Ferramenta para Identificação de Deficiência Nutricional de Café em Dispositivos Android

Lucas Rodrigues da Cunha

Universidade Federal de Uberlândia
Faculdade de Computação
Bacharelado em Sistemas de Informação

Monte Carmelo - MG
2015

Lucas Rodrigues da Cunha

Ferramenta para Identificação de Deficiência Nutricional de Café em Dispositivos Android

Trabalho de Conclusão de Curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Sistemas de Informação.

Área de concentração: Ciência da Computação

Orientador: Thiago Pirola Ribeiro

Monte Carmelo - MG
2015

Dedico esse trabalho a meus pais, Marcelo e Margarida, meus irmãos Samuel e Sara, a meus amigos, e professores. Sem eles, o sonho de concluir a graduação seria impossível.

Agradecimentos

Agradeço a Deus, autor e sustentador da vida, por ser minha força, refúgio e fortaleza. Sem Ele, nada disso seria possível.

Agradeço a minha família e amigos, por terem estado ao meu lado durante toda essa caminhada, sempre me apoiando, trazendo palavras de ânimo, mostrando que toda essa luta vale a pena.

A todos os professores e servidores do curso de Sistemas de Informação da Universidade Federal de Uberlândia Campus Monte Carmelo, em especial, ao professor Thiago Pirola Ribeiro, meu orientador, pela paciência, apoio e tempo dedicado para que esse trabalho fosse realizado.

No demais, agradeço a todos aqueles que de alguma forma estiveram e estão próximos de mim, fazendo esta vida valer cada vez mais a pena.

"Engraçado, eu acho que o destino não é o caminho dado a nós, mas o caminho que escolhemos pra nós mesmos."

(Megamente)

Resumo

Diversas são as deficiências nutricionais que atingem um cafezal, resultando em grandes perdas financeiras, além de impactar negativamente na economia nacional. O diagnóstico das deficiências quando feito em um estágio inicial, pode-se realizar a reposição nutricional do solo visando restaurar a integridade da planta. Existem métodos para realizar essa análise, através de testes de laboratórios, necessitando que amostras sejam retiradas da planta, e demandam de algum tempo para obter o resultado final. Este trabalho implementou um software para dispositivos com Android que utiliza-se de técnicas de processamento digital de imagens e estatística para identificar a existência de uma deficiência nutricional sem a retirada de amostras. O método baseia-se na obtenção de uma imagem de uma folha de café e, após a aplicação de algoritmos para a identificação e separação do objeto (folha) do fundo da imagem, a imagem resultante é convertida em tons de cinza e calcula-se a média, mediana e o terceiro momento. Os resultados obtidos com a implementação foram muito satisfatórios.

Palavras-chave: Android, Processamento Digital de Imagens, Deficiência Nutricional, Café.

Lista de ilustrações

Lista de tabelas

Lista de siglas

ABIC Associação Brasileira da Indústria de Café

ADT Ferramenta para desenvolvimento Android - *Android Development Tool*

APK Pacote Android - *Android Package*

CONAB Companhia Nacional de Abastecimento

CPU Unidade Central de Processamento - *Central Processing Unit*

ID Identificador - *Identificator*

IDE Ambiente Integrado para Desenvolvimento - *Integrated Development Environment*

PC Computador Pessoal - *Personal Computer*

SDK Kit para desenvolvimento de Software - *Software Development Kit*

SO Sistema Operacional - *Operational System*

XML Linguagem extensível de marcação - *eXtensible Markup Language*

Sumário

Capítulo 1

Introdução

1.1 Descrição do Problema

A história do café no Brasil teve inicio em 1727, com mudas trazidas da Guiana Francesa e, pelo clima favorável do país para esse tipo de planta, se espalhou por todo o território. Atualmente o Brasil mantém seu status de maior produtor e exportador mundial de café, permanecendo também como segundo maior consumidor do produto (ABIC, 2013).

Segundo a CONAB (2015) (Companhia Nacional de Abastecimento), a primeira estimativa de produção de café (arábica e conillon) para a safra 2014 indica que o país colheu mais de 45,3 milhões de sacas beneficiadas, sendo 32,3 milhões de café arábica, e 13,0 milhões de café conilon.

A área plantada de café para a safra 2015 é de 2.256.5 mil hectares, sendo 0,1% inferior à safra passada. A área em produção totaliza 1.933.120 hectares (85,7%), enquanto que a área de café em formação, ou seja, aquela que ainda não entrou no processo produtivo, totaliza 323.381,4 hectares (14,3%) (CONAB, 2015).

A expectativa ara o ano de 2015, é que o país deverá colher entre 44,11 e 46,61 milhões de sacas de 60 kg de café beneficiado. A produção do café arábica (espécie mais conhecida, e avaliada como a de "melhor qualidade"), representa 73,7% da produção total, e tem como maior produtor o estado de Minas Gerais, com mais de 50% de café beneficiado, seguido pelo estado do Espirito Santo com 28,24%.

A região do Cerrado Mineiro, composta pelas sub-regiões Triangulo Mineiro, Alto Paranaíba e Noroeste Mineiro contam com um parque cafeeiro para 2015 de 169.343 hectares em produção e 34.115 hectares em formação, sendo esperada uma produção de 5,004 milhões de sacas. Esta é a região de maior produtividade em Minas Gerais, com 29,55 sacas/hectare (CONAB, 2015).

Apesar das baixas de produção estarem ligadas à bienalidade, a queda na produção muitas vezes se deve às diversas dificuldades que são enfrentadas pelos produtores brasileiros incluindo as doenças que podem devastar uma lavoura inteira ou mesmo a deficiência

de nutrientes no solo (MALAVOLTA et al., 1993).

As deficiências nutricionais mais comuns que aparecem nos cafezais da região de Monte Carmelo, local onde foram feitos os estudos, são deficiências de nitrogênio, fósforo, potássio, cálcio, zinco, dentre outras. Essas deficiências são detectadas visualmente, pois as folhas do café começam a exibir uma tonalidade diferente de seu verde vívido (Figura 1).

Para diagnosticar essas deficiências, os produtores devem submeter amostras das folhas que estão com os possíveis problemas para laboratórios, onde serão realizados ensaios analíticos, ou uma análise foliar da cultura, para quantificação de macro e micronutrientes com o objetivo de investigar a deficiência de maneira a tomar decisões para amenizar suas consequências.

Atualmente os ensaios de análise foliar são comumente recomendados e os seus custos são relativamente pequenos comparados com os custos de outros insumos agrícolas necessários ao cultivo de café.

No entanto, esta conduta de realizar a análise foliar só é tomada quando se tem uma suspeita evidenciada visualmente, da deficiência nutricional ou doença presente na cultura, o que dependendo da relação solo/cultura/nutriente pode provocar perdas consideráveis de produtividade, devido ao estágio de identificação visual nas folhas, caule e frutos.

Figura 1 – Folha sem deficiências nutricionais (cima) e com deficiência de ferro (baixo).

A identificação precoce de doenças ou deficiências nutricionais por meio de análise automatizada das imagens das folhas se afigura como desejável, de modo que ações corretivas poderão ser tomadas ainda em um estágio inicial do desenvolvimento destas doenças ou deficiências.

1.2 Objetivos gerais

O presente projeto propõe a implementação de uma ferramenta que auxilie na identificação precoce das deficiências dos cafezais, através de imagens das folhas do café, sem a

necessidade da retirada de amostras. Utilizando-se da ferramenta para análise de imagens de deficiências nutricionais do café, o produtor poderá capturar as imagens no campo e processá-las, realizando a identificação precoce das possíveis deficiências nutricionais da planta em questão.

Essa ferramenta será desenvolvida para dispositivos móveis que utilizam o Sistema Operacional Android, plataforma a qual segundo pesquisas é a mais usada em todo o mundo (GOOGLE, 2014a).

1.3 Objetivos Específicos

- ❏ Estudo e análise das técnicas de processamento de imagens;
- ❏ Estudo da plataforma Android;
- ❏ Desenvolvimento de uma aplicação Android para detecção de deficiências nutricionais em folhas de café.

1.4 Organização da Monografia

No capítulo 2, serão apresentados os principais termos e conceitos utilizados como base para esse trabalho. Informações sobre a plataforma Android, sua arquitetura, características e funcionamento. Também será abordada a ideia geral de processamento paralelo, Tecnologia móvel e informações sobre as deficiências do café, ajudando o leitor a entender como o sistema desenvolvido poderá ser útil na solução deste problema.

No capítulo 3, alguns trabalhos semelhantes ao que foi desenvolvido aqui, são apresentados. Estes trabalhos tem como tema central processamento de imagens em Android, aplicações em Android, e processamento paralelo.

Já no capitulo 4, o tema abordado será a descrição do trabalho desenvolvido. Nele serão descritos os algoritmos desenvolvidos, a metodologia usada, e tudo o que foi construído.

No capítulo 5, serão apresentados os resultados obtidos. Capturas de tela da aplicação desenvolvida, testes realizados, problemas enfrentados.

No capítulo 6, encerrando este trabalho, apresentamos as conclusões obtidas ao final de todos os estudos, testes, e análises feitas.

Capítulo 2

Revisão Bibliográfica

Neste capítulo, serão abordados: a plataforma Android (conceitos básicos de funcionamento, arquitetura e detalhes sobre o desenvolvimento de aplicações), conceitos de Processamento Paralelo (vantagens e desvantagens) e problemas das deficiências do café.

2.1 Tecnologia Móvel

A computação esta a cada dia mais presente na vida das pessoas. O que antes era restrito a ambientes militares e laboratórios, se estendeu a empresas e bancos, culminando na criação dos computadores pessoais. Esse fato mudou a evolução dos computadores, chegando nos dias atuais, onde a computação não está restrita a computadores, mas está difundida em eletrodomésticos, eletrônicos, aparelhos médicos, matemáticos, agronômicos, e em praticamente todas as áreas do conhecimento humano.

Nesse contexto, um aparelho tem um destaque especial: os smartphones. Segundo dados de Grego (2014), em todo o mundo existem 2,7 bilhões de linhas de celulares, e, espera-se que até 2020 esse numero chegue a 6,1 bilhões. Eles representam aproximadamente 70% das vendas de aparelhos celulares no terceiro trimestre de 2014.

Não há uma definição comumente usada para smartphones. A PCMagazine (2014) diz que, "um smartphone combina telefone celular, acesso à Internet para e-mail e Web, música e filme player, câmera e filmadora, sistema de navegação GPS e uma busca por voz para fazer uma pergunta sobre qualquer coisa. Um smartphone é muito mais pessoal do que um computador pessoal, porque é com você o tempo todo quando viajar e nas proximidades, se você usá-lo como seu telefone principal."

Quando consultamos o dicionário Oxford (2014), encontramos a seguinte definição: "Um telefone celular que executa muitas das funções de um computador, normalmente com uma interface touchscreen, acesso à Internet, e um sistema operacional capaz de executar aplicativos baixados."

Podemos dividir esses aparelhos com base em seu sistema operacional. Os três principais são: Android, iOS e Windows Phone.

O Android (GOOGLE, 2014a) é um sistema operacional para plataformas móveis, desenvolvido pelo Google. Segundo dados do IDC (2014a), no segundo trimestre de 2014 as vendas de aparelhos com Android totalizaram 84,7% de todo o mercado mundial. Em todo o mundo, mais de 1 bilhão de dispositivos funcionam com a plataforma.

