Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Utilização de Filtro Laplaciano na Seleção de Imagens Sem Borramentos em Aplicações Android

Igor Figueira Pinheiro da Silva
Marcos Paulo Batalha Bispo

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientador
Prof. Dr. Marcus Vinicius Lamar

Brasília
2021

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharelado em Ciência da Computação

Coordenador: Prof. Dr. Marcelo Grandi Mandelli

Banca examinadora composta por:

Prof. Dr. Marcus Vinicius Lamar (Orientador) — CIC/UnB
Prof.ª Dr.ª Carla Maria Cavalcante e Chagas Koike — CIC/UnB
Prof. Dr. Flávio de Barros Vidal — CIC/UnB

CIP — Catalogação Internacional na Publicação

Silva, Igor Figueira Pinheiro da.

Utilização de Filtro Laplaciano na Seleção de Imagens Sem Borramentos em Aplicações Android / Igor Figueira Pinheiro da Silva, Marcos Paulo Batalha Bispo. Brasília : UnB, 2021.
83 p. : il. ; 29,5 cm.

Monografia (Graduação) — Universidade de Brasília, Brasília, 2021.

1. reconhecimento facial, 2. android, 3. detecção facial, 4. filtro laplaciano, 5. borramento

CDU 004

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Utilização de Filtro Laplaciano na Seleção de Imagens Sem Borramentos em Aplicações Android

Igor Figueira Pinheiro da Silva
Marcos Paulo Batalha Bispo

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Prof. Dr. Marcus Vinicius Lamar (Orientador)
CIC/UnB

Prof.ª Dr.ª Carla Maria Cavalcante e Chagas Koike
CIC/UnB

Prof. Dr. Flávio de Barros Vidal
CIC/UnB

Prof. Dr. Marcelo Grandi Mandelli
Coordenador do Bacharelado em Ciência da Computação

Brasília, 19 de novembro de 2021

Dedicatória

Dedicamos este trabalho a nossas famílias, amigos e a todas as pessoas que contribuíram para este trabalho.

Agradecimentos

Os autores agradecem ao Prof. Dr. Marcus Vinícius Lamar, nosso orientador, por toda a paciência e dedicação durante toda a duração deste trabalho. Uma característica que sempre se destacou foi a sua disponibilidade para ajudar e a compreensão durante todo o período de pandemia, que trouxe algumas complicações a mais.

Gratos à toda equipe do projeto F2DSys, que nos ofereceu uma grande oportunidade de aprendizado e inovação. Tivemos a oportunidade de trabalhar com uma equipe muito capacitada tecnicamente e que busca trazer inovação ao ambiente de sala de aula.

Agradecemos também à todos os nossos amigos que fizeram parte da nossa vida acadêmica, em especial: Antônio Henrique, Artur Brandão, Felipe Franco, Gabriel Mourão e Tiago Cabral.

Agradecemos à todos os participantes que cederam suas imagens para que pudéssemos realizar nossa pesquisa.

Agradecemos às nossas famílias que sempre nos apoiaram e estiveram ao nosso lado ao longo dessa jornada. Sem vocês não teríamos chegado até aqui.

Resumo

Este trabalho apresenta uma aplicação Android que é uma das partes do sistema desenvolvido para o projeto F2DSys, composto por um conjunto de ferramentas desenvolvidas com o objetivo geral de realizar monitoramento de atividades no escopo dos cursos na Fundação de Apoio à Pesquisa do Distrito Federal (FAPDF). Aqui são discutidas técnicas necessárias para que seja feita captura de imagens de boa qualidade com o intuito de, posteriormente, computar presença em atividades utilizando como meio de autenticação o reconhecimento facial dos estudantes. Além disso, esse trabalho busca resolver um dos problemas atuais da aplicação que é, eventualmente, capturar imagens com níveis elevados de borramento, o que pode impossibilitar o reconhecimento facial. Propõe-se que sejam capturadas cinco imagens nas quais é aplicado um filtro Laplaciano e, posteriormente, é calculada a variância dos pixels dessas imagens. Desta maneira, observou-se que imagens que apresentam uma variância maior, geralmente, são imagens que possuem o menor nível de borramento.

Palavras-chave: reconhecimento facial, android, detecção facial, filtro laplaciano, borramento

Abstract

This paper presents an Android application that is part of the F2DSys project, which is made of a set of tools developed to track activities in the scope of the courses offered by the Fundação de Apoio à Pesquisa do Distrito Federal (FAPDF). Here we discuss techniques needed for capturing high quality images. These images will later be used to register student classroom attendances by using face recognition as an authentication method. Besides that, this paper tries to solve one of the current application's problems, which may capture images with high levels of blur and, later on, such these images may cause errors when performing students face recognition. We propose that five images should be captured. We apply a Laplacian filter in each one and then calculate the variance of the pixels of these images. Thus, we observe that images which presented greater variance generally are images that present lower levels of blur.

Keywords: laplacian filter, android, face detection, face recognition, blur, motion blur

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

API *Application Programming Interface.*

FAPDF Fundação de Apoio à Pesquisa do Distrito Federal.

GPU *Graphics Processing Unit.*

HTTP *Hypertext Transfer Protocol.*

ICAO Organização Internacional da Aviação Civil.

ISO/IEC Organização Internacional de Padronização e pela Comissão Eletrotécnica Internacional.

JSON *JavaScript Object Notation.*

MTCNN *Multi-task Cascaded Convolutional Network.*

RAM *Read only memory.*

REST *Respresentational State Transfer.*

RGB *Red, Green and Blue.*

SaaS *Software as a Service.*

SDK *Standard Development Kit.*

UI *User Interface.*

Capítulo 1

Introdução

Reconhecimento facial é uma tarefa que os seres humanos realizam de maneira instintiva nas suas vidas diárias [8]. O primeiro passo é identificar se o que está sendo visualizado se trata de um rosto ou não e, para isso, são avaliados parâmetros como formato da cabeça ou da face, aparência facial, cor da pele ou a combinação de todas essas características. Após identificar que o que está sendo visualizado se trata de um rosto é possível diferenciar uma face de outra, identificando características como tamanho da cabeça, cor dos olhos, distância entre os olhos, largura do rosto ou a combinação de todos esses parâmetros. Essa tarefa rotineira acaba tendo um grande uso em aplicações computacionais, incluindo segurança e biometria facial.

Modelos computacionais vêm evoluindo de maneira a identificar algumas dessas características e utilizá-las como identificação de uma pessoa, ou seja, prover a possibilidade de saber quem a pessoa é ou quem ela não é baseado em suas características faciais. Mesmo com pesquisas sendo conduzidas desde os anos 60 nessa área, esse ainda é um problema complexo e não resolvido [8]. Temos algoritmos capazes de detectar rostos de maneira eficiente, no entanto, caso esse rosto detectado esteja borrado quando formos compará-lo com outro rosto dessa mesma pessoa com o intuito de realizar o reconhecimento facial os parâmetros podem ser extremamente distintos. A Figura 1.1a é um exemplo de imagem capturada sem borramento, enquanto a Figura 1.1b é um exemplo de imagem capturada com borramento devido à movimentação da câmera no momento da captura.

(a) Exemplo de imagem sem borramento (b) Exemplo de imagem com borramento

Figura 1.1: Comparação de imagem capturada sem borramento e imagem capturada com borramento.

O reconhecimento facial é um recurso importante para a automatização de processos como a presença em salas de aula ou em cursos de capacitação. A frequência em uma atividade é imprescindível para que o aluno possa concluir determinado curso e também pode ser utilizada como um registro que prova que o aluno esteve presente em determinado local em determinado horário. Um projeto que visa trazer a biometria facial para o âmbito dos cursos da Fundação de Apoio à Pesquisa do Distrito Federal (FAPDF) é o F2DSys.

Este trabalho explora a aplicação Android desenvolvida, responsável pela captura e pré-processamento de imagens que são enviadas posteriormente para o módulo de reconhecimento facial.

1.1 Problema

Este trabalho parte do seguinte problema de pesquisa: dado um conjunto de imagens contendo rostos identificados por um algoritmo de detecção facial, como podemos medir níveis de borramento e, a partir dessa medida, como escolher a imagem que possua a menor probabilidade de estar borrada.

1.2 Justificativa

A importância deste trabalho se reflete em melhorar a etapa de seleção de uma imagem de boa qualidade durante o pré-processamento de imagens em sistemas que realizam reconhecimento facial. Atualmente, há diversas aplicações que realizam esse mesmo trabalho. No entanto, na maioria das vezes todo o processo é realizado em computadores de mesa, que geralmente possuem um poder de processamento maior do que o de celulares de usuários comuns. Uma solução que leve em consideração o tamanho da aplicação final e também a possibilidade do aplicativo rodar em celulares de menor poder computacional é de grande importância quando se desenvolve um produto para um grande número de usuários.

1.3 Objetivos

1.3.1 Objetivo Geral

O objetivo geral deste trabalho é apresentar a aplicação Android que foi desenvolvida até o presente momento, incluindo as técnicas e tecnologias utilizadas, a arquitetura da aplicação e suas limitações. Também serão discutidas possíveis melhorias que podem ser introduzidas no fluxo da aplicação, levando em consideração que a aplicação deve funcionar em aparelhos de ponta mas também em aparelhos mais antigos, com poder computacional menor.

1.3.2 Objetivos específicos

Para atingir o objetivo geral, temos os seguintes objetivos específicos:

- Estudar técnicas de análise de qualidade de imagem;
- Avaliar problemas e limitações de soluções de detecção facial em dispositivos móveis;
- Propor uma solução que minimize as chances de que uma imagem com borramento seja selecionada;
- Avaliar a eficiência da solução proposta quantificando valores de borramento e fazendo testes de performance em celulares diferentes.

1.4 Organização do Trabalho

Capítulo 2 - Fundamentação Teórica. Neste capítulo são abordados os conceitos teóricos para o entendimento do trabalho.

Capítulo 3 - Sistema Proposto. Este capítulo descreve a arquitetura do sistema e as diferentes versões que foram testadas com foco no fluxo que envolve o pré-processamento de imagens.

Capítulo 4 - Resultados. Neste capítulo são apresentados os resultados obtidos nos experimentos e é feita análise da solução proposta.

Capítulo 5 - Conclusão. Este capítulo sumariza a metodologia, os resultados e a análise, apresentado as conclusões obtidas.

Capítulo 2

Fundamentação Teórica

Este capítulo apresenta os conceitos fundamentais para o entendimento do trabalho, como algoritmos para detecção de faces, frameworks utilizadas para o auxílio no desenvolvimento do aplicativo e o padrão de imagem que deve ser seguido.

2.1 F2DSys

O projeto F2DSys é realizado em parceria com a FAPDF e visa modernizar o fluxo de cadastro e chamadas dos alunos em seus cursos de capacitação. A aferição de presença em sala de aula deixa de ser realizada através de chamada oral e assinatura em punho e passa a ser feita utilizando reconhecimento facial, que autentica o aluno a partir de uma foto do seu rosto. Para identificar a atividade em questão é utilizado um de um *QRCode*. Assim como em uma sala de aula física, existem diversos papéis exercidos por pessoas que fazem com que as coisas funcionem, no sistema F2DSys isso não é diferente, há diferentes tipos de usuário com acessos diferenciados para facilitar e manter uma hierarquia no sistema. Esses usuários são:

Administrador: é responsável por manter todo o sistema em ordem, e tem o poder de cadastrar, alterar e excluir dados de todo o sistema, como usuários, informações cadastrais gerais, cursos, ocorrências em salas de aula, dentre outros.

Gestor: pode cadastrar, editar e deletar novos cursos e atividades. Também é responsável por definir o cronograma e gerar os *QRCodes* que serão utilizados na chamada de uma atividade. Além disso, o Gestor pode gerar relatórios automáticos no sistema com dados relativos aos alunos e seus respectivos cursos.

Supervisor: tem acesso a toda informação relativa às atividades que ministra. É capaz de acessar um *QRCode* e fornecer ele aos alunos para que esses possam registrar sua

presença em sala de aula. Também pode gerar relatórios do sistema referente aos cursos que ministra.

Aluno: pode acessar o aplicativo em sala de aula para ler o *QRCode* fornecido pelo professor e registrar sua presença em determinada atividade.

O sistema desenvolvido no projeto é dividido em diversos módulos. Os módulos que servem como interface com os usuários são: o sistema web e as aplicações Android e iOS. Há também os sistemas que permitem interação com o banco de dados e realizam todo o processamento necessário para computar a presença em sala de aula e outras operações essenciais do sistema, estão são o módulo de reconhecimento facial e o Back-end.

2.1.1 Aplicativos Android e iOS

Os aplicativos são utilizados por alunos e professores que desejem confirmar seu cadastro e realizar a chamada. Os alunos também podem checar a situação atual de suas atividades como número de presenças registradas até o momento e local e horário em que a presença pode ser registrada.

As aplicações mobile em Android e iOS têm um design único e tudo que é possível realizar em um também pode ser feito no outro, no entanto, para atender melhor as especificidades dos diferentes sistemas operacionais foi feita uma base de código para cada uma dessas aplicações. Neste trabalho serão abordadas apenas as técnicas e algoritmos utilizados na aplicação Android.

2.1.2 Sistema web

O sistema web é um SaaS que provê uma interface para supervisore, gestores e administradores. É possível criar, editar, ler e deletar todos os dados disponíveis no banco de dados. Logicamente, só é possível realizar operações em dados sensíveis caso o papel sendo utilizado seja o de administrador. O principal uso para professores é buscar o *QRCode* para a atividade atual para que os alunos possam lê-lo e registrar presença. Também é possível verificar quais alunos registraram presenças e emitir relatórios relativos aos dados registrados no sistema.

2.1.3 Módulo de reconhecimento facial

O módulo de reconhecimento facial é responsável pela verificação de imagens do lado do servidor. Rotinas que são muito pesadas para serem tratadas por celulares, tendo como base que nem todos os usuários possuem a última geração de dispositivos móveis,

são executadas nesse módulo. No cadastro dos alunos diversos parâmetros são checados, como nível de luminosidade da foto, ruído do fundo, posicionamento do rosto, dentre outros. Durante o processo de chamada o módulo de reconhecimento facial é responsável por comparar o rosto enviado durante a chamada com o rosto registrado no cadastro para aferir se o aluno que está realizando a chamada realmente é quem ele diz ser. Em caso de negativa é retornado o motivo da falha no reconhecimento. Esta informação pode ser acessada pelo professor ou outro tipo de usuário com mais privilégios que deseje consultar essa informação.

2.1.4 Back-end

O sistema back-end provê uma interface com o banco de dados que é utilizada por todos os outros módulos do sistema. Essa interface é implementada através de uma API REST. Dados que estão registrados no banco de dados são processados para que virem informação útil antes de serem acessados por outras interfaces e são retornados no formato JSON. Toda comunicação pode ser feita com esse módulo através de requisições HTTP e as informações que são consultadas ou dados que estejam sendo gravados passam por um tratamento antes de serem inseridos no banco ou retornados ao usuário.

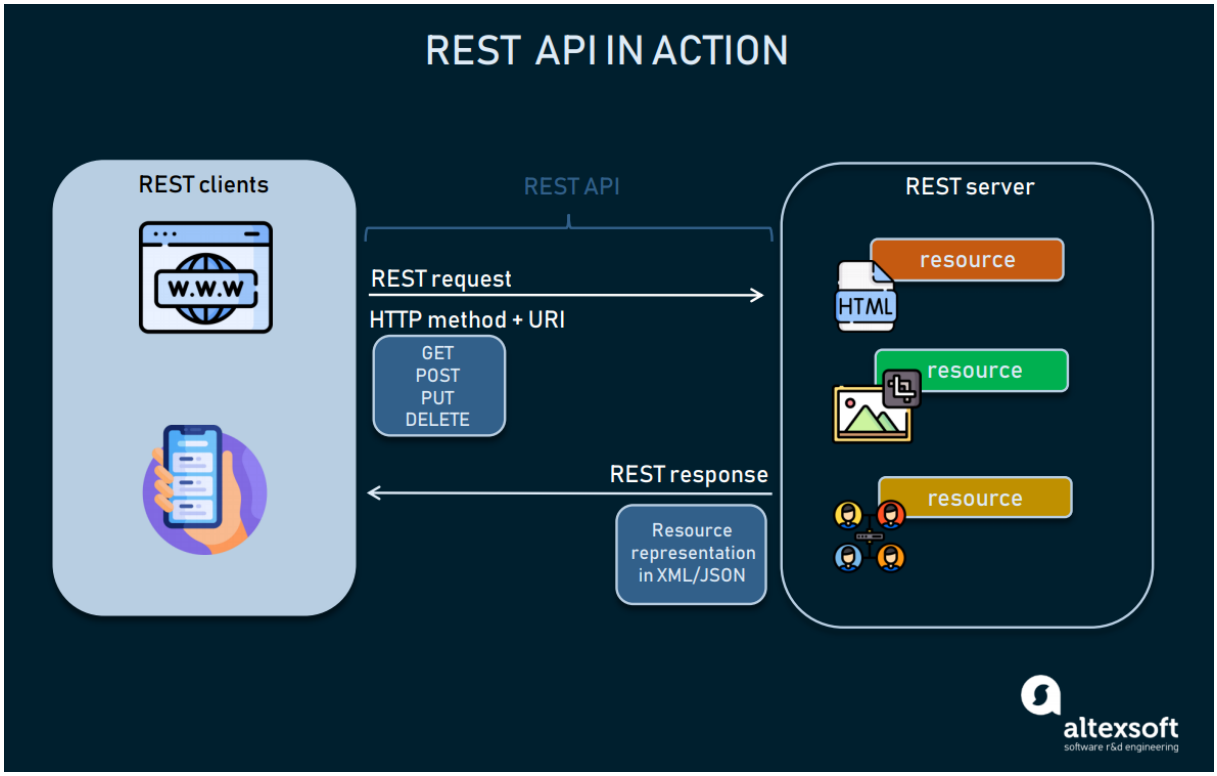


Figura 2.1: Exemplo de API REST (Fonte [1]).

