Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Aplicadas
Departamento de Computação e Sistemas

Avaliação de Desempenho de Visão Computacional em Aplicações Móveis

Luiz Henrique da Silva Cunha

TRABALHO DE CONCLUSÃO DE CURSO

ORIENTAÇÃO:

Prof. Vicente J. Peixoto de Amorim

COORIENTAÇÃO:

Prof. Igor Muzetti Pereira

Fevereiro, 2018
João Monlevade–MG

Luiz Henrique da Silva Cunha

Avaliação de Desempenho de Visão Computacional em Aplicações Móveis

Orientador: Prof. Vicente J. Peixoto de Amorim

Coorientador: Prof. Igor Muzetti Pereira

Monografia apresentada ao curso de Engenharia de Computação do Instituto de Ciências Exatas e Aplicadas, da Universidade Federal de Ouro Preto, como requisito parcial para aprovação na Disciplina "Trabalho de Conclusão de Curso II".

Universidade Federal de Ouro Preto

João Monlevade

Fevereiro de 2018

C972a Cunha, Luiz Henrique da Silva.
Avaliação de desempenho de visão computacional em dispositivos móveis [manuscrito] / Luiz Henrique da Silva Cunha. - 2018.

61f.: il.: color; grafs; tabs.

Orientador: Prof. MSc. Vicente José Peixoto Amorim.
Coorientador: Prof. MSc. Igor Muzetti Pereira.

Monografia (Graduação). Universidade Federal de Ouro Preto. Instituto de Ciências Exatas e Aplicadas. Departamento de Computação e Sistemas de Informação.

1. Visão por computador. 2. Processamento de imagens. 3. Computação móvel. I. Amorim, Vicente José Peixoto. II. Pereira, Igor Muzetti. III. Universidade Federal de Ouro Preto. IV. Titulo.

CDU: 004.932

Catalogação: ficha@sisbin.ufop.br

FOLHA DE APROVAÇÃO DA BANCA EXAMINADORA

Avaliação de Desempenho de Visão Computacional em Aplicações Móveis

Luiz Henrique da Silva Cunha

Monografia apresentada ao Instituto de Ciências Exatas e Aplicadas da Universidade Federal de Ouro Preto como requisito parcial da disciplina CEA496 – Trabalho de Conclusão de Curso II do curso de Bacharelado em Engenharia de Computação e aprovada pela Banca Examinadora abaixo assinada:

Prof. Vicente J. Peixoto de Amorim
Mestre em Ciência da Computação
DECSI - UFOP

Prof. Igor Muzetti Pereira
Mestre em Ciência da Computação
DECSI - UFOP

Prof. Sérgio Evangelista Silva
Doutor em Engenharia de Produção
Examinador
DEENP - UFOP

João Monlevade, 07 de fevereiro de 2018

ATA DE DEFESA

No dia 07 do mês de Fevereiro de 2018, às 17:30 horas, na sala C304 do Instituto de Ciências Exatas e Aplicadas, foi realizada a defesa de Monografia pelo(a) aluno(a) **Luiz Henrique da Silva Cunha**, sendo a Comissão Examinadora constituída pelos professores: Prof. Vicente J. Peixoto de Amorim, Prof. Igor Muzetti Pereira, Prof. Sérgio Evangelista Silva. O(a) candidato(a) apresentou a monografia intitulada: "**Avaliação de Desempenho de Visão Computacional em Aplicações Móveis**". A comissão examinadora deliberou, por unanimidade, pela aprovação do candidato, com nota 9,0 (nove), concedendo-lhe o prazo de 15 dias para incorporação das alterações sugeridas ao texto final. Na forma regulamentar, foi lavrada a presente ata que é assinada pelos membros da Comissão Examinadora e pelo(a) graduando(a).

Prof. Vicente J. Peixoto de Amorim
Mestre em Ciência da Computação
DECSI - UFOP

Igor Muzetti Pereira
Prof. Igor Muzetti Pereira
Mestre em Ciência da Computação
DECSI - UFOP

Prof. Sérgio Evangelista Silva
Doutor em Engenharia de Produção
Examinador(a)
DEENP - UFOP

Luiz Henrique da Silva Cunha

João Monlevade, 07 de fevereiro de 2018

TERMO DE RESPONSABILIDADE

Eu, **Luiz Henrique da Silva Cunha** declaro que o texto do trabalho de conclusão de curso intitulado "***Avaliação de Desempenho de Visão Computacional em Aplicações Móveis***" é de minha inteira responsabilidade e que não há utilização de texto, material fotográfico, código fonte de programa ou qualquer outro material pertencente a terceiros sem as devidas referências ou consentimento dos respectivos autores.

João Monlevade, 07 de fevereiro de 2018

Luiz Henrique da Silva Cunha

Este trabalho é dedicado a todos aqueles que desejam tornar o mundo um lugar cada vez melhor.

Agradecimentos

Agradeço a minha família, por me apoiarem durante essa jornada, me incentivando a nunca desistir.

Da mesma forma, agradeço ao meu orientador, Vicente J. P. Amorim e ao meu coorientador Igor Muzetti, pela paciência que tiveram, bem como o auxílio,lições e conselhos que me foram concedidos. Agradeço também cada minuto vivido dentro do laboratório iMobilis, trocando experiências com amigos que fiz para toda a vida. A todos os meus professores por me ensinarem dos mais diversos assuntos, em especial o professor Thiago Luange, por me auxiliar com os tópicos para este trabalho.

Agradeço aos integrantes das repúblicas em que morei, Thiago Tavares, Rafael Tavares, Raí Rômulo, Jardel Santos, Lucas Faria e Paulo Ricardo, que depois de tanto tempo juntos os tenho como irmãos.

A todas as amizades que fiz durante o curso em especial Marcelo Melo, Breno Gonçalves, Raul Felipe, Ricardo Coelho, Eduardo Valério, Jonas Cursage, Samuel Neves. Os agradeço pelos momentos de diversão e trabalho que compartilhamos.

Finalmente, agradeço a Universidade Federal de Ouro Preto, por me aceitar, por me ajudar a escalar mais um degrau em direção ao sucesso.

"Whatever your hand finds to do, do it with all your might,
for in the realm of the dead, where you are going,
there is neither working nor planning nor knowledge nor wisdom."
(Holy Bible, Ecclesiastes, 9:10)

Resumo

A constante evolução da tecnologia torna as pessoas cada vez mais informatizadas. Pesquisas sugerem que existem cerca de 2 *smartphones* para cada 3 cidadãos brasileiros e a tendência é que esse número aumente cada vez mais. Nesse interim, o número de aplicativos nos mercados mais populares já ultrapassa os 4 milhões. Tal mercado tem potencial ilimitado, porém nem todo tipo de aplicação é viável devido às limitações do hardware. Este trabalho compara diferentes abordagens de processamento (local e remoto) de uma aplicação OCR a fim de determinar o potencial de processamento dos dispositivos móveis disponíveis e viabilidade de aplicações desse tipo no mercado. Para tanto, foram criados sistemas responsáveis por entregar a mesma funcionalidade através de processamento remoto e local. Por fim, verificou-se que o processamento remoto trouxe pequenas vantagens em comparação ao processamento local.

Palavras-chaves: OCR. Android. Visão Computacional. Avaliação de Desempenho. Tesseract.

Abstract

The constant evolution of technology is turning people into geeks. Researches suggest there are 2 smart-phones for every three Brazilian citizens and this number tends to increase even more. Furthermore, the number of apps in the most popular Markets surpasses the 4 million mark. Therefore, such market shows an unlimited potential, however, hardware specification is a limiting factor in application building. This work presents a comparison between different processing approaches (local and remote) of an OCR application in order to determine current mobile devices' processing potential as well as OCR application market viability. Then it was verified that remote processing brought slight advantages compared to local processing.

Key-words: OCR. Android. Computer Vision. Performance Evaluation. Tesseract.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

BW Largura de Banda(*Bandwidth*)

B Bytes

Mb Megabit

s segundos

SaaS *Software as a Service*

PaaS *Product as a Service*

IaaS *Infrastructure as a Service*

API *Application Programming Interface*

LAN *Local Area Network*

VLAN *Virtual Local Area Network*

VM Máquina Virtual(*Virtual Machine*)

fps *Frames* por segundo

GPS *Global Positioning System*

API *Application Programming Interface*

HTML *HyperText Markup Language*

JSON *JavaScript Object Notation*

PHP *Personal Home Page*

IDE *Integrated Development Environment*

NDK *Android Native Development Kit*

XML *eXtensible Markup Language*

ICEA *Instituto de Ciência Exatas e Aplicadas*

Sumário

1 Introdução

Em 2002, o números de linhas de telefonia móvel ultrapassou o número de linhas de telefonia fixa em uma escala global (SRIVASTAVA, 2005). Contudo, o encanto de poder conversar por longos minutos através de um aparelho eletrônico perdeu seu brilho de outrora para dar lugar ao envio mídias digitais e interações com aplicativos diversos. Estima-se que um usuário comum gaste em média 4 horas por dia interagindo com seu *smartphone* (FLURRY, 2017). Dessa forma, os dispositivos móveis deixaram de ser uma simples ferramenta de comunicação para tornarem-se uma ferramenta com sua própria cultura (GOGGIN, 2012). Isso só é possível graças à evolução técnica do par *hardware/software*, onde o primeiro garante o poder de processamento e o último usufrui de tal poder das formas mais criativas possíveis, fornecendo experiências inovadoras ao usuário.

Entretanto, estamos chegando em um platô: a capacidade de processamento do *hardware* que outrora acompanhava a lei de Moore com precisão, em breve não conseguirá manter suas premissas (WALDROP, 2016b). O superaquecimento dos processadores, o tamanho e o tipo de material dos transistores são os principais fatores que interrompem a continuidade do progresso tecnológico. Atualmente, as indústrias vêm buscando alternativas como o aumento de núcleos e redefinição de arquitetura, porém essas técnicas não garantem uma melhora linear de eficiência e também são extremamente dependentes de um desenvolvimento de softwares que aceitam paralelismo, o que nem sempre é possível.

Uma vez que o *hardware* é limitado, deve-se dar atenção para a quantidade de recursos que uma aplicação deve exigir da máquina para que seja alcançado o melhor desempenho possível. Entretanto, em alguns casos não se pode utilizar dessa premissa, haja vista que a própria natureza da aplicação exige uma quantidade elevada de operações. Como exemplo, tem-se testes de *benchmark*, aplicações com resposta em tempo real, processamento de imagens, etc.

Tendo em vista esse cenário, é questionado se a capacidade de processamento em um dispositivo móvel alcançará capacidade dos *desktops*, se o desenvolvimento de *softwares* para o mercado de aplicativos móveis deve seguir a tendência do *cloud computing* como meio de sobrevivência e se as soluções para a questões anteriores são comercializáveis.

Este trabalho tem como objetivo responder às questões levantadas no parágrafo anterior através de uma análise comparativa de desempenho entre dois aplicativos com a mesma funcionalidade, porém, com diferentes abordagens de processamento. O primeiro realiza o processamento de um documento no próprio aparelho, enquanto o segundo envia a imagem para um servidor na nuvem e apenas exibe resultados.

2 Revisão bibliográfica

2.1 Conceitos Básicos

Este capítulo busca contextualizar o leitor em diversos conceitos de tecnologia necessários para a compreensão deste trabalho.

2.1.1 Lei de Moore

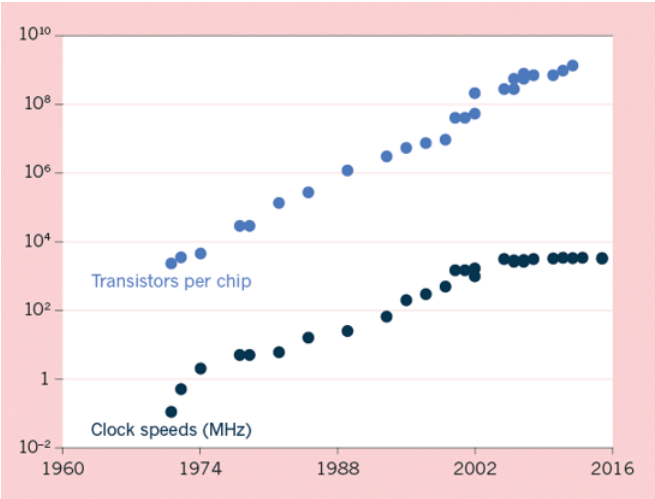
Em 1947 foi o início de uma nova era para a humanidade. A criação de transistores capazes de substituir os tubos de vácuo, aliados à miniaturização e aos circuitos integrados, permitiu o alcance de novos limites na eletrônica (THOMPSON; PARTHASARATHY, 2006). Tão grandioso foi o salto na tecnologia que em meados de 1965 a previsão feita pelo presidente da Intel foi tomada como uma espécie de regra para as indústrias. (MOORE et al., 1998) afirmava que a cada 18 meses, o número de transistores dentro dos chips dobraria e manteria o custo. Devido a sua viabilidade, essa afirmação acabou ganhando o nome de Lei de Moore e veio sendo praticada com rigor por mais de 40 anos. Entretanto, a miniaturização contínua está se tornando uma barreira para a continuação dessa lei. Transistores hoje atingem dimensões menores que 15nm. Com dimensões tão pequenas, transistores são cada vez mais suscetíveis a interferências magnética e térmica. Como consequência, tem-se a perda de desempenho nos circuitos eletrônicos.

As alternativas para o aumento de transistores dentro de um processador foram mudando. Com o aumento da temperatura, foi necessária a redução da frequência (*clock*). Depois veio o aumento de núcleos, no entanto o resultado não foi satisfatório. Uma vez que a maioria dos algoritmos não são desenvolvidos para processamento paralelo, o ganho de desempenho nesses casos não foi significativo (THOMPSON; PARTHASARATHY, 2006).

A figura 1 mostra a evolução do poder de processamento ao longo dos anos, note que o aumento de transistores em um chip a partir de 2002 não significou o aumento de velocidade de processamento. Isso se deve a introdução dos processadores de vários núcleos e suas limitações mencionadas anteriormente.

A indústria, porém, já possui rotas alternativas. Uma das novas formas de produção é apresentada por (WALDROP, 2016a). Denominada *More than Moore strategy*, é uma abordagem reversa, onde primeiro é projetada a aplicação para depois projetar o processador ideal. Também são buscadas saídas através do material utilizado. A troca do silício por materiais como o grafeno poderá trazer a resposta para o problema do calor, por exemplo (WALDROP, 2016a). Outra alternativa encontra-se na computação quântica, que apesar de não possuir produtos comercializáveis na atualidade, poderá aumentar a capacidade

Figura 1 – Quantidade de transistores por chip e velocidade de clock.



Fonte: (WALDROP, 2016b)

de processamento em parâmetros tão revolucionários quanto a própria invenção dos transistores (WALDROP, 2016a).