Na PlayStore (GOOGLE, 2014d) (loja de aplicações Android), são mais de 1,4 bilhões de aplicativos disponíveis. O sistema vem embutido em aparelhos das marcas Samsung – a que mais vende aparelhos em todo o mundo, com 23,8% de todo o mercado -, LG, Sony, Motorola, etc (IDC, 2014b).

O segundo sistema operacional da lista, o IOS, é o sistema desenvolvido pela Apple, e está presente em todos os smartphones da empresa, com 13,8% das vendas em todo o mundo. O sistema é exclusivo para aparelhos da própria marca (APPLE, 2014a).

A App Store, loja de aplicativos para IOS, possui mais de 85 mil aplicativos, e atingiu mais de 2 bilhões de dowloads. A empresa está na segunda posição na ranking de vendas de smartphones, com 12% das vendas em todo mundo, tendo o IPhone como o principal aparelho da marca (APPLE, 2014a).

O Windows Phone é o terceiro de nossa lista. O sistema produzido pela Microsoft, é o mais novo entre os três. O sistema estava em 2,7% dos aparelhos vendidos no segundo trimestre de 2014 (IDC, 2014b).

A loja de aplicativos, Windows Phone Store, possui mais de 120 mil aplicativos entre gratuitos e pagos, disponíveis para download. O sistema vem instalado em aparelhos Microsoft (MICROSOFT, 2014b).

2.1.1 Aparelhos

Há uma enorme variedade de aparelhos a venda, variando em suas especificações, as quais atendem aos mais variados gostos e necessidades. Na Tabela 1 está uma descrição dos principais aparelhos disponíveis atualmente no mercado.

Dentre os modelos descritos na Tabela 1, o único a não possuir GPU (*Graphics Processing Unit* - Unidade de Processamento Gráfico) é o iPhone 6. Os demais possuem o modelo Adreno 330.

2.2 Android

O Android SO é um sistema operacional com foco em dispositivos móveis desenvolvido pela Google em parceria com mais de 300 empresas de hardware, software e operadoras, utilizando o sistema operacional Linux como base (GOOGLE, 2014a), sendo o sistema operacional para dispositivos móveis mais usado em todo mundo, presente em aproximadamente 1 bilhão de dispositivos, e a cada dia adquire cerca de 1 milhão de novos usuários (GOOGLE, 2014a). No segundo semestre de 2014 foram vendidos 255,3 milhões de apa-

Tabela 1 – Descrição de alguns dos principais smartphones disponíveis no mercado brasileiro

Aparelho	**Sistema Operacional**	**Chipset**	**Processador**	**Memória RAM (GB)**	**Armazenamento Interno (GB)**	**Tela (pol.)**	**Câmera (Mp)**
Nokia Lumia 1520 (MICROSOFT, 2014a)	Windows Phone 8 Black	Qualcomm MSM8974 Snapdragon 800	2.2 GHz Quad Core	2	32	6	20
LG Nexus 5 (GOOGLE, 2014c)	Android 4.4.4 Kit Kat	Qualcomm MSM8974 Snapdragon 800	2.3 GHz Quad Core	2	32	4.95	8
Samsung Galaxy S5 (SAMSUNG, 2014)	Android 4.4.2 TouchWiz UI KitKat	Qualcomm Snapdragon 801 Krait 400	2.5 GHz Quad Core	2	32	5.1	16
LG G3 (LG, 2014)	Android 5.0 LG Optimus UI v3.0 Lollipop	Qualcomm MSM8975AC Snapdragon 801	2.5 GHz Quad Core	2	16	5.5	13
Sony Xperia Z3 (SONY, 2014)	Android 4.4.4 KitKat	Qualcomm MSM8974-AC Snapdragon 801	2.5 GHz Quad Core	3	16	5.2	20.7
Apple iPhone 6 (APPLE, 2014b)	iOS 8	Apple A8	1.4 GHz Dual Core	1	16 a 128	4.7	8

relhos Android, um aumento de 33,3% em um ano, dando a Google a primeira posição na lista dos mais vendidos com 84,7% do total das vendas (EXAME, 2014) .

Esses são só alguns dos números expressivos conseguidos pelo Android, que confirmam a importância desse Sistema Operacional hoje, e alimenta grandes expectativas a respeito da plataforma para os próximos anos, visto que o mesmo está implantado em TV's, equipamentos diversos e veículos automotores (GOOGLE, 2014a) .

2.2.1 Google Play

O Android conta com a maior loja online de aplicativos, a Google Play Store[1]. Nesta, qualquer usuário da plataforma Android encontrará mais de 1,3 milhões de aplicativos (TECMUNDO, 2014), além de filmes e livros.

Entre junho de 2008 e junho 2014 foram feitos aproximadamente 75 bilhões de downloads na Google Play, 15 bilhões deles feitos entre os meses de janeiro e agosto de 2014 (STATISTA, 2014).

2.2.2 Características

Uma aplicação Android é escrita na linguagem de programação Java, e em seguida é compilada pelas ferramentas do SDK (*Software Development Kit* - Kit para desenvolvimento de Software). Todo o conteúdo da aplicação é colocado em um APK (*Android Package* - Pacote Android), sendo esse, o arquivo que será instalado no dispositivo.

Cada aplicação Android é constituída de vários componentes distintos que podem ser utilizados individualmente. Esses componentes podem ser de quatro tipos diferentes: Atividades, Serviços, Provedor de Conteúdo e *Broadcast Receivers*. Cada componente pode utilizar outro componente usando um *Intent*, podendo esse componente fazer parte de outra aplicação.

Para fazer o gerenciamento das aplicações, é adotado o modelo com multiusuários, no qual cada aplicação é um usuário diferente com ID único, e concede a este a permissão exclusiva para acesso aos arquivos necessários para seu funcionamento.

Cada aplicação está isolada das outras, tendo sua própria maquina virtual e seu processo. Juntamente com cada processo ele inicia todo o conteúdo necessário à aplicação e à medida que não são mais necessários vão sendo encerrados. Caso seja necessário mais memória, um processo menos importante é encerrado para liberar espaço para um com maior importância.

É possível criar interfaces diferentes para cada tipo de aparelho. Através do XML de layout, pode-se especificar interfaces diferentes para cada tamanho de display, permitindo interfaces mais agradáveis e que usufruam ao máximo das configurações oferecidas pelo dispositivo.

O arquivo *Manifest* XML define todas as permissões, componentes e requisitos para a bom funcionamento da aplicação. As permissões se fazem necessário quando o aplicativo precisa fazer uso de um hardware especifico como a câmera ou wi-fi. Os requisitos são a versão mínima do sdk necessário, versão do Android, e caso seja necessário algum hardware especifico (GOOGLE, 2014b).

[1] https://play.google.com/store

2.2.3 Arquitetura

O Android tem como base o Linux [2]. Em seu Kernel, encontram-se os serviços básicos para o funcionamento do dispositivo: segurança, gerenciamento de memória, arquivos e conexões, drivers como o de câmera e conexão wireless, gerenciamento de energia e comunicação entre processos, conforme ilustra a Figura 2 (PORTER, 2014).

O Android possui diversas bibliotecas já embutidas. Ele utiliza uma biblioteca System C (Bionic Libc) que é derivada das BSD's Standart Library, bibliotecas escritas em linguagem C, que fornece opções básicas para sistemas operacionais como manipulação de strings, operações matemáticas, alocação de memória, etc. Ela é própria para dispositivos móveis, ja que é adaptada para CPU's com baixo pulso de clock. Utiliza também o SQLite como banco de dados, Open GL para suporte a parte gráfica, frameworks para mídia, webkit para aplicações web e outras bibliotecas (PORTER, 2014).

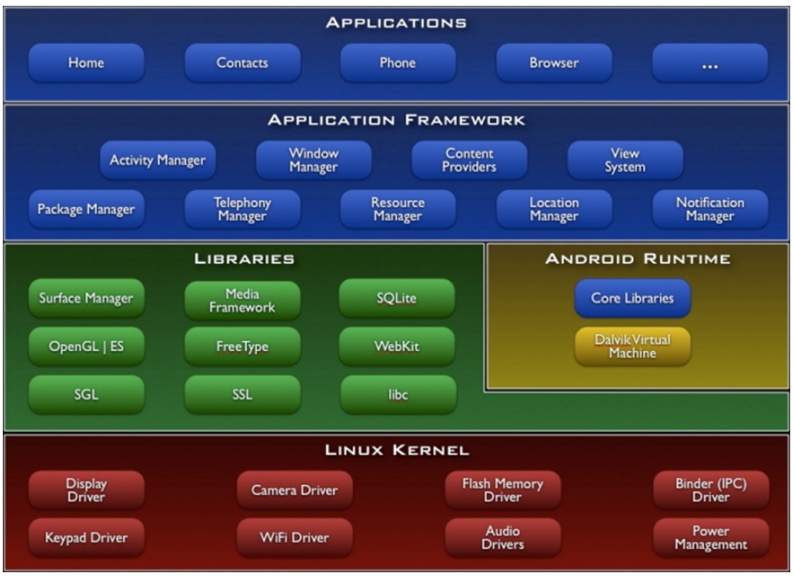

Figura 2 – Organização da arquitetura do Android (MOBILTEC, 2014).

Possui duas bibliotecas principais,as quais são encarregadas da execução das aplicações: Core Java Libraries e Dalvik Virtual Machine. Na Core Java estão as bibliotecas básicas do Java, como Java.* e Javax.*, além da Android.* e JUnit.*, que são próprias para o Android. A Dalvik Virtual Machine é a maquina virtual responsável pela execução das aplicações. Esta foi projeta para ser usada em CPU's com poder reduzido de processamento, pouca memória RAM e bateria com curta duração (PORTER, 2014).

No nível acima das bibliotecas estão os frameworks de aplicação. Nesse ponto estão o framework de gerenciamento de janelas, gerenciador de recursos não compiláveis, gerenciador de atividades, Content Provider (que faz o compartilhamento de dados entre aplicações), gerenciador de localização, gerenciador de visualizações, dentre outros (PORTER, 2014).

[2] http://www.linux.com/

No último nível estão as aplicações. Entre essas aplicações, estão as básicas em um dispositivo como, agenda, realização de chamadas, browser para acessar paginas web, menu principal, que podem ser substituídos conforme o desejo de usuário. Juntamente com essas aplicações ficarão as demais aplicações que forem instaladas no dispositivo (PORTER, 2014).

2.2.4 Desenvolvimento de Aplicações

O Google disponibiliza duas IDE's para desenvolvimento de novas aplicações: ADT plugin e o Android Studio.

O ADT (Figura 3) é um plugin que pode ser utilizado juntamente com IDE's como o Eclipse [3] ou NetBeans [4], (IDE's Open Source, utilizadas para programação). Apresenta como vantagem, o fato de ser apenas um plugin, sendo um arquivo pequeno, não necessita da instalação de uma nova IDE, podendo ser usado no ambiente de desenvolvimento em que o usuário esteja habituado. (GOOGLE, 2014b).