2.2 Detecção facial

Detecção facial pode ser classificado como um caso específico da detecção de classes de objetos. Cada classe de objetos tem suas características específicas que ajudam na classificação de uma classe, como por exemplo, círculos são redondos enquanto quadrados possuem suas extremidades com ângulos de 90 graus. Na detecção de objetos, a tarefa é encontrar a localização e o tamanho de todos os objetos em uma imagem que pertencem a uma certa classe. Como por exemplo, pedestres, carros, semáforos e bicicletas.

O objetivo da detecção facial é localizar e classificar todas as faces humanas independentes de sua posição, escala, orientação, pose e condições de iluminação [9]. Este é um problema um tanto complexo pois rostos humanos não são rígidos e possuem um alto grau de variabilidade em tamanho, formato, cor e textura. A detecção da face geralmente é a primeira etapa em processos que visam realizar o tratamento ou reconhecimento da face humana.

O modelo utilizado neste trabalho para a detecção de face é o *Multi-task Cascaded Convolutional Network* (MTCNN). Este modelo proposto por *Zhang et al.* [2] possui três redes convolucionais (P-Net, R-Net e O-Net). O modelo é capaz de detectar faces de maneira consistente e ainda assim mantém uma performance necessária para utilização em tempo real. De maneira resumida, seu funcionamento consiste em três estágios:

Estágio 1:

1. Recebe a imagem
2. Cria cópias multi-escaladas da imagem
3. Alimenta P-Net com as imagens em escala
4. Junta o output de P-Net
5. Apaga as *bounding boxes* com baixa taxa de confiança
6. Converte coordenadas de kernel 12×12 em coordenadas de "imagem sem escala"
7. Supressão não máxima para kernels em cada imagem dimensionada
8. Supressão não máxima para todos os kernels
9. Converte as coordenadas da *bounding box* em coordenadas de "imagem sem escala"
10. Remodela as *bounding boxes* para torná-las quadradas

Estágio 2

1. Preenche *out-of-bound boxes*

2. Alimenta R-Net com as imagens em escala
3. Junta o output de R-Net
4. Apaga as *bounding boxes* com baixa taxa de confiança
5. Supressão não máxima para todas as *boxes*
6. Converte as coordenadas da *bounding box* em coordenadas de "imagem sem escala"
7. Remodela as *bounding boxes* para torná-las quadradas

Estágio 3

1. Preenche *out-of-bound boxes*
2. Alimenta O-Net com as imagens em escala
3. Junta o output de O-Net
4. Apaga as *bounding boxes* com baixa taxa de confiança
5. Converte as coordenadas da *bounding box* e do ponto de referência facial em coordenadas de "imagem não dimensionada"
6. Supressão não máxima para todas as *boxes*

Como retorno, temos um retângulo que delimita o tamanho da face, um ponto para o olho esquerdo, um ponto para o olho direito, um ponto no nariz e mais dois pontos que demarcam as extremidades da boca. As três etapas da convolução podem ser vistas na Figura 2.2.

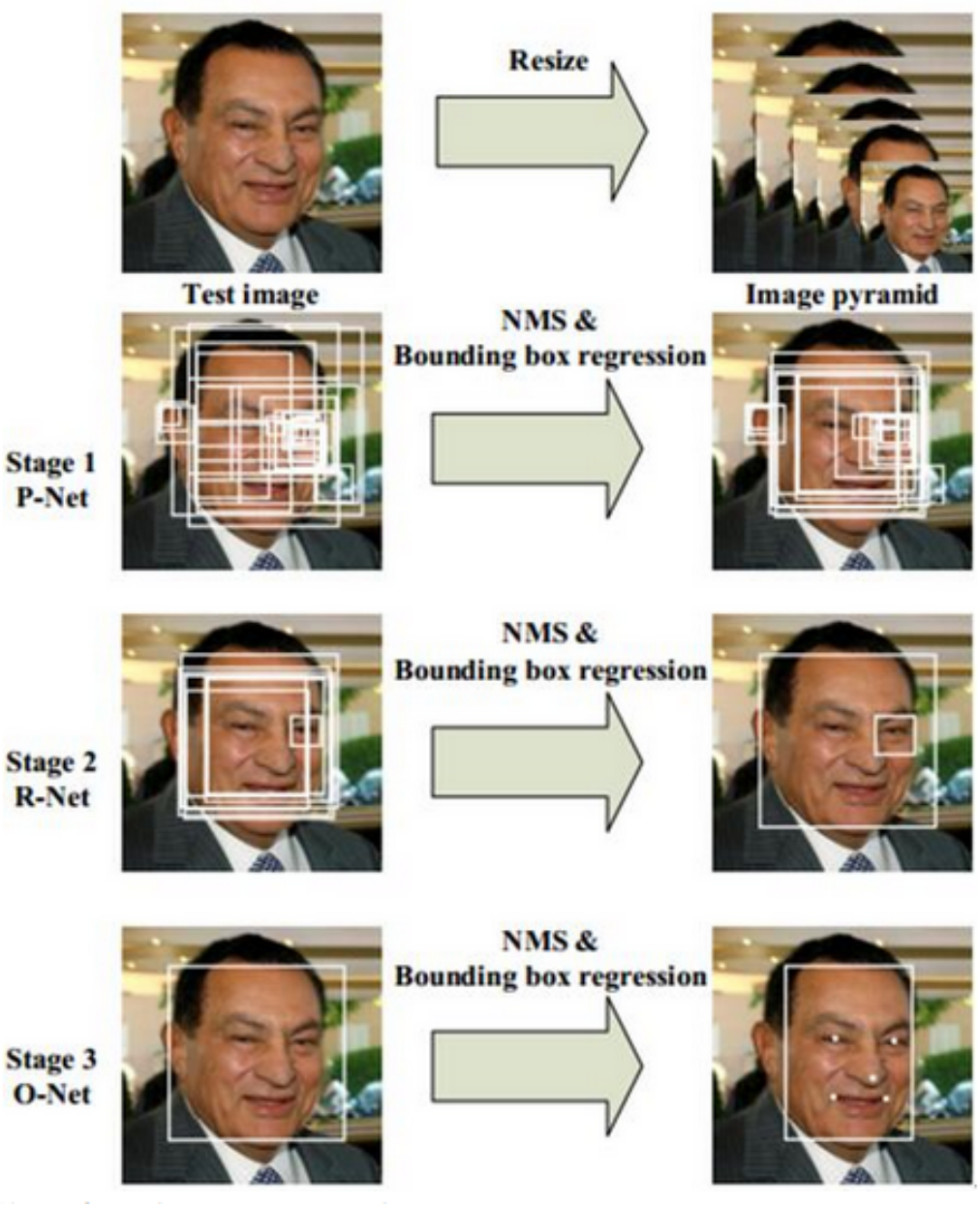


Figura 2.2: As etapas do algoritmo MTCNN (Fonte [2]).

2.3 Padrões ICAO

O Organização Internacional da Aviação Civil (ICAO) [10] define padrões para dados de biometria, mais especificamente biometria facial. Esses padrões definidos pela ICAO utilizam como base a Organização Internacional de Padronização e pela Comissão Eletrotécnica Internacional (ISO/IEC) 19794-5. Os padrões definem várias categorias como distância para a câmera, posição da cabeça, iluminação entre outros fatores. No projeto

F2DSys foram utilizados algum desses padrões para melhorar a acurácia do reconhecimento facial que é realizado pelo back-end.

Figura 2.3: Exemplo de fotos que seguem e que não seguem o padrão ICAO (Fonte [3]).

Como pode ser visto na Figura 2.3 há algumas regras que devem ser seguidas para que a foto esteja dentro do padrão estabelecido. Na primeira linha, o rosto não pode estar enquadrado nem muito distante nem muito próximo, ou seja, deve estar em uma distância aceitável da câmera. Na segunda linha é possível ver que existe um padrão mínimo de qualidade exigido. A primeira foto apresenta borramento, que pode ser ocasionado por algum tipo de movimentação durante a captura da foto ou por sujeira nas lentes da câmera. Já a segunda foto apresenta indícios de desgaste e dobraduras, que também não são aceitos por deteriorarem a qualidade da imagem.

2.4 Espaço de cores

Cor provê informações importantes que dizem respeito a detecção e reconhecimento de objetos. Diferentes espaços de cores possuem diferentes características e são adequados para diferentes tarefas visuais [11].

2.4.1 YUV

Os padrões YUV são modelos que codificam uma imagem ou vídeo levando em consideração a visão humana [12]. Os padrões de cores YUV fazem parte da família de espaços de cores YCbCr, que separa claridade (*luma*) da cor (*chroma*) [13]. Ao guardar *luma* e *chroma* separadamente, algoritmos de compressão podem conservar mais luminância, já que humanos são mais sensíveis a esse fator e descartar parte da cor sem que haja um grande impacto negativo naquilo que efetivamente está sendo visualizado. A claridade é definida pelo componente Y e a cor é definida pelos componentes U e V. Na Figura 2.4 o valor de Y é fixado em 0.5 e é possível ver a mudança na coloração do pixel YUV ao variar o valor de U (eixo X) e o valor de V (eixo Y).

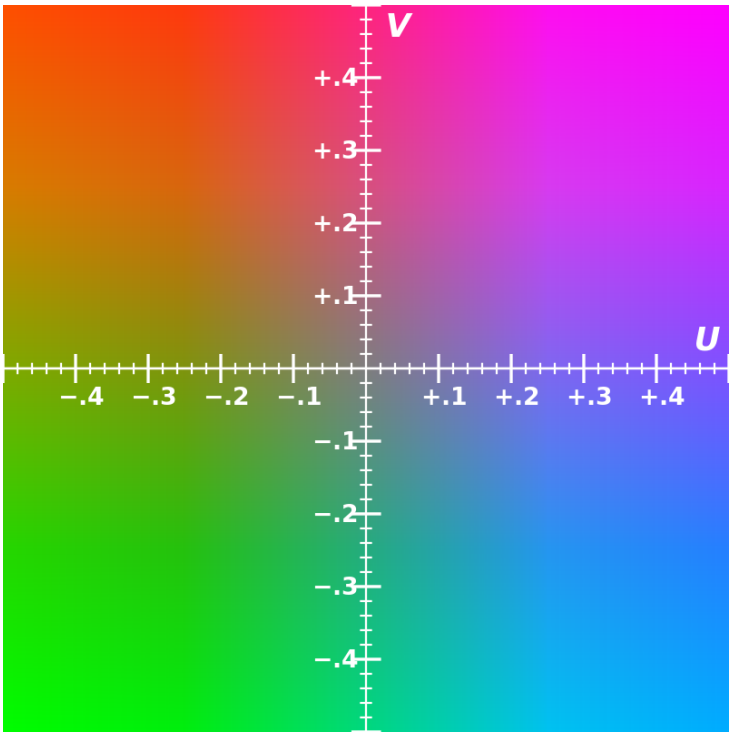


Figura 2.4: Exemplo do plano de cores U-V, valor de Y fixado em 0.5. (Fonte [4]).

2.4.2 RGB

O padrão de cor RGB é um modelo de cores aditivo [14], onde os componentes de luz vermelho(R), verde (G) e azul(B) são adicionados de diferentes maneiras para produzir uma grande variedade de cores. O maior propósito do padrão de cor RGB é para representar imagens em sistemas eletrônicos. Na Figura 2.5 é possível observar três conjuntos de cores que representam os componentes RGB em sua representatividade máxima. Separadamente é possível obter apenas as cores vermelha, verde e azul. Ao realizar a adição

de duas dessas cores é possível obter as cores amarela, ciano e magenta. E ao somar as três cores é possível obter a cor branca.

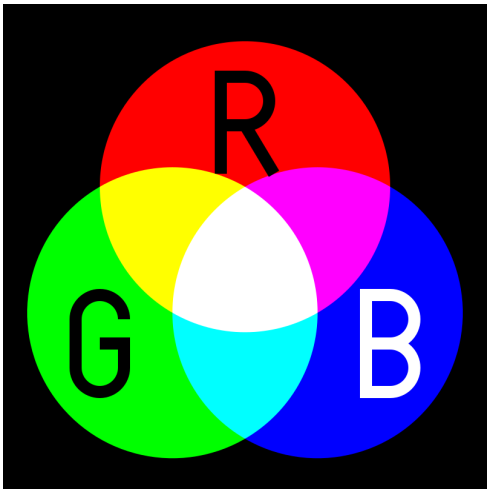


Figura 2.5: Exemplo de geração de cores aditivamente. (Fonte [5]).

2.4.3 Conversão de cores de YUV_420_888 para RGB

Em aplicações Flutter o formato padrão para exportação de frames em um *streaming* de imagens é o padrão YUV_420_888. No entanto, para que a imagem possa ser analisada pelo algoritmo MTCNN na aplicação Android é necessário que o formato seja RGB. Para isso é necessário que seja feita uma conversão entre os dois espaços de cores.

Antes deve ser realizada a conversão entre YUV_420_888 para YUV_420_NV21, embora os dois sejam padrão YUV, há uma diferença na maneira que a informação da crominância é guardada, como pode ser visto nas Figuras 2.6 e 2.7.

Figura 2.6: Formato de cores YUV_420_888 (Fonte [6]).

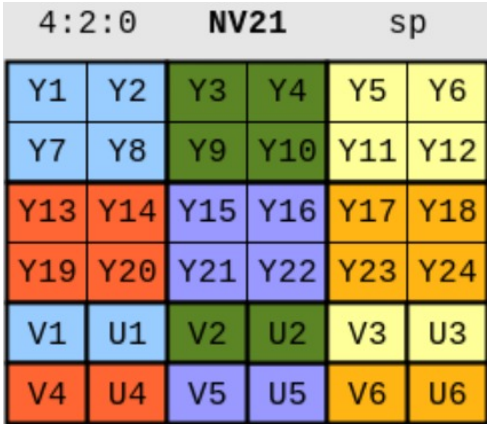


Figura 2.7: Formato de cores YUV_420_NV21 (Fonte [6]).

Sinais YUV são tipicamente criados a partir de uma fonte RGB. Valores com peso de R, G e B são somados para produzir Y, U e V são computados escalando os valores de Y, B e R Essa transformação pode ser feita utilizando a equação Equação 2.1:

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.14713 & -0.28886 & 0.436 \\ 0.615 & -0.51499 & -0.10001 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{2.1}$$

Para produzirmos uma matriz RGB a partir de uma matriz YUV basta aplicar essa relação de maneira invertida [4] e, dessa maneira, realizar a conversão de YUV_420_NV21 para RGB. Isso pode ser visto a partir da Equação 2.2:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.13983 \\ 1 & -0.39465 & -0.58060 \\ 1 & 2.03211 & 0 \end{bmatrix} \begin{bmatrix} Y \\ U \\ V \end{bmatrix} \tag{2.2}$$

2.5 Detecção de bordas

Detecção de borda inclui uma variedade de métodos matemáticos que visam identificar bordas e curvas em uma imagem digital onde a iluminação de uma imagem muda drasticamente, ou seja, detecta e carateriza descontinuidades em um domínio de imagem [15]. A detecção de bordas pode ser usada para identificar o nível de borramento de uma imagem considerando que imagens mais borradas possuem bordas menos acentuadas [16]. Uma das maneiras de se implementar a detecção de bordas é utilizando o filtro Laplaciano.

