies. *AMCIS 2006 Proceedings*, page 352, 2006.

[49] Libor Spacek. The essex faces94 database.

Anexo I

Reconhecimento facial

```
package br.usp.eesc.sel.thiagofmartins.tcc.recogattendance.recognition;

import android.graphics.Bitmap;
import android.util.Log;

import com.googlecode.javacv.cpp.opencv_contrib;
import com.googlecode.javacv.cpp.opencv_core;
import com.googlecode.javacv.cpp.opencv_core.MatVector;
import com.googlecode.javacv.cpp.opencv_imgproc;

import org.opencv.android.Utils;
import org.opencv.core.Mat;

import java.io.File;
import java.io.FileOutputStream;

import static com.googlecode.javacv.cpp.opencv_core.IPL_DEPTH_8U;
import static com.googlecode.javacv.cpp.opencv_imgproc.cvCvtColor;

public class PersonRecognizer {


    opencv_contrib.FaceRecognizer faceRecognizer;
```

```
String Path;

static  final int WIDTH= 130;
static  final int HEIGHT= 150;
private int confidence;


public PersonRecognizer(String path)
{
    faceRecognizer =  com.googlecode.javacv.cpp.opencv_contrib.create
    Path=path;
}

public void update(MatVector matVector, int[] labels, String descript
    int size = 2;

    if (size>1)
    {
        faceRecognizer.update(matVector, labels);
    }
    else {
        faceRecognizer.train(matVector, labels);
    }
    faceRecognizer.save(Facefile);

}


public int predict(Mat m) {

    int n[] = new int[1];
    double p[] = new double[1];
    opencv_core.IplImage ipl = MatToIplImage(m,WIDTH, HEIGHT);
```

```
//                IplImage ipl = MatToIplImage(m,-1, -1);

        faceRecognizer.predict(ipl, n, p);
        if (n[0]!=-1)
        {confidence = (int)p[0];}
        else
        {confidence=-1;}

        if (n[0] != -1)
        {return n[0];}
        else
        {return -2;}
    }

    opencv_core.IplImage MatToIplImage(Mat m,int width,int heigth)
    {
        Bitmap bmp=Bitmap.createBitmap(m.width(), m.height(), Bitmap.Con
        Utils.matToBitmap(m, bmp);
        return BitmapToIplImage(bmp,width, heigth);

    }

    opencv_core.IplImage BitmapToIplImage(Bitmap bmp, int width, int hei

        if ((width != -1) || (height != -1)) {
            Bitmap bmp2 = Bitmap.createScaledBitmap(bmp, width, height,
            bmp = bmp2;
        }
        opencv_core.IplImage image = opencv_core.IplImage.create(bmp.getW
        bmp.copyPixelsToBuffer(image.getByteBuffer());

        opencv_core.IplImage grayImg = opencv_core.IplImage.create(image
        cvCvtColor(image, grayImg, opencv_imgproc.CV_BGR2GRAY);
```

```
        return grayImg;
    }

    public void load2(String path)
    {
        try {
            faceRecognizer.load(path);

        } catch (Exception e) {
            Log.e("error", e.getCause() + " " + e.getMessage());
            e.printStackTrace();
        }
    }

    public int getConfidence (){
        return confidence;
    }


}
```

Anexo II

Banco de dados

```
package br.usp.eesc.sel.thiagofmartins.tcc.recogattendance.database.helpe

import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.os.Build;
import android.support.v4.util.LruCache;
import android.util.Base64;

import java.io.ByteArrayOutputStream;

/**
 * Created by 40276655893 on 22/04/2016.