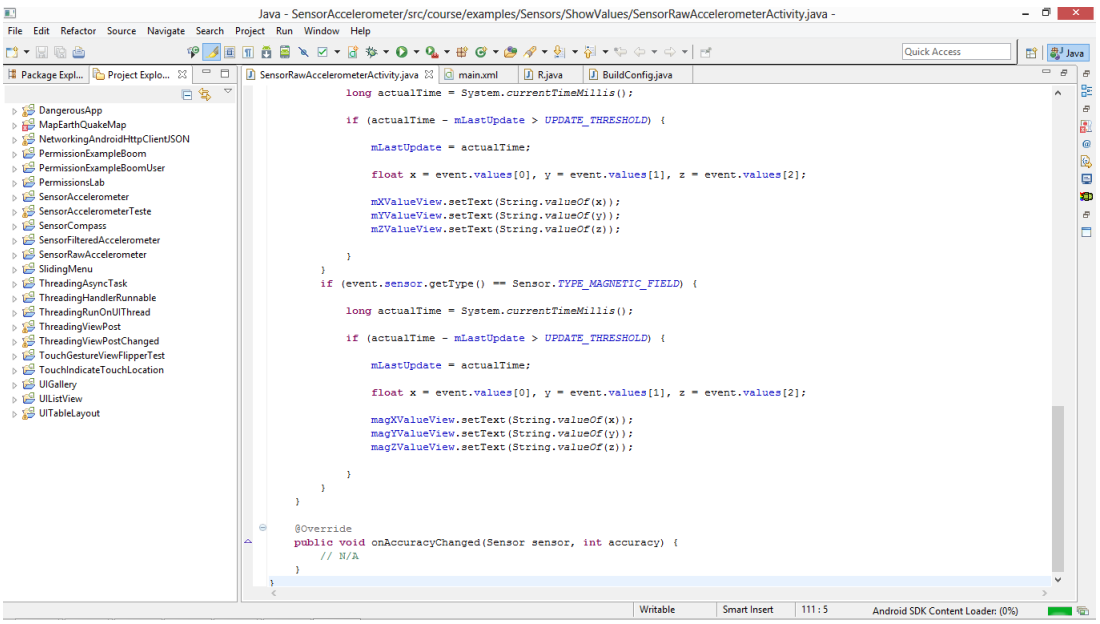

Figura 3 – ADT (Android Developement Tool).

O Android Studio (Figura 4) é uma IDE mais recente, tendo sua versão beta lançada em 2014, estando em sua versão final 1.0.2. É um ambiente criado com base no IntelliJ IDEA [5], sendo a primeira específica para desenvolvimento de aplicações Android. Possui integração com o Gradle (Sistema de automação de builds)[6] e GitHub (Serviço

[3] https://eclipse.org/
[4] https://netbeans.org/
[5] https://www.jetbrains.com/idea/
[6] https://www.gradle.org/

de Web Hosting Compartilhado para projetos que usam o controle de versionamento Git)[7].(GOOGLE, 2014b).

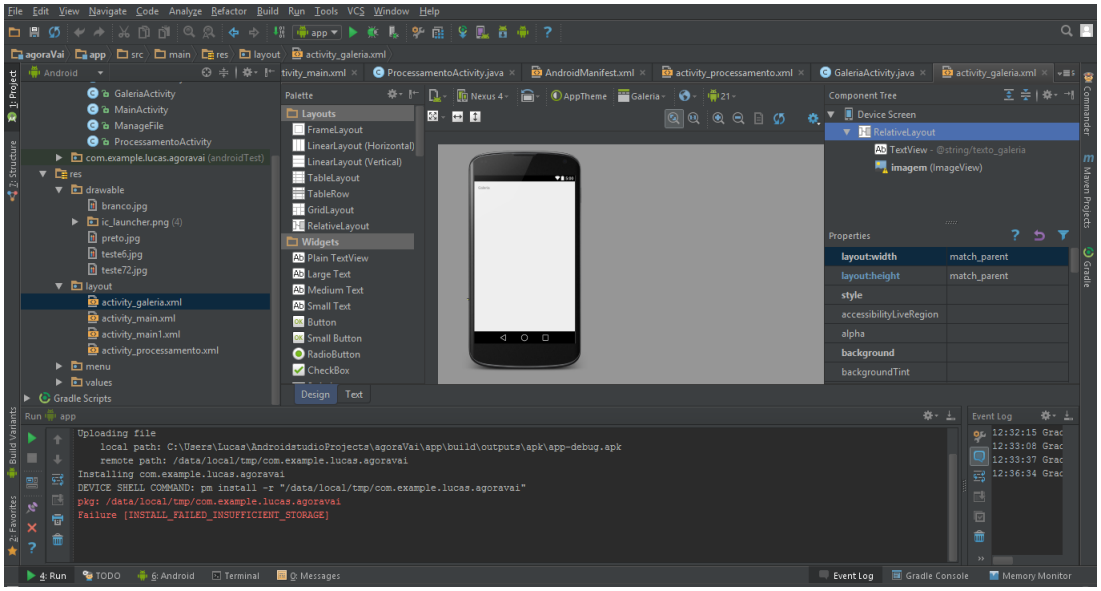

Figura 4 – Android Studio 1.0.2

Ambas as opções tem seu download disponibilizado pelo Google [8] [9] e já possuem emulador para tablets, smartphones, TV's e relógios que utilizem o sistema Android, para a realização de testes sem a necessidade do uso de um aparelho reais.

O Google também disponibiliza um site com informações para desenvolvedores. Além de informações sobre as API's, bibliotecas e interfaces disponibilizadas, estão alguns tutoriais e exemplos úteis para quem está iniciando na linguagem.

Nesse trabalho,foi utilizado ambas as plataformas durante o desenvolvimento da aplicação. O ADT foi utilizado no inicio, mas devido a problemas momentâneos enfrentados pela plataforma e vantagens oferecidas pelo concorrente,foi substituído pelo Android Studio.

2.3 Processamento Paralelo

Aumentar a capacidade de processamento em computadores é um dos objetivos principais e sempre presente entre os desenvolvedores de novas máquinas. Ano após ano, novos modelos, com novos processadores e configurações de hardware surgem, sempre superando os modelos anteriores. O grande problema nesse fato, é que existe limite físico para o hardware, limite o qual está muito próximo de ser alcançado (LLOYD, 2000).

Uma forma encontrada para contornar esse problema, foi o processamento paralelo. Ele é uma forma eficiente de processar informações, baseado na execução simultânea de

[7] https://github.com/
[8] http://developer.android.com/sdk/index.html
[9] http://developer.android.com/tools/help/adt.html

partes de um software, enquanto nos modelos antigos, isso é feito de maneira sequencial (CLUA, 2004).

Existem três formas principais de paralelismo segundo Andrews (1999): Sistema MultiThread, Sistema Distribuído e Computação Paralela.

Sistema Multi threaded, é um tipo de sistema que possui mais processos do que processadores disponíveis. Normalmente, este tipo de sistema é usado para gerenciar tarefas independentes. Normalmente os sistemas operacionais se enquadram nessa categoria.

Sistema distribuído se baseia em vários processos, divididos em várias máquinas interligadas. Cada uma realiza uma parte do processamento, trocando mensagens entre si, sendo o resultado final obtido pela junção de tudo que foi processado.

Computação paralela, consiste em dividir um programa em "várias partes" independentes, que são divididas e executas nos vários processadores disponíveis, visando resolver um problema mais rapidamente.

Para que um programa seja paralelizável, precisamos definir isso durante seu desenvolvimento. Algumas das formas mais comuns de fazer isso é processos, threads, PVM e MPI (CLUA, 2004).

2.3.1 Processo

Processo é: um programa em execução, composto por seu código, variáveis, registradores e demais conteúdo necessário para sua execução.

Os sistemas operacionais multitarefas, possuem comandos, funções, chamadas de sistema, que auxiliam na criação e gerenciamento desses processos. Caso a máquina possua mais de um núcleo, ocorre a divisão dos processos nos processadores disponíveis (TANENBAUM, 1992).

2.3.2 Thread

Em um sistema comum, o fluxo de execução segue um único caminho, sequencial, com uma única linha de controle. Nesse caso dizemos que esse é um programa monothread, ou, seja que segue apenas um fluxo. Sistemas operacionais modernos, dão suporte a mais de uma linha de controle por processo conhecidas como threads, as quais permitem que várias partes do sistema sejam executadas concomitantemente (TANENBAUM, 1992).

É uma opção muito utilizada para realizar processamento paralelo por serem simples, de baixo custo computacional, fácil de criar e gerenciar. Linguagens de programação como Java, C, C++ e Ruby possuem suporte a threads, ficando a cabo do programador organizar a estrutura do sistema nesse modelo durante a criação do código (BUENO, 2002).

2.3.3 PVM

Parallel Virtual Machine (Máquina Virtual Paralela), é a biblioteca para processamento distribuído mais utilizada. Ela permite que computadores com arquiteturas diferentes trabalhem juntos ligados por uma rede, como se fossem uma única máquina. Ela lida com o roteamento de mensagens, escalonamento de processos e conversão de dados (SCIENCE; CSMD, 2015).

A comunicação entre as máquinas é realizada através de trocas de mensagens. São utilizadas algumas primitivas para que isso seja feito, as quais foram desenvolvidas especialmente para ambientes heterogêneos (COLOURIS; DOLLIMORE; KINDBERG, 1996).

2.3.4 MPI

MPI (*Message Passing Interface* - Interface para Passagem de Mensagens) é um conjunto de funções para C e Fortran, que provê uma interface para comunicação entre as máquinas de um cluster. Tem suporte para mensagens broadcast (envia a mensagem para todas as máquinas da rede) e multicast (um conjunto específico de máquinas da rede). Tem opções mais avançadas que PVM e um melhor controle sobre o tratamento das mensagens enviadas para outro cluster (BUENO, 2002).

2.4 Problemas de Deficiências no Café

Todas as plantas necessitam de nutrientes para o seu desenvolvimento. O Café não foge a regra e necessita de cuidados, pois se uma deficiência surge, deve-se supri-la o mais rápido possível para não perder a produção toda.

Cabe ressaltar que segundo Malavolta et al. (1993), o café necessita de alguns elementos básicos para seu bom desenvolvimento:

- Macro nutrientes: nitrogênio (N), fosforo (P), potássio (K), cálcio (Ca), magnésio (Mg) e enxofre (S);
- Micronutrientes: boro (B), cloro (Cl), cobalto (Co), cobre (Cu), ferro (Fe), manganês (Mn), molibdênio (Mo), níquel (Ni), selênio (Se), silício (Si) e zinco (Zn).

A falta ou excesso de cada um desses nutrientes podem influenciar na saúde da plantação e dependendo da gravidade, na qualidade dos frutos (MALAVOLTA et al., 1993).

Na Figura (5), observa-se algumas folhas com deficiência de ferro. Essa deficiência é caracterizada pelo amarelecimento das folhas mais novas e as nervuras permanecem verdes.

A deficiência de Nitrogênio (Figura 6), também leva ao amarelecimento das folhas, sendo a principal diferença, o fato de se manifestar inicialmente nas bordas da folha e na nervura principal, além de afligir as folhas mais velhas.

Figura 5 – Folhas com deficiência de Ferro (IPNI BRASIL, 2014).

A deficiência de potássio (Figura 7) também aflige as folhas mais velhas, que apresentam amarelecimento das pontas e margens, que em estagio avançado secam, ficando marrons ou pretas (MALAVOLTA et al., 1993).

Figura 6 – Folhas com deficiência de Nitrogênio (IPNI BRASIL, 2014).

Todas essas deficiências têm como sinal de sua presença o amarelecimento das folhas, tendo como diferencial o tom de amarelo, o tipo de folha que desenvolve esse sintoma e o local do amarelecimento.