O filtro Laplaciano é utilizado para computar as segundas derivadas de uma imagem, medindo o quanto a primeira derivada muda. Isso determina se a mudança em um pixel

adjacente é de uma borda ou de uma progressão contínua [17]. O operador Laplaciano $\nabla^2 f$ é dado no conjunto $\mathbb{R}^2$ pela Equação 2.3

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}. \tag{2.3}$$

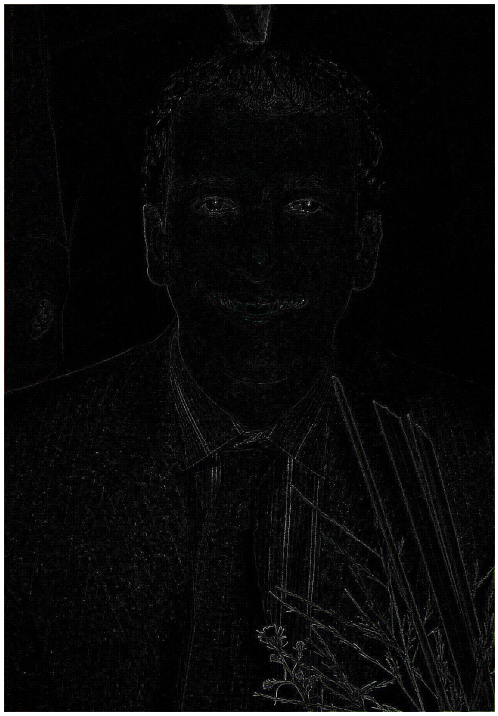
Uma vez que a imagem de entrada é representada por um conjunto discreto de pixels, é necessário encontrar um *kernel* convolucional capaz de aproximar a segunda derivada na definição da Laplaciana. Dois kernels comumente utilizados de dimensão 3×3 são apresentados nas Equações 2.4 e 2.5.

$$Kernel_1 = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \tag{2.4}$$

$$Kernel_2 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \tag{2.5}$$

Com isso podemos receber uma matriz RGB como entrada, exemplificado na Figura 2.8a e ao aplicar o filtro Laplaciano geramos uma nova matriz RGB que pode ser visualizada na Figura 2.8b.

(a) Exemplo de imagem de entrada



(b) Exemplo de imagem de saída

Figura 2.8: Exemplo de aplicação de filtro Laplaciano em uma imagem

2.6 Variância

Variância é a medida de como os dados diferem da média, ou seja, o quanto um conjunto de dados está espalhado em relação à sua média [18]. Quanto maior o valor da variância, mais os dados estão distantes da média e, quanto menor o valor da variância mais os dados estão concentrados próximo ao valor da média. O cálculo da variância é dado pela Equação 2.6

$$\sigma^2 = \frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n} \tag{2.6}$$

onde x_i são os dados, n a quantidade de dados e μ a média.

A variância é importante nesse trabalho porque a variância do Laplaciano é um método robusto de estimar o borramento de uma imagem [16].

2.7 Frameworks e Ferramentas

Para o desenvolvimento da aplicação foi utilizado o kit de desenvolvimento de Software para Android, mais conhecido como Android SDK, e a *framework* Flutter. Ambas facilitam a implementação das telas e o controle de recursos de hardware. Além disso, foram utilizadas ferramentas que auxiliem a implementação dos algoritmos de visão computacional utilizados neste trabalho. O *TensorFlow* foi utilizado na implementação da MTCNN no dispositivo Android e Open Source Computer Vision Library foi utilizado na realização dos experimentos em Desktop.

2.7.1 Kit de Desenvolvimento de Software para Android

O Kit de Desenvolvimento de Software Android, mais conhecido como Android SDK inclui ferramentas de desenvolvimento que servem como base para o desenvolvimento de qualquer aplicação Android nativa. O kit de desenvolvimento é compatível com qualquer versão moderna do Linux, Mac OS X 10.14 ou mais recente e Windows 8 ou mais recente. Nesse kit são incluídos:

- Bibliotecas;
- Debugger;
- Emulador;
- Documentação relevante para APIs Android;

- Exemplos de códigos para aplicações simples;
- Tutoriais para o sistema operacional Android.

A Android SDK é mantida pela Google e pode ser encontrado nos projetos da *Google Open Source* [19]. A versão mínima da SDK utilizada neste trabalho é a 24, que permite compatibilidade com versões do Android 7.0 ou superior.

2.7.2 Flutter

Flutter é uma *User Interface* (UI) *Framework* criada pela Google que foi lançada em maio de 2017. Ela permite que aplicativos *mobile* nativos sejam criados com uma única fonte de código. Isso significa que uma única linguagem de programação e uma única fonte de código podem gerar dois aplicativos (para iOS e Android). Flutter consiste de duas partes importantes:

- Uma SDK (*Software Development Kit*): Uma coleção de ferramentas que ajuda no desenvolvimento de aplicações. Isso inclui ferramentas para compilar o código em código de máquina nativo (para iOS e Android).
- Uma Framework (Biblioteca UI baseada em *widgets*): Uma coleção de elementos de UI reutilizáveis (botões, *inputs* de texto, etc) que podem ser personalizados conforme seja necessário.

Mesmo provendo a possibilidade de gerar uma única fonte de código, algumas aplicações exigem tratamento diferenciado dependendo da plataforma. Para isso, é possível utilizar *Flutter Platform Channels* [7], que possibilitam a criação de um canal responsável por trocar mensagens assíncronas entre a aplicação Flutter e os hosts nativos Android ou iOS. Esse processo é feito de maneira assícrona para não comprometer o que é mostrado na tela para o usuário, e dessa maneira a interface continua responsiva. Esse processo é ilustrado na Figura 2.9

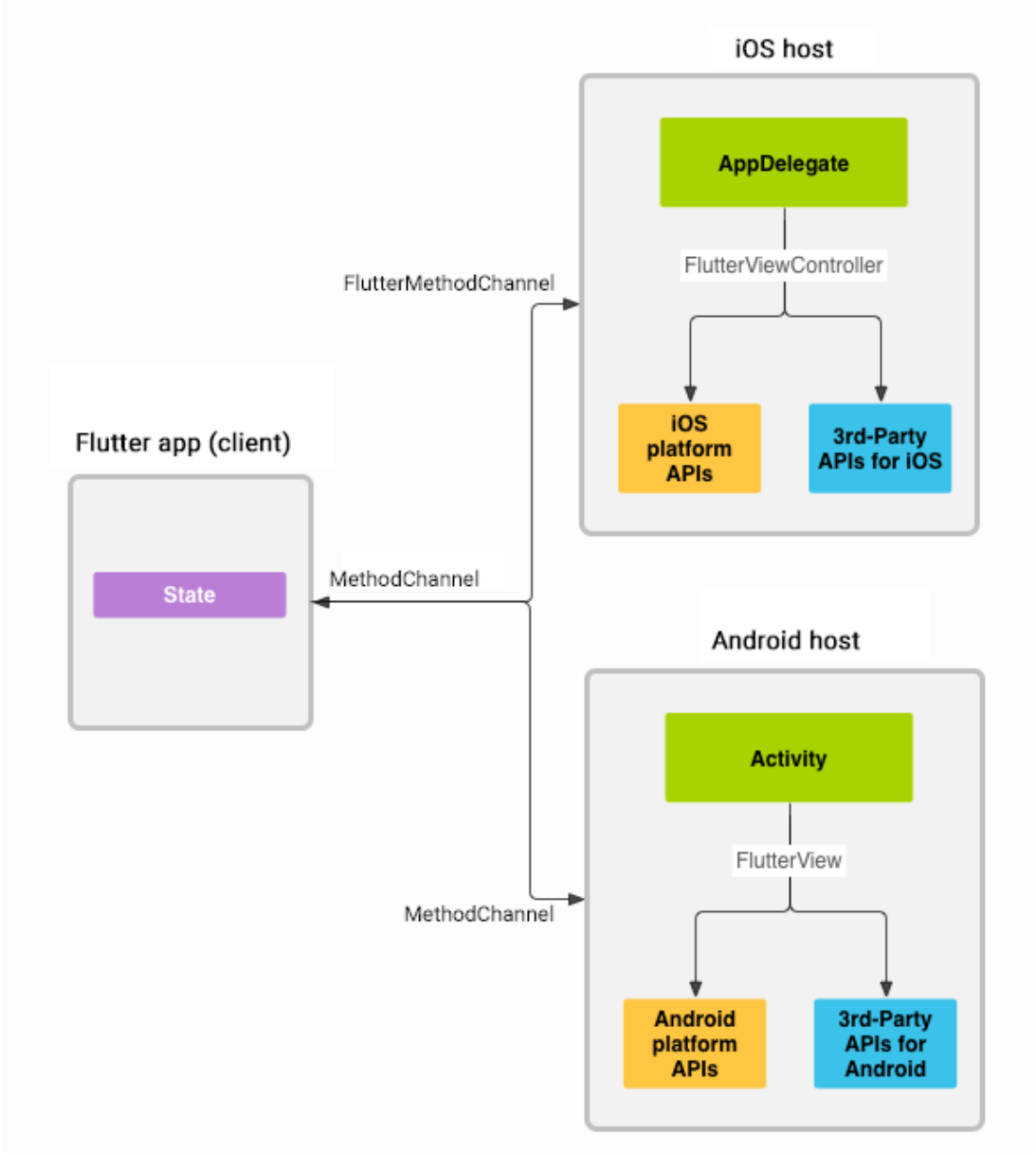


Figura 2.9: Mensagens e respostas são passadadas entre as interfaces Flutter, Android e iOS (Fonte: [7]).

2.7.3 TensorFlow

TensorFlow é uma plataforma de ponta a ponta de código aberto para aprendizado de máquina. Foi desenvolvida pela *Google Brain Team* [20] em 2015 para uso interno da Google em pesquisa e desenvolvimento e, atualmente, pode ser encontrado nos projetos da *Google Open Source* [19]. A plataforma possui uma lista diversa de ferramentas, bibliotecas e outros recursos da comunidade que permitem o desenvolvimento de aplicações poderosas [21]. Neste projeto é utilizada uma versão otimizada do *TensorFlow* para

dispositivos móveis, denominada *TensorFlow Lite*. Essa biblioteca segue 5 princípios chave:

- Latência: não há ida e volta para o servidor;
- Privacidade: Nenhum dado pessoal sai do dispotivo;
- Conectividade: conexão com a Internet não é necessária;
- Tamanho: tamanho do modelo e dos arquivos binários são reduzidos;
- Consumo de energia: inferência eficiente e não necessitar de conexões de rede, que tira a obrigatoriedade de manter o dispositivo conectado aguardando por processamento.

2.7.4 Open Souce Computer Vision Library

Open Source Computer Vision Library, mais conhecida como OpenCV é uma biblioteca de de funções focadas em aplicações que utilizam recursos de visão computacional em tempo real [22]. A biblioteca é *cross-platform* e livre para uso dentro da licença de código aberto Apache 2. Desde 2011 o OpenCV também suporta aceleração de GPU para aplicações de tempo real. As áreas de aplicação do OpenCV relevantes para esse trabalho incluem captação de movimento, detecção de objetos, segmentação e reconhecimento e rastreamento de movimentos.

A linguagem de programação primária do OpenCV é C++. Mesmo sendo escrita em C++ há diversas interfaces com outras linguagens de programação como Python e Java, que foram utilizadas nesse projeto. OpenCV suporta os seguintes sistemas operacionais desktop: como Windows, Linux, macOS, FreeBSD, NetBSD e OpenBSD. E no caso de sistemas operacionais *mobile* há suporte para Android, iOS, Maemo e BlackBerry.

Capítulo 3

Sistema Proposto

Neste capítulo é descrito o funcionamento geral da aplicação Android. Também será discutido como a aplicação tenta solucionar o problema de detectar um rosto com qualidade suficientemente boa para minimizar eventuais problemas ao realizar o reconhecimento facial no módulo de reconhecimento facial. As ações que um usuário pode tomar bem como a política de armazenamento de estado apresentados. Também é dada uma breve introdução à nova técnica introduzida que visa identificar imagens com menos borramento.

3.1 Arquitetura do sistema

Como o produto, trata-se de um aplicativo para telefones celulares (*smartphones*). Uma parte importante do seu funcionamento é a interface com o usuário. O aplicativo é divido em diversas telas de modo a facilitar a navegação do usuário. Essa lógica envolve a construção de *Widgets* [23] e um bom gerenciamento de estado da aplicação.

3.1.1 *Widgets*

Widgets são estruturas básicas para a construção das telas da aplicação. Basicamente, *Widgets* descrevem como uma tela deve ser renderizada. Isso inclui instruções para construção da tela, interação com o usuário e gerenciamento do estado daquela tela. *Widgets* também podem renderizar outras *Widgets* dentro de si, o que permite com que partes de uma tela possam ser reaproveitadas em outras telas.

3.1.2 Gerenciamento de estado

Para gerenciar o estado da aplicação foi utilizado o pacote *provider* [24]. Esse pacote permite com que o estado de uma *Widget* seja propagado para todas as outras *Widgets* que estejam contidas dentro dela. Informações como a sessão do usuário ativo e outros

dados que são buscados através de requisições ao *back-end* podem ser compartilhados nos módulos da aplicação onde eles são necessários sem que sejam feitas novas requisições a todo momento.

3.2 Captura de imagens

Para realizar a captura de imagens, foi utilizado o pacote *camera* [25]. Com isso, é possível detectar quais câmeras estão disponíveis para uso no celular e criar um controlador de câmera. Ao criar um controlador de câmera selecionamos qual câmera do celular será utilizada (frontal ou traseira), é feita a escolha da resolução de captura da imagem, é habilitado ou não a opção de áudio para o caso de gravação de vídeo e, por último, é selecionado o formato de saída da imagem que será processada.

Os formatos de imagem disponíveis para Android são YUV_420_888 e JPEG. Selecionar o formato JPEG facilita o processamento das imagens para a detecção da face, porém foi verificado que alguns aparelhos apresentaram problemas em suas câmeras quando esse foi formato foi selecionado. Por conta disso, o formato YUV_420_888 foi selecionado para abranger o maior número modelos de celulares possível.

Como o intuito da aplicação é realizar a captura da foto de maneira automática sem que o usuário tenha que realizar nenhuma ação além de posicionar a câmera, foi utilizado o método *startImageStream*, que permite acesso ao *frame* capturado em tempo real. Os três planos Y, U e V são mapeados para um *array* de bytes e em seguida são passados para o módulo de detecção facial juntamente com a orientação do sensor da câmera, a largura e a altura da imagem capturada. Essa etapa pode ser vista no Código-fonte I.1.

3.3 Pré-processamento de imagens

Quando uma imagem é capturada, esta deve passar por diversas verificações para assegurar que haja um rosto na imagem. Também é um requisito que esse rosto siga um padrão de qualidade aceitável de acordo com o padrão ICAO para que a imagem possa ser enviada para cadastro ou para registro de chamada. Para isso, é necessário passar por diversas etapas que são realizadas dentro da aplicação Android nativa.

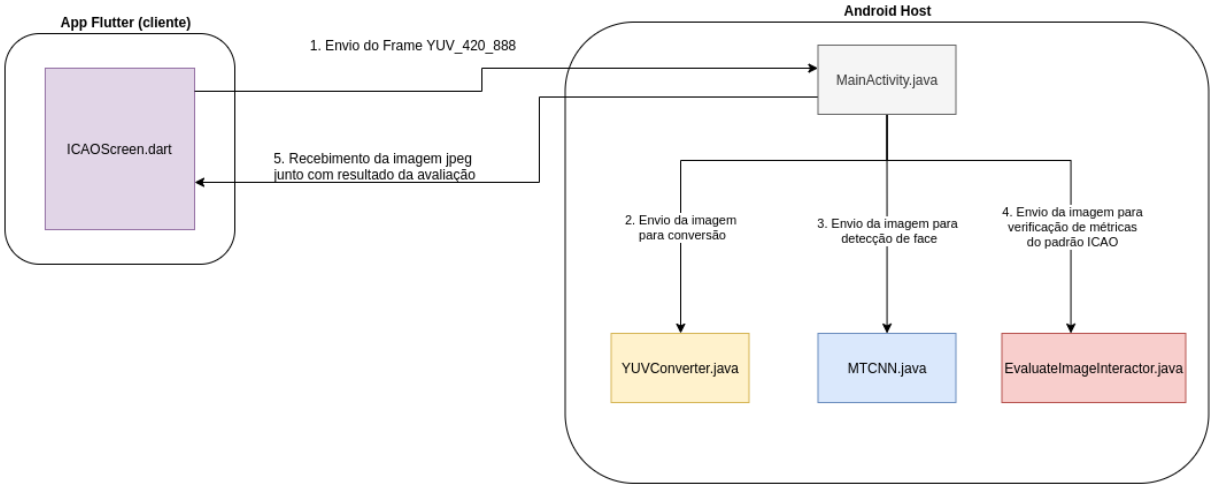


Figura 3.1: Diagrama que ilustra a comunicação entre o módulo Flutter e o Android *host* nativo.

A Figura 3.1 ilustra o processo geral do pré-processamento. Inicialmente é feita a captura de imagens dentro do módulo Flutter *Flutter*. Após isso, é feita uma troca de mensagens com o Android *host*, onde é passado *frame* capturado pela câmera. Esse frame passa pelas três etapas de pré-processament, que são: conversão de cores, detecção da face e verificação do padrão ICAO.

3.3.1 Etapa 1: Conversão de cores

Cada *frame* capturado é processado diretamente em código Java nativo. Isso é feito utilizando *Flutter Platform Channels*. Inicialmente é feita a chamada da classe *YUVConverter*, que fará a conversão de YUV_420_888 para YUV_420_NV21. Após isso, é feita outra conversão, desta vez de YUV_420_NV21 para RGB.