2.2 Computação em Nuvem

Atualmente o termo *nuvem* vem se popularizando no meio tecnológico. Isso se deve a praticidade e forma dinâmica que um modelo de negócios baseado em nuvem pode oferecer. O termo Computação em Nuvem (*Cloud Computing*) está diretamente relacionado às aplicações como serviços fornecidos através da Internet, bem como aos recursos de *hardware* e *software* utilizados para fornecer esse serviço (ARMBRUST et al., 2010).

Simplificando, *Cloud Computing* é uma mudança geográfica do processamento (OGRAPH; MORGENS, 2008). A aplicação passa a ser independente das características do *hardware* cliente e este passa a necessitar apenas de um navegador *web*. Como consequência, o cliente gastaria menos na aquisição do *hardware* sem perder qualidade no serviço. Entretanto, para que a qualidade seja mantida, uma boa largura de banda (*Bandwidth* ou *BW*) pode ser necessária.

As vantagens desse modelo de processamento são inúmeras. Isso é notado facilmente através do modelo de negócios *Software as a Service* (*Saas*) adotado por grandes marcas como Netflix, Microsoft Azure, Amazon EC2 e Google. (CUSUMANO, 2010) afirma que, com *SaaS*, empresas alugam seus equipamentos para hospedar aplicações/dados de terceiros e podem cobrar valores proporcionais ao que foi consumido ou vender seu serviço por um preço fixo mensal. Para o autor, *SaaS* é o modelo de negócios que ganha maior destaque na atualidade. (CUSUMANO, 2010) ainda afirma que a ideia de ser cobrado apenas pelo que utilizar, conhecida como *pay-as-you-go*, gera uma dinamização de fluxo

de caixa atrativa para gestores. Uma vez que os gastos com tecnologia se tornam mais previsíveis tanto em valor quanto em demanda, gestores podem direcionar investimentos para outros setores da empresa. Quanto à questão tecnológica, a centralização do *hardware* traz praticidade no gerenciamento e manutenção. Um *software* local depende da atenção do cliente para se manter atualizado e funcional, depende também da qualidade do *hardware* que o suporta, sistema operacional, licenças, etc. A nuvem elimina a maioria desses fatores e foca na praticidade para o cliente.

No entanto, existem algumas barreiras para a computação em nuvem não dominar o mercado em sua totalidade. Uma delas por exemplo é a largura de banda que nem sempre é suficiente por parte do cliente. Mais importante que isso é a questão da segurança dos dados do cliente. As grandes empresas ainda não conseguem confiar seus dados a terceiros pois a segurança dessas informações nem sempre é garantida. Além disso, há também o receio quanto a disponibilidade do serviço: uma interrupção de algumas horas para uma empresa pode custar milhões em prejuízo para o contratante. Uma alternativa não tão viável é que cada empresa possua sua própria nuvem, porém isso requereria um grande investimento para um possível futuro abandono de tecnologia, fator que a nuvem veio eliminar.

A tabela 1 traz uma breve ilustração de alguns obstáculos e oportunidades para que a computação em nuvem venha a se consolidar cada vez mais no mercado. Infelizmente, alguns desses desafios como a questão da disponibilidade e da confidencialidade serão sempre tópicos em pauta neste tema, haja vista que segurança da informação e privacidade são temas de debate contínuos no meio tecnológico.

Tabela 1 – Obstáculos e Oportunidades para a Computação em Nuvem.

Obstáculos	**Oportunidades**
Disponibilidade	Uso de Múltiplos Provedores
Fidelidade	Padronização de APIs
Confidencialidade	Criptografia de dados, VLANs, *Firewalls*
Limites de Dados	Entrega física e maiores BW
Imprevisibilidade de Performance	Melhorias no suporte a VM, memórias *Flash*
Armazenamento Escalável	Desenvolver o conceito
Bugs em Sistemas Distribuídos	Desenvolver *software debugger* para sistemas distribuídos
Escalabilidade Ágil	Desenvolver *software* para autogerenciamento baseado em ML
Reputação	Oferecer serviço de qualidade como os serviços de email
Licenceamento de *Software*	Licenças *pay-for-use*

Fonte: (ARMBRUST et al., 2010)

2.3 Dispositivos Móveis

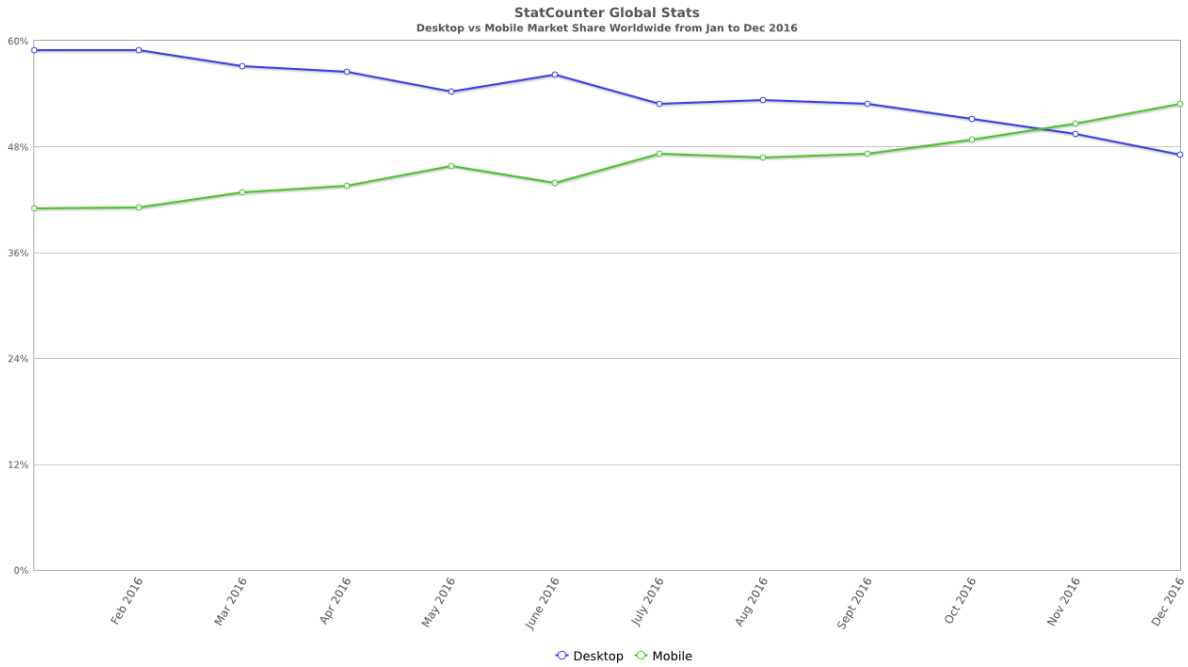
Dispositivo móvel é o termo designado a computadores portáteis como os *smartphones*, *handhelds*, *tablets* ou *MP3 players*. Esses dispositivos vêm revolucionando a forma

como interagimos com computadores através das mais diversas aplicações (MALLADI; AGRAWAL, 2002). Economia de energia e tempo, conectividade, menor custo e maior nível de interação com o usuário são algumas das vantagens dessa troca de *desktops* para dispositivos móveis. Entretanto, quanto mais simples o dispositivo, menor é o seu poder de processamento. Existem desafios que são considerados intermitentes pois evoluem juntamente à tecnologia. Em seu artigo (SATYANARAYANAN, 1996) menciona, dentre outras, algumas desvantagens relacionadas a segurança tanto física quanto digital do usuário e ao consumo de energia dos aparelhos.

Em um estudo realizado com diversos estudantes de uma universidade, (SARKER; WELLS, 2003) tenta identificar características para a aceitação dos dispositivos móveis pela sociedade. Em suma, o pesquisador afirma que a adoção desses aparelhos não é oriunda simplesmente das comodidades e eficiência em comunicação, mas também dos fatores socioculturais do indivíduo.

De fato, cultura e *marketing* têm um grande papel na adoção de uma tecnologia e por vezes o mercado arrisca anunciar produtos que nem mesmo foram desenvolvidos por completo, como foi o caso de sucesso do iPhone. Mas isso é uma boa estratégia: uma pré venda consegue avaliar o nível de aceitação do produto, evitando um possível prejuízo para a empresa.

Figura 2 – Participação de mercado por dispositivos móveis e *desktops*.

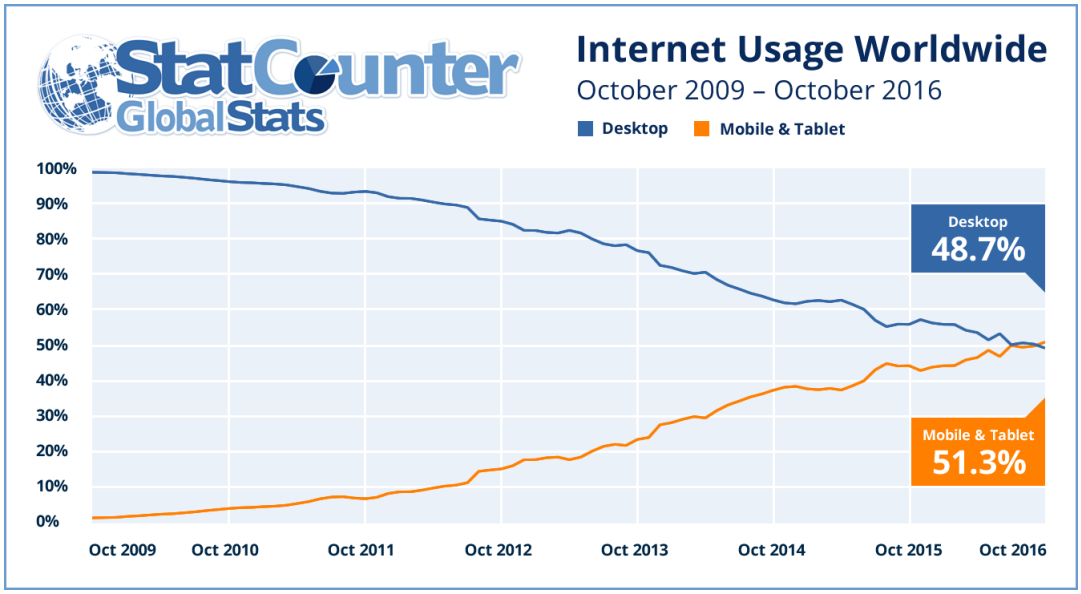


Fonte: (STATCOUNTER, 2017)

O gráfico da figura 2 ilustra a participação de Mercado por dispositivos móveis e por *desktops*. No final de 2016 os dispositivos móveis passaram a liderar as vendas no mercado, superando os *desktops*, o que mostra a importância desses dispositivos na

atualidade. Podemos encontrar outra evidência na figura 3, onde os acessos à internet através de dispositivos móveis superam os *desktops*.

Figura 3 – Acesso à Internet: dispositivos móveis e *desktops*.



Fonte: (STATCOUNTER, 2016)

Em suma, o uso desses dispositivos vem crescendo cada vez mais, por isso se faz necessário maior atenção no setor de desenvolvimento de *software* para tais dispositivos. Tal setor é essencial para que o crescimento do uso de dispositivos móveis continue.

2.4 Processamento Digital de Imagens

A popularização de dispositivos capazes de capturar imagens e o avanço de desempenho dos computadores colocaram o processamento de imagens em alta. O que era antes feito em laboratórios especializados, hoje pode ser feito por pessoas que não têm conhecimento avançado no assunto (BURGERN; BURGER, 2009). Basta uma visita em sua *App Store* preferida para ver dezenas de aplicativos capazes de fazer montagens ou colocar filtros em suas fotos.

O Processamento Digital de Imagens tem por função alterar o estado original de uma imagem. (QUEIROZ; GOMES, 2001) destaca que um dos passos mais importantes para obtenção de um bom resultado é a fase de pré processamento da imagem. Esta fase busca separar os objetos de interesse (*foreground*) do resto da imagem (*background*). Para tanto, é necessário reduzir ao máximo os elementos indesejados que são caracterizados como ruído. Isso ocorre através de operações matemáticas aplicadas sobre cada pixel da imagem original. Esse processo é conhecido como filtragem, e tem como resultado transformações como suavização, detecção de contornos, limiarização, mudança de escala, ajuste de contraste, etc. Dentre essas, uma das operações mais utilizadas é a limiarização.

2.4.1 Limiarização

A limiarização (QUEIROZ; GOMES, 2001), também conhecida como binarização, busca traduzir uma imagem em diferentes tons de cinza para uma imagem com apenas duas cores (daí o nome binarização). Esse processo é resultado da aplicação da equação 2.1 sobre todos os píxels da imagem.

$$f(x) = \begin{cases} a & , x \geq n \\ 0 & , caso\ contrário \end{cases} \tag{2.1}$$

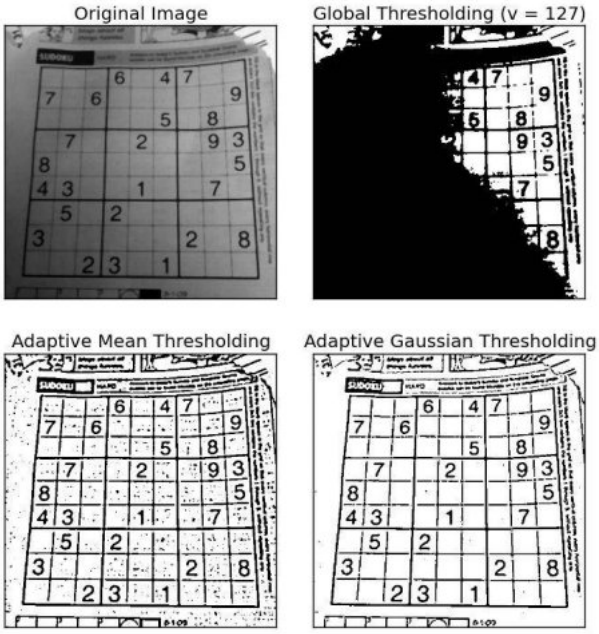
Na equação 2.1, temos que, dado um ponto x imagem de entrada, o mesmo ponto será escrito na cor a, dado que o valor de x seja maior ou igual a um tom de cinza representado por n, caso contrário será escrito na cor 0.

Tal processo pode não ser eficiente em imagens que possuem variação intensa de iluminação(ex, imagens com sombra) e portanto, para que haja uma limiarização de maior precisão, podemos determinar a constante n da equação 2.1 através de uma operação local sobre os pixels adjacentes como é demonstrado na equação 2.2. Nesse caso, temos uma binarização adaptativa.

$$f(x) = \begin{cases} a & , x \geq thresh(x) \\ 0 & , caso\ contrário \end{cases} \tag{2.2}$$

Na figura 4 temos exemplos de aplicações de diferentes tipos de binarização sobre uma imagem. Podemos ver claramente a diferença de uma binarização simples para uma binarização adaptativa quando a iluminação não é uniforme na imagem original. A figura 4 mostra a imagem original na parte superior esquerda. A imagem superior direita representa uma binarização simples com limiar de cinza em 127 em uma escala de cinza de 0 (preto) a 255 (branco). A imagem inferior esquerda representa uma binarização adaptativa pela média. A imagem inferior direita representa uma binarização adaptativa gaussiana.