 */
public class DatabaseUtil {

    private static LruCache<String, Bitmap> photoMemoryCache;

    public static String bitmapToString(Bitmap bitmap){
        ByteArrayOutputStream baos=new ByteArrayOutputStream();
        bitmap.compress(Bitmap.CompressFormat.PNG,100, baos);
        byte [] b=baos.toByteArray();
        String temp=Base64.encodeToString(b, Base64.DEFAULT);
        return temp;
```

```
    }

    /**
     * @param encodedString
     * @return bitmap (from given string)
     */
    public static Bitmap stringToBitmap(String encodedString){
        try {
            byte [] encodeByte=Base64.decode(encodedString,Base64.DEFAUL
            Bitmap bitmap= BitmapFactory.decodeByteArray(encodeByte, 0,
            return bitmap;
        } catch(Exception e) {
            e.getMessage();
            return null;
        }
    }

    public static void initPhotoCache() {
        final int maxMemory = (int) (Runtime.getRuntime().maxMemory() /
        final int cacheSize = maxMemory / 10;

        photoMemoryCache = new LruCache<String, Bitmap>(cacheSize) {
            @Override
            protected int sizeOf(String key, Bitmap bitmap) {
                if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOM
                    return bitmap.getByteCount() / 1024;
                } else {
                    return ((bitmap.getRowBytes() * bitmap.getHeight())/1
                }
            }
        };
    }
```

```
    public static LruCache getPhotoCache(){
        return photoMemoryCache;
    }

    public static void addBitmapToMemoryCache(String key, Bitmap bitmap)
        if (getBitmapFromMemCache(key) == null) {
            getPhotoCache().put(key, bitmap);
        }
    }

    public static Bitmap getBitmapFromMemCache(String key) {
        return (Bitmap)getPhotoCache().get(key);
    }

}
```

Anexo III

Tela de chamada

```
package br.usp.eesc.sel.thiagofmartins.tcc.recogattendance.view;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.hardware.Camera;
import android.os.Bundle;
import android.os.Environment;
import android.os.Handler;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.WindowManager;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ImageButton;
import android.widget.ImageView;
import android.widget.LinearLayout;
import android.widget.TextView;
import android.widget.ToggleButton;
```

```
import org.opencv.android.BaseLoaderCallback;
import org.opencv.android.CameraBridgeViewBase;
import org.opencv.android.LoaderCallbackInterface;
import org.opencv.android.OpenCVLoader;
import org.opencv.core.Core;
import org.opencv.core.Mat;
import org.opencv.core.MatOfRect;
import org.opencv.core.Point;
import org.opencv.core.Rect;
import org.opencv.core.Scalar;
import org.opencv.core.Size;
import org.opencv.imgproc.Imgproc;
import org.opencv.objdetect.CascadeClassifier;
import org.opencv.objdetect.Objdetect;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.GregorianCalendar;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import br.usp.eesc.sel.thiagofmartins.tcc.recogattendance.database.helpe
import br.usp.eesc.sel.thiagofmartins.tcc.recogattendance.database.helpe
import br.usp.eesc.sel.thiagofmartins.tcc.recogattendance.database.model
```

```
import br.usp.eesc.sel.thiagofmartins.tcc.recogattendance.database.model
import br.usp.eesc.sel.thiagofmartins.tcc.recogattendance.database.model
import br.usp.eesc.sel.thiagofmartins.tcc.recogattendance.recognition.La
import br.usp.eesc.sel.thiagofmartins.tcc.recogattendance.recognition.Ol
import br.usp.eesc.sel.thiagofmartins.tcc.recogattendance.recognition.Pe
import br.usp.eesc.sel.thiagofmartins.tcc.recogattendance.recognition.Pr
import br.usp.eesc.sel.thiagofmartins.tcc.recogattendance.recognition.Tu

//import java.io.FileNotFoundException;
//import org.opencv.contrib.FaceRecognizer;

/**
 * Created by Thiago on 1/31/2016.