Com base nesses diferenciais apresentados, fica evidente que é possível diagnosticar se uma amostra de café está deficiente ou não, podendo chegar a identificação de qual

Figura 7 – Folhas com deficiência de Potássio (IPNI BRASIL, 2014).

deficiência.

Capítulo 3

Trabalhos Relacionados

Neste capítulo, será apresentada a descrição de alguns trabalhos relacionados ao realizado. Será uma descrição simples, dando um enfoque maior nos pontos mais importantes e que contribuíram para o desenvolvimento desse trabalho.

3.1 BioMobile: Sistema Biométrico de Identificação de Usuários em Dispositivos Móveis na Plataforma Android

O BioMobile é um trabalho desenvolvido para avaliar a viabilidade do desenvolvimento de um sistema para proteção de informações em dispositivos móveis com base no reconhecimento facial por vídeo.

O sistema foi desenvolvido por Farina e Marana (2012) para aparelhos que usem a plataforma Android. Essa escolha foi feita com base em dados estatísticos, os quais mostraram claramente que o Android, é o sistema operacional mais usado em dispositivos móveis e também por ser um software livre e de código aberto.

O sistema pode ser divido nas seguintes etapas:

- Aquisição do vídeo: o usuário faz a captura da imagem facial através de um vídeo, permitindo que o próprio algoritmo escolha os quadros a serem utilizados;
- Extração dos quadros: o algoritmo seleciona um quadro do inicio, um do meio e um do final do vídeo, variando assim a posição, condições de iluminação;
- Detecção facial e pré processamento: em seguida é feita a detecção facial utilizando o algoritmo Viola-Jones, já implementado na biblioteca OpenCV. Se uma face é detectada, a imagem é pre processada para ajustá-la para a extração das características;

- ❏ Extração das características: é utilizado o operador LBP (*Local Binary Patterns*), que é um ótimo descritor de texturas;
- ❏ Reconhecimento: um banco de dados com imagens já cadastradas é usado, com uma busca sendo feita de qual face cadastrada se aproxima mais da face que esta sendo analisada. Assim que um registro é encontrado, é feita um teste para garantir a autenticidade da pessoa, utilizando o chi-quadrado.

O grande diferencial deste método com os demais, é que todo o processamento e analise, é feito no dispositivo, não necessitando de um banco de dados e recursos externos. Os resultados obtidos com os testes realizados, mostraram que mesmo com as limitações de memória e processamento, é possível fazer o reconhecimento fácil de usuários com smartphones ou tablets (FARINA; MARANA, 2012).

3.2 Detection, Extraction and Text Translation in Digital Images using Android Platform

Esse trabalho realizado por Rabachini et al. (2014), teve como objetivo desenvolver uma aplicação para plataformas Android, que fosse capaz de ler e traduzir textos presentes em placas de rodovias, aeroportos, restaurantes, sinalizações, auxiliando indivíduos que não compreenda perfeitamente o idioma do país visitado.

Podemos dividir a aplicação nos seguintes módulos:

- ❏ Aquisição da imagem: o usuário captura uma imagem do local analisado, utilizando a câmera do próprio dispositivo;
- ❏ Detecção de regiões de texto: para essa etapa foram utilizados vários algoritmos. Para detecção de textos é utilizada a biblioteca de robótica Literate PR2, mais precisamente o algoritmo para detecção de textos em ambientes internos. Também foram utilizados algoritmos de processamento de imagens como detecção de bordas e blur da biblioteca OpenCV, além de alguns algoritmos desenvolvidos especificamente para esse trabalho, os quais utilizaram a própria API do Android;
- ❏ Extração de texto: depois de localizar os caracteres, é preciso converter isso em texto. Para isso é utilizado o Tesseract OCR (desenvolvido em C/C++, mas que possui uma versão para Android), que é aplicado após alguns ajustes na imagem feitos com filtro de mediana, máscara, remoção de ruído;
- ❏ Tradução do texto: após o obter os caracteres, é preciso traduzir a mensagem. Primeiramente, a mensagem é verifica por um corretor ortográfico, Enchant. Para a tradução, foi utilizada a API MyMemory, de acesso publico e com suporte a diversos idiomas;

- ❑ Apresentação do resultado: o resultado é exibido em uma imagem similar a original, tendo os caracteres apagados e a mensagem traduzida sendo exibida nesse lugar.

O sistema foi desenvolvido para realizar traduções do inglês para português, obtendo resultados satisfatórios, com um tempo de processamento um pouco alto se comparado ao de concorrentes, mas dentro do limite aceitável.

Um fator de grande influência no tempo gasto, é o fato de a tradução ser feita em um servidor, o que demanda acesso à rede e um tempo maior de resposta. Todo o resto do processamento é feito no próprio dispositivo (RABACHINI et al., 2014).

3.3 DNA Shot - Reconhecimento de DNA através de imagens

Nesse trabalho, Beltzac, Hess e Yamaguchi (2014) desenvolveram uma aplicação que tinha como objetivo ler uma cadeia de caracteres que representam um DNA, diretamente de uma fotografia, em seguida analisar esse DNA em uma base de dados e mostrar o resultado da pesquisa para o usuário.

A consulta seria feita online, onde a principal vantagem oferecida pela aplicação, seria a de que o usuário não teria mais de digitar toda a sequência de caracteres referentes a um DNA, sequencia a qual pode passar de 100 caracteres.

Para obter o resultado desejado, foram feitos estudos sobre a compressão de imagens e a perda de qualidade, testes com diferentes fontes e testes de iluminação .

Foi utilizado o algoritmo thresholding para diminuir sombreados nas imagens e o algoritmo OCR para detecção dos caracteres presentes na imagem, juntamente com filtros desenvolvidos para limpar a imagem.

O aplicativo conseguiu cumprir seu papel, conseguindo ler a sequencia de caracteres tanto por uma imagem arquivada, quando por uma fotografia tirada na hora, processar a imagem e extrair a sequência de caracteres, em seguida, buscar pelo DNA informado na base de dados BLAST(*Basic Local Alignment Search Tool* - Ferramenta básica para busca local de alinhamento) e retornar esse resultado para o usuário (BELTZAC; HESS; YAMAGUCHI, 2014).

3.4 Processamento de áudio em tempo real em dispositivos não convencionais

Esse trabalho apresenta um estudo de caso entre do processamento de áudio em tempo real em três plataformas distintas: Arduino, GPGPU e Android. Neste estudo, Bianchi (2011) visava encontrar os limites computacionais de cada método para o processamento de audio.

Nos sistemas desenvolvidos, foram utilizados filtros básicos de suavização ou diferenças, diversos tipos de efeitos, reverberação, morphing em tempo real, phase vocode e auralisação.

O processamento paralelo é de grande utilidade para processamento de sinais, onde através da divisão do algoritmos em partes, podemos executá-las simultaneamente, obtendo o resultado final mais rapidamente.

Dentre as plataformas disponíveis, o processamento paralelo foi implantado com sucesso na GPGPU, já que ela apresenta ferramentas e estruturas próprias para essa atividade.

No caso do Android, foi analisada apenas a possibilidade da implementação, a qual foi comprovada ser possível, utilizando GPU presente no próprio Smartphone (BIANCHI, 2011).

Todos estes trabalhos analisados foram úteis por provar a viabilidade do desenvolvimento da aplicação. Além disso, elas nos mostram problemas possíveis de ocorrer, assim foi possível evitá-los.

Capítulo 4

Desenvolvimento

Neste capítulo, serão abordados aspectos do funcionamento do sistema. Inicialmente é apresentada uma visão geral do sistema, principais conceitos, a estrutura geral do funcionamento, sendo posteriormente detalhados.

4.1 Descrição geral

O sistema desenvolvido recebe a imagem de uma folha de café e como resultado, informa se essa folha está deficiente ou saudável. Para alcançar esse resultado, são feitas várias análises e processamentos na imagem. Essas etapas estão representadas no esquema da Figura 8 e serão discutidas em seguida.

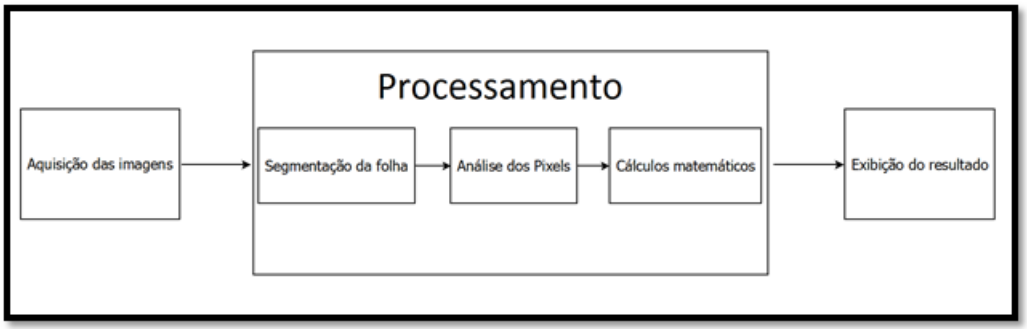

Figura 8 – Fluxograma detalhando as etapas do algoritmo

4.1.1 Aquisição de imagens

O primeiro passo e o mais importante dentro do sistema é a aquisição das imagens. O resultado final será obtido com base nessa imagem, sendo que caso haja problemas na luminosidade, resolução, ou qualquer outro fator que possa interferir na qualidade da imagem, pode consequentemente interferir na análise e no resultado final.

4.1.2 Processamento

O processamento pode ser dividido em três etapas: segmentação da folha, análise dos pixels e cálculos matemáticos.

4.1.2.1 Segmentação da folha

Assim que a folha é selecionada (e nesse ponto, assume-se que foram tomadas as devidas precauções para garantir a qualidade de imagem), precisa-se extrair apenas a região ocupada pela folha, a parte que realmente será útil na análise. Para isso, foi desenvolvido um filtro que através dos canais RGB, consegue fazer essa separação.

Para a construção desse filtro foram feitos varios testes variando os parametros de canal (vermelho,verde e azul) e cor do pixel (0 a 256). O o canal que obteve o melhor resultado foi o azul e o valor 130 fui utilizado como limiar para o filtro. Com esse valor foi obtido um melhor aproveitamento da imagem, visto que com valores maiores partes da folha eram eliminadas e com valores menores as sombras ao redor das bordas da folha permaneciam.

A imagem é percorrida pixel por pixel, sendo feita em cada um, uma verificação utilizando o filtro. Se for maior que 130, esse valor é desprezado, ele não faz parte da folha e o valor desse pixel é setado para 0 (em RGB, um pixel de valor 0 tem a cor preto). Se for menor que 130, esse pixel faz parte da folha e tem seu valor setado para 255 (em RGB, um pixel de valor 255 tem a cor branco).

Esse filtro foi suficiente para remover as sombras que porventura possam acompanhar a folha, juntamente com tudo aquilo que não pertence a amostra.

Agora, têm-se duas imagens, a original e a binária. Para obter a imagem que realmente será processada, é feita uma subtração de imagens:

- ❏ se o valor do pixel na imagem binária for 0: mantenho o pixel como 0;
- ❏ se o valor do pixel na imagem binária for 1: mantenho o valor do pixel da imagem original.

Ao final desse processo, obtêm-se a imagem original sem o fundo, sem sombras, apenas com a folha.

4.1.2.2 Análise dos pixels

Após garantir que todo o resto foi removido da imagem, que agora restou apenas a folha, a imagem está pronta para ser analisada.

A imagem será novamente percorrida e o valor de cada canal do RGB é somado e armazenado para cálculos posteriores.