Figura 4 – Imagem original e seus diferentes níveis de binarização.



Fonte: (OPENCV, 2016)

2.5 Reconhecimento Ótico de Caracteres

Existe um ramo da visão computacional que tenta fazer análise de imagens digitais a fim de extrair informações que sejam processáveis como tipos de dados pelos computadores. Quando tal reconhecimento é voltado para caracteres escritos à mão ou texto impresso, temos a definição de Reconhecimento Ótico de Caracteres (do inglês, *Optical Character Recognition* - OCR) (KORNAI, 2003).

Podemos encontrar aplicações para OCR em sistemas de leitura de placas veiculares, correção de gabaritos, sistemas de aposta automatizados, tradutores e até mesmo ferramentas de acessibilidade para pessoas com deficiência, sendo um tema bastante pesquisado na computação.

O processo para reconhecimento de caracteres pode ser dividido em etapas. Em seu trabalho, os autores (GUPTA; AHUJA; AICH, 2009) definem um modelo amplamente utilizado, demonstrado na figura 5 com o seguinte pipeline:

Hand Written Text: texto escrito a mão, ou texto utilizado como base para o processo;

Optical Scanning: captura de imagem do texto através de um dipositivo óptico (e.g. câmera digital);

Location Segmentation: segmentação da imagem obtida em regiões de interesse, i. e. regiões que contêm caracteres para reconhecimento;

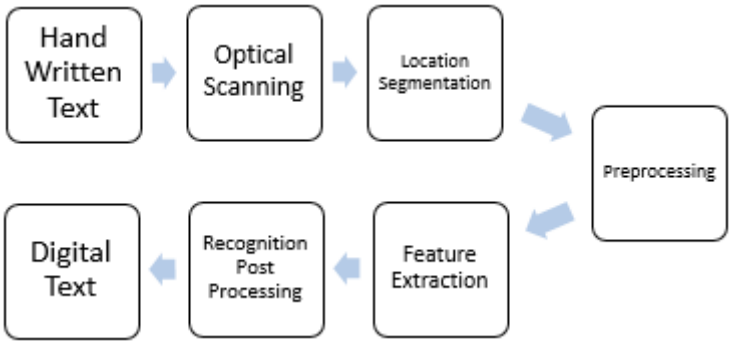
Preprocessing: aplicação de filtros para eliminar ruídos indesejados da imagem a fim de obter uma maior taxa de reconhecimento;

Feature Extraction: extração de caracteres da imagem;

Recognition Post Processing: etapa de pós processamento, onde pode ocorrer a correção de palavras obtidas, realizando uma comparação do texto com um dicionário da linguagem em questão.

Digital Text: O texto é retornado em formato digital.

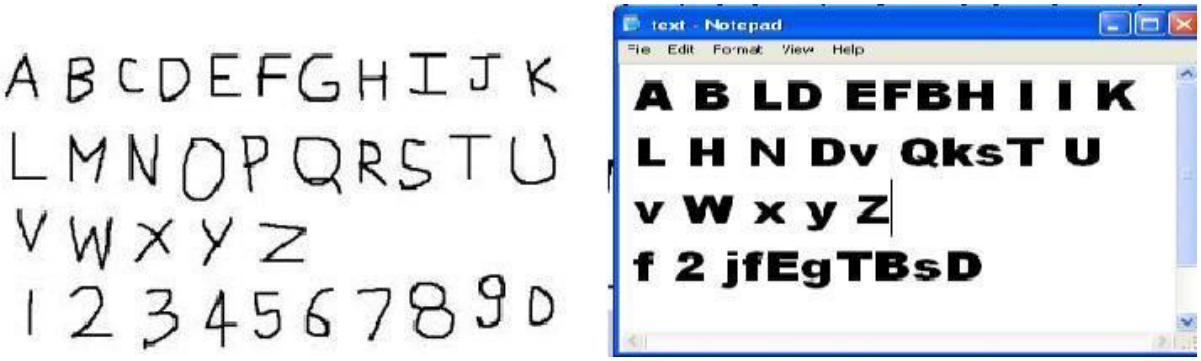
Figura 5 – Etapas de um processo de OCR



Fonte: (GUPTA; AHUJA; AICH, 2009)

A figura 6 mostra um exemplo de OCR com uma imagem digital de entrada e o reconhecimento obtido na saída. Como podemos notar, há caracteres que não foram reconhecidos corretamente. Isso pode ocorrer devido uma falha em uma das etapas de reconhecimento descritas anteriormente.

Figura 6 – Exemplo de entrada (texto à esquerda) e saída (texto à direita) em aplicações OCR.



Fonte: (GUPTA; AHUJA; AICH, 2009)

2.5.1 Luck Checker

Este aplicativo foi desenvolvido no laboratório iMobilis (IMOBILIS, 2017) com o intuito de fornecer uma maneira de gerenciar apostas lotéricas feitas por um jogador. Para

isso, o aplicativo armazena os resultados dos jogos feitos pelo usuário e além disso, possui um módulo de verificação de cupons lotéricos premiados. Para que um usuário confira o resultado do jogo, basta que seja tirada uma foto do cupom lotérico, que os dados serão obitidos através de OCR. Dessa forma, o usuário poderia conferir seus jogos de maneira mais ágil e evitar novas apostas com números já sorteados.

As taxas de reconhecimento não alcaçaram um nível de confiança alto suficiente para que o mesmo fosse comercializado. No entanto, devido a quantidade de processamento necessária para que um cupom fosse reconhecido, utilizou-se como base desse trabalho o módulo de reconhecimento desse aplicativo.

2.6 Trabalhos Relacionados

Devido ao recente salto na evolução dos dispositivos móveis, a literatura sobre esse tipo de comparação é escassa, porém existem muitas obras que trazem indícios dessa evolução computacional constante e que são partes fundamentais desse trabalho.

Um grupo de pesquisadores de Stanford desenvolveu um aplicativo capaz de reconhecer números de cupons lotéricos através de um dispositivo móvel operando sobre Android OS. No entanto, o trabalho de (GUPTA; ZOU, 2012) apresenta algumas limitações como por exemplo, a utilização do modelo cliente-servidor, onde o dispositivo móvel é responsável apenas pela captura da imagem e envio para um servidor local da universidade. Além disso, o processamento foi feito por um *software* robusto de licença comercializável com alto custo (MATLAB). Com resultados positivos no reconhecimento, o grupo também mostrou a viabilidade do sistema cliente-servidor para uma aplicação móvel.

Como já sabemos, existem alguns desafios (i.e., limitação de recursos) no desenvolvimento de *software* para os dispositivos móveis (MEDVIDOVIC et al., 2003). O trabalho de (PARK; KWON, 2009), porém, nos mostra um algoritmo OCR adaptado para dispositivos embarcados sem perda de qualidade e com ganho no tempo de execução quando comparado ao algoritmo original que operava sobre um *desktop*. Isso foi possível devido ao isolamento das funcionalidades do algoritmo original, substituição dos cálculos de ponto flutuante, aproximação linear de funções matemáticas e um sistema dinâmico de aplicação de *templates* para o reconhecimento. Em suma o trabalho desses autores traz evidências de que é possível realizar processamento OCR moderadamente complexo em dispositivos móveis.

Na busca por uma arquitetura de desenvolvimento mais eficiente, (HAMAD; SAAD; ABED, 2010) apresenta uma comparação entre dois tipos de *web services*. O primeiro é o *Simple Object Access Protocol* (*SOAP*), um protocolo padrão em tal tipo de atividade. O segundo é visto apenas como uma arquitetura de desenvolvimento conhecida como *Representational State Transfer* (*REST*). Ela se utiliza de requisições através de uma *url*

utilizando o protocolo HTTP para retornar o resultado. O serviço implementado sobre um dispositivo móvel faz requisições para dois tipos de operação, uma soma e uma concatenação. As métricas utilizadas para avaliação de desempenho foram tempo de resposta e tamanho da mensagem enviada para dois tipos de dados distintos. Como resultado, verificou-se que a arquitetura REST foi claramente mais eficiente. Os autores ainda afirmam que a robustez do SOAP pode ter sido o motivo da sua queda de desempenho.

Por tratar-se de uma aplicação com finalidades simples(consultas), cogitou-se a utilização de servidores móveis, como os vistos em (MIZOUNI et al., 2011). Nesse trabalho, foi proposta uma arquitetura para *web services* móveis (*m-service*) e em seguida, um teste de desempenho de aplicações baseadas na arquitetura proposta. Como conclusão, os autores, assim como em outros estudos, apoiam o uso da arquitetura REST e acreditam que com a evolução tecnológica será cada vez mais simples a construção de um *m-service* de qualidade. No entanto, as aplicações desse tipo de serviço são exclusivas para casos em que a mobilidade é necessária, como por exemplo, o rastreio de pacotes entregues pelo correio. Para casos em que é possível a utilização de computação estática, os autores enfatizam que os *m-services* não devem ser utilizados como substitutos. Isso se deve ao poder de processamento limitado que os dispositivos móveis têm em comparação aos *desktops*.

3 Desenvolvimento

3.1 Motivação e Objetivos

3.1.1 Motivação

A cada dia que passa, podemos contemplar a crescente inclusão de dispositivos móveis a nossa volta. O custo desses dispositivos está cada vez menor, o que os torna fortes candidatos a sucessores dos computadores atuais. Além disso, tais dispositivos vêm acompanhados de diversos periféricos como microfones, acelerômetros, câmeras, etc. que podem aumentar a funcionalidade de um aparelho. Sistemas de entrada alternativos fornecem uma maneira mais prática e divertida de interação para os usuários.

Um desses sistemas de entrada que vêm ganhando atenção dos desenvolvedores são as câmeras. O motivo para isso é que uma imagem, além de possuir grande densidade de informação, pode ser de mais fácil compreensão para a população em geral, independentemente de sua origem ou educação. Além disso, é muito mais rápido pressionar o botão do obturador do que digitar uma única sentença. Um dos ramos da ciência que busca interpretar os dados do mundo real através de lentes é conhecido como Visão Computacional.

Para que haja reconhecimento de dados em uma imagem é necessário que a mesma seja dividida e analisada em pequenas partes que são processadas individualmente através dos mais diversos algoritmos. Devido a sua complexidade, tais algoritmos geralmente exigem um *hardware* mais robusto.

Apesar de um dispositivo móvel atual ser capaz de aplicar tais algoritmos, o poder de processamento de um *desktop* ainda é muitas vezes superior ao de um *smartphone* ou *tablet*. Esses últimos, ainda que possuam frequências de processamento comercial igual aos computadores, não conseguem fornecer seu poder máximo de processamento pois podem aquecer muito e com isso, danificar o aparelho móvel podendo até oferecer perigo aos seus utilizadores.

Sendo assim, é interessante a proposta de um dispositivo com um poder de processamento mediano, que opere com o auxílio de um processamento externo ao aparelho. A solução viável atualmente é a computação em nuvem (BUYYA et al., 2009). Com ela, podemos utilizar o conceito de "terminais burros" que tem como função principal exibir o resultado do processamento de uma máquina superior em capacidades computacionais (e.g. servidores).

Para consolidar esta ideia, este estudo buscou verificar a necessidade de aplica-

ção de processamento remoto para aplicativos que requerem um custo considerável de processamento. Foi tomado como base um aplicativo Android que utiliza conceitos de visão computacional (SHAFAIT; KEYSERS; BREUEL, 2008), que por natureza requerem um *hardware* razoavelmente robusto. Mais especificamente, o aplicativo busca reconhecer caracteres em cupons lotéricos através da captura de uma imagem por uma câmera a fim de verificar se o cupom é premiado ou não, dessa forma, um usuário poderia fazer uma verificação ágil e poderia ainda manter uma base de dados sobre suas apostas anteriores. Tal processo de reconhecimento requer uma série de etapas desde a preparação da imagem (e.g. redução de ruído, binarização (SAUVOLA; PIETIKÄINEN, 2000), posicionamento, projeção, etc.) até o tratamento dos dados obtidos (e.g. ferramenta ocr (BIENIECKI; GRABOWSKI; ROZENBERG, 2007) (MAJUMDER, 2009), correção por dicionário, etc.). Este aplicativo foi portado para um serviço web que realiza a mesma tarefa de processamento remotamente e, em seguida, foram feitas análises comparativas entre ambos em relação ao seu desempenho em geral.

Foi obtido como resultado esperado a criação de um novo serviço web, tornando o aplicativo portável para diversas plataformas. Além disso, buscou-se um novo ponto de vista em relação aos tipos de processamento escolhidos que possivelmente, pôde contribuir para o esclarecimento de dúvidas quanto a abordagem de implementação de um aplicativo robusto em *hardwares* menores. Esperou-se também que o processamento remoto fosse mais eficiente (GUPTA; ZOU, 2012), o que tornaria interessante a mescla do aplicativo com o serviço web, permitindo que o método mais eficiente fosse escolhido com base em informações como tamanho da imagem e conectividade do dispositivo.

3.1.2 Objetivos

O objetivo deste estudo é analisar o desempenho do módulo de reconhecimento de cupons lotéricos do aplicativo Luck Checker, desenvolvido no laboratório (IMOBILIS, 2017). Para tanto, uma versão do aplicativo móvel com processamento local foi comparada com uma versão do seu código portado para um serviço *web* e, por conseguinte, processado fora do dispositivo móvel. Como objetivos específicos do projeto, são destacados:

- Desenvolvimento de um serviço web para o reconhecimento de cupoons lotéricos.
- Desenvolvimento de um aplicativo móvel que realiza reconhecimento de cupons lotéricos no próprio dispositivo.
- Desenvolvimento de um aplicativo móvel que utiliza os serviços web e exibe os valores obtidos na tela.

- Análise comparativa do desempenho geral dos aplicativos referente ao seu custo computacional, tempo total de execução, portabilidade, viabilidade comercial e consumo energético.

3.2 Materiais e métodos

Neste capítulo serão abordados a metodologia de desenvolvimento deste trabalho e as ferramentas utilizadas no mesmo.