No Código-fonte I.2 está definida a função YUVtoNV21, que recebe como entrada uma lista de bytes representando um frame YUV_420_888 e tem como retorno uma lista de bytes representando um frame YUV_420_NV21. Também está definida a função NV21toJPEG que comprime o frame YUV_420_NV21 para JPEG, que posteriormente é utilizado para a construção de uma matriz RGB.

3.3.2 Etapa 2: Detecção da face

Após a conversão de cores temos como saída uma imagem RGB. Essa imagem passa a ser a entrada no módulo de detecção de face que é realizada utilizando a MTCNN. Nessa etapa é verificado se existe uma face na imagem que está sendo processada. Caso a face

seja encontrada, são extraídos os pontos relativos aos olhos, boca e nariz, e também é dado retângulo que delimita o tamanho da face. A saída é ilustrada na Figura 3.2.

Figura 3.2: *Landmarks* do rosto extraídos utilizando a MTCNN.

3.3.3 Etapa 3: Verificação do padrão ICAO

Após a detecção da face e dos pontos e a caixa delimitadora da face podemos analisar o posicionamento do rosto em relação ao restante da imagem. Os parâmetros verificados nessa etapa são baseados nos requisitos necessários para atender o padrão ICAO. Estes são:

Distância entre os olhos e o topo da imagem

A partir das coordenadas dos olhos, verifica-se se estes estão muito próximos de zero no eixo Y, o que corresponde ao topo da imagem, exemplificado na Figura 3.3. Para requisito, a distância da média dos olhos deve ser menor que 0.28 vezes a altura da imagem.

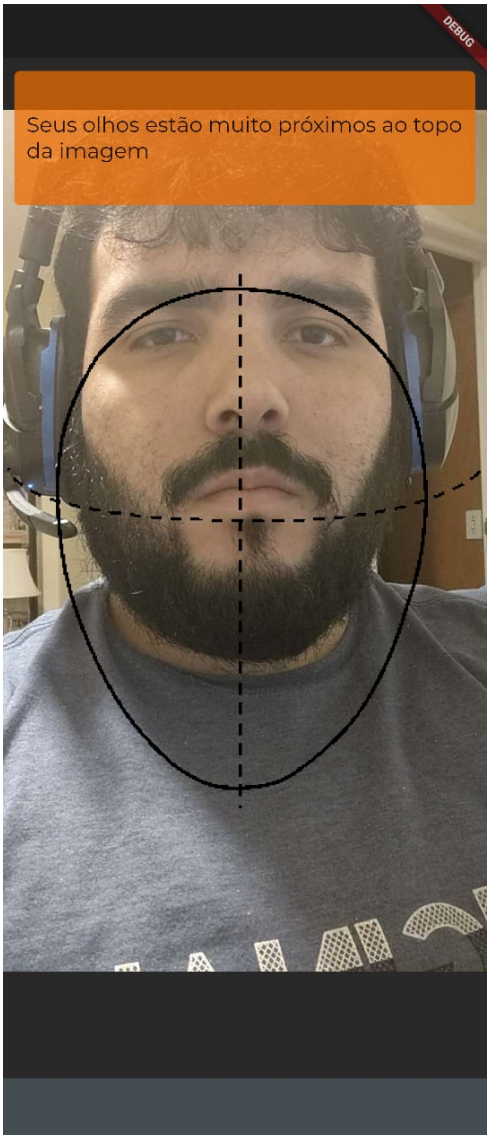


Figura 3.3: Exemplo de imagem com rosto muito próximo ao topo da imagem.

Distância entre os olhos e a parte inferior da imagem

A partir das coordenadas dos olhos, verifica-se se estes não estão muito próximos da altura da imagem no eixo Y, o que corresponde à parte debaixo da imagem, exemplificado na Figura 3.4. Para esse requisito, a distância média dos olhos deve ser no máximo 0.51 vezes a altura da imagem.

Figura 3.4: Exemplo de imagem com rosto muito próximo à parte debaixo da imagem.

Rosto muito distante da câmera

Para esse parâmetro, o retângulo que delimita o tamanho da face é comparado com o tamanho da imagem. Se a razão entre a altura do retângulo e o tamanho da imagem for muito alta, isso significa que o rosto está muito distante da câmera. Este caso é ilustrado na Figura 3.5. Para o cálculo desse requisito, a razão entre a altura do quadrado do rosto e a altura da imagem deve ser de no mínimo 50.

Figura 3.5: Exemplo de imagem com rosto muito distante da câmera.

Rosto muito próximo da câmera

Neste parâmetro é feito o mesmo cálculo para verificar se o rosto está muito distante da câmera. A diferença é que agora é checado se a razão entre a altura do retângulo e a altura da imagem não é muito baixa. Este caso é ilustrado na Figura 3.6. Para o cálculo desse requisito, a razão entre a altura do quadrado do rosto e a altura da imagem deve ser de no máximo 70.

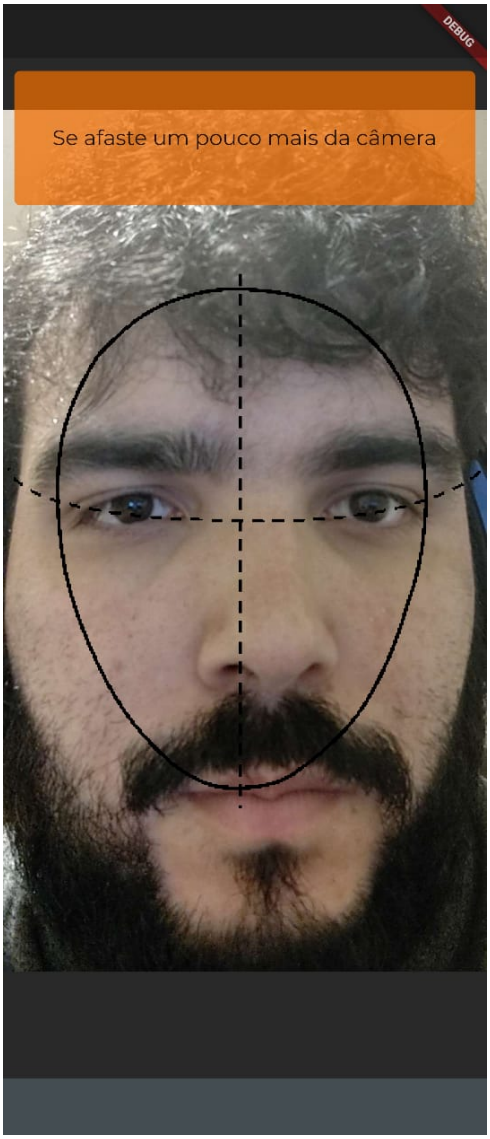


Figura 3.6: Exemplo de imagem com rosto muito próximo da câmera.

Alinhamento dos olhos

Aqui é feita uma comparação entre as coordenadas dos dois olhos. Por padrão é definida uma tolerância de 10% na diferença entre a altura dos olhos. Caso a diferença seja maior do que esse valor, os olhos são considerados desalinhados, como pode ser verificado na Figura 3.7. Para o cálculo do alinhamento dos olhos, foi estabelecido um nível de tolerância de 10%. Caso a distância absoluta entre as coordenadas y dos olhos ultrapasse esse valor, os olhos estão desalinhados.

Figura 3.7: Exemplo de imagem com olhos desalinhados.

Alinhamento do rosto

Por último, é verificado o posicionamento do rosto em relação ao centro da imagem. Imagens onde o rosto se encontra descentralizado não passam na verificação, como ilustrado na figura Figura 3.8. Para o cálculo do alinhamento do rosto, foi estabelecido um nível de tolerância de 10%. Caso a distância entre o centro da face e a largura da imagem divido por 2 ultrapasse a tolerância, o rosto é considerado desalinhado.

Figura 3.8: Exemplo de imagem com rosto não centralizado.

3.4 Quantificando níveis de borramento

Como foi mostrado anteriormente, ao utilizar a MTCNN é possível extrair pontos da face. Com esses pontos podemos fazer um pré-processamento para verificar o posicionamento do rosto em relação a imagem é adequado. No entanto, a MTCNN também é capaz de detectar rostos em condições adversas de borramento, como ilustrado na Figura 3.9.

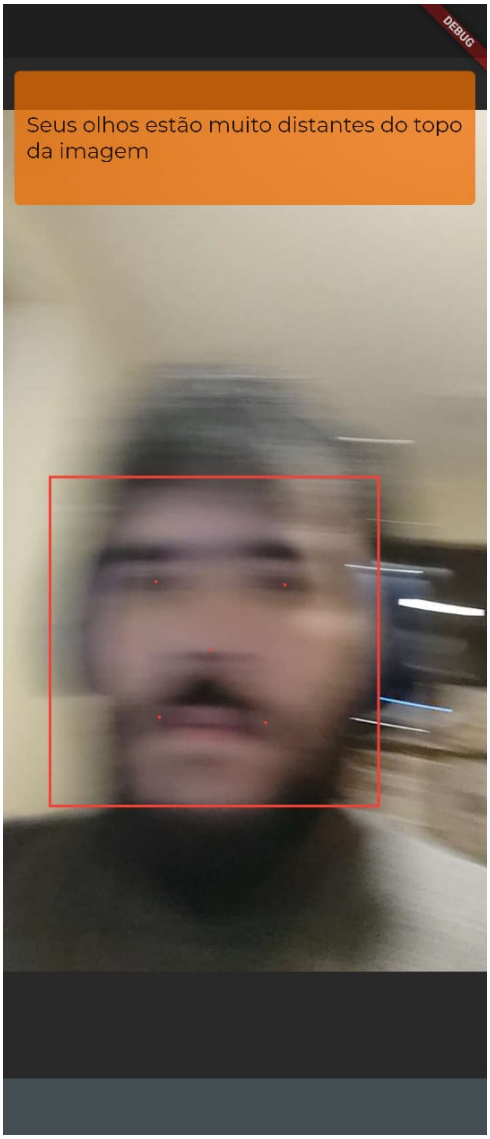


Figura 3.9: Exemplo de imagem com rosto borrado sendo detectado utilizando MTCNN.

O borramento é um problema comum ao manipular a câmera do celular e, uma vez que o retângulo e os pontos da face estejam dentro dos limiares estabelecidos essa foto vai ser aprovada. Para tentar mitigar esse problema, foi adotado um método de aprovação manual onde o usuário pode verificar a foto capturada e aprová-la ou tentar tirar uma novo foto. Porém, além de aumentar o número de tarefas a serem executadas pelo usuário, a aprovação manual do mesmo não garante que a imagem esteja sem borramento. Por conta disso, foi adicionada mais uma etapa no pré-processamento da imagem, onde é aplicado um filtro Laplaciano na imagem aprovada no padrão ICAO. Após a aplicação do filtro, são feitas algumas medidas na imagem modificada com o intuito de quantificar o nível de borramento da imagem.

3.4.1 Etapa 1: Aplicando o filtro Laplaciano

Como explicado na Seção 2.5, temos como entrada um conjunto discreto de pixels, cuja informação é armazenada em um *Bitmap* da biblioteca *Android Graphics*, que faz parte da Android SDK. Por questões de performance, a aplicação do filtro é feita em uma imagem reescalada para dimensões menores. Após, é feita a extração dos componentes de cor RGB do Bitmap e é aplicada a matriz de convolução nos pixels para obtermos os componentes de cor da matriz RGB de saída. A implementação da aplicação do filtro Laplaciano pode ser vista no Código-fonte I.3. O kernel utilizado no Código-fonte I.3 é o definido na Equação 2.5. A implementação é uma adaptação do código desenvolvido por Robert Sedgewick e Kevin Wayne [26].

3.4.2 Etapa 2: Quantificando o valor do borramento

Para quantificar o valor do borramento, a matriz com a aplicação do filtro Laplaciano é percorrida em um laço de repetição. Primeiramente são extraídos os componentes RGB do pixel que está sendo avaliado e em seguida é feito o cálculo do seu componente de iluminação. Com isso é feito o cálculo da média e da variância desse componente de iluminação da matriz. O algoritmo utilizado pode ser verificado no Código-fonte I.4.

3.4.3 Etapa 3: Sugestão da imagem mais adequada

Para que seja feita a sugestão da imagem com menor quantidade de borramento, ao invés de enviar a primeira imagem que esteja dentro do padrão ICAO para que o usuário avalie são selecionadas cinco imagens com um intervalo de 0,5 segundo entre elas. Embora isso efetivamente torne a experiência do usuário alguns segundos mais lenta, a captura das cinco imagens mostrou-se eficiente em relação a captura de imagens com o menor nível de borramento possível.

O intervalo de 0,5 segundo foi escolhido com o intuito de dar uma chance ao usuário de fazer pequenos ajustes e estabilizar a câmera. Caso contrário, *frames* subsequentes seriam selecionados aumentando as chances de se capturar uma imagem com parâmetros muito parecidos ao da imagem anterior e, caso a primeira imagem apresentasse borramento, a segunda teria uma chance mais elevada de estar borrada também. Dentre as cinco imagens é selecionada a imagem que possui o Laplaciano com maior variância, que corresponde a imagem com o maior número de bordas bem definidas após a aplicação do filtro Laplaciano.

Diferente de aplicações que utilizam técnicas para estimar um limiar que defina a presença ou ausência de borramento, a aplicação Android tem a vantagem de ser a fonte das imagens. Portanto, ao invés de verificar se uma única imagem está acima de um

limiar, é proposto que dentre cinco imagens, selecionar aquela que apresenta o Laplaciano com maior variância, corresponde à imagem com mais chance de estar nítida.

3.5 Fluxo principal do usuário

Embora a aplicação tenha outras funcionalidades, neste trabalho é dado foco nos processos de cadastro e chamada, que constituem o fluxo principal do usuário do sistema.

3.5.1 Cadastro

Todos os usuários são pré-cadastrados no sistema utilizando o sistema web, todos os dados cadastrais como nome, matrícula, gênero, dentre outros já são fornecidos. O aplicativo é responsável por complementar esse cadastro fornecendo fotos que estejam de acordo com com o que é estabelecido no padrão ICAO. Para que isso seja feito, o usuário deve inserir seus dados cadastrais para se autenticar no sistema, assim como ilustrado na Figura 3.10.

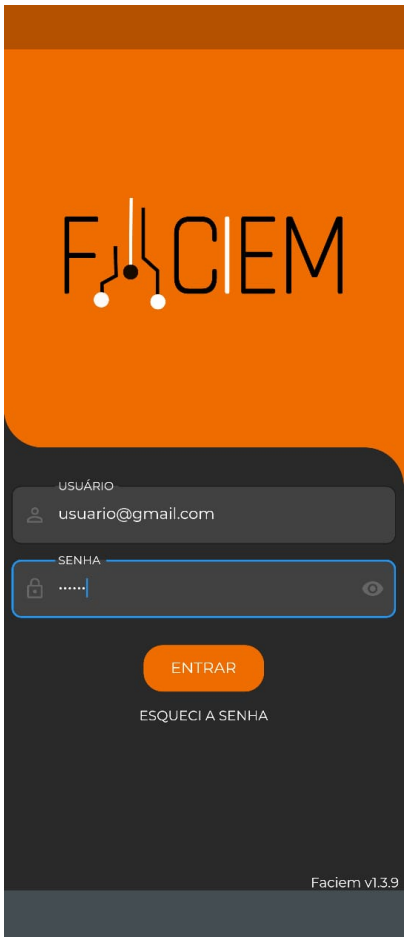
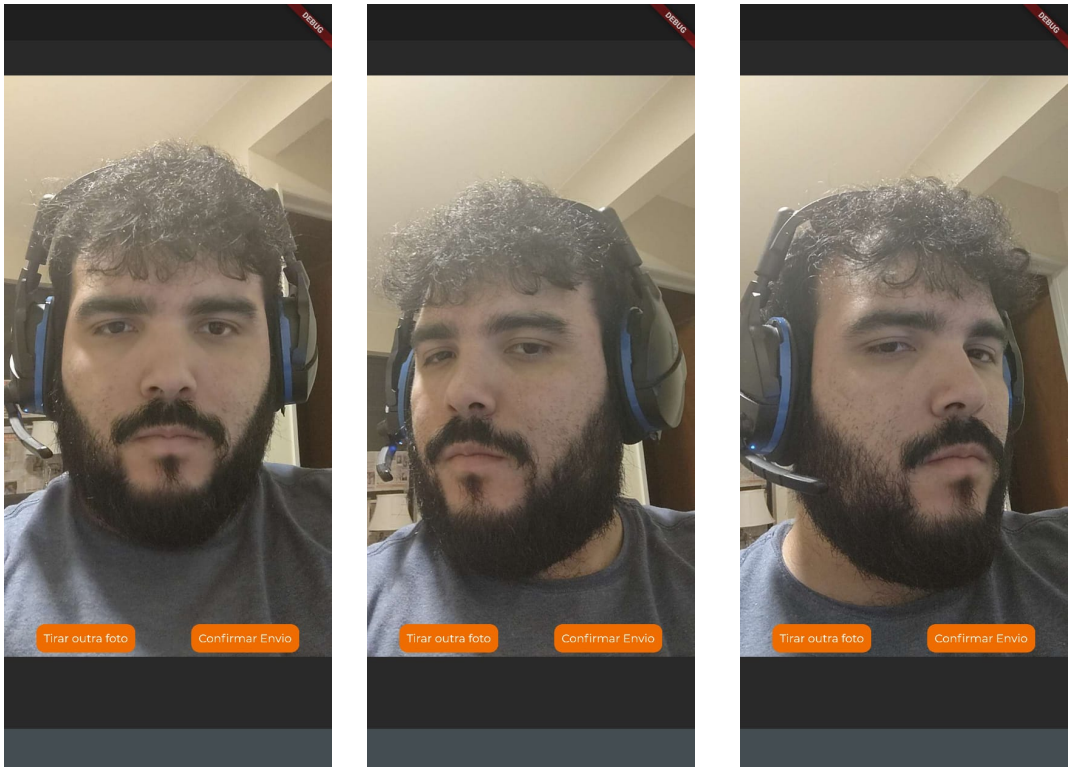


Figura 3.10: Tela de autenticação do usuário.