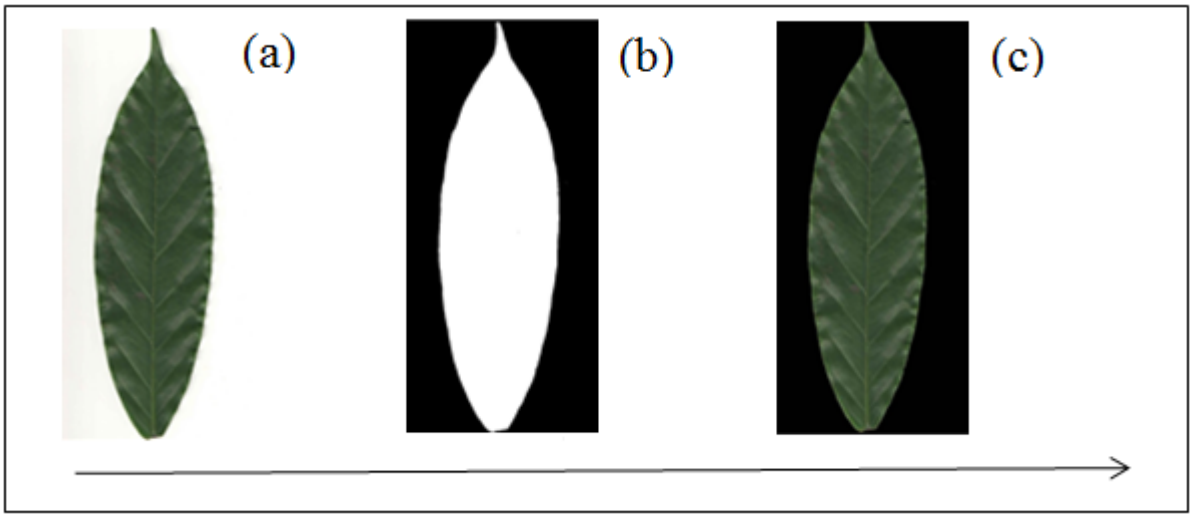

Figura 9 – Todas as transformações que são feitas em cada folha. Imagem original (a), imagem binária (b), imagem segmentada (c).

4.1.2.3 Cálculos matemáticos

Na busca para encontrar algum padrão, ou valor que fosse suficiente para distinguir folhas saudáveis de folhas deficientes, foram realizadas analises com cálculos estatísticos.

Inicialmente, optou-se pela média aritmética, mas o resultado não foi satisfatório. Em seguida, foi testado o valor do desvio padrão, também sem sucesso. O terceiro calculo testado foi o do terceiro momento, o qual gerou resultados satisfatórios.

Os detalhes destes testes estão disponíveis em Cunha et al. (2013), um estudo realizado anteriormente, utilizado como base para este trabalho.

Média Aritmética:

$$\bar{X} = \frac{\sum_{i=1}^{n} Xi}{n}$$

Desvio Padrão:

$$s = \sqrt{\frac{(Xi - \bar{X})^2}{n}}$$

Terceiro Momento:

$$\mu_3 = \frac{1}{L} \sum_{i=0}^{L-1} (z_i - m)^3$$

4.1.3 Exibição do Resultado

Após obter o valor do terceiro momento, é feita uma análise com esse número:

$$\begin{cases} \mu_3 \geq 0,7 \longrightarrow Saudável \\ < 0,7 \longrightarrow Deficiente \end{cases}$$

O limiar 0,7 para o terceiro momento também foi encontrado após testes. 145 amostras entre saudáveis e deficientes colhidas em cafezais da região de Monte Carmelo, foram

utilizadas para a realização de testes. As amostras saudáveis tinham o valor do terceiro momento maior ou igual a 0,7; as que possuíam alguma deficiência tinha esse valor menor que o limiar.

O resultado é mostrado na tela com uma mensagem dizendo se amostra está realmente com alguma deficiência ou não.

4.2 Android

A aplicação faz acesso à câmera para permitir ao usuário fotografar a amostra escolhida para ser analisada. Para isso, necessita-se seguir alguns passos, os quais estão definidos na documentação da linguagem.

4.2.1 Android Manifest

Android Manifest é um arquivo localizado na raiz do projeto, que contêm todas as configurações necessárias para a execução do da aplicação, como o nome do pacote, das atividades e demais configurações.

```
<uses-permission android:name="android.permission.CAMERA" />
```

O código anterior precisa ser adicionado ao arquivo Android Manifest. Ele "diz" ao dispositivo que esta aplicação pode querer utilizar a câmera.

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

Esse outro código também deve ser colocado no arquivo Manifest. Ele informa o dispositivo que a aplicação pode precisar armazenar informações na memória externa. Esse comando pede a permissão para que as imagens capturadas pela câmera sejam armazenadas e depois processadas.

Após alterar o arquivo de configurações, a próxima etapa é a invocação da câmera dentro da aplicação.

Primeiro cria-se uma intenção de uso da câmera. Um objeto do tipo Intent é criado exatamente conforme o próximo código.

```
Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
```

Após criar a intenção, defini-se o local onde a imagem será armazenada e o nome da imagem. No próximo código, a imagem esta sendo armazenada na mídia externa.

```
fileUri = getOutputMediaFileUri(MEDIA_TYPE_IMAGE);
intent.putExtra(MediaStore.EXTRA_OUTPUT, fileUri);
```

Após criar a intenção e definir as opções de armazenamento, utiliza-se o próximo código para chamar a atividade padrão da câmera para dispositivos Android.

```
startActivityForResult(intent, CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE);
```

A resposta da câmera é passada para o método *onActivityResult* juntamente com três valores: dois valores inteiros de controle e um intent contendo o endereço onde a imagem foi armazenada, o mesmo que foi passado na intenção da câmera.

Verifica-se os dois valores de controle e caso eles sejam respectivamente 100 e 200, a fotografia foi armazenada com sucesso. O código exemplifica os passos descritos.

```
protected void onActivityResult(int requestCode, int resultCode, Intent data)
if (requestCode == 100) {
if (resultCode == 200){
// Imagem capturada e armazenada com sucesso
} else if (resultCode == RESULT_CANCELED) {
// Usuario cancelou a fotografia
} else {
// Falha na captura da imagem
}
}
}
```

4.2.2 Acesso à galeria

Outra opção disponível ao usuário é utilizar uma imagem da galeria do dispositivo. Essa opção permite uma maior flexibilidade ao usuário na hora de adquirir as imagens, podendo receber a imagem de outras fontes.

Para isso, foi criada uma intenção, similar a usada para acessar a câmera. Nela define-se o tipo de dados que se quer receber (imagens) e informa-se que é uma ação para aquisição de conteúdo, juntamente com os demais argumentos necessários.

```
intent.setType("image/*");
intent.setAction(Intent.ACTION_GET_CONTENT);
startActivityForResult(Intent.createChooser(intent,"Select␣Picture"),
SELECT_PICTURE);
```

O resultado também é retornado para o método *onActivityResult*, mesmo método usado para o retorno da câmera, com uma diferença em dos valores retornados, justamente o que me informa que o arquivo veio da memória interna e não da câmera.

Depois disso, basta utilizar o terceiro valor, o que vem com a intent, ler o caminho onde a imagem está armazenada na forma de uma *stream*, decodificar essa *stream* transformando uma imagem do tipo *Bitmap*, pronta para ser utilizada.

```
protected void onActivityResult(int requestCode, int resultCode, Intent data)
if (requestCode == CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE){
if (resultCode == RESULT_OK){
fis = new FileInputStream(imageFile);
Bitmap picture = BitmapFactory.decodeStream(fis);
} else if (resultCode == RESULT_CANCELED) {
// Usuario cancelou a leitura
```

```
} else {
// Falha na captura da imagem
}
}
}
```

4.2.3 Threads

Uma das formas mais simples de realizar processamento paralelo em Android é usando Threads. Através da interface *Runnable*, consegue-se implementar e gerenciar threads.

Assim como no Java, quando implementa-se a interface *Runnable*, necessita-se reescrever o método *run*, método no qual se descreve o que se deseje que a Thread realize.

Possui os métodos usuais para thread: start, stop, sleep, resume, isAlive, destroy, métodos comuns para quem usa Java. A principal diferença da implementação de thread em Android para a implementação em Java, é que as aplicações possuem uma thread principal, que tem o papel de gerenciar a interface gráfica e processamentos leves. Quando se quer realizar algum processamento mais pesado utiliza-se uma thread secundária. As alterações feitas pela thread secundária são enviadas para a thread principal através de um *handler*. Um *hand*ler tem o papel de levar as mudanças feitas por threads secundárias, mas que só podem ser efetuadas pela thread principal.

```
public void onClick(View v) {
new Thread(new Runnable() {
public void run() {
Bitmap b =
loadImageFromNetwork("http://example.com/image.png");
mImageView.setImageBitmap(b);
}
}).start();
}
```

O exemplo anterior mostra a criação de uma thread e reescrita do método run. Nesse exemplo, a thread criada está fazendo a leitura de uma imagem diretamente da web e carregando essa imagem em um *ImageView*.

Capítulo 5

Resultados

Esse capítulo apresentará todos os resultados obtidos: valores obtidos nos testes, capturas de tela da aplicação e demais artefatos desenvolvidos.

5.1 Aplicação para detecção de deficiência de café

Ao final deste trabalho, conseguiu-se desenvolver a aplicação proposta inicialmente, uma aplicação desenvolvida para smartphones que utilizam Android e, que fosse capaz de informar ao usuário se uma amostra de folha de café possui alguma deficiência nutricional ou não.

Figura 10 – Tela Inicial da Aplicação

A tela inicial (Figura 10) possui uma imagem de um cafezal, que ajuda a identificar o sistema, juntamente com o nome e dois botões, que levam o usuário diretamente para a etapa da aquisição das imagens: o botão “Galeria” e o botão “Fotografar”.

Os rótulos são bem simples e específicos, deixando bem claro para o usuário o que vai acontecer se ele clicar ali. Se clicar no botão “Galeria”, a galeria de imagens do aparelho será exibida na tela (Figura 11), para que seja escolhida qual imagem será analisada. Caso clique em “Fotografar”, a câmera padrão do dispositivo será iniciada para a captura da imagem da amostra.

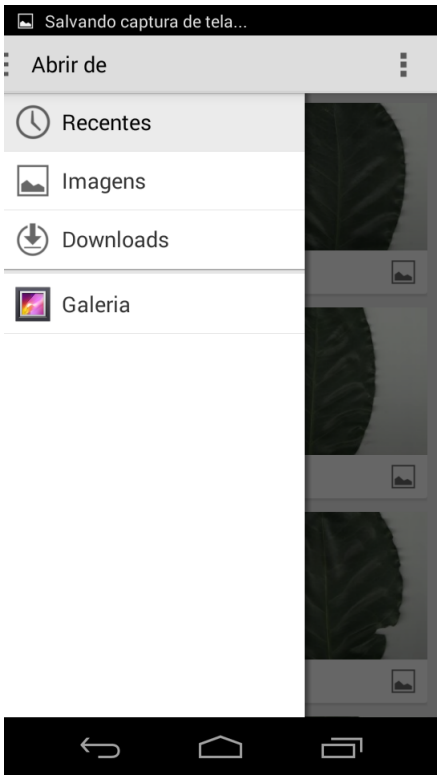

Figura 11 – Botão "Galeria"foi pressionado e a galeria de imagens do aparelho foi disponibilizada para escolha da imagem.