 */


public class ChamadaActivity extends Activity implements CameraBridgeView



    private Camera mCamera;
    private TutorialCamera mOpenCvCameraView;
    private static final String    TAG                 = "OCVSample::Act
    private static final Scalar FACE_RECT_COLOR     = new Scalar(0, 255,
    public static final int        JAVA_DETECTOR       = 0;
    public static final int        NATIVE_DETECTOR     = 1;

    public static final int TRAINING= 0;
    public static final int SEARCHING= 1;
    public static final int IDLE= 2;

    private static final int frontCam =1;
    private static final int backCam =2;
```

```
    private int faceState=IDLE;

//
    private MenuItem               nBackCam;
    private MenuItem               mFrontCam;
    private MenuItem               mEigen;


    private Mat mRgba;
    private Mat                    mGray;
    private File mCascadeFile;
    private CascadeClassifier mJavaDetector;
    Olhos olhos = new Olhos();

    private File                   eyeCascade1File;
    private File                   eyeCascade2File;
    public CascadeClassifier      eyeCascade1;
    public CascadeClassifier      eyeCascade2;

    private int                    mDetectorType       = JAVA_DETECTOR;
    private String[]               mDetectorName;

    private float                  mRelativeFaceSize   = 0.2f;
    private int                    mAbsoluteFaceSize   = 0;
    private int mLikely=999;

    String mPath="";


    private int mChooseCamera = backCam;
```

```
EditText text;
TextView textresult;
private ImageView Iv;

ImageButton imCamera;

com.googlecode.javacv.cpp.opencv_contrib.FaceRecognizer faceRecogniz


long MAXIMG=9;

ProcessaImagem meuProcesso = new ProcessaImagem();

PersonRecognizer fr;
String Path="";
String Facefile="";
Labels labelsFile;
LinearLayout layout;
LinearLayout cam;
String resultName;
int who;

TextView name;
TextView id;
ImageView image;


Photo photo = new Photo();
Student student = new Student();
Boolean canPredict;
public List<Long> presentStudentsList = new ArrayList<Long>();

private List<Integer> averagePerson= new ArrayList<Integer>();
```

```
public List<Student> studentList= new ArrayList<Student>();
Mat processada;
Lesson lesson = new Lesson();

private BaseLoaderCallback mLoaderCallback = new BaseLoaderCallback(
    @Override
    public void onManagerConnected(int status) {
        switch (status) {
            case LoaderCallbackInterface.SUCCESS:
            {
                Log.i(TAG, "OpenCV loaded successfully");

                // Load native library after(!) OpenCV initialization
                //    System.loadLibrary("detection_based_tracker");


                try {
                    // load cascade file from application resources
                    InputStream is = getResources().openRawResource(R
                    File cascadeDir = getDir("cascade", Context.MODE
                    mCascadeFile = new File(cascadeDir, "lbpcascade_
                    FileOutputStream os = new FileOutputStream(mCasc

                    byte[] buffer = new byte[4096];
                    int bytesRead;
                    while ((bytesRead = is.read(buffer)) != -1) {
                        os.write(buffer, 0, bytesRead);
                    }
                    is.close();
                    os.close();

                    mJavaDetector = new CascadeClassifier(mCascadeFi
                    if (mJavaDetector.empty()) {
```

```
            Log.e(TAG, "Failed to load cascade classifier
            mJavaDetector = null;
        } else
            Log.i(TAG, "Loaded cascade classifier from "

        //                  mNativeDetector = new Detectio

        cascadeDir.delete();

    } catch (IOException e) {
        e.printStackTrace();
        Log.e(TAG, "Failed to load cascade. Exception thr
    }

    try {
        // load cascade file from application resources
        InputStream isa = getResources().openRawResource(
        File cascadeDira = getDir("cascade", Context.MOD
        eyeCascade1File = new File(cascadeDira, "haarcasc
        FileOutputStream osa = new FileOutputStream(eyeC

        byte[] buffera = new byte[4096];
        int bytesRead;
        while ((bytesRead = isa.read(buffera)) != -1) {
            osa.write(buffera, 0, bytesRead);
        }
        isa.close();
        osa.close();

        eyeCascade1 = new CascadeClassifier(eyeCascade1F
        if (eyeCascade1.empty()) {
            Log.e(TAG, "Failed to load cascade classifier
            eyeCascade1 = null;
```

```
            } else
                Log.i(TAG, "Loaded cascade classifier from "

            //mJavaDetector = new CascadeClassifier(faceCasc

            cascadeDira.delete();

        } catch (IOException e) {
            e.printStackTrace();
            Log.e(TAG, "Failed to load cascade. Exception thr
        }


        try {
            // load cascade file from application resources
            InputStream isb = getResources().openRawResource(
            File cascadeDirb = getDir("cascade", Context.MOD
            eyeCascade2File = new File(cascadeDirb, "haarcasc
            FileOutputStream osb = new FileOutputStream(eyeC

            byte[] bufferb = new byte[4096];
            int bytesRead;
            while ((bytesRead = isb.read(bufferb)) != -1) {
                osb.write(bufferb, 0, bytesRead);