3.2.1 Materiais Utilizados

3.2.1.1 Servidor

O laptop Dell Inspiron N7110 foi utilizado como base para o desenvolvimento e também como servidor para o *web service*, suas características principais são:

- 8GB RAM DDR3 1333MHz
- Processador Intel Core i5-2410M CPU @ 2.3Ghz
- Intel HD Graphics 3000
- HD 750GB 7200 rpm
- Intel Centrino Wireless-N1030, 1x2 bgn (2.4GHz)

3.2.1.2 Dispositivos Móveis

O dispositivo móvel principal utilizado para testes foi o modelo Motorola Moto G5, que possui as seguintes características:

- S.O. Android 7.1 Nougat
- Câmera 13Mpx
- Display 5’ 1080 x 1920
- Chipset Qualcomm MSM8937 Snapdragon 430
- Processador Octa-core 1.4 GHz Cortex-A53
- GPU Adreno 505
- 3GB RAM
- Memória: 32GB expansível

Além disso, foram feitos testes com máquinas virtuais com características similares ao *smartphone* Google Nexus 4:

- CPU/ABI: Google APIs Intel Atom (x86)
- Resolução 768x1280
- Memória: 1,5GB

3.2.1.3 Android Studio

Android Studio é a IDE oficial para desenvolvimento de aplicativos Android (STUDIO, 2014). Além de ser a ferramenta padrão atual para desenvolvimento para a as aplicações da Google Play Store, ela é compatível com C++ e Android Native Development Kit (NDK), ambos necessários para a utilização de bibliotecas específicas para o processo de OCR, como é o caso da OpenCV. Além disso, visando o maior desempenho possível, o algoritmo de reconhecimento utilizou a linguagem C++ em ambas as plataformas em que foi desenvolvido.

3.2.1.4 Xampp

XAMPP é um ambiente de desenvolvimento PHP popular, de fácil instalação, que abriga as ferramentas MySQL, PHP e Perl (FRIENDS, 2017). Através dele podemos disponibilizar servidores para abrigar APIs e *web services* como foi feito neste trabalho.

3.2.1.5 Google Tesseract

Tesseract foi inicialmente desenvolvido pela Hewlett-Packard em 1985. É uma ferramenta capaz de reconhecer mais de 100 linguagens e pode de ser treinada para reconhecer outras linguagens caso necessário (SMITH, 2007). Em meados de 2005, teve seu código aberto e vem sendo mantido pela Google desde 2006.

Como esperado, o processo de OCR realizado por essa ferramenta segue as premissas mencionadas na seção 2.5. É importante mencionar que para maior taxa de acertos, a imagem de entrada deve ser filtrada e binarizada com antecedência. Em sua fase inicial, o Tesseract faz a detecção de possíveis candidatos ao reconhecimento, em seguida é feita a segmentação das regiões em busca de palavras, que são segmentadas em caracteres individuais e então é feita uma primeira classificação de caracteres por casamento de padrão. Durante essa etapa são atribuídos valores de satisfação de reconhecimento para cada caractere e uma lista de palavras reconhecidas. Após a primeira passada, podem haver palavras com baixo grau de satisfação de reconhecimento e por isso, é feita uma nova passada nessas palavras com os dados do reconhecimento obtidos anteriormente. O Tesseract possui tratamento de diversas situações como a inclinação das palavras, caracteres

incompletos ou mesmo contexto linguístico de uma palavra, entretanto são tratamentos básicos e portanto, não devemos nos tornar dependentes deles para o desenvolvimento de um aplicativo. É de suma importância que a imagem de entrada seja a mais correta possível, otimizando o processamento feito pela ferramenta.

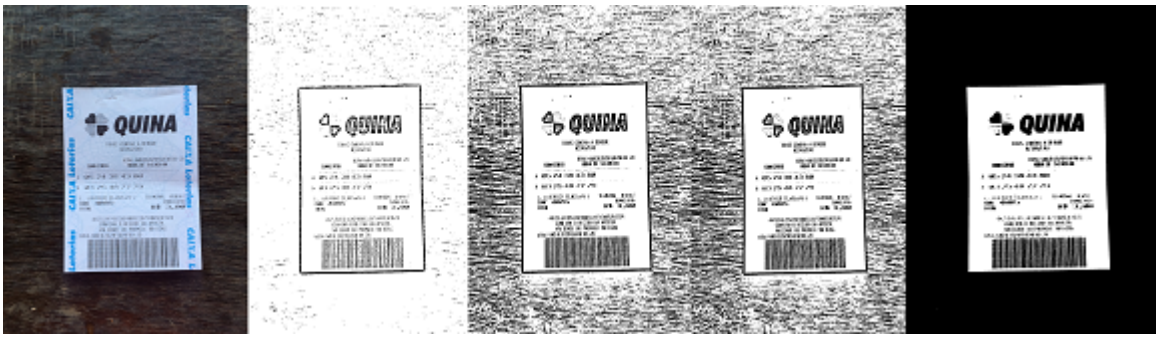
Seus desenvolvedores disponibilizam interação com C++, fazendo com que o Tesseract seja a ferramenta ideal para a parte de OCR remota deste trabalho. Além disso, há também fácil integração com Android através de bibliotecas disponibilizadas especificamente para a plataforma. Essas características foram determinantes na utilização dessa ferramenta no desenvolvimento das aplicações aqui estudadas.

3.2.1.6 OpenCV

OpenCV (Open Source Computer Vision Library) é uma biblioteca de código aberto para *softwares* relacionados a visão computacional e aprendizagem de máquina. Ela foi desenvolvida para fornecer estruturas comuns a aplicações de visão computacional, facilitando o desenvolvimento de aplicações comerciais que necessitam deste tipo de funcionalidade. A OpenCV possui licença e código aberto, permitindo que instituições acadêmicas ou comerciais a utilizem conforme sua necessidade. Além disso, ela fornece interfaces nas linguagens C, C++, Java e Python, para os mais diversos sistemas operacionais.

Neste trabalho, utilizamos essa biblioteca para realizar a captura e o pré-processamento da imagem, fornecendo assim uma entrada de dados refinada para o Tesseract. A figura 7 ilustra diversas saídas resultantes do processamento de imagem através da biblioteca OpenCV.

Figura 7 – Imagem original (esquerda) e diferentes formas de processamento com OpenCV



Fonte: Elaborada pelo autor

3.2.1.7 Microsoft Visual Studio Community 2013

O Visual Studio é uma das IDEs mais robustas do mercado, permitindo desde o desenvolvimento de aplicações *desktop* até aplicações *web* e móvel. Com a sua versão *Community*, a Microsoft introduz todo o poder da tradicional ferramenta com uma licença sem custo, buscando atrair a nova geração de desenvolvedores e incentivar o

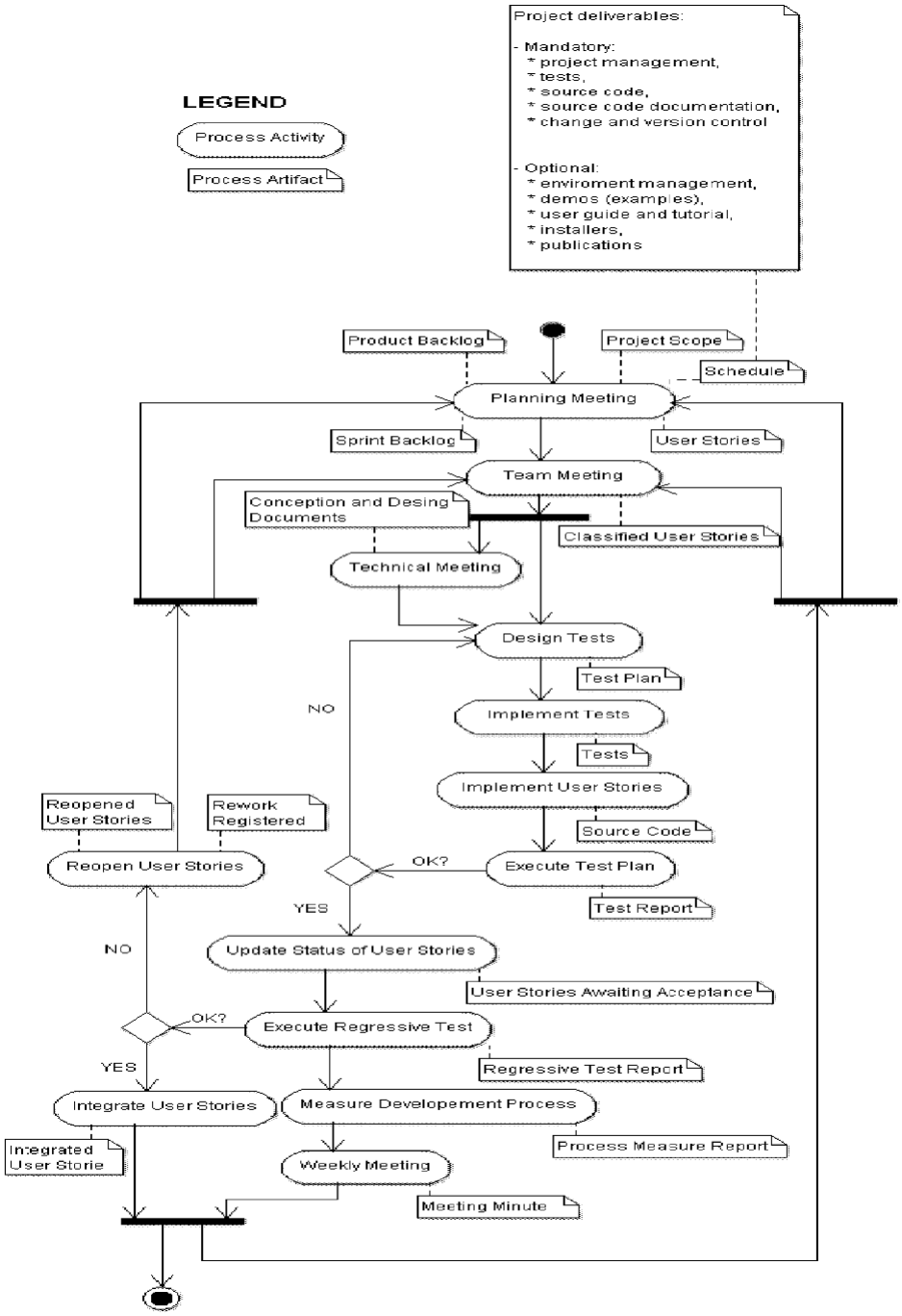
desenvolvimento com suas tecnologias. Algumas vantagens dessa plataforma são a instalação de *plug-ins* através do gerenciador nuGet, inteliSense, alta customização do ambiente de desenvolvimento, *build* específicas para *debug* e publicação do *software* e integração com a ferramenta Git.

3.2.2 Metodologia de Desenvolvimento

O trabalho foi desenvolvido nas dependências do iMobilis (IMOBILIS, 2017) - o Laboratório de Computação Móvel e Sistemas Embarcados da Universidade Federal de Ouro Preto - localizado no campus João Monlevade. O iMobilis tem como padrão um processo de desenvolvimento ágil conhecido como BOPE(PEREIRA; CARNEIRO; PEREIRA, 2013), desenvolvido para auxiliar estudantes de graduação a gerar produtos com qualidade. Ele foi idealizado e projetado por um dos atuais coordenadores do laboratório iMobilis da Universidade Federal de Ouro Preto, campus de João Monlevade.

O BOPE contempla características de processos como Scrum, XP e PMBoK. Algumas de suas características fundamentais são as reuniões frequentes entre aluno e orientador e a produção de artefatos que levam ao produto final. A figura 8 ilustra as etapas deste processo de desenvolvimento.

Figura 8 – Processo de Desenvolvimento BOPE.



Fonte: (PEREIRA; CARNEIRO; PEREIRA, 2013)

3.2.3 Escopo e Limitações

Para realizar os testes de computação em nuvem, utilizou-se uma rede *wireless* local. Porém, uma rede local geralmente é mais rápida e possui um tráfego de dados menor do que uma rede externa. Portanto, é esperado que os resultados dos mesmos testes em um cenário de rede externa e real aumentem o tempo de execução de aplicativos dessa natureza proporcionalmente aos fatores como largura de banda, latência e tráfego na rede. Em contrapartida, o servidor que suporta um serviço em nuvem geralmente tem seu poder de processamento maior do que um *desktop* e portanto, é possível que haja um tempo de processamento menor do que o registrado durante este trabalho em um cenário real.

3.2.4 Teste de Aplicação com Processamento Local

Esse teste consiste na utilização do mecanismo de reconhecimento utilizado no *app* Luck Checker, desenvolvido no laboratório (IMOBILIS, 2017). O núcleo do aplicativo foi portado para uma aplicação móvel de testes e, ao fornecer uma imagem de cupom lotérico aleatória, foram coletados os dados a respeito do tempo de processamento através de *logs* e mecanismos de *debug* do Android Studio.

O mesmo código foi portado para o servidor *web* e, através de um navegador, a mesma imagem fornecida na aplicação local foi testada. Dessa vez, as informações sobre o tempo de processamento foram coletadas em *scripts* php e exibidos na interface *web*. O teste foi realizado para 34 amostras de cupons.

3.2.5 Teste de Aplicação com Processamento Remoto

Foi desenvolvida uma aplicação que faz upload de uma imagem do dispositivo móvel para o servidor *web* - o mesmo servidor utilizado em 3.2.4. Os dados sobre o processamento foram coletados dos logs do Android Studio. Vale ressaltar que, como mencionado em 3.2.3, todos os testes foram realizados em rede *wireless* local, com tráfego de dados leve através de um roteador popular de 150Mbps. O teste foi realizado para 34 amostras de cupons.

4 Resultados

Este capítulo mostra o resultado obtido da coleta de dados sobre os dois aplicativos.

4.1 Tendência de Mercado e Comercialização

O custo envolvido considerado em ambos os casos levou em conta toda a infraestrutura para fornecer um serviço, bem como os custos de manutenção de software para uma única plataforma a longo prazo.

Avaliando o custo envolvido em cada abordagem, nota-se que o processamento remoto é mais custoso em sua fase inicial devido aos componentes de infraestrutura envolvidos. Portanto do ponto de vista do fornecedor, esse tipo de processamento mostrou-se mais viável em casos de software como serviço, uma vez que os subsídios serão constantes, permitindo a continuidade do produto. Do ponto de vista do consumidor, um *hardware* popular seria suficiente para executar a aplicação.

No cenário especificado, aplicações desenvolvidas para processamento local tem um custo inicial similar ao custo da parte de *software* de processamento remoto apenas, o que a torna mais viável finaceiramente. O modelo de licença comercial mostrou-se mais atrativo do ponto de vista do fornecedor, uma vez que o custo de manutenção seria limitado apenas aos custos envolvidos com o desenvolvimento do software. Dessa forma, as aplicações podem possuir ciclo de vida menor, dando oportunidade para comercialização de novas versões. Do ponto de vista do consumidor, um *hardware* mais robusto seria necessário para executar a aplicação.

Devido as baixas taxas de reconhecimento, o algoritmo necessita de maior refinamento para uma possível comercialização.

