Universidade Federal de Pernambuco
Centro de Informática

Graduação em Ciência da Computação

Uma Ferramenta de Reconhecimento Visual de Fontes de Texto para Dispositivos Móveis

Pedro Augusto Silva Lucena

Trabalho de Graduação

Recife
20 de janeiro de 2017

Universidade Federal de Pernambuco
Centro de Informática

Pedro Augusto Silva Lucena

Uma Ferramenta de Reconhecimento Visual de Fontes de Texto para Dispositivos Móveis

Trabalho apresentado ao Programa de Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: *Cleber Zanchettin*

Recife
20 de janeiro de 2017

Dedico este trabalho a todos os professores que tive em minha vida, em especial os do curso de ciência da computação. A educação é o combustível que nos leva mais longe.

Agradecimentos

Agradeço aos meus amigos e colegas de classe, por me apoiarem ao longo do desenvolvimento deste trabalho e em toda a graduação. Agradeço também aos meus pais, Carlos Lucena e Nerivânia Lucena, que sempre enfatizaram a importância dos estudos para o crescimento pessoal, e me ofereceram todo o suporte necessário para que eu pudesse seguir estudando. Por fim, agradeço ao meu orientador Cleber Zanchettin, por ter aceitado essa proposta e guiado o desenvolvimento do projeto.

Words are, in my not-so-humble opinion, our most inexhaustible source of magic. Capable of both inflicting injury, and remedying it.

—ALBUS DUMBLEDORE (J.K. ROWLING)

Resumo

O Reconhecimento Visual de Fontes (VFR) é a identificação de tipografia a partir de imagens de textos, este conceito é tratado pelas sub áreas de inteligência artificial tais como visão computacional e aprendizagem de máquina. Ferramentas de reconhecimento de fontes a partir de imagens são utilizadas principalmente por designers para identificar uma fonte desconhecida para utilização posterior em um possível trabalho, mas atualmente existem poucas soluções que oferecem este serviço especialmente no âmbito móvel, e as opções existentes requerem uma quantidade alta de entradas por parte do usuário para permitir a análise da fonte. O objetivo deste trabalho é oferecer uma ferramenta de VFR para dispositivos móveis que seja mais confiável, simples de usar e com funcionalidades mais refinadas que as soluções atuais, utilizando conhecimento adquirido em artigos acadêmicos.

Palavras-chave: Reconhecimento Visual de Fontes, Visão Computacional, Mobilidade, Usabilidade.

Abstract

Visual Font Recognition (VFR) is the process of identifying typography from text pictures. This concept uses knowledge based in sub areas of artificial intelligence such as computer vision and machine learning. Font recognition from image tools are used mainly by designers to identify an unknown font for later work usage, but there aren't many applications offering this service specially in the mobile field and such tools require too many inputs for the user to allow the analysis and identification. This work's goal is to offer a mobile software which operates VFR in a reliable way, easy to use and with a better performance than the current tools, using theoretical knowledge in the form of academic articles.

Keywords: Visual Font Recognition, Computer Vision, Mobility, Usability.

Sumário

Lista de Figuras

Lista de Tabelas

CAPÍTULO 1

Introdução

1.1 Contexto

A tipografia, do grego *typos* – forma e *graphein* – escrita, é a técnica de formatação de texto com intuito de tornar a linguagem escrita legível e atraente. Na área de design gráfico é considerada o principal elemento de qualquer texto, seu uso correto implica na facilidade de interpretação pelo leitor e transmite confiabilidade e credibilidade de acordo com o objetivo destinado. Assim como a utilização incorreta pode causar problemas de comunicação e gerar transtornos. A representação visual da tipografia é alcançada por meio de fontes tipográficas, que são padrões ou coleções de caracteres com o mesmo desenho ou atributos e com o mesmo corpo. Por questões de simplicidade fontes tipográficas serão chamadas apenas de fontes de texto, ou fontes, ao longo deste trabalho.