            }
            isb.close();
            osb.close();

            eyeCascade2 = new CascadeClassifier(eyeCascade2F
            if (eyeCascade2.empty()) {
                Log.e(TAG, "Failed to load cascade classifier
                eyeCascade2 = null;
            } else
```

```
                        Log.i(TAG, "Loaded cascade classifier from "

                        //mJavaDetector = new CascadeClassifier(faceCasc

                        cascadeDirb.delete();

                    } catch (IOException e) {
                        e.printStackTrace();
                        Log.e(TAG, "Failed to load cascade. Exception thr
                    }


                    mOpenCvCameraView.enableView();

                } break;
                default:
                {
                    super.onManagerConnected(status);
                } break;


            }
        }
    };



    public ChamadaActivity() {
        mDetectorName = new String[2];
        mDetectorName[JAVA_DETECTOR] = "Java";
        mDetectorName[NATIVE_DETECTOR] = "Native (tracking)";

        Log.i(TAG, "Instantiated new " + this.getClass());
```

```
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Inte
    {
        if (resultCode==RESULT_OK) {
            Long studentCode = getIntent().getLongExtra("studentCode", 0
            String image = getIntent().getStringExtra("face");
            Bitmap face = DatabaseUtil.stringToBitmap(image);

            if (studentCode > 0) {
                presentStudentsList.add(studentCode);
            }
        }
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        Log.i(TAG, "called onCreate");
        super.onCreate(savedInstanceState);
        getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_
        setContentView(R.layout.activity_chamada);
        mOpenCvCameraView = (TutorialCamera) findViewById(R.id.tutorialCa
        mOpenCvCameraView.setCvCameraViewListener(this);
        //imCamera=(ImageButton)findViewById(R.id.changeCamera);
        Button btn_accept = (Button)findViewById(R.id.acceptButton);
        Button btn_decline = (Button)findViewById(R.id.declineButton);
        Button btn_add = (Button)findViewById(R.id.addButton);
        Button btn_save = (Button)findViewById(R.id.saveChamada);
        layout = (LinearLayout)findViewById(R.id.frameLayout2);
        cam = (LinearLayout)findViewById(R.id.changeC);
        layout.setVisibility(View.INVISIBLE);
        name = (TextView)findViewById(R.id.nameView);
```

```
id = (TextView)findViewById(R.id.idView);
image = (ImageView)findViewById(R.id.chamadaView);
final String courseCode = getIntent().getStringExtra("courseCode
final Long courseID = getIntent().getLongExtra("courseID", 0);
Path=Environment.getExternalStorageDirectory()+"/EESC-FACE/faces
Facefile=Environment.getExternalStorageDirectory()+"/EESC-FACE/fa
DateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy HH:mm")
Date date = new Date();


lesson.setDate(dateFormat.format(date));
lesson.setCourseId(courseID);

fr=new PersonRecognizer(Path);
if (Facefile!=null) {
    fr.load2(Facefile);
}

List<Student> allStudentsList = DatabaseInteractor.getAllStudents
if (allStudentsList.size()>0){canPredict=true;}
else{canPredict=false;}
allStudentsList.clear();

btn_accept.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Long studentCode = student.getStudentCode();
        presentStudentsList.add(studentCode);
        Photo dbPhoto = new Photo();
        Mat a = new Mat();
        Imgproc.resize(processada,a,new Size (30,40));
        dbPhoto.setImage(DatabaseUtil.bitmapToString(ProcessaImag
        dbPhoto.setStudentId(student.getId());
```

```
            dbPhoto.setLessonId(lesson.getId());
            DatabaseInteractor.savePhoto(ChamadaActivity.this,dbPhoto


            setVisibility(layout, false);
            setVisibility(cam, true);
        }
    });

    btn_decline.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            setVisibility(layout,false);
            setVisibility(cam, true);
        }
    });

    btn_add.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            Intent i = new Intent(ChamadaActivity.this, SelectPeople
            i.putExtra("courseCode", courseCode);
            startActivityForResult(i, 0);


        }
    });

    btn_save.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            Intent intent = new Intent(ChamadaActivity.this, FinalAc

            for(int i = 0;i<presentStudentsList.size();i++)
```

```
                {
                    student = DatabaseInteractor.getStudentByCode(Chamac
                    studentList.add(student);
                    //DatabaseInteractor.addStudentToLesson(ChamadaActiv
                }

                lesson.setStudentList(studentList);
                DatabaseInteractor.saveLesson(ChamadaActivity.this, lesso
                List<Lesson> lessonList = new ArrayList<Lesson>();
                lessonList = DatabaseInteractor.getLessonByCourseId(Cham
                Lesson lastLesson = lessonList.get(lessonList.size()-1);
                intent.putExtra("courseCode", courseCode);
                intent.putExtra("courseId",courseID);
                intent.putExtra("LessonID",lastLesson.getId());
                //// TODO: 5/14/2016 adicionar novas fotos de alunos adic
                startActivity(intent);