Após ser realizada a autenticação, três fotos da face devem ser capturadas, uma frontal, uma lateral esquerda e uma lateral direita, assim como ilustrado na Figura 3.11. As três fotos obrigatoriamente devem passar na lógica de pré-processamento. Cada captura deve ser confirmada pelo usuário antes de ser enviada para o cadastro.



(a) Frontal (b) Lateral esquerda (c) Lateral direita

Figura 3.11: Exemplos de imagens capturadas durante a fase de confirmação de cadastro.

Após a confirmação das três imagens, elas são enviadas para a avaliação do módulo de reconhecimento facial que verificará se a qualidade da imagem é suficiente para o cadastro.

3.5.2 Chamada

Uma vez que as três fotos de cadastro foram devidamente capturadas e aprovadas pelo módulo de reconhecimento facial, os alunos e professores estão aptos a registrar suas presenças nas atividades dos cursos em que foram cadastrados. Para realizar a chamada, primeiramente é necessário fazer a leitura de um *QR Code*, assim como ilulstrado na Figura 3.12, que contém a informação da atividade e do curso.

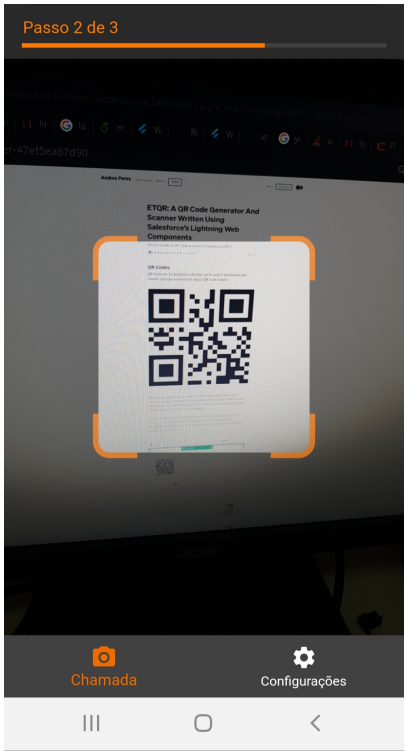


Figura 3.12: Exemplo de leitura de *QR Code* feita pelo usuário.

Após isso será feita a captura frontal do rosto do usuário. Caso o usuário esteja corretamente cadastrado e ainda não tenha realizado a chamada da mesma atividade a foto do usuário será capturada e sujeita à avaliação do módulo de reconhecimento facial. Posteriormente, o usuário receberá uma notificação informando o resultado do processamento da sua foto. Caso o reconhecimento facial seja feito com sucesso e o usuário esteja dentro da área limite da atividade, a sua presença será computada corretamente.

Capítulo 4

Resultados

Neste capítulo são descritos os testes e resultados obtidos utilizando o sistema proposto. Foram feitos testes de desempenho utilizando o sistema F2Dsys e testes de comparação da variância do Laplaciano utilizando o OpenCV na linguagem Python.

4.1 Análise de desempenho

Para a aplicação do filtro Laplaciano, o tamanho da imagem é um fator que impacta diretamente o tempo necessário ao seu cálculo. Em dispositivos mais antigos, com menor poder computacional, esse tempo pode não ser aceitável.

Foram selecionados três celulares e medido o tempo de execução do conjunto de funções i) redimensionamento da imagem do rosto extraído da foto, ii) aplicação do filtro Laplaciano e iii) cálculo da variância de seus valores. O tempo foi medido utilizando a função `uptimeMillis`, recomendada para medir um intervalo entre dois pontos de execução do aplicativo. A função `uptimeMillis` mede o tempo total desde que o sistema foi iniciado em milissegundos [27]. Usando essa função é simples escolher quais trechos de código a serem medidos.

Os três aparelhos celulares usados foram:

- Samsung Galaxy S10 lite(SM-G770F)
  Processador: Qualcomm Snapdragon 855 2,8 GHz Octa-Core 64bits
  Memória RAM: 6 GB LPDDR4X
  GPU:Qualcomm Adreno 640
  Sistema Operacional: Android 10

- Motorola MOTO G100
  Processador: Qualcomm Snapdragon 870 3,2 GHz Octa-Core 64bits

Memória RAM: 12 GB LPDDR5
GPU:Qualcomm Adreno 650
Sistema Operacional: Android 11

- Samsung A51(SM-A515F/T)
  Processador: Samsung Exynos 9611 Octa-Core 4× 2,3GHz,4× 1,7GHz
  Memória RAM: 4 GB LPDDR4X
  GPU: ARM Mali-G72 MP3
  Sistema Operacional: Android 11

O tempo de execução foi medido em função do tamanho da imagem redimensionada obtida a partir do retângulo do rosto detectado. Os tamanhos utilizados no teste foram 2×2, 3×3, 4×4, 5×5, 10×10, 20×20, 40×40, 80×80, 160×160 , 320×320 e 640×640 pixels, conforme mostrado na Figura 4.1 .

(a) 2×2 (b) 3×3 (c) 4×4

(d) 5×5 (e) 10×10 (f) 20×20

(g) 40×40 (h) 60×60 (i) 80×80

(j) 160×160 (k) 320×320 (l) 640×640

Figura 4.1: Exemplos das imagens cortadas e redimensionadas para os tamanhos de teste

Para cada tamanho da imagem e para cada modelo de celular foram medidos 15 tempos com imagens diferentes.

4.1.1 Resultados - Samsung Galaxy S10 lite

A Tabela 4.1 apresenta as médias, máximos e mínimos das medidas de tempo dos trinta testes feitos para os tamanhos menores no aparelho Galaxy S10 lite.

Tabela 4.1: Médias, máximos e mínimos das medidas de tempo (ms) para os tamanhos menores no aparelho S10 lite.

	2×2	3×3	4×4	5×5	10×10	15×15	20x20
Máximo	11	12	11	10	12	14	12
Mínimo	9	9	9	9	9	10	11
Média	9,46	9,46	9,60	9,46	9,96	10,83	11,63

A Tabela 4.2 apresenta as médias, máximos e mínimos das medidas de tempo dos trinta testes feitos para os tamanhos maiores no aparelho Galaxy S10 lite.

Tabela 4.2: Médias, máximos e mínimos das medidas de tempo (ms) para os tamanhos maiores no aparelho S10 lite.

	40×40	60×60	80×80	160×160	320×320	640×640
Máximo	20	34	51	177	685	2723
Mínimo	19	31	50	173	677	2707
Média	19,16	32,40	50,20	174,46	678,43	2716,76

É observado que para imagens menores que 20×20 não há mais praticamente nenhum ganho de desempenho e o tamanho da imagem deixa de ser o fator determinante.

4.1.2 Resultados - Motorola MOTO G100

A Tabela 4.3 apresenta as médias, máximos e mínimos das medidas de tempo dos trinta testes feitos para os tamanhos menores no aparelho MOTO G100.

Tabela 4.3: Médias, máximos e mínimos das medidas de tempo (ms) para os tamanhos menores no aparelho MOTO G100.

	2×2	3×3	4×4	5×5	10×10	15×15	20×20
Máximo	15	26	9	8	22	9	10
Mínimo	7	7	7	7	7	8	9
Média	8,63	8,26	7,40	7,20	8,73	8,16	9,06

A Tabela 4.4 apresenta as médias, máximos e mínimos das medidas de tempo dos trinta testes feitos para os tamanhos maiores no aparelho MOTO G100.

Tabela 4.4: Médias, máximos e mínimos das medidas de tempo (ms) para os tamanhos maiores no aparelho MOTO G100.

	40×40	60×60	80×80	160×160	320×320	640×640
Máximo	30	33	50	153	571	2259
Mínimo	15	26	41	144	561	2238
Média	16,60	27,06	42,76	145,73	563,03	2242,73

É observado que o MOTO G100 teve desempenho ligeiramente melhor ao S10 lite, mais rápido em todos dos tamanhos. Para imagens menores que 20×20 não há mais praticamente nenhum ganho de desempenho assim como S10 lite.

4.1.3 Resultados - Samsung A51

A Tabela 4.5 apresenta as médias, máximos e mínimos das medidas de tempo dos trinta testes feitos para os tamanhos menores no aparelho A51.

Tabela 4.5: Médias, máximos e mínimos das medidas de tempo (ms) para os tamanhos menores no aparelho A51.

	2×2	3×3	4×4	5×5	10×10	15×15	20×20
Máximo	27	25	23	28	31	30	47
Mínimo	17	17	16	17	17	18	20
Média	19,33	18,30	18,23	19,06	19,10	19,96	22,26

A Tabela 4.6 apresenta as médias, máximos e mínimos das medidas de tempo dos trinta testes feitos para os tamanhos maiores no aparelho A51.

Tabela 4.6: Médias, máximos e mínimos das medidas de tempo (ms) para os tamanhos maiores no aparelho A51.

	40×40	60×60	80×80	160×160	320×320	640×640
Máximo	45	62	101	320	1231	4845
Mínimo	32	53	81	280	1079	4293
Média	35,36	54,66	83,26	293,90	1121,10	4387,40

A Figura 4.2 apresenta mudança da média do tempo de execução em função do tamanho da imagem para os três aparelhos testados.

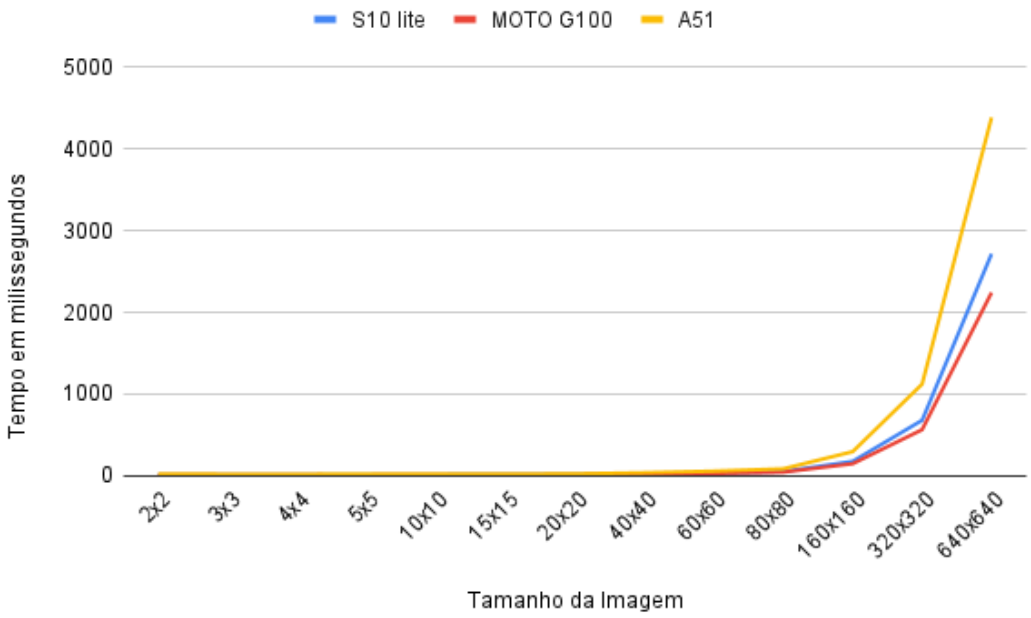


Figura 4.2: Variação da média do tempo de execução em função do tamanho da imagem para os três aparelhos testados.

O aparelho A51 demonstrou melhor a inviabilidade de se usar os tamanhos de imagem 320×320 e 640×640, pela duração ser longa no ponto de vista do usuário para um processamento de imagem de vários quadros.Também podemos utilizar esse aparelho como limite superior considerando que ainda existem em circulação diversos aparelhos com menor capacidade computacional.

Nos três aparelhos podemos observar que para tamanhos de imagens menores que 20×20 não há mais praticamente nenhum ganho de desempenho e o tamanho da imagem deixa de ser o fator determinante. O aparelho MOTO G100 teve o melhor desempenho dos trés, com desempenho um pouco melhor comparado ao S10 lite.

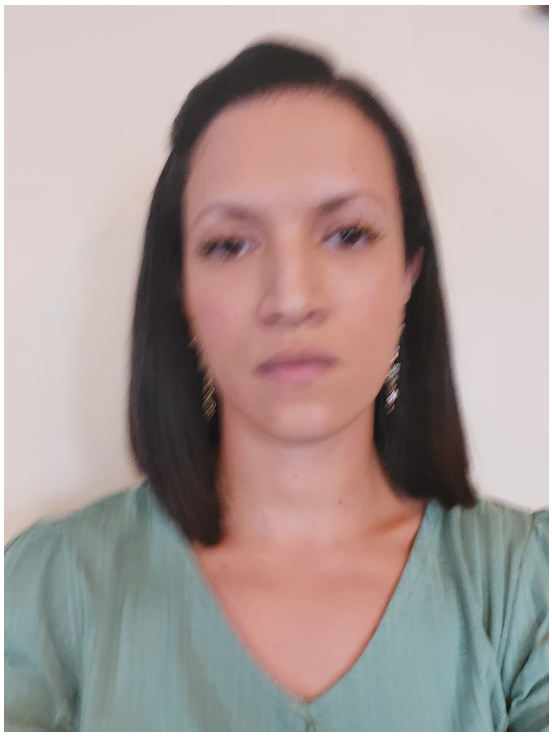
Os três aparelhos apresentam um comportamento de função quadrática considerando o aumento de tempo do tamanho 320×320 para 640×640.

4.2 Análise da variância do Laplaciano

O objetivo deste teste é comparar imagens consideradas borradas com imagens consideradas não borradas e analisar a variância dos pixels dessas imagens. É considerado que quanto mais borrada a imagens menor é a variância do Laplaciano. Como imagens menores possuem a variância maior a análise só pode ser feita comparando imagens do mesmo tamanho. É considerado que a imagem de maior variância de um determinado tamanho

é a imagem escolhida pelo sistema proposto. Para esse teste cinco pessoas tiraram dez fotos cada, cinco fotos inserindo borramento com o movimento e cinco fotos normais. As fotos tiradas inserindo borramento foram classificadas como fotos borradas e as demais como fotos normais. A Figura 4.3a mostra um exemplo de foto tirada sem borramento e a Figura 4.3b é um exemplo de foto tirada com borramento.

(a) Exemplo de imagem sem borramento



(b) Exemplo de imagem com borramento

Figura 4.3: Comparação de imagem que não contém borramento e imagem que contém borramento

Após a geração das imagens foi feito um programa em python usando a biblioteca OpenCV e a biblioteca FaceRecognition para detectar as faces nas imagens, gerar uma nova imagem a partir do retângulo da face detectada, redimensionar a nova imagem gerada, aplicar o filtro Laplaciano e calcular a variância da imagem filtrada assim como é feito no aplicativo. O processo de aplicação do filtro Laplaciano e extração da variância foi feito para todos os tamanhos citados anteriormente. O python foi usado para o teste pela praticidade em modificar e salvar as informações sobre as imagens bem como pela maior velocidade de processamento de um *desktop* em relação a um *smartphone*. A Figura 4.4a e a Figura 4.4b exemplificam o comportamento de uma imagem com borramento e sem borramento, respectivamente, após a aplicação do filtro Laplaciano.

(a) Aplicação do filtro Laplaciano em imagem não borrada

(b) Aplicação do filtro Laplaciano em imagem borrada

Figura 4.4: Comparação de imagem que não contém borramento e imagem que contém borramento após aplicação do filtro Laplaciano

Na Figura 4.4 é possível observar pixels com maior intensidade de branco na imagem da Figura 4.4a em relação a imagem borrada, Figura 4.4b. Isto possivelmente indica que a imagem normal terá uma variância maior do que a imagem borrada.