Dada a importância da tipografia na comunicação escrita, designers gráficos naturalmente possuem interesse em fontes para seus trabalhos e costumam se inspirar tanto em outros designers quanto em conteúdo encontrado no dia-a-dia, na maioria das vezes de maneira aleatória, como por exemplo o texto de um menu de restaurante ou de um outdoor. A partir desta necessidade identifica-se uma solução na forma de um reconhecedor dos tipos de fontes de fácil acesso e que funcione de maneira instantânea.

Esta solução pode ser alcançada utilizando o Reconhecimento Visual de Fontes, ou VFR (sigla para *Visual Font Recognition*). VFR não é um algoritmo ou modelo único, e sim um conceito que pode ser alcançado de diferentes maneiras utilizando conhecimentos da área de inteligência artificial tais como visão computacional [Szeliski, 2010], aprendizagem de máquina [Smola and Vishwanathan, 2008] e reconhecimento de padrões. VFR consiste no entendimento das propriedades tipográficas de um texto por um software a partir da captura de uma imagem deste texto.

Reconhecimento óptico de caracteres, ou OCR (*Optical Character Recognition)*) é uma técnica análoga ao VFR, no sentido de utilizar conhecimentos semelhantes, apesar dos resultados distintos. No OCR o resultado final é a identificação do caractere, e não a fonte utilizada. De acordo com [Zhu et al., 1999] várias técnicas de reconhecimento óptico de caracteres já foram propostas e algumas foram comercializadas, mas o reconhecimento de fontes, que pode ser por exemplo uma questão fundamental na análise e reconhecimento de documentos (e muitas vezes é uma tarefa difícil e demorada), é levado em conta por poucas técnicas. Apesar da clara importância do reconhecimento automático de fontes, apenas alguns pesquisadores abordaram a questão.

Os objetivos desde trabalho não envolvem o desenvolvimento de um VFR com o objetivo de auxiliar reconhecimento de caracteres. Porém, a opinião dos autores nesse ponto é importante

principalmente para salientar a relativa escassez de abordagens na área, comparada a outras ramificações da visão computacional, por exemplo.

1.2 Motivação

O primeiro passo para a validação da ideia deste projeto foi definir o problema para então poder trabalhar em uma solução viável. Como citado na seção anterior, o problema definido é o de identificação de tipos de fontes instantaneamente no mundo real.

Atualmente existem algumas ferramentas que propõem-se a oferecer o reconhecimento visual de fontes, porém a grande maioria delas funciona em plataformas web, como por exemplo o What Font Is[1], o Identifont[2] e o FontSquirrel[3]. As três soluções oferecem o mesmo nível de reconhecimento, sem distinções ou funcionalidades únicas. Além disso, a proposta de um website de VFR não resolve o problema de maneira imediata, visto que o usuário precisa tirar uma foto em determinado momento e então ao chegar em um lugar com computador e disponibilidade, transferir esta foto para o site e aguardar o resultado. Ou então acessar o website diretamente pelo smartphone, e fazer o *upload* da imagem. Opção dificultada, considerando que a maioria destas soluções não possui uma página otimizada para dispositivos móveis.

A partir deste ponto, fica claro que para atacar o problema de maneira efetiva é preciso desenvolver uma ferramenta para dispositivos móveis. A única ferramenta em formato móvel que possui alguma popularidade, o suficiente para ser encontrada na revisão, é o WhatTheFont Mobile[4] que se propõe a resolver o problema definido, porém não oferece a melhor qualidade de utilização ao usuário, por exemplo requerendo uma quantidade alta de ações redundantes em diversas telas para permitir a análise da fonte. Estes problemas de usabilidade também são compartilhados pelas soluções web citadas anteriormente. Mais recentemente o projeto DeepFont, definido no artigo de [Wang et al., 2015], traz uma solução de VFR bastante eficiente, desenvolvida por pesquisadores da Adobe, e que possui planos para uma versão móvel, apesar de ainda não disponibilizar esta versão. Neste projeto, o DeepFont se encaixa tanto como uma referência na literatura acadêmica, quanto como um possível "concorrente" comercial a partir do momento que este se torne um produto de fato. Atualmente sua distribuição está atrelada ao pacote profissional da aplicação Photoshop, para computadores.