Após obter a imagem, o sistema realiza todo o processamento (já descrito) e assim que o resultado do terceiro momento é calculado, o usuário obtêm a resposta positiva (Figura 12), que a amostra está deficiente, ou negativa (Figura 13), que a amostra está saudável.

5.1.1 Análise de IHC

Para o desenvolvimento da interface, foi feita uma análise de IHC (Interação Humano Computador), objetivando desenvolver uma interface mais amigável e de fácil uso, não necessitando de manuais e treinamentos.

Como demostrado na seção anterior, a interface é bem intuitiva, com mensagens claras e objetivas.

Figura 12 – Tela exibindo resultado. Nesse caso, o resultado foi positivo.

Figura 13 – Tela exibindo resultado. Nesse caso, o resultado foi negativo.

O usuário precisará apenas iniciar a aplicação, escolher qual a forma da entrada das imagens, se vai fotografar ou usar uma imagem já pronta e todo o resto é feito pela própria

aplicação. Poucas opções de caminho diminuem a chance de conflitos e do usuário se perder dentro da aplicação.

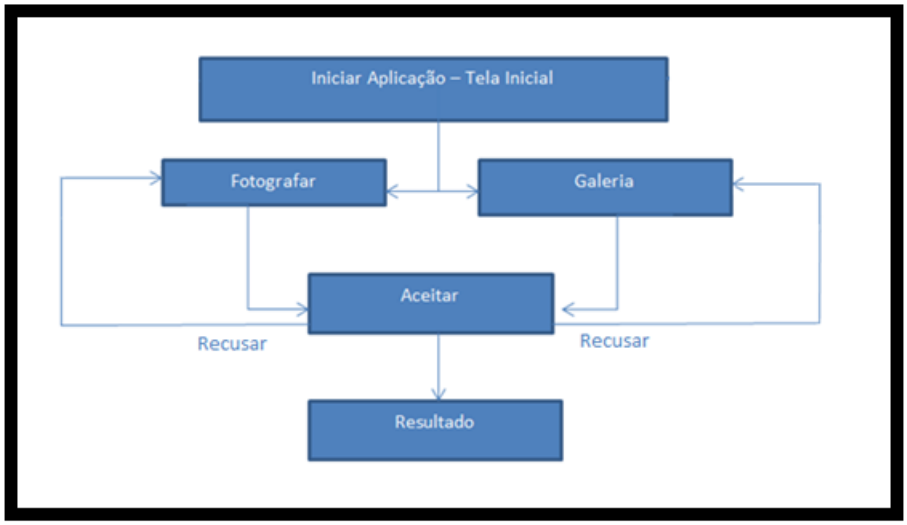

Figura 14 – Fluxograma do uso da aplicação

O fluxograma apresentado na figura 14 mostra todo o mapa da aplicação, com todos os caminhos possíveis.

Uma persona é um personagem fictício, que possui características de usuários típicos, usado para prever possíveis ações dentro da aplicação (BENYON, 2011). Foram criadas várias dessas personas, todas elas representam possíveis situações de utilização da aplicação. O objetivo dessa técnica é antever prováveis tipos de usuários e usos para a aplicação e tentar prever possíveis problemas ou melhorias para que estes consigam tirar máximo proveito da ferramenta.

Na análise do tipo de usuário, é possível encontrar usuários com uma maior familiaridade com smartphones e aplicações e também usuários que não têm tanto costume com novas tecnologias. Esse foi um dos fatores que mais influenciaram para que a interface fosse simples e objetiva, para que possa alcançar todos os tipos de usuários, desde o mais jovem ao mais velho, desde o que possui um curso superior na área, quanto àquele que possui apenas a experiência própria.

Quanto ao local de uso, um smartphone pode ser levado para qualquer lugar. Por não depender de nenhum componente externo e nem de acesso a redes de comunicação, a aplicação pode ser utilizada em qualquer lugar, desde que o usuário se lembre de garantir a qualidade da imagem caso escolha adquiri-la usando a câmera.

O esquema de cores, vermelho para amostra deficiente e verde para amostra saudável, é algo familiar. Nos semáforos, por exemplo, o vermelho indica pare e o verde indica siga. Em diversos outros tipos de aplicações o mesmo principio é utilizado. O vermelho, uma cor mais chamativa, usada para uma resposta ruim, ou que precise de mais atenção, enquanto o verde indica que está tudo bem.

5.2 Testes para Validação do Sistema

Para comprovar a eficiência do Sistema desenvolvido, foram realizados diversos testes controlados com amostras colhidas em cafezais da região de Monte Carmelo. Os resultados obtidos são mostrados a seguir:

Tabela 2 – Testes com o Matlab

Amostras	**Média**	**Desvio Padrão**	**Terceiro Momento**
Amostra 1	20,6286	28,2239	0,9157
Amostra 2	20,8991	28,6071	0,9367
Amostra 3	23,0194	29,0765	0,7423
Amostra 4	23,8282	28,8122	0,6696
Amostra 5	26,7482	31,9072	0,6365
Amostra 6	21,7228	28,7776	0,8694
Amostra 7	27,3931	30,9520	0,5207
Amostra 8	21,6953	29,2635	0,8746
Amostra 9	20,1233	29,5219	1,0779
Amostra 10	22,4975	29,3894	0,8231
Amostra 11	25,6642	32,7780	0,7373
Amostra 12	26,2114	32,0049	0,6555
Amostra 13	27,2478	31,7229	0,6082
Amostra 14	19,4557	28,2622	1,0592
Amostra 15	23,2768	29,5016	0,7758
Amostra 16	22,9586	30,0571	0,8438
Amostra 17	25,3310	31,8309	0,6970
Amostra 18	23,8361	30,1278	0,7313
Média	23,4743	30,0454	0,7875
Desvio Padrão	2,498378337	1,452150959	0,151285626
Variância	6,241894317	2,108742407	0,022887341

Os testes (Tabela 2) foram realizados no Matlab (INC, 1991), com 18 folhas saudáveis. Dentre todas as amostras, 4 delas foram classificadas erradas, representando uma taxa de erro de 22,22%.

Já no teste 2 (Tabela 3), realizado em um aparelho com Android, utilizou-se das mesmas amostras já utilizadas para os testes no Matlab. A taxa de acerto foi de 100%, tendo classificadas todas corretamente.

A variância média entre o Matlab e o Android no valor do terceiro momento foi 5,6%.

5.3 Testes de Tempo

Um método utilizado para avaliar o desempenho de um sistema é o tempo gasto para responder o usuário. Apesar desse tempo variar de acordo com a máquina utilizada, qual processamento está sendo realizado, é de grande importância que esse intervalo seja

Tabela 3 – Testes no Android

Amostras	**Média**	**Desvio Padrão**	**Terceiro Momento**
Amostra 1	54,3578	13,9611	1,1675
Amostra 2	54,5385	15,0192	1,0086
Amostra 3	55,2112	13,7168	1,0983
Amostra 4	54,1866	13,6287	1,1662
Amostra 5	57,6913	16,4979	0,9855
Amostra 6	55,5991	14,1496	1,3013
Amostra 7	56,1287	15,2362	0,9871
Amostra 8	56,0582	14,3909	0,9779
Amostra 9	58,2885	15,4359	1,0659
Amostra 10	56,2981	14,1533	1,1949
Amostra 11	60,6405	16,3023	0,8489
Amostra 12	59,1746	15,5517	0,9607
Amostra 13	57,4389	15,9041	1,2988
Amostra 14	56,5880	14,0377	1,2449
Amostra 15	55,6165	14,7869	1,1269
Amostra 16	56,8300	15,5565	1,0253
Amostra 17	59,4977	15,0559	0,8202
Amostra 18	57,5510	13,7695	1,0725
Média	56,7608	14,8419	1,0751
Desvio Padrão	1,818118503	0,909856164	0,138461745
Variância	3,305554892	0,827838238	0,019171655

o mais breve possível. É sem dúvida um dos fatores mais analisados pelos usuários (SCHRÖEDER, 2005).

Inicialmente, foi calculado o tempo gasto pelo Matlab(INC, 1991) para realizar os processamentos na imagem. Para estes testes, foi utilizado um computador Lenovo G400s, com processador Intel Core i7 3632QM, 8 Gb de memória RAM, sistema operacional Windows 8 64 bits.

O cálculo de tempo foi feito para o algoritmo de segmentação da folha (etapa 1) e para a análise dos pixels (etapa 2), resultando no tempo total com a soma do tempo gasto nas duas etapas. O tempo médio para a etapa 1 foi de 1,017 segundos e uma variância de 0,029, para a etapa 2 o tempo foi de 0,858 segundos e a variância de 0,029, com um tempo total médio de processamento no valor 1,875 segundos e uma variância de 0,111, conforme mostrado na (Tabela 4).

Em seguida, foram realizados os mesmos testes, agora em um smartphone. Para esse teste, foi utilizado um aparelho (Tabela 5) da empresa Motorola.

A arquitetura dos dispositivos móveis possui algumas limitações de memória e processamento, o que influencia diretamente no tempo gasto para realizar tarefas mais complexas. No caso, a aplicação desenvolvida tem um tempo médio de execução de 14,422 segundos para a etapa 1 com uma variância de 6,367, na etapa 2 foram 7,713 segundos

Tabela 4 – Testes de Tempo no Matlab(em segundos)

Amostras	**Etapa 1**	**Etapa 2**	**Total**
Amostra 1	2,504	2,177	4,682
Amostra 2	2,715	2,367	5,082
Amostra 3	2,508	2,164	4,672
Amostra 4	3,046	2,871	5,918
Amostra 5	2,087	1,798	3,886
Amostra 6	1,694	1,432	3,127
Amostra 7	2,579	2,248	4,828
Amostra 8	2,859	2,504	5,364
Amostra 9	2,037	1,853	3,890
Amostra 10	1,725	1,478	3,204
Amostra 11	2,004	1,736	3,740
Amostra 12	2,556	2,403	,4,959
Amostra 13	2,384	1,858	4,242
Amostra 14	1,622	1,386	3,009
Amostra 15	2,238	2,029	4,267
Amostra 16	2,998	2,817	5,816
Amostra 17	2,473	2,219	4,693
Amostra 18	2,422	2,245	4,667
Média	2,358	2,088	4,447
Desvio Padrão	0,415	0,421	0,836

Tabela 5 – Motorola Moto E - Especificações

Especificações	Valores
Sistema Operacional	Android 4.4 Kit Kat
Chipset	Qualcomm Snapdragon 200
Processador	1.2 GHz Dual Core A7
GPU	Adreno 302
Memória RAM	1 GB
Armazenamento Interno	4 GB
Tela	4,3 pol
Câmera	5 Mp

com a variância de 1,965, com um tempo médio de 22,136 segundos e a variância de 8,33, conforme Tabela 6.

Para diminuir o tempo gasto no processamento, foi desenvolvida uma outra versão do aplicativo. Nessa nova versão, o processamento não é feito pela thread principal da aplicação, mas agora é feito por uma thread secundária.

Nessa nova versão, conforme Tabela 7, houve uma aparente diminuição no tempo gasto em ambas as etapas: na etapa 1, o tempo médio foi de 13,994 segundo com desvio padrão de 2,673; na etapa 2 foram 7,569 segundos com desvio padrão de 1,385; no total, a média de tempo foi 21,563 segundos e um desvio padrão de 4,058.