4.2 Processamento

Os testes demonstraram que um tempo similar de interação com o usuário pode não siginificar a mesma carga de processamento. Nota-se na tabela 2 que o tempo gasto em processamento local foi ligeiramente superior ao processamento remoto, contudo, as chamadas do sistema apresentaram um tempo aproximadamente igual. Além disso, constatou-se que quanto maior o uso de CPU, maior o consumo energético. Isso se mostra na tabela 2, onde o consumo energéritco é da ordem de aproximadamente 20% maior para o processamento local. Por fim, o tamanho do aplicativo com processamento local é maior que o tamanho do aplicativo de processamento remoto, devido ao numero maior de

bibliotecas utilizadas para o processamento local.

Tabela 2 – Comparação geral entre diferentes tipos de processamento

Tipo de Processamento	Local	Remoto
Tempo total de execução	16m23s	15m4s
Tempo de CPU por usuário	12m1s244ms	2m47s437ms
Tempo de CPU pelo sistema	45s250ms	46s70ms
Consumo estimado de Bateria	0,89%	0,19%
Tempo de aplicação ativa	12m45s865ms	10m26s342ms
Tamanho do Aplicativo	53,53MB	3,8MB

Fonte: Elaborada pelo autor

4.2.1 Processamento Local

A tabela 3 lista as informações gerais do teste envolvendo todas as amostras. Surpreendentemente, o tempo de execução desse tipo de processamento mostrou-se ligeiramente maior que o processamento remoto. Isso se explica devido a utilização de uma rede local nos testes do outro tipo de processamento, bem como um processador mais poderoso no servidor.

Tabela 3 – Informações Gerais de Processamento Local

Tempo de execução	16m23s
Tamanho da massa de dados transferida	0
Tamanho médio de arquivo	2.44MB
Tempo médio de processamento por arquivo	28.91s
Consumo estimado de bateria	0.89%

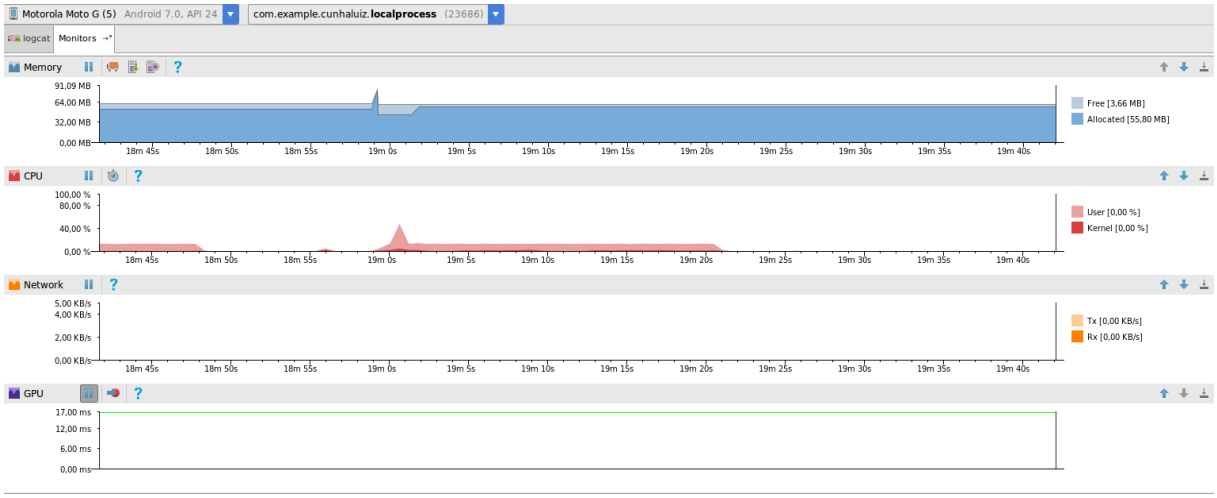
Fonte: Elaborada pelo autor

Analisando detalhadamente uma das amostras fornecidas ao aplicativo de processamento local, o tempo de duração, do lançamento do aplicativo até o seu encerramento, foi de aproximadamente 30 segundos e o módulo nativo consumiu em seu tempo total de execução cerca 21 segundos (figura 9), aproximadamente 60% do tempo total da aplicação.

A figura 9 ilustra o processo em maiores detalhes. Para que seja feito o processamento de uma imagem, deve haver memória suficiente no dispositivo e o sistema de gerenciamento da mesma deve ser eficiente. Em um caso de sucesso, não foi necessário grande quantidade de memória, porém a liberação desse recurso não foi imediata. Ainda sobre o monitor de memória, a figura 10 evidencia a quantidade de memória necessária para esse tipo de aplicação. O espaço com coloração mais clara no gráfico representa a quantidade de memória alocada livre durante a execução dos testes. A figura 11 mostra o momento em que uma nova imagem é selecionada e um novo ciclo de OCR tem início.

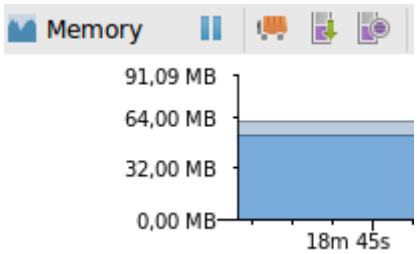
Ainda na figura 9, o gráfico do monitor de CPU traz o tempo em que a aplicação utilizou o algoritmo de processamento de imagens. A partir dos 19 minutos, temos um pico

Figura 9 – Análise dos recursos de *hardware* do dispositivo durante o processamento local



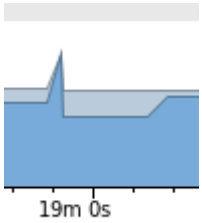
Fonte: Elaborada pelo autor

Figura 10 – Quantidade de Memória Reservada para Aplicação de Processamento Local



Fonte: Elaborada pelo autor

Figura 11 – Seleção de Imagem em Aplicação de Processamento Local

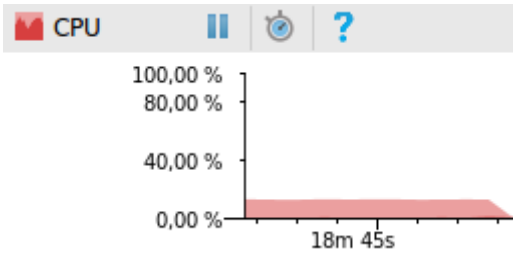


Fonte: Elaborada pelo autor

que informa a chamada da função que captura uma imagem e em seguida, a quantidade de recursos de CPU utilizados no processamento e na exibição dos resultados na tela. Como podemos ver nas figuras 12 e 13, a maior parte dos recursos de CPU foi utilizada pelo usuário representada pela cor clara, porém há uma parte que é reservada ao do Kernel do Android representada pela cor escura.

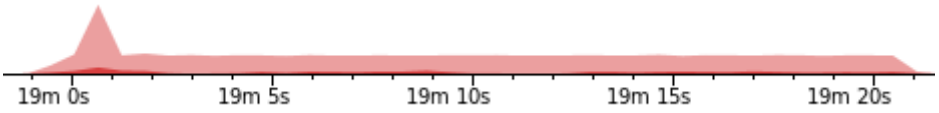
Como não há uma filtragem da imagem em tempo real, o uso da GPU é pequeno e breve. Notou-se que a aplicação executa na maior parte do tempo respeitando a margem de 60fps denotada pela linha verde. Essa linha só foi ultrapassada nos testes com dispositivos móveis legados. Valores acima desta linha podem significar um tempo de processamento

Figura 12 – Uso de CPU em Aplicação de Processamento Local



Fonte: Elaborada pelo autor

Figura 13 – Tempo de CPU em Aplicação de Processamento Local



Fonte: Elaborada pelo autor

muito maior e uma resposta com menor taxa de frames por segundo. Por se tratar de processamento local não houve atividade significante na rede, como mostra o gráfico Network da figura 9.

A tabela 4 mostra o tempo gasto na operação de reconhecimento de caracteres de cada arquivo. O tempo médio constatado nesse tipo de operação foi de aproximadamente 20 segundos. Porém, devido ao seu alto valor de desvio padrão. Isso ocorre devido a fragilidade do reconhecimento de caracteres, onde diversos fatores intrínsecos e extrínsecos são extremamente relevantes no resultado final.

Tabela 4 – Tempo de Processamento Local por Arquivo

Arquivo	Tempo da Operação (ms)	Erro da média
IMG_2275.JPG	16758	-17,398
IMG_2276.JPG	28782	31,64631
IMG_2277.JPG	20845	5,619771
IMG_2273.JPG	25244	22,0664
IMG_2274.JPG	6679	-194,558
IMG_2271.JPG	19856	0,918821
IMG_2272.JPG	9526	-106,525
IMG_2267.JPG	11982	-64,1926
IMG_2268.JPG	12232	-60,8368
IMG_2270.JPG	10686	-84,1059
IMG_2264.JPG	38710	49,17706
IMG_2266.JPG	34547	43,05277
IMG_2263.JPG	32774	39,97205
IMG_2261.JPG	23502	16,28985
IMG_2262.JPG	14322	-37,366
IMG_2258.JPG	10180	-93,257
IMG_2259.JPG	15950	-23,3452
IMG_2260.JPG	22956	14,29884
IMG_2255.JPG	11261	-74,7053
IMG_2256.JPG	17527	-12,2472
IMG_2257.JPG	11045	-78,1219
IMG_2252.JPG	29078	32,34212
IMG_2253.JPG	15607	-26,056
IMG_2254.JPG	11653	-68,8283
IMG_2249.JPG	32618	39,68496
IMG_2250.JPG	9625	-104,401
IMG_2251.JPG	47029	58,16718
IMG_1875.JPG	23382	15,86024
IMG_2248.JPG	14658	-34,2172
5.jpg	28983	32,12035
IMG_1851.JPG	13336	-47,5222
IMG_1862.JPG	10603	-85,5471
IMG_1874.JPG	17739	-10,9057
4.jpg	19226	-2,32788
média	19673,56	
desvio padrão	7816,246	

Fonte: Elaborada pelo autor

4.2.2 Processamento Remoto

A tabela 5 mostra as informações gerais do processamento remoto. Como mencionado na seção 4.2.1, o tempo de processamento desse método foi menor, diferente do que era esperado. Isso ocorre devido as condições favoráveis do ambiente de testes (rede local de alta velocidade e servidor dedicado). Em contrapartida, existe carga na rede e

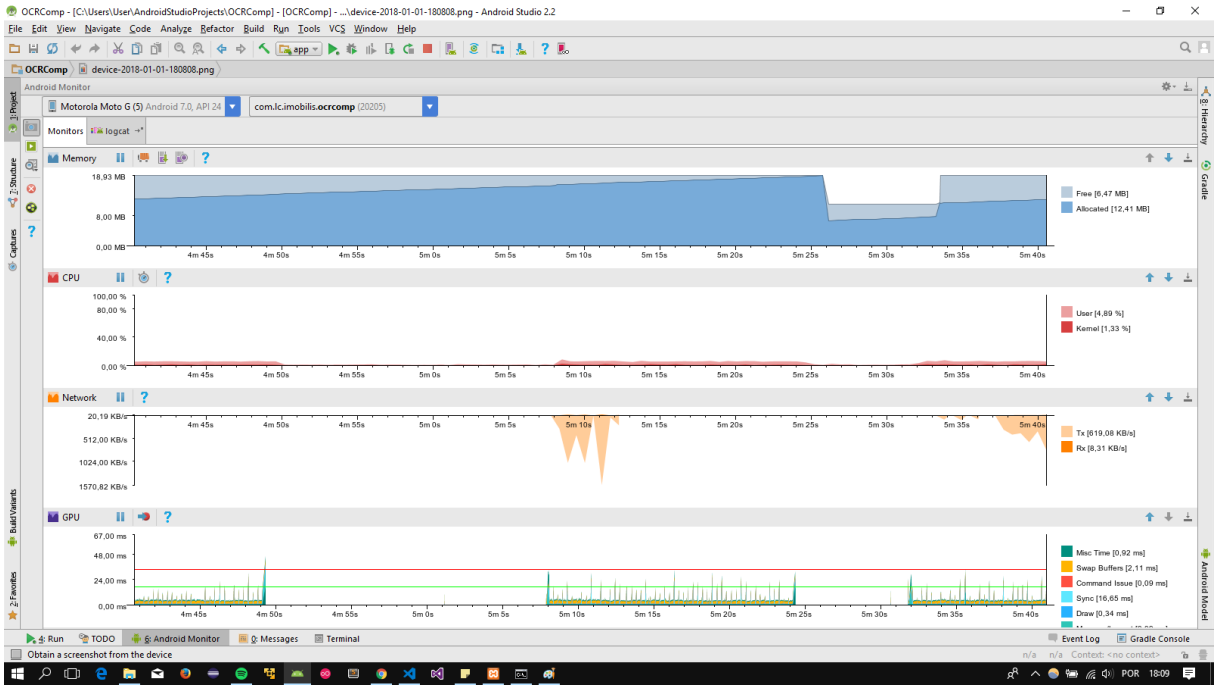
o consumo energético não leva em consideração os outros componentes envolvidos, que podem ter grande contribuição no custo final do produto.

Tabela 5 – Informações Gerais de Processamento Remoto

Tempo de execução	15m4s
Tamanho da massa de dados transferida	83MB
Tamanho médio de arquivo	2,44MB
Tempo médio de processamento por arquivo	25,58s
Consumo estimado de bateria	0,19%

Fonte: Elaborada pelo autor

Figura 14 – Análise dos recursos de hardware do dispositivo durante o processamento remoto

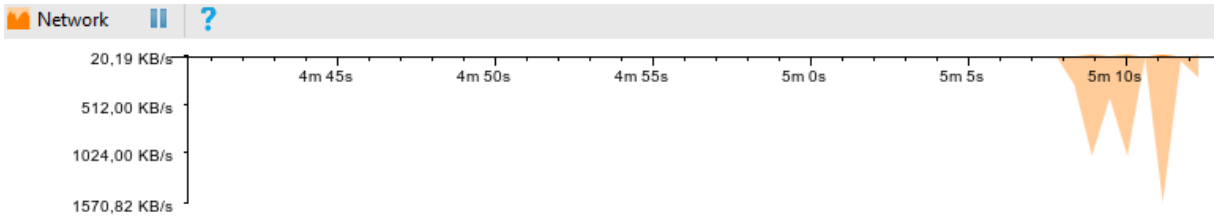


Fonte: Elaborada pelo autor

Analisando detalhadamente um cupom aleatório fornecido ao aplicativo de processamento remoto representado pela figura 14, foi obtido um tempo de processamento de aproximadamente 16 segundos e o tempo total de execução de aproximadamente 18 segundos, valor inferior ao tempo de execução do processamento local. Isso se deve principalmente a diferença dos poderes de processamento, uma vez que a imagem de teste foi a mesma. Essa alteração pode ser visualizada no gráfico de rede na figura 15, onde, diferente do que constatou-se em 4.2.1, há dados sendo transferidos.