Investigando o universo de aplicações voltadas para a identificação de tipografia por imagem, é perceptível a defasagem na questão comercial. Na pesquisa acadêmica existem alguns artigos – a serem detalhados no capítulo seguinte – oferecendo novas soluções, menos custosas e consequentemente mais rápidas, porém, especialmente orientadas à teoria, sem prospecção de se tornarem produtos. Fazendo um paralelo entre essas duas visões surgiu a proposta deste trabalho de graduação. Oferecer uma ferramenta de reconhecimento visual de fontes para dispositivos móveis com acabamento de nível comercial e voltado para um usuário real, funcionalidades refinadas e facilitar o uso a partir de ideias e soluções discutidas em artigos acadêmicos.

[1] http://www.whatfontis.com
[2] http://www.identifont.com
[3] https://www.fontsquirrel.com/matcherator
[4] https://www.myfonts.com/WhatTheFont/mobile/

1.3 Objetivos

Este trabalho de graduação objetiva construir um reconhecedor visual de fontes automático para dispositivos móveis. Esta aplicação se propõe a oferecer o reconhecimento de fontes de texto de maneira rápida e menos custosa para o usuário a partir de um sistema que requeira o mínimo de informação de entrada com bom desempenho de classificação. Dentre os objetivos específicos, estão:

- Criação de uma aplicação para dispositivos móveis para captura e edição (corte) de fotos de texto;
- Implementação de um servidor que receba imagens e as use como entrada em um classificador, tendo como saída a fonte identificada;
- Treinamento e verificação do classificador;
- Validação e testes do sistema, além de comparação com os produtos existentes.

1.4 Estrutura

Este trabalho está dividido em 4 capítulos. O capítulo 2 oferece uma visão mais detalhada do Reconhecimento Visual de Fontes (VFR) a partir de um conjunto de conceitos relevantes. O capítulo 3 relaciona o conhecimento teórico adquirido no capítulo 2 com o desenvolvimento da ferramenta proposta. Explica em detalhes o projeto e a implementação. O capítulo 4 traz os experimentos e resultados da aplicação além de uma comparação entre esta e os trabalhos relacionados. Por fim, no capítulo 5 são mostradas as contribuições e possíveis trabalhos futuros, bem como as dificuldades encontradas no projeto.

CAPÍTULO 2

Revisão de Literatura

O reconhecimento visual de fontes utiliza-se dos conceitos de visão computacional e aprendizagem de máquina para avaliar uma imagem à procura de texto. Mas ao invés de identificar os caracteres deste texto para posteriormente entender o que está escrito – como feito no reconhecimento óptico de caracteres (OCR) – leva em consideração apenas a forma dos caracteres e procura uma conexão entre as fontes em um banco de dados e as características extraídas da imagem.

Existem algumas abordagens possíveis para alcançar os resultados desejados no reconhecimento de fontes, visto que existem maneiras diferentes de reconhecer texto em imagem, bem como diversas técnicas de classificação e de reconhecimento de padrões. Apesar disso, este é um tema um tanto negligenciado e pouco explorado academicamente, comparando-se com OCR por exemplo. Por isso, a quantidade de literatura pertinente para revisão para este trabalho é, de certa forma, limitada.

2.1 Abordagens

O reconhecimento de fontes em [Khoubyari and Hull, 1996] é utilizado para facilitar o posterior reconhecimento de caracteres, e tem foco na identificação da tipografia predominante de um documento, ou seja, a entrada consiste de uma imagem de uma ou mais páginas de texto. As palavras em língua inglesa são localizadas e agrupadas de acordo com semelhança. Palavras comuns como artigos e conjunções são utilizadas como marcadores da presença de ruído. Então protótipos dos grupos são gerados utilizando as imagens com menos ruídos e estes protótipos são combinados à base de dados para obter um resultado.

Esta abordagem considera que as imagens de entrada são por exemplo documentos escaneados, ou seja, figuras de alta resolução que transmitem fielmente o documento, e quaisquer ruídos existentes na teoria devem ser provenientes do documento original. Para o objetivo proposto neste trabalho, a solucão de [Khoubyari and Hull, 1996] não é ótima, mas apesar disso, o fluxo do algoritmo pode ser tomado como base. Por esta razão este artigo é tido como referência para o reconhecimento visual de fontes.