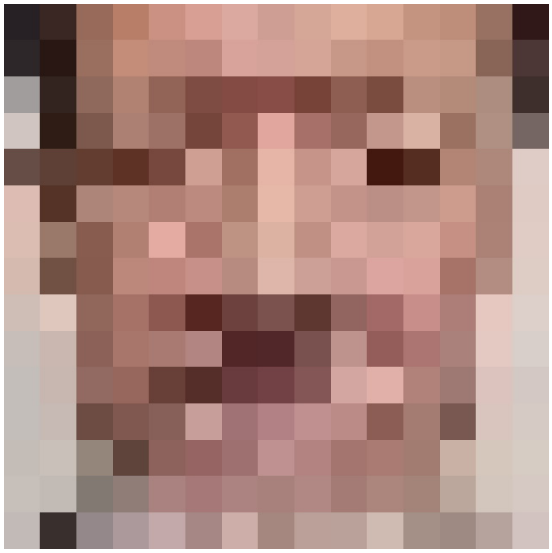
Usando como exemplo a pessoa 4, a Tabela 4.7 apresenta o comportamento da variância do Laplaciano em função do tamanho da imagem. As imagens que possuem um "B" antes do nome do arquivo são as imagens classificadas como borradas, em negrito são destacados os maiores valores por tamanho.

Tabela 4.7: Variância das imagens de tamanhos menores da pessoa 4.

Imagens	2×2	3×3	4×4	5×5	10×10	15×15	20×20
205943	28086	26107,88	**24403,88**	40197,39	19954,69	16632,22	13206,91
205814	54350	30518,89	16458,44	11778,15	17877,22	15806,43	11249,38
205838	36878	10955,51	8032,36	42593,65	22754,87	13833,23	**12657,17**
205826	19076	10403,95	15204,69	16831,03	**25688,94**	15107,00	10779,25
205831	25846	9222,10	19545,48	43929,11	17300,86	15208,17	11727,07
B205954	35984	15376,91	8898	12492,30	15990,51	10148,79	6934,47
B205951	**80822**	6900,62	15177,44	11922,86	18005,91	10939,68	8296,49
B205850	6582	5078,44	11715,36	19883,64	11316,23	5473,44	3409,90
B205843	17624	**32376,02**	21214	**48360,74**	23653,05	**16876,57**	13092,15
B205957	62622	11675,78	11453,13	20038,89	16501,64	10185,61	6608,60

É observado que a imagem com maior variância muda a cada tamanho, devido à grande perda de informação da operação de redimensionamento da imagem. Nestes casos então a estimativa fica inconsistente. Essa inconsistência está presente em todas as imagens das cinco pessoas para os tamanhos menores. Podemos ver a perda de informação na Figura 4.5.

(a) Imagem original



(b) Imagem após o corte e redimensionamento para 15×15

Figura 4.5: Comparação entre uma imagem original e uma redimensionada da pessoa 4

Com essa perda de informação a Figura 4.5 teve a maior variância para o tamanho 15×15 mesmo sendo visivelmente borrada.

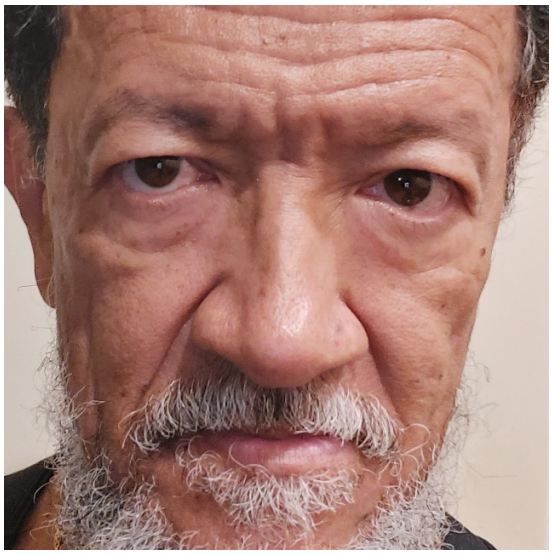
Em contraste, para os tamanhos maiores é observado que a imagem de maior variância para quase todos os casos é do grupo de imagens classificadas como normais (sem borramento), conforme apresentado na Tabela 4.8.

Tabela 4.8: Variância das imagens de tamanhos maiores da pessoa 4.

Imagens	40×40	60×60	80×80	160×160	320×320	640×640
205943	4685,28	3206,63	2155,91	1059,18	574,46	314,51
205814	4929,74	3335,42	2443,38	1284,54	465,88	166,78
205838	**6836,31**	**5110,90**	**4174,19**	**3024,67**	**1763,88**	**509,00**
205826	5164,51	3946,24	3070,04	1849,40	777,98	227,63
205831	5059,44	3754,91	3227,61	2079,53	1040,68	297,68
B205954	2115,14	1105,95	723,88	365,28	235,98	127,08
B205951	2723,18	1579,66	1112,98	554,54	295,14	146,52
B205850	783,39	425,86	275,77	173,79	142,31	100,20
B205843	4861,34	3006,49	2090,23	1208,91	737,54	364,01
B205957	2229,37	1342,14	924,31	424,32	215,16	113,20

No caso da pessoa 4 é observado um caso em que para todos os tamanhos maiores a foto de maior variância é a mesma (imagem 205838). Para as pessoas 2, 3 e 5 a variância de tamanhos maiores variou entre duas ou três imagens, porém a imagem de maior variância foi sempre uma das imagens normais. Na Figura 4.6 podemos verificar que a perda de informação é menor com o maior tamanho da imagem redimensionada e que a escolha é acertada como uma imagem sem borramento.

(a) Imagem original



(b) Imagem após o corte e redimensionamento para 640x640

Figura 4.6: Comparação entre a imagem original e a imagem redimensionada da pessoa 4

Para a pessoa 1 ocorreu uma escolha errada para o tamanho 640×640. Uma imagem que teve baixa variância relativa para os outros tamanhos foi a imagem escolhida para o maior tamanho testado e também foi a única imagem classificada como borrada que foi escolhida nos tamanhos maiores.

(a) Imagem escolhida para o tamanho 320×320(b) Imagem escolhida para o tamanho 640×640

Figura 4.7: Comparação de imagens escolhidas da pessoa 1

Com a Figura 4.7 é observado que mesmo para imagens de tamanho maior é possível a ocorrência de erro na estimativa do borramento da imagem quando é utilizada a variância do Laplaciano como estimador de borramento.

Na Figura 4.8 é apresentado um gráfico comparativo das médias das imagens normais com as médias das imagens borradas de todos as pessoas que participaram do texto. A Figura 4.9 a mesma comparação para os tamanhos maiores para uma melhor visibilidade.

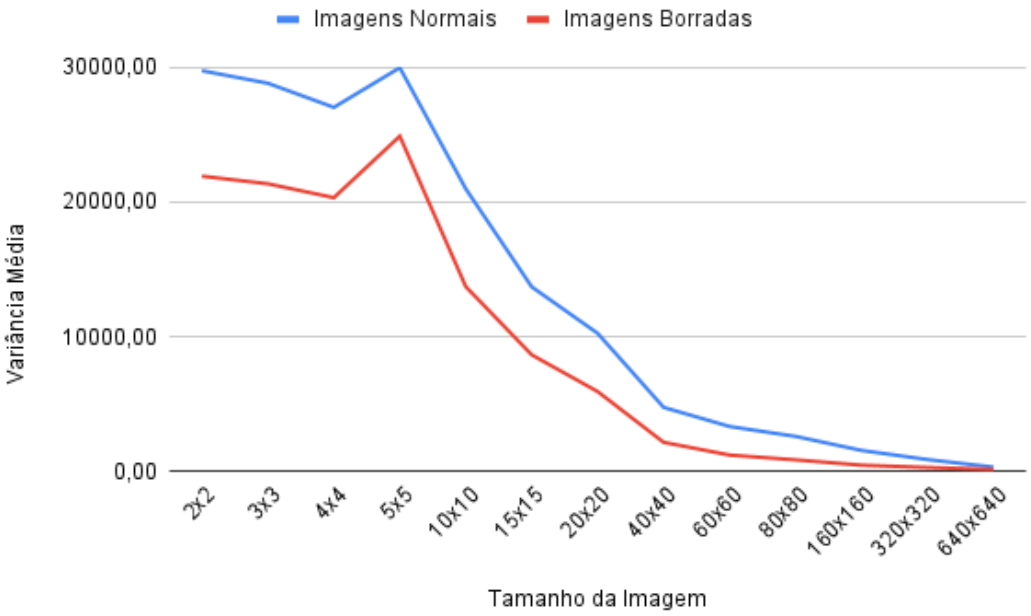


Figura 4.8: Comparação das variâncias médias de todas as imagens normais com todas as imagens borradas.

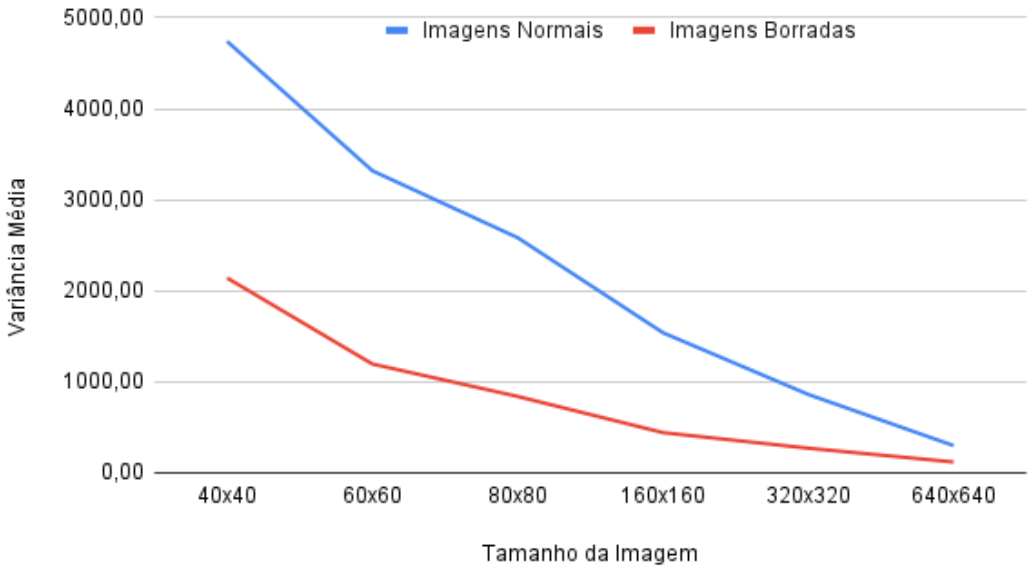


Figura 4.9: Comparação das variâncias médias de todas as imagens normais com todas as imagens borradas para os tamanhos maiores.

Mesmo com casos adversos é observado na Figura 4.8 que a média das variâncias das imagens normais é maior do que das imagens borradas para todos os tamanhos, porém o mais importante é a diferença na média para os tamanhos maiores como é observado na Figura 4.9. Devemos notar também que a diferença entre as variâncias médias diminui com o aumento de seu tamanho. A variância do Laplaciano pode falhar em definir exatamente qual imagem é menos borrada porém os testes demonstraram que essa técnica pode ser usada para fazer uma estimativa e recomendar qual foto é a menos borrada. A Tabela 4.9 apresenta a taxa de falhas para cada tamanho, é considerado que ocorreu uma falha quando a imagem que tem maior variância é uma imagem classificada como borrada. Considerando as cinco pessoas do grupo, caso houver 5 falhas para um determinado tamanho da imagem teremos 100% de falhas.

Tabela 4.9: Taxa de falhas para cada tamanho.

Tamanho	Taxa de falha
2×2	20%
3×3	40%
4×4	20%
5×5	60%
10×10	0%
15×15	20%
20×20	0%
40×40	0%
60×60	0%
80×80	0%
160×160	0%
320×320	0%
640×640	20%

É observado que os tamanhos menores tiveram a taxa falha mais elevada enquanto os tamanhos maiores praticamente não tiveram falhas com exceção da falha já mencionada.

Como grupo de teste é pequeno, é necessário comparar as variâncias das imagens normais com as imagens borradas. As Figuras 4.10,4.11,4.12 e 4.13 apresentam o diagrama de caixa para comparar a variância de todas as imagens normais (caixas azuis) com todas as imagens borradas (caixas vermelhas) para cada tamanho.As caixas representam a metade dos valores mais próximos da média ou seja os valores menos discrepantes.O traço cruzando a caixa representa a mediana, os traços horizontais acima e abaixo da caixa representam os pontos máximos e mínimos respectivamente e os pontos que não estão nem

na caixa nem nas linhas verticais são pontos discrepantes que por terem valores muito distantes dos valores da caixa podem ser considerados casos consideravelmente atípicos.

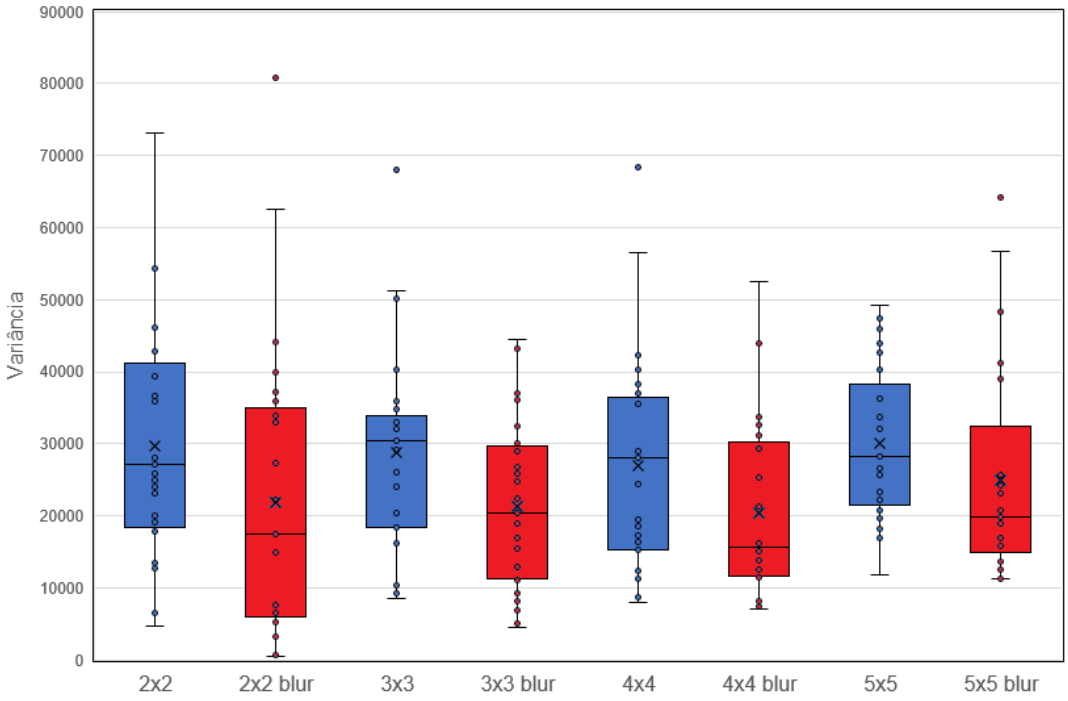


Figura 4.10: Diagrama de caixa para comparar a distribuição da variância das imagens normais com as imagens borradas para os tamanhos 2×2, 3×3,4×4 e 5×5.

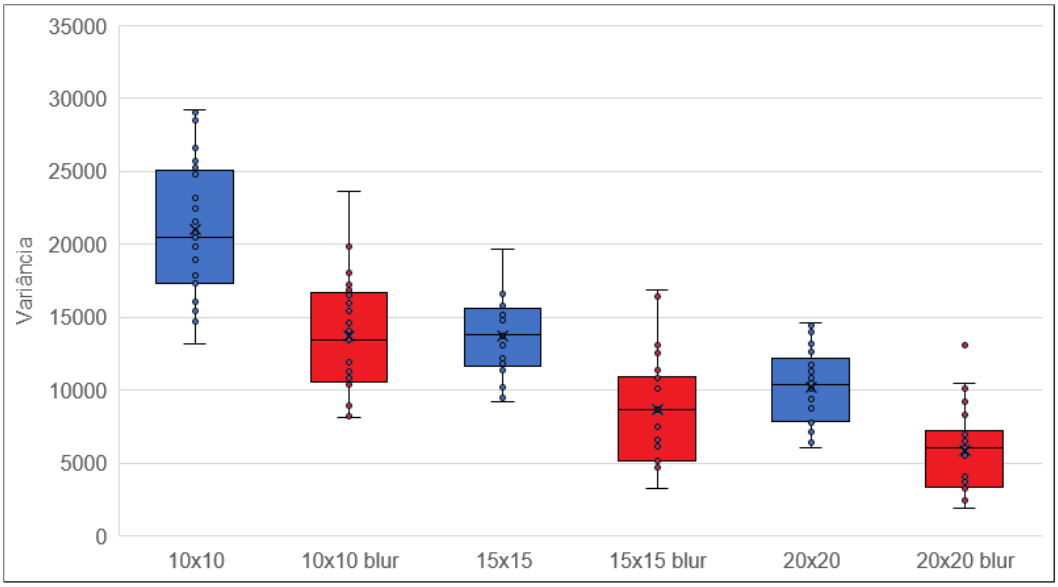


Figura 4.11: Diagrama de caixa para comparar a distribuição da variância das imagens normais com as imagens borradas para os tamanhos 10×10,15×15 e 20×20 .