Quanto a memória, o limite utilizado foi menor do que em 4.2.1 (figura 16) e em poucos momentos houve memória disponível. Isso ocorreu porque menos entidades poderiam ser liberadas pelo *garbage collector* do Android, afinal, não há processamento

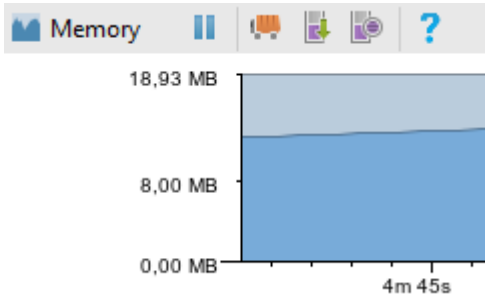
Figura 15 – Análise de Tráfego de Rede em Processamento Remoto



Fonte: Elaborada pelo autor

robusto, apenas requisição e resposta. Além disso, a maior parte dessa memória é utilizada em conjunto com a GPU.

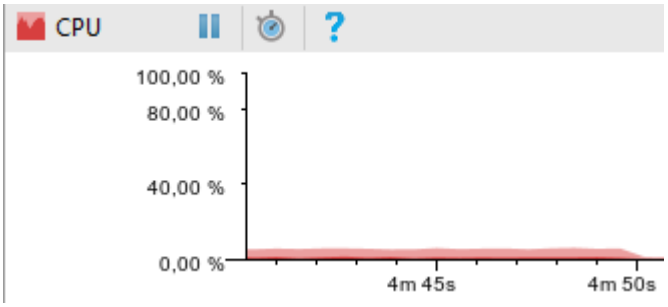
Figura 16 – Consumo de Memória em Processamento Remoto



Fonte: Elaborada pelo autor

As figuras 17 e 18 indicam respectivamente a quantidade de recursos e o tempo de CPU consumidos durante o processamento de um cupom. Assim como em 4.2.1, o tempo de CPU utilizado pelo usuário (cor clara) foi maior que o tempo de CPU para chamadas do sistema (cor escura).

Figura 17 – Uso de CPU em Processamento Remoto



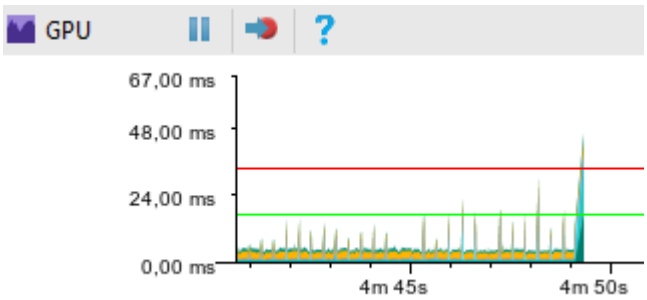
Fonte: Elaborada pelo autor

Figura 18 – Tempo de CPU em Processamento Remoto para um cupom

Fonte: Elaborada pelo autor

O monitor de GPU representado na figura 19, diferente do caso de processamento local, teve maior participação no processo, pois, apesar de não haver grande interação com usuário, o aplicativo exibe uma animação informando o usuário que está sendo feito um processamento e mais tarde modifica a tela novamente para exibir o resultado. A cor laranja em destaque representa o processamento feito pela entidade, enquanto a cor azul, é o tempo gasto desenhando na tela.

Figura 19 – Uso de GPU em Processamento Remoto



Fonte: Elaborada pelo autor

A tabela 6 lista maiores detalhes do tempo da operação gasto em cada cupom e sua margem de erro em relação ao tempo médio de processamento. Dessa vez, a média obtida foi estimada em aproximadamente 15 segundos. Ao levar-se em consideração que o módulo de processamento é idêntico para ambos os casos, a justificativa para essa diferença de resultados encontra-se no hardware dos mesmos. Contudo, a utilização desse modelo, de acordo com o cenário de teste definido e considerando o ponto de vista do fornecedor do serviço, resultou em maior custo de manutenção do que o modelo apresentado em 4.2.1.

Tabela 6 – Tempo de Processamento Remoto Por Arquivo

Arquivo	Tempo da operação (ms)	Erro da média (%)
IMG_2275.JPG	23786	34.56
IMG_2276.JPG	16315	4.59
IMG_2277.JPG	20140	22.71
IMG_2273.JPG	14411	-8.02
IMG_2274.JPG	15118	-2.96
IMG_2271.JPG	15029	-3.57
IMG_2272.JPG	12644	-23.11
IMG_2267.JPG	16366	4.89
IMG_2268.JPG	14655	-6.22
IMG_2270.JPG	13616	-14.32
IMG_2264.JPG	17479	10.94
IMG_2266.JPG	16235	4.12
IMG_2263.JPG	20525	24.16
IMG_2261.JPG	14868	-4.70
IMG_2262.JPG	17534	11.22
IMG_2258.JPG	14652	-6.24
IMG_2259.JPG	14689	-5.97
IMG_2260.JPG	16186	3.83
IMG_2255.JPG	16249	4.20
IMG_2256.JPG	18449	15.63
IMG_2257.JPG	16663	6.58
IMG_2252.JPG	14237	-9.34
IMG_2253.JPG	14308	-8.79
IMG_2254.JPG	13966	-11.46
IMG_2249.JPG	13525	-15.09
IMG_2250.JPG	13340	-16.69
IMG_2251.JPG	13818	-12.65
IMG_1875.JPG	11537	-34.92
IMG_2248.JPG	13145	-18.42
5.jpg	18442	15.59
IMG_1851.JPG	15768	1.28
IMG_1862.JPG	14831	-4.96
IMG_1874.JPG	11644	-33.68
4.jpg	15077	-3.24
Média	15566.08	
Desvio Padrão	1894.80	

Fonte: Elaborada pelo autor

4.3 Interface Web

O teste realizado na interface web utilizou em seu núcleo o mesmo algoritmo das seções 4.2.1 e 4.2.2, porém seus dados foram coletados através de algoritmos em PHP. Nesse caso, não houveram filtros apenas para números, tendo como resultado na tela o cupom como um todo. O tempo de duração do processamento foi o mesmo obtido no

capítulo 4.2.2, como esperado do algoritmo.

4.4 Discussão

Considerando o cenário mencionado em 4.1, a tabela 7 mostra uma visão geral das vantagens dos tipos de processamento obtidas nesse trabalho.

Tabela 7 – Processamento Local x Processamento: Vantagens

	Local	Remoto
Custo	X	
Comercialização	Licença Comercial	SaaS
Consumo Energético(Cliente)		X
Manutenção	X	
Desempenho		X
Flexibilidade		X

Fonte: Elaborado pelo autor

O custo para o desenvolvimento de um produto do ponto de vista do fornecedor, quando limitado a um único sistema operacional, é maior para uma aplicação em nuvem devido a infraestrutura que envolve o *software*.

A comercialização de um *software* local é mais simples e direta, uma vez que o tempo de suporte ao produto pode ser menor do que um serviço em nuvem. Por isso, uma licença comercial aparentou ser a melhor opção para aplicações locais. Para o serviço em nuvem, a disponibilidade do serviço pode ser custosa, e portanto, uma renda contínua parece mais atrativa.

O consumo energético nos dispositivo em que a aplicação executa foi menor para o caso de aplicação em nuvem. Logo, para o cliente é mais atrativo a aplicação que consome menos bateria, mantendo o aparelho ativo por mais tempo.

No caso de uma abordagem *SaaS*, o fornecedor pode ter um custo deveras contínuo de manuteção e atualização do seu equipamento, enquanto um software de licença comercial não possui custos para manutenção de *hardware*, apenas de *software*.

Para o cliente a computação em nuvem exige menos recursos de *hardware*, o que pode dar a impressão de que a aplicação remota é mais eficiente (possui melhor desempenho) caso a conexão com o serviço forneça condições favoráveis para tal.

Por fim, caso o fornecedor queira expandir seus serviços, a aplicação remota poderá atender mais plataformas sem maior esforço, uma vez que o acesso ao serviço pode ser feito em navegadores independentes de plataforma, tendo sua distribuição mais flexível do que uma aplicação local.

5 Conclusão

Este trabalho apresentou a avaliação de desempenho entre diferentes abordagens de processamento de dados para uma aplicação móvel. Em um caso, no processamento no próprio dispositivo, foi explorada a capacidade do dispositivo móvel de receber uma imagem, aplicar filtros e reconhecer os caracteres de um cupom; no outro, foi testada a mesma funcionalidade sendo processada por um servidor na rede e por fim, aplicamos o mesmo teste no servidor.

Apesar do servidor possuir um maior poder de processamento, o tempo de execução dos aplicativos foi similar, com uma leve vantagem para o processamento remoto. Além disso, utilizou-se menos recursos de memória e CPU. Outra grande vantagem desse método foi o tamanho final da aplicação, uma vez que, para a utilização da biblioteca OpenCV é necessário instalação de arquivos adicionais. Além disso, os testes indicaram que a aplicação em nuvem forneceu maior flexibilidade na manutenção e distribuição da sua funcionalidade em forma de API para outras aplicações. Entretanto, para um fornecedor, o custo financeiro de manutenção desse modelo mostrou-se maior do que o custo de manutenção para processamento local, uma vez que a aplicação exige uma infraestrutura completa a ser mantida.

Quanto ao processamento local, foi apresentado nos testes menor uso de GPU e o não uso da rede, fornecendo uma maior disponibilidade do aplicativo e um custo significativamente menor, uma vez que não existe outros componentes para dar suporte a solução.

O algoritmo escolhido se mostrou ideal para a avaliação de desempenho, pois, devido as suas características intrínsecas, os testes puderam mostrar que a memória de um dispositivo móvel ainda pode ser muito limitada, fazendo com que a aplicação quebrasse em menor tempo que a aplicação remota em caso de erro.

Em suma, o estudo mostrou que o poder de processamento de um dispositivo móvel na atualidade pode não se apresentar mais como único grande fator no desenvolvimento de software. Ao seu lado, agora deve ser levada em consideração a melhor abordagem possível para uma aplicação, e considerando que não há um modelo perfeito para tal tarefa, fatores como o público alvo e cultura regional devem servir de apoio para a tomada de decisão na escolha do modelo de software.

Referências

ARMBRUST, M. et al. A view of cloud computing. *Communications of the ACM*, ACM, v. 53, n. 4, p. 50–58, 2010. Citado 2 vezes nas páginas 18 e 19.

BIENIECKI, W.; GRABOWSKI, S.; ROZENBERG, W. Image preprocessing for improving ocr accuracy. In: IEEE. *Perspective Technologies and Methods in MEMS Design, 2007. MEMSTECH 2007. International Conference on*. [S.l.], 2007. p. 75–80. Citado na página 28.

BURGERN, W.; BURGER, M. *Principles of Digital Image Processing Fundamental Techniques*. [S.l.]: Springer-Verlag London, 2009. Citado na página 21.

BUYYA, R. et al. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, Elsevier, v. 25, n. 6, p. 599–616, 2009. Citado na página 27.

CUSUMANO, M. Cloud computing and saas as new computing platforms. *Communications of the ACM*, ACM, v. 53, n. 4, p. 27–29, 2010. Citado na página 18.

FLURRY, A. B. *U.S. Consumers Time-Spent on Mobile Crosses 5 Hours a Day*. 2017. Artigo da Web. Disponível em: <http://flurrymobile.tumblr.com/post/157921590345/us-consumers-time-spent-on-mobile-crosses-5>. Acesso em: 09 dez. 2017. Citado na página 16.

FRIENDS, A. Xampp. *Apache Friends*, 2017. Disponível em: <https://www.apachefriends.org/pt_br/index.html>. Citado na página 30.

GOGGIN, G. *Cell phone culture: Mobile technology in everyday life*. [S.l.]: Routledge, 2012. Citado na página 16.

GUPTA, A.; ZOU, T. Mobile lottery ticket recognition using android phone. 2012. Citado 2 vezes nas páginas 25 e 28.

GUPTA, T.; AHUJA, C.; AICH, S. Optical character recognition. *IJETAE*, v. 4, n. 58, p. 402–405, 2009. ISSN 2250-2459. Disponível em: <http://www.ijetae.com/files/Volume4Issue9/IJETAE_0914_58.pdf>. Citado 2 vezes nas páginas 23 e 24.

HAMAD, H.; SAAD, M.; ABED, R. Performance evaluation of restful web services for mobile devices. *Int. Arab J. e-Technol.*, v. 1, n. 3, p. 72–78, 2010. Citado na página 25.

IMOBILIS. *iMobilis - Laboratório de computação móvel e embarcada da UFOP-JM*. 2017. Página da Web. Disponível em: <http://www.imobilis.ufop.br/>. Acesso em: 21-05-2017. Citado 4 vezes nas páginas 24, 28, 32 e 34.

KORNAI, A. Optical character recognition. 2003. Citado na página 23.

MAJUMDER, A. Image processing algorithms for improved character recognition and components inspection. In: IEEE. *Nature & Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on*. [S.l.], 2009. p. 531–536. Citado na página 28.

MALLADI, R.; AGRAWAL, D. P. Current and future applications of mobile and wireless networks. *Communications of the ACM*, ACM, v. 45, n. 10, p. 144–146, 2002. Citado na página 20.

MEDVIDOVIC, N. et al. Software architectural support for handheld computing. *Computer*, IEEE, v. 36, 2003. Disponível em: <http://gen.lib.rus.ec/scimag/index.php?s=10.1109/mc.2003.1231196>. Citado na página 25.

MIZOUNI, R. et al. Performance evaluation of mobile web services. In: IEEE. *Web Services (ECOWS), 2011 Ninth IEEE European Conference on*. [S.l.], 2011. p. 184–191. Citado na página 26.

MOORE, G. E. et al. Cramming more components onto integrated circuits. *Proceedings of the IEEE*, v. 86, n. 1, p. 82–85, 1998. Citado na página 17.

OGRAPH, B. T.; MORGENS, Y. R. Cloud computing. *Communications of the ACM*, v. 51, n. 7, p. 9–11, 2008. Citado na página 18.

OPENCV. *OpenCV - Image Thresholding*. 2016. Artigo da Web. Disponível em: <http://docs.opencv.org/3.2.0/d7/d4d/tutorial_py_thresholding.html>. Acesso em: 12 abr. 2017. Citado na página 23.

PARK, J.; KWON, Y.-B. An embedded ocr: A practical case study of code porting for a mobile platform. In: IEEE. *Pattern Recognition, 2009. CCPR 2009. Chinese Conference on*. [S.l.], 2009. p. 1–5. Citado na página 25.

PEREIRA, I. M.; CARNEIRO, T. G. de S.; PEREIRA, R. R. Developing innovative software in brazilian public universities: Tailoring agile processes to the reality of research and development laboratories. In: *Proceedings of the 4th Annual Conference on Software Engineering and Applications*. [S.l.: s.n.], 2013. Citado 2 vezes nas páginas 32 e 33.