A abordagem de [Chen et al., 2014] é importante de ser entendida para uma aplicação de nível comercial, por tratar de reconhecimento em larga escala. O projeto possui uma base de mais de 2000 fontes com amostras de 1000 palavras da lingua inglesa para cada uma. A extração de características fica por conta de um processo chamado pelos autores de *Local Feature Embedding*, ou incorporação de características locais. O algoritmo funciona extraindo, para cada palavra, uma combinação de dados esparsos, suficientes para diferenciá-la em níveis ge-

rais, com um reconhecimento fino nas áreas de bordas. Isso reduz a quantidade de dados para tornar a solução mais eficiente, porém mantém um alto nível de singularidade com objetivo de facilitar o reconhecimento.

O classificador desta solução utiliza medidas de distância entre os agrupamentos de dados de cada palavra como parâmetros para o reconhecimento. Para o projeto descrito neste relatório, o reconhecimento por caractere, ou seja, cada letra sendo avaliada individualmente, parece uma melhor opção pela independência e demanda de uma quantidade menor de dados de entrada. [Chen et al., 2014] então perde espaço para as duas abordagens seguintes, que propõem soluções que se encaixam com mais facilidade à proposta deste trabalho.

[Sexton et al., 2000] define um modelo que particiona repetidas vezes imagens binárias (contendo apenas pixels pretos e brancos) dos caracteres em quatro blocos de tamanhos variáveis. O tamanho do bloco é definido a partir do centro de gravidade da imagem e as características extraídas aparecem são representadas pelos centróides de cada imagem gerada e seus respectivos níveis na árvore quaternária resultante, como se observa na Figura 2.1.

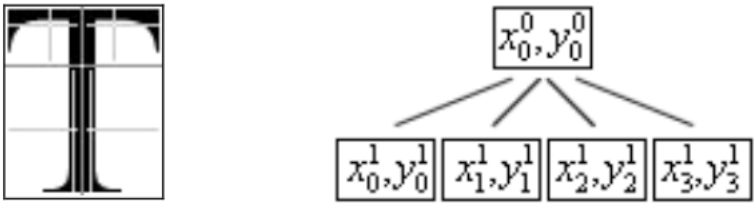


Figura 2.1 Decomposição da imagem e estrutura de árvore quaternária de dois níveis. Fonte: [Sexton et al., 2000].

A base de dados das fontes neste modelo também consiste de uma árvore onde cada nó contém um par de pontos de referência e um par de subárvores. O par de pontos de referência divide o espaço métrico em dois com base na proximidade relativa de pontos no espaço para os pontos de referência. As sub-árvores então recursivamente indexam os sub-espaços correspondentes. O resultado é uma árvore binária que, embora desequilibrada, reduziu com sucesso um problema de busca de linear para um de complexidade logarítmica, segundo os autores.

A abordagem trazida por [Bui and Collomosse, 2015] também trata de caracteres individualmente. Cada imagem, escalada para possuir uma mesma quantidade de pixels passa por algoritmo que extrai um conjunto de descritores de Histograma de Gradientes Orientados, ou HOG (sigla para *Histogram of Oriented Gradient*). Um descritor HOG é uma janela definida por uma quantidade pré-definida de pixels em duas variáveis, tamanho do bloco e tamanho da célula, passando pelas bordas do objeto da imagem, neste caso um caractere, como mostra a Figura 2.2. Ao fim, cada imagem possui por volta de 30 características distintas.

Uma pequena parte aleatória do conjunto de treinamento é agrupada de acordo com suas características utilizando o algoritmo *k-médias* criando *k* grupos. O restante é dividido entre esses grupos a partir do algoritmo de vizinhos mais próximos (*k-NN*). Os grupos que possuem maior frequência de caracteres correspondem às formas que possuem menor distinção entre fontes, ou seja, não são bons parâmetros para reconhecimento. Estes elementos são apagados