                finish();
            }
        });
        imCamera=(ImageButton)findViewById(R.id.changecam2);
imCamera.setOnClickListener(new View.OnClickListener() {

            public void onClick(View v) {

                if (mChooseCamera == frontCam) {
                    mChooseCamera = backCam;
                    mOpenCvCameraView.setCamBack();
                } else {
                    mChooseCamera = frontCam;
                    mOpenCvCameraView.setCamFront();

                }
            }
```

```
        });
    }


    @Override
    public void onPause()
    {
        super.onPause();
        if (mOpenCvCameraView != null)
            mOpenCvCameraView.disableView();
    }

    @Override
    public void onResume()
    {
        super.onResume();
        //OpenCVLoader.initAsync(OpenCVLoader.OPENCV_VERSION_2_4_3, this
        if (!OpenCVLoader.initDebug()) {
            Log.d(TAG, "Internal OpenCV library not found. Using OpenCV
            OpenCVLoader.initAsync(OpenCVLoader.OPENCV_VERSION_2_4_3, thi
        } else {
            Log.d(TAG, "OpenCV library found inside package. Using it!")
            mLoaderCallback.onManagerConnected(LoaderCallbackInterface.SU
        }

    }


    public void onDestroy() {
        super.onDestroy();

        mOpenCvCameraView.disableView();
    }
```

```
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 input
    mRgba = inputFrame.rgba();
    mGray = inputFrame.gray();

    //if (getResources().getConfiguration().orientation==0);

    //Mat equalizedImg = new Mat();



    if (mAbsoluteFaceSize == 0) {
        int height = mGray.rows();
        if (Math.round(height * mRelativeFaceSize) > 0) {
            mAbsoluteFaceSize = Math.round(height * mRelativeFaceSize
        }
        //  mNativeDetector.setMinFaceSize(mAbsoluteFaceSize);
    }


    /*********************************************Detect face – LI

    //Size maxFeatureSize = new Size();// IMPORTANT second option: ne
```

```
//Size minFeatureSize = new Size(20, 20);
MatOfRect faces = new MatOfRect();
mJavaDetector.detectMultiScale(mGray, faces, 1.1f, 2, Objdetect.C
//mJavaDetector.detectMultiScale(mGray, faces, 1.1, 2, 2,

Rect[] faceArray = faces.toArray();


if (faceArray.length>0) {

    Rect r = faceArray[0];
    Mat face = mRgba.submat(r);
    if (face.width() <= 0) {
        Point leftEye = new Point();
    }
    Mat packet = face.clone();

    Point leftEye = new Point();
    Point rightEye = new Point();
    olhos = detectBothEyes(face, eyeCascade1, eyeCascade2, olhos


    leftEye = olhos.leftEye;
    rightEye = olhos.rightEye;
    //adicionar processamento da imagem!!!!!
    if (leftEye.x >= 0 && rightEye.x >= 0)

    {
        Core.rectangle(mRgba, faceArray[0].tl(), faceArray[0].br

        processada = meuProcesso.gray(face);
        Imgproc.equalizeHist(processada, processada);
```

```
        //if(fr.canPredict())
        if (canPredict) {

            int who = fr.predict(processada);

            if (who > 0) {

                student = DatabaseInteractor.getStudentByCode(Ch




                resultName = student.getName();
                photo = student.getPhotoList().get(0);


                setVisibility(layout, true);
                setVisibility(cam, false);
                setText(name, resultName);
                //setText(id, Integer.toString(fr.getConfidence(
                setText(id, Integer.toString((int)student.getStud
                setImage(image, DatabaseUtil.stringToBitmap(photo

            }


        }


    } else {
        setVisibility(layout, false);
        setVisibility(cam, true);
    }
```

```
        }

        return mRgba;

    }

    public static <T> T mostCommon(List<T> list) {
        Map<T, Integer> map = new HashMap<>();

        for (T t : list) {
            Integer val = map.get(t);
            map.put(t, val == null ? 1 : val + 1);
        }

        Map.Entry<T, Integer> max = null;

        for (Map.Entry<T, Integer> e : map.entrySet()) {
            if (max == null || e.getValue() > max.getValue())
                max = e;
        }

        return max.getKey();
    }

    private void setImage(final ImageView image,final Bitmap value){
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
               image.setImageBitmap(value);
            }
        });
    }
    private void setText(final TextView text,final String value){
```

```
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            text.setText(value);
        }
    });
}

private void setVisibility(final LinearLayout layout,final Boolean fl
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            if (flag)
                layout.setVisibility(View.VISIBLE);
            else
                layout.setVisibility(View.GONE);
        }
    });
}


public Bitmap readBitmap(int id, String name) {
    Bitmap a = null;
    File f = Environment.getExternalStorageDirectory();
    File image = new File(f + "/EESC-Face/faces", name + "-" + Integ
    BitmapFactory.Options options = new BitmapFactory.Options();
    //options.inPreferredConfig = Bitmap.Config.ARGB_8888;
    a = BitmapFactory.decodeFile(image.getAbsolutePath(), options);

    return a;
}
```

```
public Rect detectLargestObject( Mat img, CascadeClassifier cascade,
{
    // Only search for just 1 object (the biggest in the image).