QUEIROZ, J. E. R. de; GOMES, H. M. Introdução ao Processamento Digital de Imagens. *Rita*, v. 8, n. 1, p. 1–31, 2001. ISSN 85-240-0762-1. Disponível em: <http://files.multimedia-unicv.webnode.com/200000006-16be917b69/Rita-Tutorial-PDI.pdf>. Citado 2 vezes nas páginas 21 e 22.

SARKER, S.; WELLS, J. D. Understanding mobile handheld device use and adoption. *Communications of the ACM*, ACM, v. 46, n. 12, p. 35–40, 2003. Citado na página 20.

SATYANARAYANAN, M. Fundamental challenges in mobile computing. In: ACM. *Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing*. [S.l.], 1996. p. 1–7. Citado na página 20.

SAUVOLA, J.; PIETIKÄINEN, M. Adaptive document image binarization. *Pattern recognition*, Elsevier, v. 33, n. 2, p. 225–236, 2000. Citado na página 28.

SHAFAIT, F.; KEYSERS, D.; BREUEL, T. M. Efficient implementation of local adaptive thresholding techniques using integral images. In: INTERNATIONAL SOCIETY FOR OPTICS AND PHOTONICS. *Electronic Imaging 2008*. [S.l.], 2008. p. 681510–681510. Citado na página 28.

SMITH, R. An overview of the tesseract ocr engine. In: IEEE. *Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on*. [S.l.], 2007. v. 2, p. 629–633. Citado na página 30.

SRIVASTAVA, L. Mobile phones and the evolution of social behaviour. *Behaviour & Information Technology*, Taylor and Francis Group, v. 24, 2005. Disponível em: <https://doi.org/10.1080/01449290512331321910>. Citado na página 16.

STATCOUNTER. *Mobile internet usage surpasses desktop usage for the first time in history*. 2016. Artigo da Web. Disponível em: <http://gs.statcounter.com/press/mobile-and-tablet-internet-usage-exceeds-desktop-for-first-time-worldwide>. Acesso em: 09 abr. 2017. Citado na página 21.

STATCOUNTER. *Desktop vs Mobile Market Share Worldwide*. 2017. Artigo da Web. Disponível em: <http://gs.statcounter.com/platform-market-share/desktop-mobile/worldwide/#monthly-201603-201703>. Acesso em: 09 abr. 2017. Citado na página 20.

STUDIO, A. The official ide for android. *Android Studio*, 2014. Disponível em: <https://developer.android.com/studio/index.html>. Citado na página 30.

THOMPSON, S. E.; PARTHASARATHY, S. Moore's law: the future of si microelectronics. *Materials today*, Elsevier, v. 9, n. 6, p. 20–25, 2006. Citado na página 17.

WALDROP, M. M. The chips are down for moore's law. *Nature*, v. 530, n. 7589, p. 144–147, 2016. Citado 2 vezes nas páginas 17 e 18.

WALDROP, M. M. More than moore. *Nature*, Nature Publishing Group, v. 530, n. 7589, p. 144, 2016. Citado 2 vezes nas páginas 16 e 18.

Apêndices

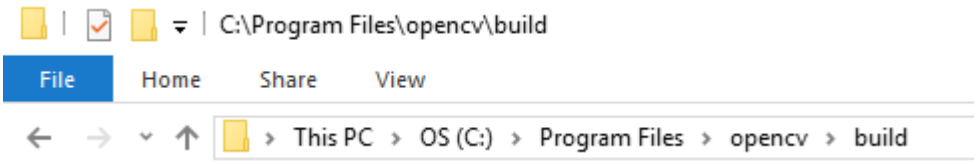
APÊNDICE A – Preparação de Ambientes de Desenvolvimento

A.1 Instalação das Bibliotecas OpenCV e Tesseract no Windows

A.1.1 OpenCV

O download da biblioteca OpenCV pode ser feito através do site <http://opencv.org/releases.html>. São fornecidos os arquivos pré compilados para Windows. Basta descompactá-los em uma pasta específica do computador.

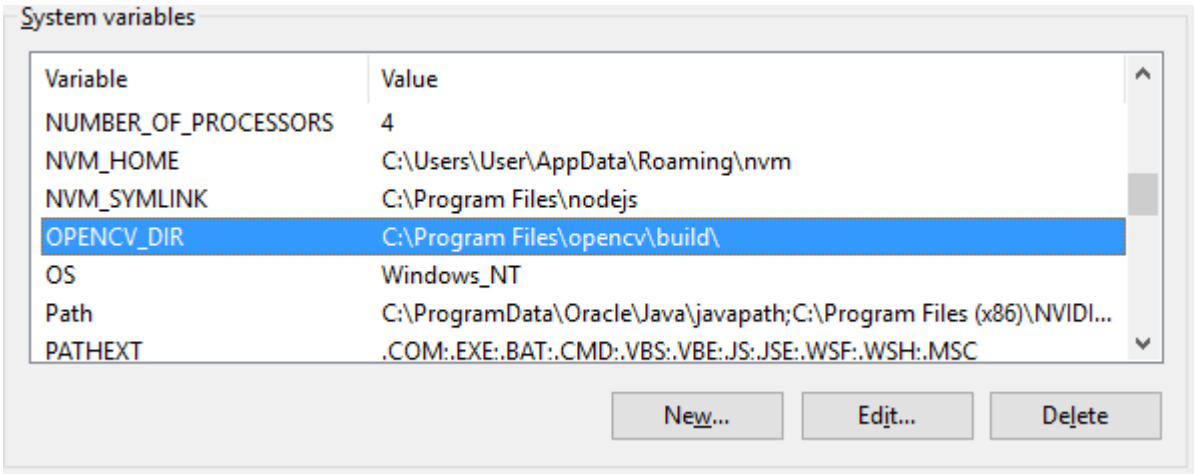
Figura 20 – Diretório de instalação dos arquivos da biblioteca OpenCV



Fonte: Elaborada pelo autor

Em seguida configurar uma variável de ambiente apontando para o diretório *build*. Para configurar uma variável de ambiente, basta clicar com o botão direito do mouse no menu iniciar->configurações-> Configurações Avançadas do Sistema. Na guia *Avançado*, clique em *Variáveis de Ambiente* e na seção Variáveis de Sistema, clique em Novo e preencha com o caminho desejado. A figura 21 ilustra uma variável de ambiente configurada de acordo com as instruções anteriores.

Figura 21 – Configuração de variável de ambiente para a biblioteca OpenCV



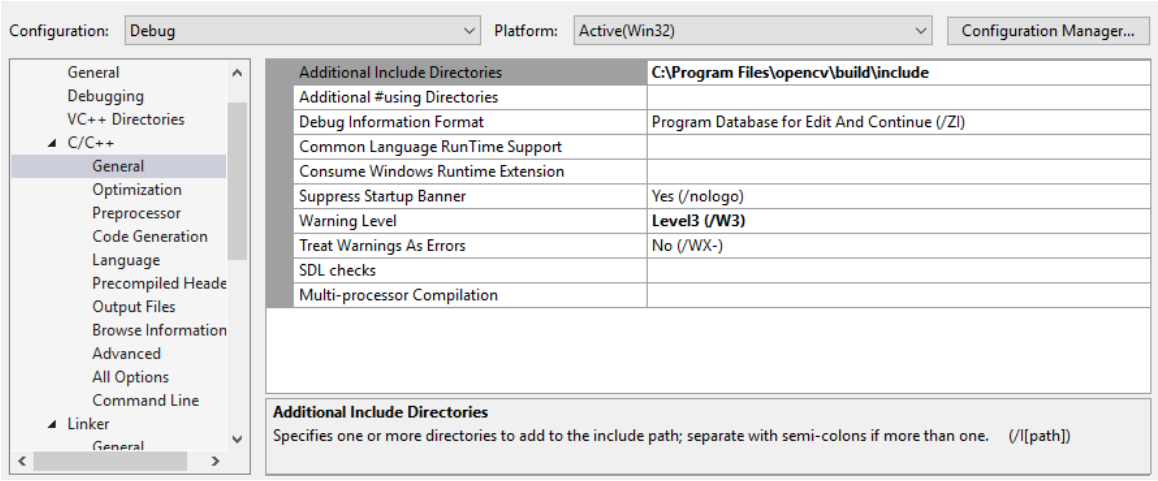
Fonte: Elaborada pelo autor

A.1.1.1 Criação de Projeto OpenCV no Visual Studio Community 2013

No Visual Studio, vá em Arquivo->Novo->Projeto, selecione *Win32 Console Application*, dê um nome para a sua aplicação e crie o projeto. Na tela seguinte, selecione *Console Application* e marque *Empty Project*. Em seguida, em *Solution Explorer*, clique com o botão direito do mouse sobre o arquivo do projeto (crie um arquivo de configurações, caso o Visual Studio não fornecer um padrão) e exiba suas propriedades.

Em C/C++->Geral adicione o caminho do seu diretório */include*. Se uma variável de ambiente estiver configurada, basta referenciá-la, caso contrário, escreva o caminho por extenso como na figura 22.

Figura 22 – Configurações da Seção C/C++



Fonte: Elaborada pelo autor

Em *VC++ Directories*, insira o caminho para sua pasta *include* no campo *Include Directories* e o caminho para a pasta *lib* no campo *Library Directories* como ilustrado na figura 23.

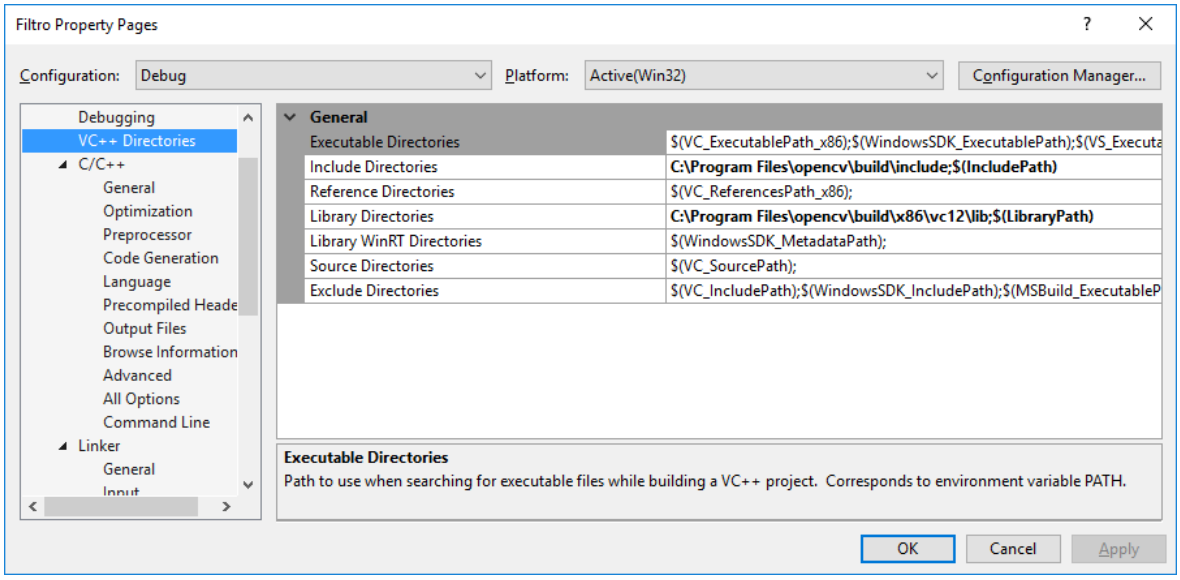
Em *Linker->Input* preencha o campo *Additional Dependencies* com os arquivos com sufixo *d.lib* contidos nas pastas *lib* e *staticlib*. A figura 24 ilustra o resultado desse procedimento.

O mesmo processo poderá ser feito com o modo *Release*, com a diferença da seção *Linker*, onde os arquivos devem ter extensão *.lib* sem a letra 'd'. Por fim, copie o arquivo *opencv_world300.dll* para a pasta do projeto. A partir desse momento, o Visual Studio está configurado para desenvolvimento de aplicações OpenCV na linguagem C++.

A.1.2 Tesseract

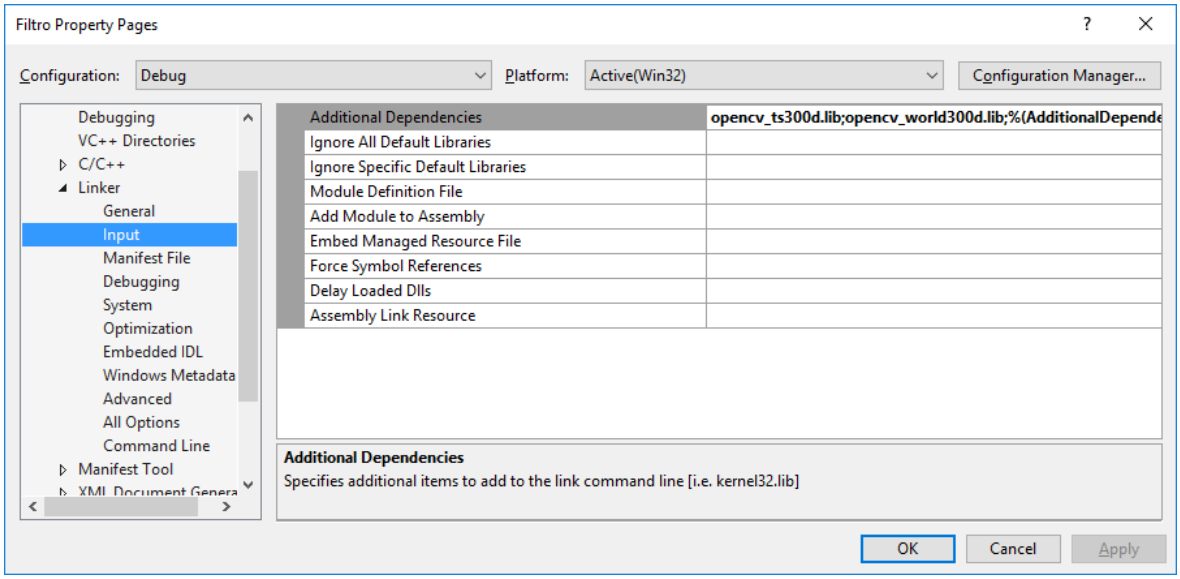
Existe um instalador para o sistema operacional Windows que pode ser adquirido através do site da biblioteca. Basta executá-lo para que sejam descompactadas as bibliotecas pré-compiladas para o sistema e que seja também criada ima variável de sistema.