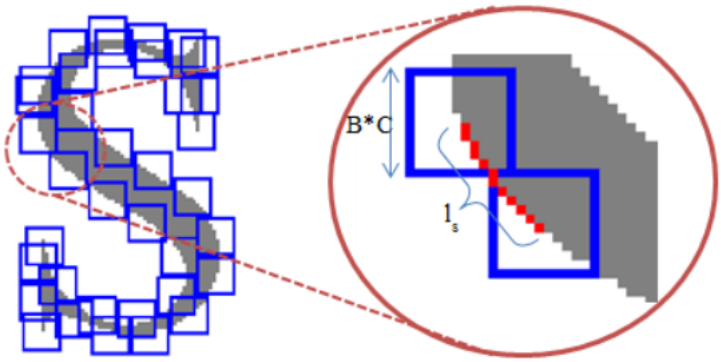


Figura 2.2 Descritores HOG em azul computados a partir de centros equidistantes ao redor das bordas do caractere. B e C indicam os tamanhos do bloco e da célula de cada janela, em pixels. Fonte: [Bui and Collomosse, 2015].

do conjunto de treinamento. O treinamento se dá utilizando um classificador de regressão logística, onde cada classe corresponde a uma fonte distinta.

O teste, dada uma string de caracteres, também avalia a presença de elementos não classificáveis, isto é, aqueles pertencentes aos grupos de díficil distinção, e os elimina da possibilidade de combinação.

Em adição a estes trabalhos, mais recentemente [Wang et al., 2015] trouxe uma solução, denominada DeepFont, que utiliza *deep learning*, um ramo do aprendizado de máquina que basicamente consiste em utilizar uma quantidade grande de camadas intermediárias em uma rede neural. Outros pontos interessantes do trabalho são a construção e disponibilização de uma base de dados mais rica que qualquer uma criada anteriormente, dada a quantidade de fontes especificadas, e um método de compressão para trazer o reconhecimento em tempo hábil.

Além disso, o DeepFont, que foi criado por engenheiros da Adobe[1], possui um caráter fortemente comercial, tendo sido muito recentemente incluído na versão mais nova da aplicação Photoshop. O projeto ainda possui em sua página um protótipo para dispositivos móveis, não encontrado para download até o momento. Por estas razões o DeepFont se torna o principal aspecto de comparação com este trabalho, sendo citado ao longo do relatório como o objetivo a ser alcançado, em geral.

A partir da revisão bibliográfica, ficou decidido que a implementação da ferramenta se daria com uma versão simplificada do reconhecedor mostrado em [Bui and Collomosse, 2015], onde as características serão extraídas das imagens utilizando descritores HOG e o classificador será o que obtiver os melhores resultados, dentre uma rede neural de reconhecimento de padrões e a técnica dos vizinhos mais próximos. Em caráter comparativo, a ferramenta também busca ser uma alternativa tão confiável quanto o DeepFont, e significativamente mais acessível. As questões específicas da implementação assim como os detalhes dos algoritmos estão explicitados no capítulo seguinte.

[1] http://www.adobe.com/br/

CAPÍTULO 3

Uma Ferramenta de Reconhecimento Visual de Fontes de Texto para Dispositivos Móveis

Este capítulo destina-se à especificação, desenvolvimento e implementação técnica da ferramenta proposta. As seções a seguir estão organizadas de maneira a refletir o fluxo natural do processo de desenvolvimento, começando pelo levantamento de requisitos, passando projeto da interface e arquitetura e chegando à implementação propriamente dita. Um simples manual de utilização também está incluso.

3.1 Requisitos

A especificação técnica do projeto pode ser descrita e documentada na forma de requisitos. Requisitos são definições de características que o software deve possuir, sejam estas funcionalidades ou propriedades. Estes são divididos em Requisitos Funcionais e Não-Funcionais.

3.1.1 Requisitos Funcionais

Em engenharia de software, um requisito funcional define uma função de um sistema de software ou seu componente. Uma função é descrita como um conjunto de entradas, seu comportamento e as saídas. Os requisitos funcionais podem ser cálculos, detalhes técnicos, manipulação de dados e de processamento e outras funcionalidades específicas que definem o que um sistema, idealmente, será capaz de realizar.