Tabela 6 – Testes de Tempo no Android (em segundos)

Amostras	**Etapa 1**	**Etapa 2**	**Total**
Amostra 1	14,284	7,711	21,995
Amostra 2	15,67	8,583	24,253
Amostra 3	14,745	7,758	22,503
Amostra 4	17,817	9,451	27,268
Amostra 5	13,277	7,105	20,382
Amostra 6	9,923	5,322	15,245
Amostra 7	16,349	8,824	25,173
Amostra 8	17,035	9,257	26,292
Amostra 9	12,762	6,772	19,534
Amostra 10	10,302	5,548	15,85
Amostra 11	12,738	6,926	19,664
Amostra 12	16,008	8,528	24,536
Amostra 13	15,06	8,048	23,108
Amostra 14	10,657	5,223	15,88
Amostra 15	13,924	7,491	21,415
Amostra 16	19,162	10,268	29,43
Amostra 17	15,051	8,036	23,087
Amostra 18	14,836	7,999	22,835
Média	14,422	7,713	22,136
Desvio Padrão	2,523	1,402	3,925

Para verificar se realmente houve um diminuição, foi realizado o teste estatístico T-Student para as duas amostras, com um intervalo de confiança de 95%. O valor de t foi 0,431, o valor de p foi 0,6692. Como o valor de p é maior que 0,05 (intervalo de confiança), ficou provado que não há diferença entre as duas amostras, na versão com thread e na versão sem thread.

Uma outra versão do algoritmo foi testada visando diminuir o tempo de processamento. Nessa nova versão da aplicação, as duas etapas do processamento (segmentação da folha e análise dos pixels) foram transformadas em apenas uma.

Nessa nova versão é feita a verificação do canal azul (maior ou menor que 130), caso seja maior já é somado o valor daquele pixel, não necessitando gerar uma nova imagem, nem percorrê-la novamente.

Os valores obtidos nessa nova versão estão disponíveis nas tabelas 8, 9 e 10 .

Nos testes realizados com o novo algoritmo no Matlab, houve uma diminuição aparente no valor médio de 2 segundos de diferença da antiga versão para a nova (com apenas uma etapa de processamento onde antes eram duas).

Para verificar se diminuição é verdadeira, foi feito o teste estatístico T Student comparando as duas versões para um intervalo de confiança de 95%. O valor encontrado para p foi de 0,000000000275, valor menor que 0,05 usado como referência, comprovando que realmente houve uma diminuição do tempo gasto para o processamento com uma etapa

Tabela 7 – Testes de Tempo no Android com uso de Thread (em segundos)

Amostras	**Etapa 1**	**Etapa 2**	**Total**
Amostra 1	13,74	7,511	21,251
Amostra 2	15,374	8,379	23,753
Amostra 3	14,505	7,556	22,061
Amostra 4	17,31	9,254	26,564
Amostra 5	12,48	6,89	19,37
Amostra 6	9,398	5,156	14,554
Amostra 7	16,073	8,842	24,915
Amostra 8	16,659	9,022	25,681
Amostra 9	12,164	6,663	18,827
Amostra 10	9,914	5,468	15,382
Amostra 11	12,316	6,803	19,119
Amostra 12	15,646	8,422	24,068
Amostra 13	14,882	7,947	22,829
Amostra 14	9,403	5,173	14,576
Amostra 15	13,964	7,407	21,371
Amostra 16	19,04	10,126	29,166
Amostra 17	14,504	7,816	22,327
Amostra 18	14,523	7,808	22,331
Média	13,994	7,569	21,563
Desvio Padrão	2,673	1,385	4,058

a menos realizado no Matlab. A diminuição no tempo de processamento foi de um valor médio de 45%.

Os testes realizados com o novo algoritmo no Android também demostraram uma diminuição no tempo de processamento. Nesse caso, a diferença foi de um valor médio de 11,296 segundos.

Para avaliar se essa diminuição no tempo de processamento era realmente verdadeira, foi realizado o mesmo teste que foi feito com as amostras do Matlab, o T Student, também para 95% de confiança. O resultado obtido para p foi 0,000000000005419, muito inferior ao valor referencia 0,05, confirmando que também houve diminuição no tempo de processamento no Android com uma etapa a menos. A diminuição nesse caso, foi de um valor médio de 51%.

Também foram feitos testes com esse novo algoritmo na versão com thread. Como nos demais testes anteriores, também houve diminuição no tempo gasto para processamento.

Para verificar se essa diminuição se confirmava, também foi utilizado o mesmo teste dos casos anteriores, T Student com 95% de confiança. O valor encontrado para p foi de 0,00000000002138, também inferior a 0,05, confirmando a diminuição do tempo. Nesse caso, a diminuição foi de aproximadamente 51%.

Tabela 8 – Testes de Tempo no Matlab - Uma Etapa (em segundos)

Amostras	**Versão Inicial**	**Uma Etapa**	**Diferença**
Amostra 1	4,682	2,327	2,354
Amostra 2	5,082	2,510	2,571
Amostra 3	4,672	2,433	2,239
Amostra 4	5,918	3,011	2,907
Amostra 5	3,886	2,091	1,795
Amostra 6	3,127	1,664	1,462
Amostra 7	4,828	2,658	2,169
Amostra 8	5,364	2,850	2,513
Amostra 9	3,890	2,141	1,749
Amostra 10	3,204	1,690	1,514
Amostra 11	3,740	2,126	1,613
Amostra 12	4,959	2,982	,1,976
Amostra 13	4,242	2,574	1,668
Amostra 14	3,009	1,721	1,288
Amostra 15	4,267	2,375	1,891
Amostra 16	5,816	3,390	2,425
Amostra 17	4,693	2,604	2,089
Amostra 18	4,667	2,629	2,037
Média	4,447	2,432	2,014
Desvio Padrão	0,856	0,474	0,435

5.4 Dificuldades Encontradas

Durante o desenvolvimento do trabalho, surgiram diversas dificuldades, relatadas nesta seção.

Inicialmente, a aplicação estava sendo desenvolvido na IDE Eclipse utilizando o plugin ADT. Essa era a plataforma recomendada pelo Google, tendo seu download disponibilizado gratuitamente na pagina web do Android, juntamente com todos os demais recursos necessários.

Uma das atualizações disponibilizadas para o SDK do plugin, a versão 23, continha um erro que não permitia que as aplicações fossem executadas. Uma mensagem de erro era sempre mostrada e não havia nada que os usuários pudessem fazer para contornar esse problema.

O bug é descrito em vários fóruns, inclusive vários artigos foram postados alertando sobre o problema. O erro foi corrigido na versão 23.0.2, lançada algum tempo depois.

Antes do lançamento da versão 23.0.2 do SDK, foi lançado o Android Studio, a primeira IDE especifica para Android e desenvolvido pelo Google. Foi um lançamento muito aguardado, já que diversas ferramentas para facilitar o desenvolvimento vinham embutidas.

A versão 0.8 beta, disponível desde junho de 2014, continha muitos erros, ainda não

Tabela 9 – Testes de Tempo no Android - Uma Etapa (em segundos)

Amostras	**Versão Inicial**	**Uma Etapa**	**Diferença**
Amostra 1	21,995	11,82	10,175
Amostra 2	24,253	12,206	12,047
Amostra 3	22,503	10,882	11,621
Amostra 4	27,268	13,052	14,216
Amostra 5	20,382	9,525	10,857
Amostra 6	15,245	7,621	7,624
Amostra 7	25,173	11,608	13,565
Amostra 8	26,292	13,14	13,152
Amostra 9	19,534	9,89	9,644
Amostra 10	15,85	7,768	8,082
Amostra 11	19,664	9,589	10,075
Amostra 12	24,536	11,932	12,604
Amostra 13	23,108	10,97	12,138
Amostra 14	15,88	7,677	8,203
Amostra 15	21,415	10,447	10,968
Amostra 16	29,43	14,531	14,899
Amostra 17	23,087	11,268	11,819
Amostra 18	22,835	11,185	11,65
Média	22,136	10,839	11,296
Desvio Padrão	3,921	1,923	2,068

corrigidos. Em especial, cito um erro no emulador de dispositivos, ferramenta que emulava aparelhos sem a necessidade de realizar testes em dispositivos físicos, realizando os mesmos no próprio ambiente de desenvolvimento. Essa ferramenta não funcionava perfeitamente nessa versão, não sendo capaz de emular.

Posteriormente uma nova versão foi lançada, a versão 1.0, a primeira versão estável da IDE. Ela continha um problema no emulador, similar ao da versão 0.8. Nessa versão, um dispositivo era criado, mas não podia ser emulado. Esse e outros bugs, foram corrigidos na versão 1.0.2.

A aplicação desenvolvida utiliza a câmera e para testes, o emulador permite que uma webcam seja utilizada para tal tarefa. Novamente o emulador não funcionou perfeitamente e os testes eram interrompidos.

Inicialmente, achava-se que a aplicação era finalizada devido a erros no algoritmo, fazendo o mesmo ser revisado e alterado diversas vezes sem sucesso. Algum tempo depois, um teste realizado em um smartphone mostrou que a aplicação funcionava perfeitamente e que o problema estava no emulador. A partir desse momento, o emulador foi desprezado e os novos testes passaram a ser feitos diretamente em aparelhos.

Outro problema durante o desenvolvimento foi a diferença de resultados numéricos entre a aplicação no Android e o algoritmo em Java. Ambas as linguagens tem a mesma base, com um alto grau de similaridade, portanto, era esperado números mais próximos.

Tabela 10 – Testes de Tempo no Android com uso de Thread - Uma Etapa (em segundos)

Amostras	**Versão Inicial**	**Uma Etapa**	**Diferença**
Amostra 1	21,251	10,813	10,438
Amostra 2	23,753	12,035	11,718
Amostra 3	22,061	10,542	11,519
Amostra 4	26,564	12,828	13,736
Amostra 5	19,37	9,336	10,034
Amostra 6	14,554	7,331	7,223
Amostra 7	24,915	11,463	13,452
Amostra 8	25,681	12,771	12,91
Amostra 9	18,827	9,691	9,136
Amostra 10	15,382	7,615	7,767
Amostra 11	19,119	9,501	9,618
Amostra 12	24,068	11,662	12,406
Amostra 13	22,829	10,838	11,991
Amostra 14	14,576	7,419	7,157
Amostra 15	21,371	10,384	10,987
Amostra 16	29,166	14,338	14,828
Amostra 17	22,32	10,932	11,388
Amostra 18	22,331	10,939	11,392
Média	21,563	10,580	10,983
Desvio Padrão	4,055	1,898	2,195

Tabela 11 – Testes realizados em Android e em Java

Amostra	**Média**	**Desvio Padrão**	**Terceiro Momento**
Android	54,5851	15,1901	0,9434
Java	55,3479	14,9373	0,9363

Como mostrado nos testes (Tabela 11), os valores tiveram uma pequena variação, a qual não foi suficiente para interferir no resultado dos testes realizados, mas é passível de erros no resultado.

Para corrigir tal problema, testamos o uso do *BigDecimal*, uma forma de trabalhar com números longos em Java, com uma taxa de precisão maior que de outros tipos de variáveis. É muito utilizado em aplicações onde a precisão do numero é de extrema importância, como por exemplo, aplicações bancárias.