Figura 23 – Configurações da Seção *VC++ Directories*



Fonte: Elaborada pelo autor

Figura 24 – Configurações da Seção *Linker*



Fonte: Elaborada pelo autor

A.1.2.1 Configuração do Tesseract no Visual Studio

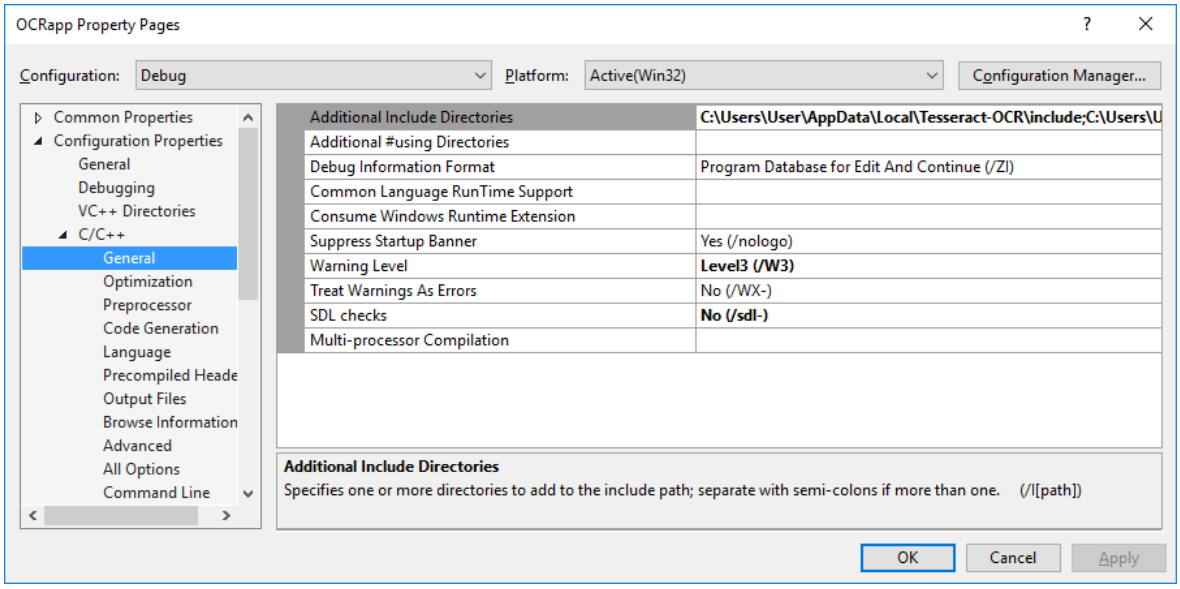
Similar ao que foi feito na seção A.1.1.1, é necessário incluir as referências das bibliotecas no projeto.

Em *C++ -> General -> Additional Include Directories*, adicione a referência a pasta *include* do Tesseract como mostrado na figura 25.

Em *Linker->General->Additional Library Directories*, adicione a referência para a pasta *lib* do Tesseract como ilustrado na figura 26.

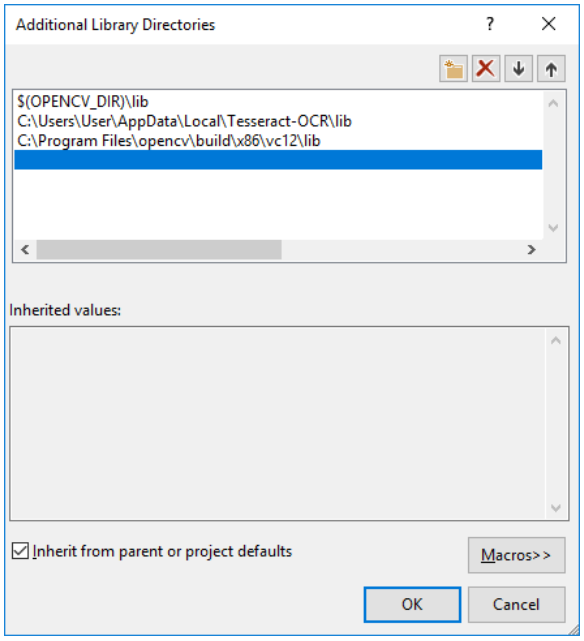
Em *Linker->Input->Additional Dependencies*, inclua todas as bibliotecas dom

Figura 25 – Configurações da Seção *C++/General*



Fonte: Elaborada pelo autor

Figura 26 – Configurações da Seção *Linker/General*



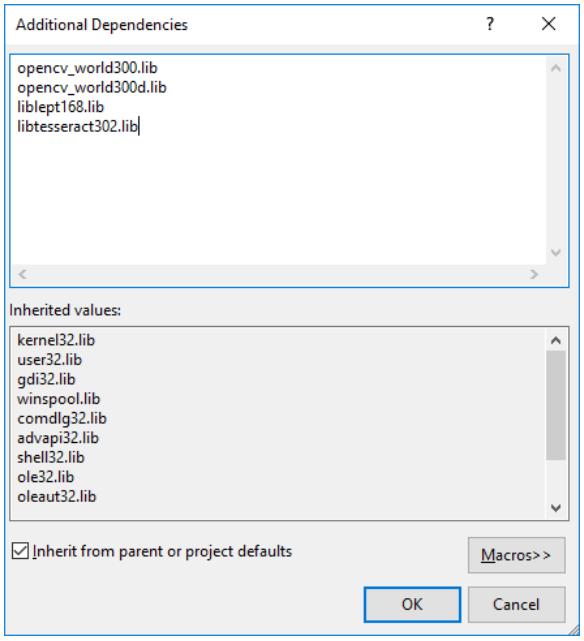
Fonte: Elaborada pelo autor

sufixo *d.lib*, como em 27.

Caso o sistema acuse falta de qualquer biblioteca durante o processo de *build*, elas podem ser baixadas separadamente no repositório da comunidade e adicionadas através do processo demonstrado. Por fim, copie a dll *libtesseract302d.dll* para a pasta do projeto.

As bibliotecas necessárias para o módulo de leitura do projeto são estão ilustradas na figura 28.

Figura 27 – Configurações da Seção textitLinker/Input



Fonte: Elaborada pelo autor

Figura 28 – Bibliotecas para o módulo de leitura do projeto

```
#include<tesseract/baseapi.h>
#include <tesseract/strngs.h>
#include<leptonica/allheaders.h>
```

Fonte: Elaborada pelo autor

A.2 Instalação e Configuração do XAMPP

O conjunto de ferramentas do XAMPP possui instalador executável que pode ser encontrado no site <https://www.apachefriends.org/pt_br/index.html>. O processo de instalação é simples e sem nenhuma configuração em especial, basta seguir o assistente de instalação para ter um servidor básico pré configurado. Para iniciar ou parar qualquer serviço da ferramenta, basta ativá-lo ou desativá-lo no painel de controle do XAMPP.

No local de instalação do XAMPP, acesse o diretório *htdocs*. Nele, crie uma pasta para o seu projeto com a seguinte estrutura interna. Os arquivos são referentes aos *scripts* de filtragem e leitura dos caracteres de uma imagem recebida pela rede.

```
MyProjectFolder.
|
\---public
    |   ExecutionTime.php
    |   Filtro.exe
    |   filtro.php
```

```
|   index.html
|   Ler.exe
|   libtesseract302.dll
|   opencv_world300.dll
|   opencv_world300d.dll
|
+---AndroidFileUpload
|   |   fileUpload.php
|   |
|   \---uploads
+---extras
|   |   ExecutionTime.php
|   |   Filtro.exe
|   |   filtro.php
|   |   Ler.exe
|   |   libtesseract302.dll
|   |   opencv_world300.dll
|   |   opencv_world300d.dll
|   |   UploadToServer.php
|   |
|   +---proc
|   |
|   \---uploads
|
|
+---output
|
|
\---uploads
```

Em \xampp\apache\conf\extra, edite o arquivo *httpd-vhosts.conf* comentando todo o seu conteúdo adicionando uma referência para a pasta do projeto. Isso permitirá a criação de um *host virtual* para acesso ao servidor através do endereço *http://localhost*.

Por motivos de segurança, o XAMPP limita o tamanho de *upload* no servidor. Esse tamanho pode ser alterado editando a propriedade *upload_max_filesize* para o tamanho desejado. Esta propriedade se encontra no arquivo *php.ini* no diretório \xampp\php. Após a edição desse arquivo, reinicie o servidor. Está pronto o *web service* para leitura de imagens.

A.3 Instalação das Bibliotecas OpenCV e Tesseract no Linux

A.3.1 OpenCV

Inicialmente, são necessárias algumas dependências. Para instalá-las, digite as seguintes linhas no console:

```
$ sudo apt-get -y install libopencv-dev build-essential cmake
  git libgtk2.0-dev pkg-config python-dev python-numpy
  libdc1394-22 libdc1394-22-dev libjpeg-dev libpng12-dev
  libtiff4-dev libjasper-dev libavcodec-dev libavformat-dev
  libswscale-dev libxine-dev libgstreamer0.10-dev libgstreamer-
  plugins-base0.10-dev libv4l-dev libtbb-dev libqt4-dev libfaac
  -dev libmp3lame-dev libopencore-amrnb-dev libopencore-amrwb-
  dev libtheora-dev libvorbis-dev libxvidcore-dev x264 v4l-
  utils unzip
```

Em seguida, faça o download da biblioteca com os seguintes comandos:

```
$ mkdir opencv
$ cd opencv
$ wget https://github.com/Itseez/opencv/archive/3.0.0-alpha.zip
  -O opencv-3.0.0-alpha.zip
$ unzip opencv-3.0.0-alpha.zip
```

Acesse o diretório com os arquivos descompactados e instale a biblioteca:

```
$ cd opencv-3.0.0-alpha
$ mkdir build
$ cd build
$ cmake -D CMAKE\_BUILD\_TYPE=RELEASE -D CMAKE\_INSTALL\_PREFIX
  =/usr/local -D WITH\_TBB=ON -D WITH\_V4L=ON -D WITH\_QT=ON -D
   WITH\_OPENGL=ON ..
$ make -j \$(nproc)
$ sudo make install
```

Por fim, adicione as seguintes variáveis de ambiente:

```
$ sudo /bin/bash -c 'echo "/usr/local/lib" > /etc/ld.so.conf.d/
  opencv.conf'
$ sudo ldconfig
```

Reinicie o sistema para finalizar a instalação

A.3.2 Tesseract

Assim como na seção A.3.1, devem ser instaladas as dependências através dos comandos:

```
$ sudo apt-get -qq install -y libpng12-dev libjpeg62-dev
  libtiff4-dev zlib1g-dev
$ sudo apt-get -qq install -y libicu-dev libpango1.0-dev
  libcairo2-dev
$ sudo apt-get -qq install -y gcc g++
$ sudo apt-get -qq install -y autotools-dev autoconf automake
  libtool checkinstall build-essential
$ cd /tmp
$ wget http://www.leptonica.org/source/leptonica-1.72.tar.gz
$ tar -xvf leptonica-1.72.tar.gz
$ cd leptonica-1.72
$ ./configure
$ make
$ sudo make install
```

Com as dependências instaladas, instale o programa:

```
$ cd /tmp
$ wget https://github.com/tesseract-ocr/tesseract/archive
  /3.04.00.tar.gz
$ tar -xvf 3.04.00.tar.gz
$ cd tesseract-3.04.00
$ ./autogen.sh
$ ./configure
$ make
$ sudo make install
$ make training
$ sudo make training-install
$ sudo ldconfig
```

Por fim, instale os pacotes de idioma:

```
$ cd /usr/local/share/tessdata
$ wget https://github.com/tesseract-ocr/tessdata/raw/master/eng.
  traineddata
$ wget https://github.com/tesseract-ocr/tessdata/raw/master/por.
  traineddata
```

A.4 Configuração do OpenCV e Tesseract no Android Studio

A.4.1 Tesseract

O Tesseract possui um fork no GitHub no seguinte endereço <https://github.com/rmtheis/tess-two>. Para utilizá-lo, basta adicionar a dependência `compile 'com.rmtheis:tess-two:7.0.0' compile 'com.rmtheis:tess-two:7.0.0'`

no arquivo *build.gradle* do projeto.

Para mantermos o mesmo arquivo de processamento tanto para o servidor, quanto para o aplicativo, será necessário utilização de código nativo no Android Studio. Isso ocorre através da instalação no NDK

A.4.2 OpenCV

A página da biblioteca fornece o *download* do *Software Development Kit – SDK* para Android. Após fazer o *download* do arquivo e extraí-lo para uma pasta (para maior clareza na explicação, vamos chama-la de pasta Android-SDK), crie um projeto no Android Studio com suporte a API 21 ou superior. Em seguida vá em Arquivo->Novo->Importar Módulo, informe o caminho para a pasta Android-SDK/Opencv-sdk/sdk/java e clique em *OK*.

Com a pasta do projeto selecionada, pressione F4 para abrir a janela de estrutura do projeto. Na parte esquerda dessa janela, clique sobre a pasta do projeto e em seguida, clique na aba Dependências e adicione o módulo da pasta da OpenCV.

Crie uma classe Java com o nome dos métodos oriundos do código nativo utilizando os modificadores *public static native* e de Build no projeto. No console fornecido pela IDE, navegue até a pasta principal do projeto e digite:

```
$ javah -d jni -classpath ../../build/intermediates/classes/
    debug com.example.project.MinhaClasse
```

Na pasta jni gerada, crie um arquivo com extensão .cpp e um com estensão .h, caso não existam e complete-os com os códigos nativos. Ainda na pasta jni, crie um arquivo Android.mk com o seguinte conteúdo:

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
OPENCVROOT:= <Caminho para a pasta Android-SDK>
OPENCV_CAMERA_MODULES:=on
OPENCV_INSTALL_MODULES:=on
OPENCV_LIB_TYPE:=SHARED
include ${OPENCVROOT}/sdk/native/jni/OpenCV.mk
LOCAL_SRC_FILES := <arquivo.cpp>
LOCAL_LDLIBS += -llog
LOCAL_MODULE := <Minha Biblioteca>
include $(BUILD_SHARED_LIBRARY)
```

Adicione também um arquivo Application.mk com o seguinte conteúdo:

```
APP_STL := gnustl_static
APP_CPPFLAGS := -frtti -fexceptions
APP_ABI := armeabi-v7a
APP_PLATFORM := android-16
```

No Gradle, adicione as seguintes linhas:

```
sourceSets.main {
    jni.srcDirs = []
    }
task ndkBuild(type: Exec, description: 'Compile JNI source via
   NDK') {
    commandLine "<caminho para o arquivo ndk-build.cmd>",
    'NDK_PROJECT_PATH=build/intermediates/ndk',
    'NDK_LIBS_OUT=src/main/jniLibs',
    'APP_BUILD_SCRIPT=src/main/jni/Android.mk',
    'NDK_APPLICATION_MK=src/main/jni/Application.mk'
}
tasks.withType(JavaCompile) {
    compileTask -> compileTask.dependsOn ndkBuild
}
```

Na classe *Main* do projeto, adicione uma referência ao seu código nativo com o seguinte comando:

```
static {
    System.loadLibrary("opencv\_java3");
    System.loadLibrary("Minha Biblioteca");
}
```

Faça o *build* do projeto e desenvolva a aplicação desejada.