Ao longo do projeto, os requisitos iniciais foram sendo modificados, além de surgirem novos requisitos. A lista final com todos eles se encontra a seguir:

- O software móvel deve ser capaz de acessar a câmera do smartphone e capturar imagens.
- O software móvel deve cortar a foto no formato da palavra de entrada.
- O software móvel deve ser capaz de fazer requisições POST para o servidor, enviando imagens.
- O software móvel deve receber do servidor, na mesma requisição, os resultados da análise da imagem.
- O software móvel deve mostrar o resultado da análise em sua interface.

- O servidor deve receber a imagem e rodar um *script* que analisa-a e retorna a fonte da palavra de entrada.
- O *script* deve processar a imagem, separar os caracteres, e rodá-los num classificador. O resultado final a ser retornado deve ser definido por uma heurística que leve em consideração a credibilidade de cada resultado individual.
- O classificador deve ser treinado utilizando-se um banco de dados de fontes de texto distintas.
- O classificador deve obter um combinação satisfatória dos dados de entrada do usuário com os dados próprios, e retornar um resultado condizente.

3.1.2 Requisitos Não-Funcionais

Requisitos não-funcionais descrevem características chave de um projeto. É mais difícil mensurar a corretude de um requisito não-funcional, pois estes tendem a conter regras subjetivas relacionadas ao uso da aplicação, em termos de desempenho, usabilidade, confiabilidade, segurança e disponibilidade, por exemplo.

Os requisitos não-funcionais deste projeto incluem:

- O software móvel deve ser de fácil utilização.
- O software móvel deve ter uma interface limpa e centrada no conteúdo.
- O software móvel deve ser familiar para os usuários alvo.
- O software móvel deve realizar suas funções em tempo satisfatório.

3.2 Interface

A aplicação para dispositivos móveis deste projeto foi desenvolvida na linguagem Swift para dispositivos com o sistema operacional iOS, da Apple. O ambiente de desenvolvimento Xcode oferece um framework de interface de usuário embutido para a linguagem, chamado *Cocoa Touch*. Este framework facilita a ilustração da interface, visto que todo o design pode ser pensado e implementado sem levar em consideração a implementação, mas ao mesmo tempo quando for necessário, os elementos de design integram-se ao código a partir de um sistema de *drag & drop*.

A interface do projeto foi então desenvolvida utilizando diretamente o framework *Cocoa Touch*. O primeiro protótipo possuia 4 telas, são elas:

- Tela inicial (câmera).
- Tela de corte da foto.
- Tela de confirmação do envio.

- Tela de retorno das informações.

A primeira tela contem apenas uma visualização de câmera simples e um botão de captura de foto. Ao capturar a foto, a aplicação passa para a segunda tela, que mostra a foto capturada e gera uma *bounding box* redimensionável (sem perder a proporção) e reposicionável acima da imagem. Um botão no canto inferior modifica o formato da caixa entre algumas opções distintas de proporções. Um segundo botão ao lado do primeiro confirma que a foto pode ser cortada no formato definido após a modificação livre da caixa por parte do usuário. Uma vez que o corte foi feito, a terceira tela mostra a imagem resultante e pergunta ao usuário se este deseja confirmar o envio da foto ao servidor, ou cortar a foto novamente. Caso a confirmação aconteça, o sistema aguarda os resultados da avaliação dos dados pelo servidor para abrir a quarta tela, com as informações sobre a fonte encontrada.

Apesar de realizar o trabalho, após alguns testes, foi percebido que existiam redundâncias neste layout inicial. Primeiramente, havia seis diferentes formatos para a *bounding box*, e só um deles estava sendo utilizado na maioria das vezes. Esta proporção é a 5x2, que se assemelha ao espaço ocupado por uma palavra de em média 6-7 letras. O segundo ponto observado foi que tanto a terceira tela, com o corte da imagem, quanto a quarta, com a fonte resposta, possuiam uma quantidade grande de espaço sem uso, tornando as telas visualmente pobres e menos profissionais.