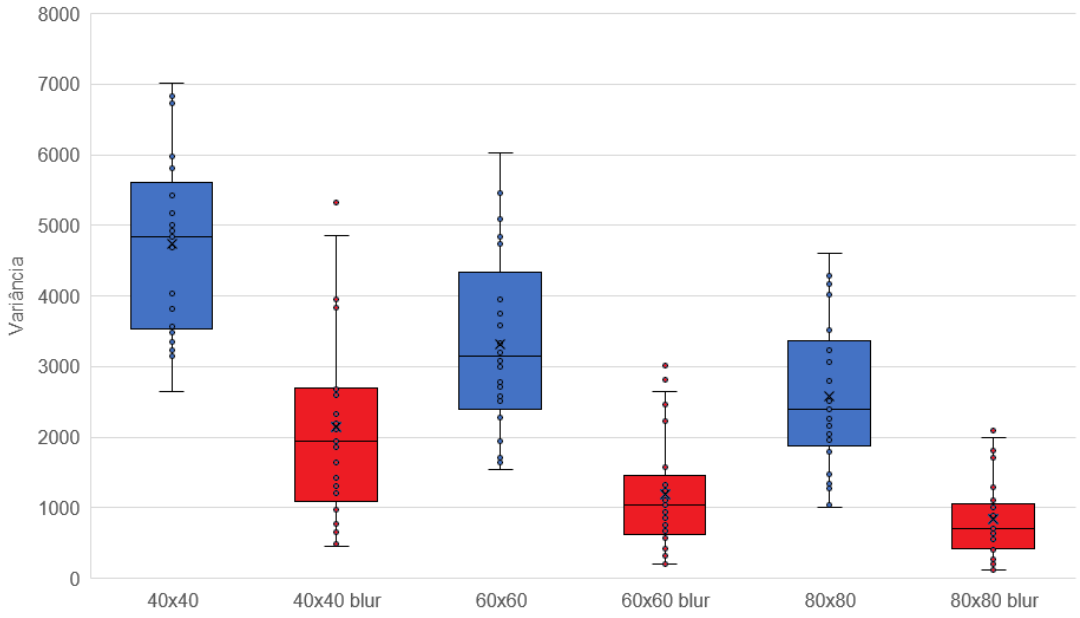


Figura 4.12: Diagrama de caixa para comparar a distribuição da variância das imagens normais com as imagens borradas para os tamanhos 40×40,60×60 e 80×80 .

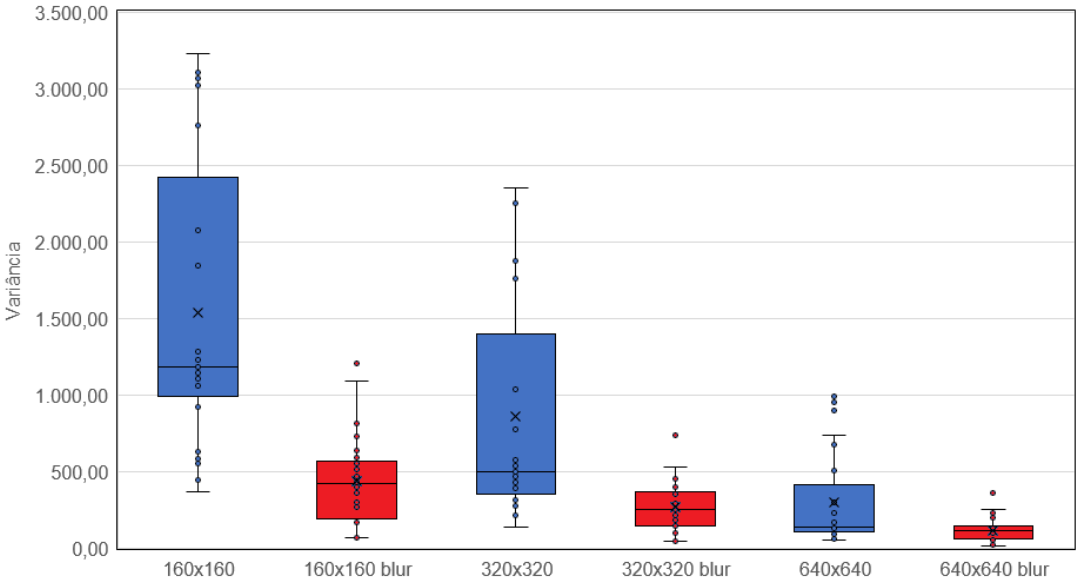


Figura 4.13: Diagrama de caixa para comparar a distribuição da variância das imagens normais com as imagens borradas para os tamanhos 160×160,320×320 e 640×640 .

É observado que para os tamanhos menores as caixas das imagens normais são semelhantes as imagens com borramento, com o aumento do tamanho da imagem somente a cauda superior das caixas das imagens com borramento alcançam o valor das caixas das imagens normais ou seja somente valores mais altos que são atípicos em relação ao resto dos valores medidos. Esse padrão é bem visível para o tamanho 160×160 na Figura 4.13. Isto indica que o método proposto não é 100% seguro, e que uma imagem atípica borrada pode ser selecionada como imagem não borrada. Porém, seu uso deve reduzir significativamente o número de vezes que um usuário precisará enviar nova foto devido à borramentos.

Capítulo 5

Conclusão

Neste trabalho foi apresentado uma aplicação Android que é uma das partes do sistema desenvolvido para o projeto F2Dsys. Este projeto oferece uma solução que utiliza biometria facial para aumentar a segurança e diminuir a chance de fraudes em cursos e eventos. Considerando que o foco do sistema é o reconhecimento e a identificação facial foi apresentado uma solução para aumentar a probabilidade que o usuário envie uma foto sem borramentos. A estimativa de borramento pode ser feita pelo sistema *back-end*, porém, como o usuário interage diretamente com a aplicação, o auxílio direto ao usuário pode ajudar na usabilidade da aplicação.

Foi verificado também que um dos problemas para usar um método de análise de borramento é a sua complexidade computacional. Em dispositivos móveis, que possuem menor capacidade de processamento, é imprescindível achar um meio para execução mais rápida das funções utilizadas no aplicativo. A redução do tamanho das imagens se mostrou eficiente para diminuir o tempo de execução da aplicação do filtro Laplaciano.

Uma imagem perde informação quando é diminuída de tamanho, então foi analisado visto que imagens maiores que 20×20 mantém a consistência na análise da variância da imagem filtrada pelo filtro Laplaciano.

Dos testes é observado que os tamanhos 60×60 até 160×160 apresentaram a melhor consistência na variância do Laplaciano e um tempo de execução adequado nos dispositivos móveis testados.

O maior problema da solução apresentada é a consistência da variância do Laplaciano como método para detectar borramento em condições adversas. Apesar de ter certa robustez em condições normais, o filtro Laplaciano é sensível tanto a ruído como a saturação e outros fatores como a iluminação podem afetar as bordas da imagem [16]. Exatamente por esse motivo a estimativa do borramento é usada somente para apresentar ao usuário a imagem que tem menor probabilidade de estar borrada entre um conjunto de cinco imagens.

5.1 Trabalhos Futuros

É sugerido que em trabalhos futuros seja feito um teste da eficiência do Laplaciano como estimador de borramento em uma base de dados maior. Como este trabalho foi realizado durante um contexto de pandemia da COVID19, a base de dados utilizada foi reduzida. Tendo uma base maior seria possível observar efeitos que podem causar algum tipo de alteração nas bordas, como iluminação, movimento e qualidades de imagens diferentes.

Também seria de grande valor testar o desempenho do sistema em celulares com menor poder computacional para abranger um maior número de sistemas onde este trabalho pode ser aplicado. Também sugere-se testar outras técnicas de estimativa de borramento para fazer a comparação com a eficiência da variância do Laplaciano. Outro trabalho que poderia ser realizado seria a implementação de técnicas de aprendizado de máquina para estimativa de borramento.

Referências

[1] altexsoft software r&d engineering. About yuv formats. https://www.altexsoft.com/blog/rest-api-design. [Online; acessado 01-11-2021]. x, 7

[2] Kaipeng Zhang, Zhanpeng Zhang, Zhifeng Li, e Yu Qiao. Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE Signal Processing Letters*, 23(10):1499–1503, Oct 2016. x, 8, 10

[3] International Civil Aviation Organization. Icao photo standards. http://miami.itamaraty.gov.br/en-us/photo_specifications.xml. [Online; acessado 01-11-2021]. x, 11

[4] Wikipedia. Yuv. https://en.wikipedia.org/wiki/YUV. [Online; acessado 01-11-2021]. x, 12, 14

[5] Wikipedia. Rgb color model. https://en.wikipedia.org/wiki/RGB_color_model. [Online; acessado 01-11-2021]. x, 13

[6] Jean-Marie Baran. About yuv formats. https://gist.github.com/Jim-Bar/3cbba684a71d1a9d468a6711a6eddbeb. [Online; acessado 01-11-2021]. x, 13, 14

[7] Flutter. Writing custom platform-specific code. https://flutter.dev/docs/development/platform-integration/platform-channels. [Online; acessado 01-11-2021]. x, 17, 18

[8] Stan Z. Li e Anil K. Jain. *Handbook of Face Recognition*. Springer, 2004. 1

[9] Ming-Hsuan Yang e Rebecca R Ahuja, Narendra. *Face Detection and Gesture Recognition for Human-Computer Interaction*. Springer Science & Business Media, 2001. 8

[10] ICAO. Technical report: Portrait quality. https://www.icao.int/Security/FAL/TRIP/Documents/TR%20-%20Portrait%20Quality%20v1.0.pdf. [Online; acessado 10-11-2021]. 10

[11] Jian Yang, Chengjun Liu, e Lei Zhang. Color space normalization: Enhancing the discriminating power of color spaces for face recognition. *Pattern Recognition*, 43(4):1454–1466, 2010. 11

[12] LLC Discovery Scientific. Yuv, ycbcr, ypbpr color spaces. https://discoverybiz.net/enu0/faq/faq_YUV_YCbCr_YPbPr.html. [Online; acessado 01-11-2021]. 12

[13] LLC Discovery Scientific. Ycbcr. hhttps://api.video/what-is/ycbcr. [Online; acessado 01-11-2021]. 12

[14] Robert Hirsch. *Exploring Colour Photography: A Complete Guide*. Laurence King Publishing, 2004. 12

[15] Encyclopedia of Mathematics. Edge detection. https://encyclopediaofmath.org/index.php?title=Edge_detection&oldid=17883. [Online; acessado 01-11-2021]. 14

[16] Miguel Angel Garcia Said Pertuz, Domenec Puig. Analysis of focus measure operators for shape-from-focus. *Pattern Recognition*, 46(5):1415–1432, 2013. 14, 16, 52

[17] Institute of Radio Astronomy e Astrophysis National Autonomous University of Mexico. Apply laplacian filters. https://www.irya.unam.mx/computo/sites/manuales/IDL/Content/GuideMe/ImageProcessing/LaplacianFilters.html. [Online; acessado 01-11-2021]. 15

[18] P.L. Meyer e R. de Carvalho Bergström Lourenço Filho. *Probabilidade: aplicações à estatística*. LTC, 2006. 16

[19] Google. Google open source. https://opensource.google/. [Online; acessado 10-11-2021]. 17, 18

[20] Google. Google brain team. https://research.google/teams/brain/. [Online; acessado 10-11-2021]. 18

[21] TensorFlow. Why tensorflow. https://www.tensorflow.org/. [Online; acessado 01-11-2021]. 18

[22] Wikipedia. Opencv. https://en.wikipedia.org/wiki/OpenCV. [Online; acessado 01-11-2021]. 19

[23] Flutter. Introduction to widgets. https://flutter.dev/docs/development/ui/widgets-intro. [Online; acessado 05-11-2021]. 20

[24] Dash Overflow. provider package. https://pub.dev/packages/provider. [Online; acessado 05-11-2021]. 20

[25] Flutter. camera package. https://pub.dev/packages/camera. [Online; acessado 06-11-2021]. 21

[26] Robert Sedgewick and Kevin Wayne. Laplacefilter.java. https://introcs.cs.princeton.edu/java/31datatype/LaplaceFilter.java.html. [Online; acessado 10-11-2021]. 31

[27] Google. Systemclock. https://developer.android.com/reference/android/os/SystemClock. [Online; acessado 06-11-2021]. 35

Anexo I

Códigos Fonte

```
@override
void initState() {
  super.initState();

  _cameraController = CameraController(
    widget.cameras[1],
    _resolution,
    enableAudio: false,
  );

  _cameraController.initialize().then((_) {
    if (!mounted) {
      return;
    }

    _cameraController.startImageStream((CameraImage img) async {
      if (_isDetecting || _showFacePreview) {
        return;
      }
      _isDetecting = true;

      List<int> strides = new Int32List(img.planes.length * 2);
      int index = 0;
      // We need to transform the image to Uint8List so that the native
    code could
      // transform it to byte[]
      List<Uint8List> data = img.planes.map((plane) {
        strides[index] = (plane.bytesPerRow);
        index++;
        strides[index] = (plane.bytesPerPixel);
        index++;
```

```
31          return plane.bytes;
        }).toList();
33        _detectFace(
            strides,
35          data,
            img.width,
37          img.height,
            _cameraController.description.sensorOrientation,
39        );
      });
41  });
}
```

Código I.1: Exemplo de inicialização de controlador de câmera e envio de *frames* para pré-processamento

```
package com.unb.f2dsys;
44
import android.graphics.ImageFormat;
46 import android.graphics.Rect;
import android.graphics.YuvImage;
48
import java.io.ByteArrayOutputStream;
50 import java.nio.ByteBuffer;
import java.util.List;
52
// Implementation based on https://github.com/tomerblecher/YUV_2_RGB
54 public class YuvConverter {
    /**
56   * Converts an NV21 image into JPEG compressed.
     * @param nv21 byte[] of the input image in NV21 format
58   * @param width Width of the image.
     * @param height Height of the image.
60   * @param quality Quality of compressed image(0-100)
     * @return byte[] of a compressed Jpeg image.
62   */
    public static byte[] NV21toJPEG(byte[] nv21, int width, int height, int
    quality) {
64      ByteArrayOutputStream out = new ByteArrayOutputStream();
        YuvImage yuv = new YuvImage(nv21, ImageFormat.NV21, width, height,
    null);
66      yuv.compressToJpeg(new Rect(0, 0, width, height), quality, out);
        return out.toByteArray();
68  }
```

```
70      /**
        * Format YUV_420 planes in to NV21.
72      * Removes strides from planes and combines the result to single NV21
    byte array.
        * @param planes  List of Bytes list
74      * @param strides contains the strides of each plane. The structure :
        *                strideRowFirstPlane,stridePixelFirstPlane,
    strideRowSecondPlane
76      * @param width   Width of the image
        * @param height  Height of given image
78      * @return NV21 image byte[].
        */
80     public static byte[] YUVtoNV21 (List<byte[]> planes, int[] strides, int
     width, int height) {
            Rect crop = new Rect(0, 0, width, height);
82          int format = ImageFormat.YUV_420_888;
            byte[] data = new byte[width * height * ImageFormat.getBitsPerPixel
    (format) / 8];
84          byte[] rowData = new byte[strides[0]];
            int channelOffset = 0;
86          int outputStride = 1;
            for (int i = 0; i < planes.size(); i++) {
88              switch (i) {
                    case 0:
90                      channelOffset = 0;
                        outputStride = 1;
92                      break;
                    case 1:
94                      channelOffset = width * height + 1;
                        outputStride = 2;
96                      break;
                    case 2:
98                      channelOffset = width * height;
                        outputStride = 2;
100                     break;
                }
102
                ByteBuffer buffer = ByteBuffer.wrap(planes.get(i));
104             int rowStride;
                int pixelStride;
106             if(i ==0 ) {
                    rowStride = strides[i];
108                 pixelStride = strides[i+1];
                }
110             else {
```

```
                rowStride = strides[i *2 ];
                pixelStride = strides[i *2 +1];
            }
            int shift = (i == 0) ? 0 : 1;
            int w = width >> shift;
            int h = height >> shift;
            buffer.position(rowStride * (crop.top >> shift) + pixelStride *
    (crop.left >> shift));
            for (int row = 0; row < h; row++) {
                int length;
                if (pixelStride == 1 && outputStride == 1) {
                    length = w;
                    buffer.get(data, channelOffset, length);
                    channelOffset += length;
                } else {
                    length = (w - 1) * pixelStride + 1;
                    buffer.get(rowData, 0, length);
                    for (int col = 0; col < w; col++) {
                        data[channelOffset] = rowData[col * pixelStride];
                        channelOffset += outputStride;
                    }
                }
                if (row < h - 1) {
                    buffer.position(buffer.position() + rowStride - length)
    ;
                }
            }
        }
        return data;
    }
}
```