    int flags = Objdetect.CASCADE_FIND_BIGGEST_OBJECT; // | CASCADE_I

    // Smallest object size.

    Size minFeatureSize =  new Size(20, 20);
    // How detailed should the search be. Must be larger than 1.0.
    float searchScaleFactor = 1.3f;
    // How much the detections should be filtered out. This should de
    // minNeighbors=2 means lots of good+bad detections, and minNeig
    int minNeighbors = 2;

    // Perform Object or Face Detection, looking for just 1 object (

    List<Rect> objectsa = new ArrayList<>();
    //List<Mat> matList = new ArrayList<>();
    MatOfRect objects = new MatOfRect();


    detectObjectsCustom(img, cascade, objects, scaledWidth, flags, m

    if (objects.toArray().length>0) {
        // Return the only detected object.
        faceRect = objects.toArray()[0];
    }
    else {
        // Return an invalid rect.
        faceRect = new Rect(-1,-1,-1,-1);
    }
    return faceRect;
```

```
}

public void detectObjectsCustom(final Mat img, CascadeClassifier casc
{
    // If the input image is not grayscale, then convert the BGR or B
    //List<Mat> matList = new ArrayList<>();
    Mat gray = new Mat();
    if (img.channels() == 3) {
        Imgproc.cvtColor(img, gray, Imgproc.COLOR_BGR2GRAY);

    }
    else if (img.channels() == 4) {
        Imgproc.cvtColor(img, gray, Imgproc.COLOR_BGRA2GRAY);
    }
    else {
        // Access the input image directly, since it is already gray
        gray = img;
    }

    // Possibly shrink the image, to run much faster.
    Mat inputImg = new Mat();

    float scale = img.cols() / (float)scaledWidth;
    if (img.cols() > scaledWidth) {
        // Shrink the image while keeping the same aspect ratio.

        int scaledHeight = Math.round(img.rows() / scale);

        Imgproc.resize(gray, inputImg, new Size(scaledWidth, scaledH
    }
    else {
        // Access the input image directly, since it is already smal
        inputImg = gray;
```

```
}

// Standardize the brightness and contrast to improve dark images
Mat equalizedImg = new Mat();
Imgproc.equalizeHist(inputImg, equalizedImg);
Bitmap a = ProcessaImagem.MattoBitmap(equalizedImg);
Size maxFeatureSize = new Size (equalizedImg.width(),equalizedImg



cascade.detectMultiScale(equalizedImg, objects, searchScaleFactor
// Detect objects in the small grayscale image.
//cascade.detectMultiScale(equalizedImg, objects, searchScaleFact
//cascade.detectMultiScale(equalizedImg, objects, searchScaleFact

// Enlarge the results if the image was temporarily shrunk before
if (img.cols() > scaledWidth) {
    for (int i = 0; i < objects.toArray().length; i++ ) {
        int x = objects.toArray()[i].x;
        objects.toArray()[i].x =
                objects.toArray()[i].x = Math.round(objects.toAr
        objects.toArray()[i].y = Math.round(objects.toArray()[i]
        objects.toArray()[i].width = Math.round(objects.toArray(
        objects.toArray()[i].height = Math.round(objects.toArray
    }
}

// Make sure the object is completely within the image, in case
for (int i = 0; i < objects.toArray().length; i++ ) {
    if (objects.toArray()[i].x < 0)
        objects.toArray()[i].x = 0;
    if (objects.toArray()[i].y < 0)
        objects.toArray()[i].y = 0;
    if (objects.toArray()[i].x + objects.toArray()[i].width > img
```

```
                objects.toArray()[i].x = img.cols() - objects.toArray()[
            if (objects.toArray()[i].y + objects.toArray()[i].height > im
                objects.toArray()[i].y = img.rows() - objects.toArray()[
        }

        // Return with the detected face rectangles stored in "objects".