A partir destes testes, uma segunda interface foi prototipada, com a intenção de simplificar a aplicação e remover redundâncias de design. Neste protótipo as quatro telas anteriores foram combinadas em apenas duas. A primeira tela possui agora uma visualização de camera com uma caixa 5x2 no centro, a imagem da câmera fora dessa caixa é escurecida. Fica implícito que o objeto a ser capturado, no caso uma palavra, deve ser enquadrada dentro desta caixa; caso a palavra seja muito longa ou curta, não há problemas em capturar uma imagem com bordas vazias ou não centralizar a palavra alvo, por exemplo. O sistema é robusto o suficiente para reconhecer o texto. Clicando o botão de captura, a foto já cortada é enviada para a tela seguinte. A segunda tela no novo protótipo é apenas uma combinação das duas últimas telas do protótipo anterior. Inicialmente apenas a imagem cortada e o botão de confirmação aparecem, mas ao confirmar, um indicador de carregamento é mostrado até que um *card* com as informações da fonte é carregado na mesma página.

O segundo protótipo se mostrou mais eficiente e obteve sucesso em eliminar processos de menor relevância, atendendo os requisitos não-funcionais definidos na seção anterior. A Figura 3.1 demonstra o protótipo feito no framework *Cocoa Touch*.

3.2.1 Design

Uma vez definidos os aspectos técnicos da interface, ou seja, os elementos que tem um relacionamento direto com o código, o passo seguinte é definir a interface em termos visuais. São três as características a serem especificadas neste ponto: o esquema de cores do aplicativo, a tipografia e o nome.

A cor principal da aplicação, que define o ícone e a barra de navegação é o verde[1]. Estudos em psicologia das cores mostram que o verde possui a capacidade de melhorar a leitura e

[1] http://www.huffingtonpost.com/2012/04/03/green-colors-creative_n_1386190.html

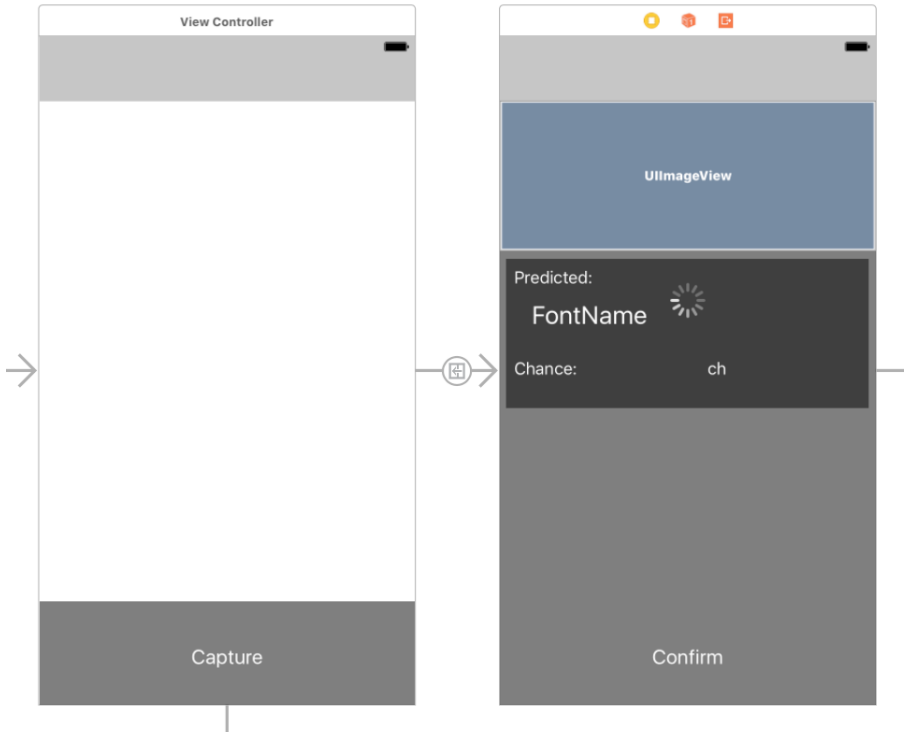


Figura 3.1 Protótipo final da interface de usuário desenvolvido no ambiente de desenvolvimento Xcode utilizando o framework *Cocoa Touch*.

desperta criatividade e inventividade, habilidades relacionadas à proposta do software e ao seu público alvo. A cor secundária, que preenche a maior parte das telas é o preto, que foi escolhido por criar uma noção de familiaridade com outros apps que utilizam câmera, e até com a câmera padrão dos smartphones com o sistema iOS.