Tabela 12 – Testes usando o tipo Double e BigDecimal

Amostra	**Média**	**Desvio Padrão**	**Terceiro Momento**
Usando Double	55.3479379073	14.9534534275	0.9363955598
Usando BigDecimal	55.3479379072	14.9373154036	0.9363955597

O resultado com o *BigDecimal* e com o *Double* foram muito próximos, mantendo a

mesma diferença anterior entre as duas plataformas.

Capítulo 6

Conclusão

O presente trabalho conseguiu mostrar que ao se utilizar o canal azul da imagem, utilizando em seguida os pixels com valor menor ou igual a 130, consegue-se processar apenas a folha, desprezando o fundo e as sombras.

Após os testes realizados, foi possível afirmar que as medidas estatísticas media e desvio padrão não são suficientes para saber se a folha possui alguma deficiência. Isso só foi encontrado com a medida Terceiro Momento.

Foi possível determinar se a imagem da folha de um pé de café apresenta ou não deficiência nutricional, através de técnicas do processamento digital de imagens e matemática.

Comparou-se o método já desenvolvido no Matlab com o método implementado para a plataforma Android e pôde-se verificar que ocorreu uma pequena diferença de resultados, tendo sua variância 5,6%. Verificou-se que essa diferença ocorre devido à quantidade de casas decimais trabalhadas pelo Java. Fez-se a análise de uma solução com *BigDecimal*, porém não ocorreu grande variação entre os resultados.

A solução implementada para dispositivos móveis utilizando Android ficou intuitiva e clara para qualquer usuário do sistema Android.

A aplicação conseguiu classificar corretamente todas as 18 amostras disponíveis, indicando se elas possuíam alguma deficiência ou não.

O uso de threads aparentemente diminuía o tempo de processamento, mas após os testes com o T Student ficou confirmado com 95% de certeza que não houve diferença de tempo entre as duas versões.

A segunda versão do algoritmo, onde a folha é percorrida apenas uma vez, demanda menos tempo que a versão antiga onde a mesma era percorrida duas vezes.

Seguindo o desenvolvimento do sistema, para trabalhos futuros, diversas melhorias podem ser implementadas:

- ❏ reconhecer não apenas a existência ou não de deficiência, mas indicar também de qual é a deficiência, quando o resultado for positivo;

- ❏ utilização de amostras controladas, cultivadas em ambiente controlado, especificamente para a realização desses testes e validação do método;
- ❏ desenvolvimento de algoritmos de extração de características e classificação.

Referências

ABIC, A. a. B. d. I. d. C. **A história do café**. 2013. Disponível em: <http://www.abic.com.br/publique/cgi/cgilua.exe/sys/start.htm?sid=38>. Citado na página 11.

ANDREWS, G. R. **Foundations of Multithreaded, Parallel, and Distributed Programming**. 1 edition. ed. [S.l.: s.n.], 1999. 664 p. Citado na página 21.

APPLE. **App Store, Loja de Aplicativos da Apple, Atinge 2 Bilhões de Downloads**. 2014. Disponível em: <http://www.apple.com/br/pr/library/2009/09/28Apples-App-Store-Downloads-Top-Two-Billion.html>. Citado na página 15.

_____. **iPhone 6 - Apple**. 2014. Disponível em: <https://www.apple.com/br/iphone-6/>. Citado na página 16.

BELTZAC, A.; HESS, R. C.; YAMAGUCHI, S. Y. **DNA Shot - Reconhecimento de DNA através de imagens**. Curitiba: [s.n.], 2014. Disponível em: <http://dspace.c3sl.ufpr.br:8080/dspace/bitstream/handle/1884/36975/TCCDNA_SHOT.pdf?sequence=1&isAllowed=y>. Citado na página 27.

BENYON, D. **interação Humano Computador**. Segunda ed. [S.l.]: Pearson, 2011. 442 p. Citado na página 38.

BIANCHI, A. J. **Processamento de áudio em tempo real em dispositivos não convencionais**. 2011. Disponível em: <http://www.ime.usp.br/~ajb/projeto/qualificacao-ajb.pdf>. Citado 2 vezes nas páginas 27 e 28.

BUENO, A. D. Introdução ao Processamento Paralelo e ao Uso de Clusters de Workstacions em Sistemas GNU/LINUX. nov. 2002. Disponível em: <http://www.redhat.com/archives/fedora-users-br/2006-December/pdfXbBDApNiw1.pdf>. Citado 2 vezes nas páginas 21 e 22.

CLUA, E. W. G. **Impostores com relevo**. Rio de Janeiro: [s.n.], 2004. Disponível em: <http://www.maxwell.vrac.puc-rio.br/5409/5409_1.PDF>. Citado na página 21.

COLOURIS, G.; DOLLIMORE, J.; KINDBERG, T. **Distributed Sytems – Concepts and Design**. Segunda ed. [S.l.: s.n.], 1996. Citado na página 22.

CONAB, C. N. d. A. Acompanhamento da Safra Brasileira. Moitoramento Agrícola - Café - Safra 2015. 2015. Disponível em: <http://www.conab.gov.br/OlalaCMS/uploads/

arquivos/15_01_14_11_57_33_boletim_cafe_janeiro_2015.pdf>. Citado na página 11.

CUNHA, L. R. da et al. Desenvolvimento de sistema para analise foliar de café. **III Encontro de Iniciação Científica e Tecnológica da UFU**, p. 1–29, 2013. Citado na página 31.

EXAME. **Android abasteceu 85% do mercado mundial de smartphones**. 2014. Disponível em: <http://exame.abril.com.br/tecnologia/noticias/android-abasteceu-85-do-mercado-mundial-de-smartphones>. Citado na página 16.

FARINA, A. M.; MARANA, A. N. **BioMobile: Sistema Biométrico de Identificação de Usuários em Dispositivos Móveis na Plataforma Android**. 2012. Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/wvc/2012/0048.pdf>. Citado 2 vezes nas páginas 25 e 26.

GOOGLE. **Android**. 2014. Disponível em: <http://www.android.com/>. Citado 3 vezes nas páginas 13, 15 e 16.

_____. **Android Developer**. 2014. Disponível em: <http://developer.android.com/about/index.html>. Citado 3 vezes nas páginas 17, 19 e 20.

_____. **Nexus 5**. 2014. Disponível em: <http://www.google.com/nexus/5/>. Citado na página 16.

_____. **Play Store**. 2014. Disponível em: <https://play.google.com/>. Citado na página 15.

GREGO, M. **14 números que mostram a veloz popularização dos smartphones**. 2014. Disponível em: <http://exame.abril.com.br/tecnologia/noticias/14-numeros-que-mostram-a-veloz-popularizacao-dos-smartphones>. Citado na página 14.

IDC. Worldwide Smartphone Growth Forecast to Slow from a Boil to a Simmer as Prices Drop and Markets Mature, According to IDC. **IDC.com**, 2014. Disponível em: <http://www.idc.com/getdoc.jsp?containerId=prUS25282214>. Citado na página 15.

_____. Worldwide Smartphone Shipments Increase 25.2% in the Third Quarter with Heightened Competition and Growth Beyond Samsung and Apple, Says IDC. **IDC.com**, 2014. Disponível em: <http://www.idc.com/getdoc.jsp?containerId=prUS25224914>. Citado na página 15.

INC, T. M. W. **MATLAB for Windows User's guide**. 1991. Disponível em: <http://www.mathworks.com/products/matlab/>. Citado 2 vezes nas páginas 39 e 40.

IPNI BRASIL, I. P. N. I. B. **Cafezal**. 2014. Disponível em: <http://www.ipni.net/ppiweb/gbrazil.nsf/$webindex/article=88EF02F083256D8100453652C2A63AF7>. Citado 3 vezes nas páginas 6, 23 e 24.

LG. **LG G3 - LG**. 2014. Disponível em: <http://www.lge.com/br/smartphones-g3-lg/index.html?cmpid=BR-alwaysonmc_mc_sem_google____g3_lgg3___alwayson2014-mc-lgg3-institucional&gclid=CO7wuc2pq8ICFSRo7AodCUIApg>. Citado na página 16.

LLOYD, S. **Ultimate Physical Limits of Computation**. 2000. Disponível em: <http://www.nature.com/nature/journal/v406/n6799/full/4061047a0.html>. Citado na página 20.

MALAVOLTA, E. et al. Encarte técnico. Seja doutor do seu cafezal. **Arquivo do Agrônomo. Potafos.**, 1993. Disponível em: <http://brasil.ipni.net/ipniweb/region/brasil.nsf/0/1870E4C8386104EE83257AA0003B6C81/\protect\T1\textdollarFILE/cafezal2edicao.pdf>. Citado 3 vezes nas páginas 12, 22 e 23.

MICROSOFT. **Lumia 1520 - Nokia**. 2014. Disponível em: <http://www.microsoft.com/pt-br/celulares/celular/lumia1520/>. Citado na página 16.

_____. **Windows Phone Store**. 2014. Disponível em: <http://www.windowsphone.com/pt-br/store>. Citado na página 15.

MOBILTEC. **Desenvolvimento Mobile nas Plataformas Android e Ios**. 2014. Disponível em: <http://www.mobiltec.com.br/blog/index.php/desenvolvimento-mobile-nas-plataformas-android-e-ios>. Citado 2 vezes nas páginas 6 e 18.

OXFORD, D. **Smartphone - Definition**. [s.n.], 2014. Disponível em: <http://www.oxforddictionaries.com/definition/english/smartphone>. Citado na página 14.

PCMAGAZINE. **Encyclopedia - Smartphone**. 2014. Disponível em: <http://www.pcmag.com/encyclopedia/term/51537/smartphone>. Citado na página 14.

PORTER, A. **Programming Mobile Applications for Android Handheld Systems**. [S.l.]: Universidade de Maryland, 2014. Citado 2 vezes nas páginas 18 e 19.

RABACHINI, L. P. et al. **Detection, Extraction and Text Translation in Digital Images using Android Platform**. 2014. Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/wvc/2014/0015.pdf>. Citado 2 vezes nas páginas 26 e 27.

SAMSUNG. **Galaxy S5 - Samsung**. 2014. Disponível em: <http://www.samsung.com/br/consumer/cellular-phone/smartphones/galaxy-s5/SM-G900MZDAZVV>. Citado na página 16.

SCHRÖEDER, C. d. S. Critérios e indicadores de desempenho para sistemas de treinamento corporativo virtual: um modelo para medir resultados. **UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL**, 2005. Disponível em: <http://www.ufrgs.br/gpnavi/artigos/disserta_cris.pdf>. Citado na página 40.

SCIENCE, C.; CSMD, M. D. **PVM (Parallel Virtual Machine)**. 2015. Disponível em: <http://www.csm.ornl.gov/pvm/>. Citado na página 22.

SONY. **Xperia Z3 - Sony**. 2014. Disponível em: <http://www.sonymobile.com/br/products/phones/xperia-z3/>. Citado na página 16.

STATISTA. **Cumulative number of apps downloaded from the Apple App Store from June 2008 to June 2014 (in billions)**. 2014. Disponível em: <http://www.statista.com/statistics/263794/number-of-downloads-from-the-apple-app-store/>. Citado na página 17.

TANENBAUM, A. S. **Modern Operating Systems**. [S.l.: s.n.], 1992. Citado na página 21.

TECMUNDO. **iOS, Android e Windows Phone: números dos gigantes comparados [infográfico]**. 2014. Disponível em: <http://www.tecmundo.com.br/sistema-operacional/60596-ios-android-windows-phone-numeros-gigantes-comparados-infografico.htm>. Citado na página 17.