    }

    public Olhos detectBothEyes(Mat face, CascadeClassifier eyeCascade1,
    {

        final float EYE_SX = 0.16f;
        final float EYE_SY = 0.26f;
        final float EYE_SW = 0.30f;
        final float EYE_SH = 0.28f;

        int leftX = Math.round(face.cols() * EYE_SX);
        int topY = Math.round(face.rows() * EYE_SY);
        int widthX = Math.round(face.cols() * EYE_SW);
        int heightY = Math.round(face.rows() * EYE_SH);
        int rightX = (int)Math.round(face.cols() * (1.0 - EYE_SX - EYE_SW
// Start of right-eye corner

        Mat topLeftOfFace = new Mat (face,(new Rect(leftX, topY, widthX,
        Mat topRightOfFace = new Mat (face,(new Rect(rightX, topY, widthX
        Rect leftEyeRect = new Rect();
        Rect rightEyeRect = new Rect();

        if (searchedLeftEye!=null)//////////////////////////////////////
            searchedLeftEye = new Rect(leftX, topY, widthX, heightY);
        if (searchedRightEye!=null)/////////////////////////////////////
            searchedRightEye = new Rect(rightX, topY, widthX, heightY);
```

```
leftEyeRect = detectLargestObject(topLeftOfFace, eyeCascade1, lef
rightEyeRect = detectLargestObject(topRightOfFace, eyeCascade1, 

boolean try_twice = false;
if (try_twice) {
    if (leftEyeRect.width <= 0 && !eyeCascade2.empty()) {
        leftEyeRect = detectLargestObject(topLeftOfFace, eyeCasca

    }

    if (rightEyeRect.width <= 0 && !eyeCascade2.empty()) {
        rightEyeRect = detectLargestObject(topRightOfFace, eyeCa

    }
}

if (leftEyeRect.width > 0) {
    leftEyeRect.x += leftX;
    leftEyeRect.y += topY;
    leftEye = new Point(leftEyeRect.x + leftEyeRect.width/2, left
}
else {
    leftEye = new Point(-1, -1);
}

if (rightEyeRect.width > 0) { // Check if the eye was detected.
    rightEyeRect.x += rightX; // Adjust the right-eye rectangle,
    rightEyeRect.y += topY;  // Adjust the right-eye rectangle b
    rightEye = new Point(rightEyeRect.x + rightEyeRect.width/2, 
}
else {
    rightEye = new Point(-1, -1);    // Return an invalid point
}
```

```
        Olhos olhos = new Olhos();
        olhos.leftEye = leftEye;
        olhos.rightEye = rightEye;
        olhos.searchedRightEye = searchedRightEye;
        olhos.searchedLeftEye = searchedLeftEye;
        return olhos;
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        Log.i(TAG, "called onCreateOptionsMenu");
        if (mOpenCvCameraView.numberCameras()>1)
        {
            nBackCam = menu.add("front");
            mFrontCam = menu.add("back");
//          mEigen = menu.add("EigenFaces");
//          mLBPH.setChecked(true);
        }
        else
        {imCamera.setVisibility(View.INVISIBLE);

        }
        //mOpenCvCameraView.setAutofocus();
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        Log.i(TAG, "called onOptionsItemSelected; selected item: " + item

        nBackCam.setChecked(false);
        mFrontCam.setChecked(false);
        if (item == nBackCam)
```

```
        {
            mOpenCvCameraView.setCamFront();
            mChooseCamera=frontCam;
        }
        else if (item==mFrontCam)
        {
            mChooseCamera=backCam;
            mOpenCvCameraView.setCamBack();
        }
        item.setChecked(true);
        return true;
    }

    private void setMinFaceSize(float faceSize) {
        mRelativeFaceSize = faceSize;
        mAbsoluteFaceSize = 0;
    }

}
```