Código I.2: Trecho de código responsável pela conversão de uma matriz YUV para uma matriz RGB

```
// Implemetation based on https://introcs.cs.princeton.edu/java/31datatype/
    LaplaceFilter.java.html
public static Bitmap laplacianFilter(Bitmap srcBitmap) {
    int width  = srcBitmap.getWidth();
    int height = srcBitmap.getHeight();

    Bitmap dstBitmap = Bitmap.createBitmap(srcBitmap);

    for (int y = 1; y < height - 1; y++) {
        for (int x = 1; x < width - 1; x++) {
```

```
            int c00 = srcBitmap.getPixel(x-1, y-1);
            int c01 = srcBitmap.getPixel(x-1, y);
            int c02 = srcBitmap.getPixel(x-1, y+1);
            int c10 = srcBitmap.getPixel(x, y-1);
            int c11 = srcBitmap.getPixel(x, y);
            int c12 = srcBitmap.getPixel(x, y+1);
            int c20 = srcBitmap.getPixel(x+1, y-1);
            int c21 = srcBitmap.getPixel(x+1, y);
            int c22 = srcBitmap.getPixel(x+1, y+1);
            int r = - Color.red(c00) - Color.red(c01)
                    - Color.red(c02) - Color.red(c10)
                    + 8*Color.red(c11) - Color.red(c12)
                    - Color.red(c20) - Color.red(c21)
                    - Color.red(c22);
            int g = - Color.green(c00) -  Color.green(c01)
                    - Color.green(c02) - Color.green(c10)
                    + 8*Color.green(c11) - Color.green(c12)
                    + -Color.green(c20) -  Color.green(c21)
                    - Color.green(c22);
            int b = - Color.blue(c00) - Color.blue(c01)
                    - Color.blue(c02) - Color.blue(c10)
                    + 8*Color.blue(c11) - Color.blue(c12)
                    - Color.blue(c20) - Color.blue(c21)
                    - Color.blue(c22);
            r = Math.min(255, Math.max(0, r));
            g = Math.min(255, Math.max(0, g));
            b = Math.min(255, Math.max(0, b));
            int color = Color.rgb(r, g, b);
            dstBitmap.setPixel(x, y, color);
        }
    }
    return dstBitmap;
}
```

Código I.3: Trecho de código responsável por aplicar um filtro Laplaciano em uma matriz RGB.

```
public static double calculateVariance(Bitmap bitmap) {
    double sum = 0;
    double num_of_elements = bitmap.getWidth() * bitmap.getHeight();
    for (int x = 0; x < bitmap.getWidth(); x++) {
        for (int y = 0; y < bitmap.getHeight(); y++) {
            int r = Color.red(bitmap.getPixel(x, y));
            int g = Color.green(bitmap.getPixel(x, y));
            int b = Color.blue(bitmap.getPixel(x, y));
```

```
190
                sum += 0.299 * r + 0.587 * g + 0.114 * b;
192         }
        }
194
        double mean = sum / num_of_elements;
196
        sum = 0;
198     for (int x = 0; x < bitmap.getWidth(); x++) {
            for (int y = 0; y < bitmap.getHeight(); y++) {
200             int r = Color.red(bitmap.getPixel(x, y));
                int g = Color.green(bitmap.getPixel(x, y));
202             int b = Color.blue(bitmap.getPixel(x, y));
                double rgb = 0.299 * r + 0.587 * g + 0.114 * b;
204             sum += Math.pow((rgb - mean), 2);
            }
206     }
        return sum / num_of_elements;
208 }
```

Código I.4: Trecho de código que define o método que recebe uma matriz e calcula a variância dos seus pixels.

Anexo II

Banco de Imagens

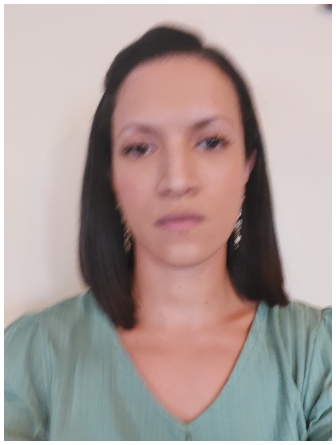
Pessoa 1 Imagem 1

Pessoa 1 Imagem 2

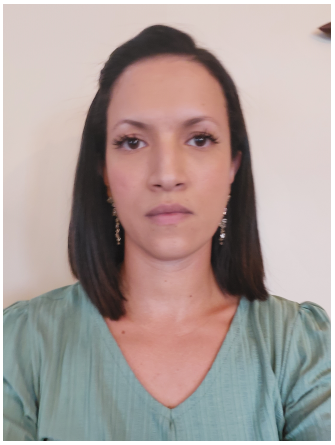
Pessoa 1 Imagem 3

Pessoa 1 Imagem 4

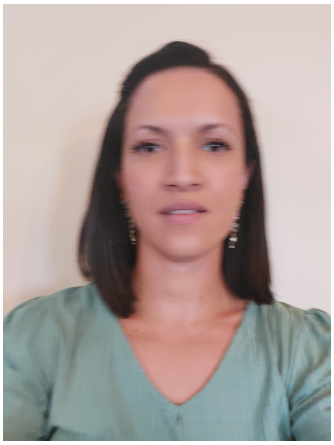
Pessoa 1 Imagem 5



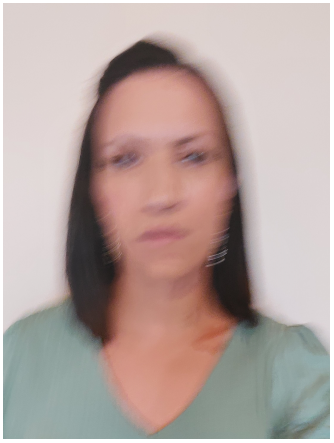
Pessoa 1 Imagem 6



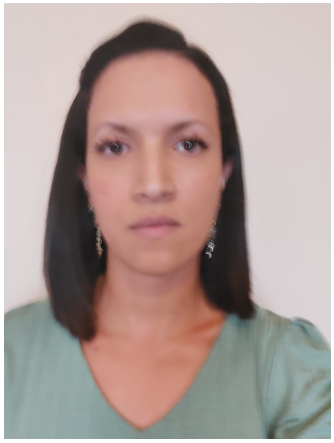
Pessoa 1 Imagem 7



Pessoa 1 Imagem 8



Pessoa 1 Imagem 9



Pessoa 1 Imagem 10

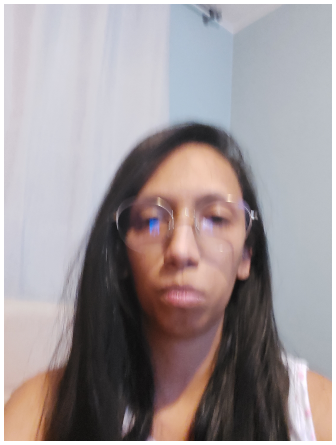
Pessoa 2 Imagem 1

Pessoa 2 Imagem 2

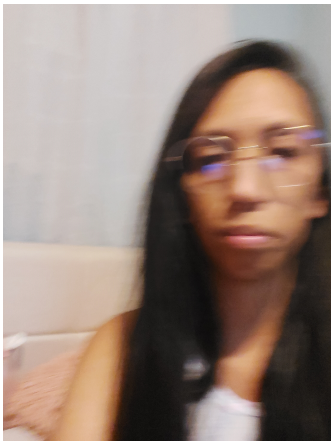
Pessoa 2 Imagem 3

Pessoa 2 Imagem 4

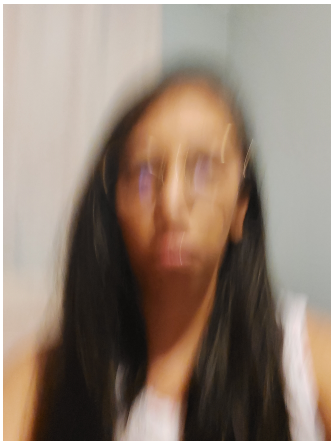
Pessoa 2 Imagem 5



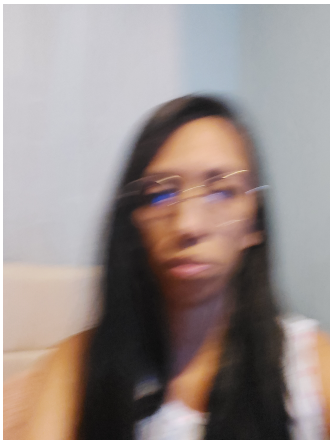
Pessoa 2 Imagem 6



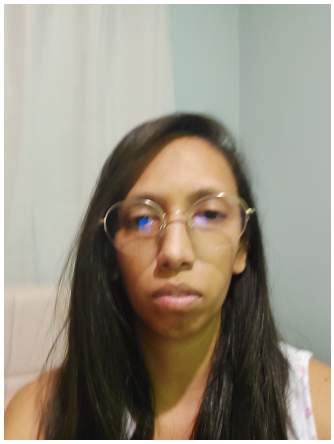
Pessoa 2 Imagem 7



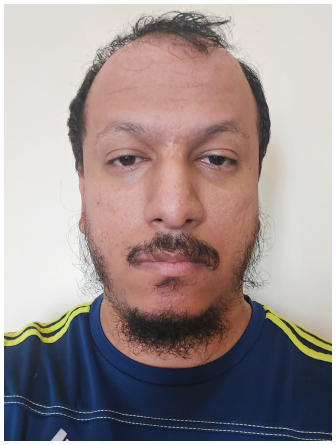
Pessoa 2 Imagem 8



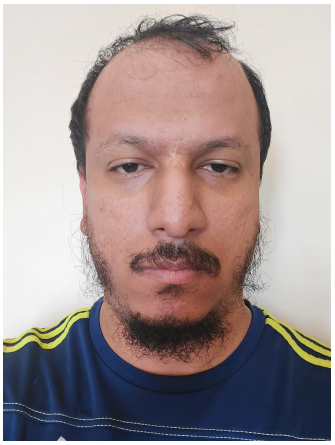
Pessoa 2 Imagem 9



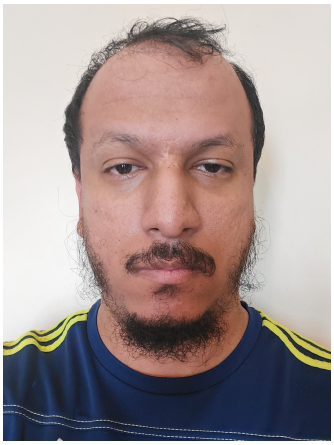
Pessoa 2 Imagem 10



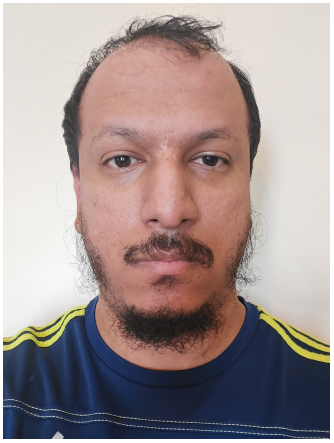
Pessoa 3 Imagem 1



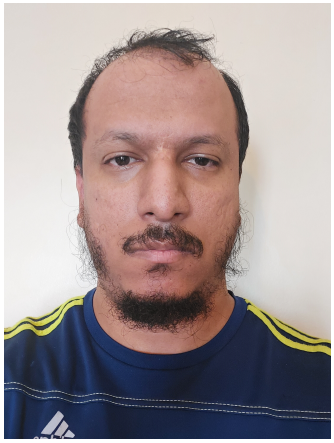
Pessoa 3 Imagem 2



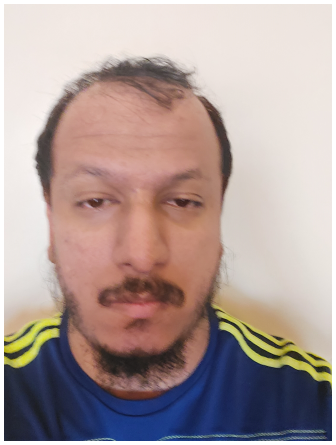
Pessoa 3 Imagem 3



Pessoa 3 Imagem 4

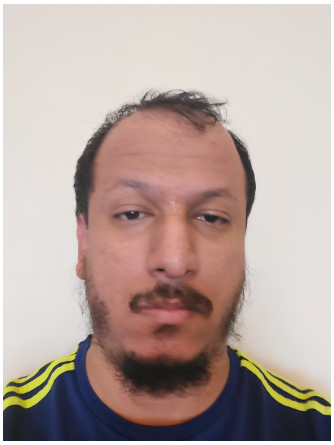


Pessoa 3 Imagem 5

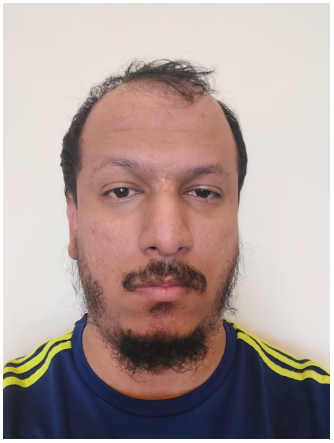


Pessoa 3 Imagem 6

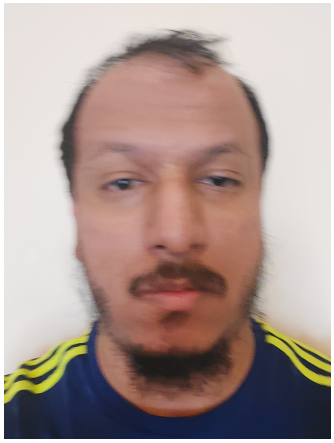
Pessoa 3 Imagem 7



Pessoa 3 Imagem 8



Pessoa 3 Imagem 9



Pessoa 3 Imagem 10

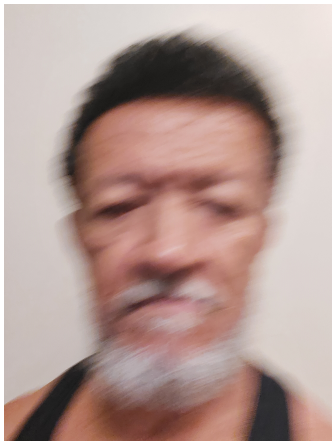
Pessoa 4 Imagem 1

Pessoa 4 Imagem 2

Pessoa 4 Imagem 3

Pessoa 4 Imagem 4

Pessoa 4 Imagem 5



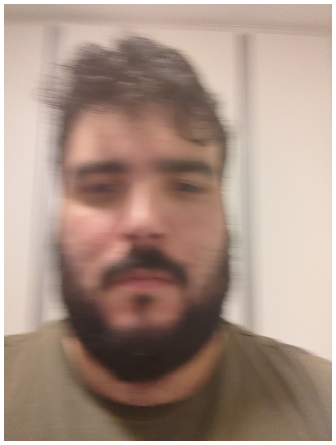
Pessoa 4 Imagem 6

Pessoa 4 Imagem 7

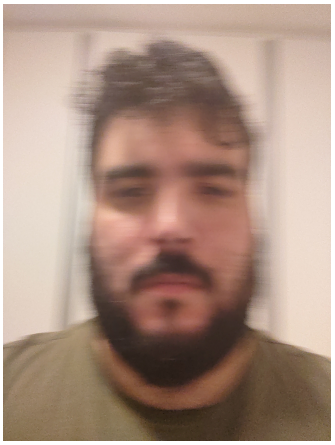
Pessoa 4 Imagem 8

Pessoa 4 Imagem 9

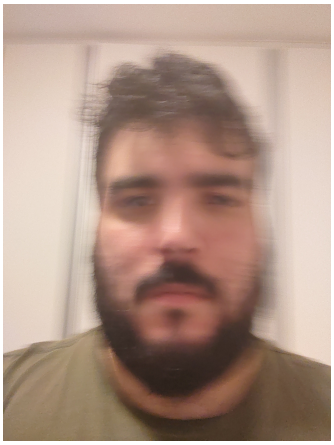
Pessoa 4 Imagem 10



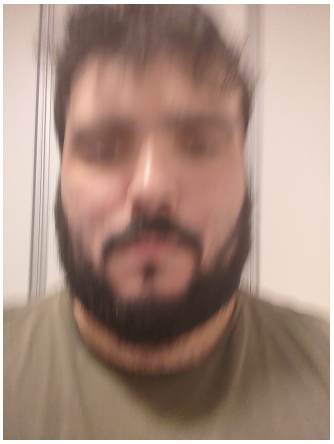
Pessoa 5 Imagem 1



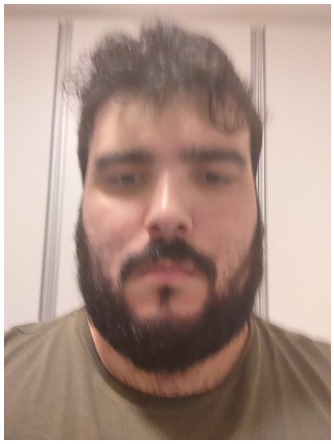
Pessoa 5 Imagem 2



Pessoa 5 Imagem 3



Pessoa 5 Imagem 4



Pessoa 5 Imagem 5

Pessoa 5 Imagem 6

Pessoa 5 Imagem 7

Pessoa 5 Imagem 8

Pessoa 5 Imagem 9

Pessoa 5 Imagem 10