A tipografia padrão foi mantida na maior parte do aplicativo, também por questão de familiaridade e padronização do sistema. A exceção está no logo, que se utiliza de um mix de fontes diferentes, para enfatizar a funcionalidade da aplicação como reconhecedor de fontes.

Por fim, o nome do aplicativo acabou se originando de uma simplificação e um jogo de palavras com o primeiro identificador do projeto, que era "Font Recognizer" e se tornou *Fontr*. As Figuras 3.2 e 3.3 mostram as duas telas após as intervenções estéticas.

3.3 Arquitetura

A arquitetura de um software descreve primariamente a estrutura utilizada na construção do projeto, a disciplina de criar estas estruturas e a documentação destas. Esta descrição se dá de maneira abstrata, ou "alto nível" e não entra em detalhes de implementação. A arquitetura se distancia desses detalhes com a finalidade de oferecer entendimento sobre o funcionamento da aplicação de forma independente para as tecnologias utilizadas. Distanciar-se dos detalhes também é importante porque reduz a quantidade de dados sem desviar do objetivo, que é prover

Figura 3.2 Interface da tela inicial da aplicação finalizada.

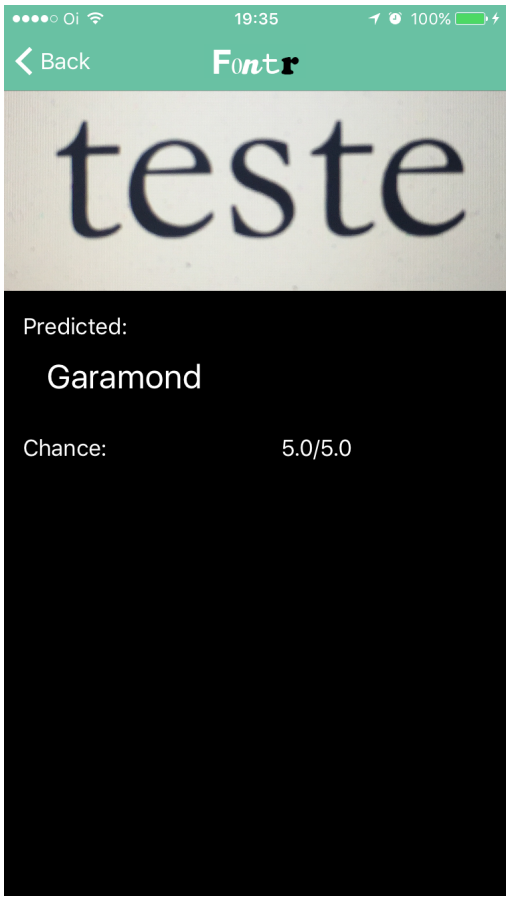


Figura 3.3 Interface da tela de resposta da aplicação finalizada.

uma visão geral sobre a estrutura do software. Em resumo, arquitetura de software é sobre fazer escolhas estruturais fundamentais que são caras para mudar uma vez implementado.

Neste trabalho, as noções de definição de arquitetura foram utilizadas de maneira solta e informal, sem haver a preocupação de representar corretamente todos os pontos necessários, considerando o tamanho da aplicação e sua representação simplificada para fins acadêmicos.

A partir da especificação dos requisitos funcionais e não-funcionais do projeto, e seguindo príncipios do desenvolvimento ágil de software temos uma análise arquitetural que leva a definição do padrão de arquitetura *Cliente-Servidor*. Concluindo esta definição, o objetivo se torna implementar o software seguindo o padrão até ter uma versão funcional que deve ser testada e modificada de forma incremental.

O modelo Cliente-Servidor é uma estrutura de aplicativo distribuída que divide tarefas em módulos ou processos distintos. Um processo é responsável pela manutenção da informação, chamados servidores e outros responsáveis pela obtenção dos dados, chamados clientes. Clientes e servidores se comunicam através de uma rede de computadores em hardware separado, mas tanto o cliente como o servidor podem residir no mesmo sistema. Um host de servidor executa um ou mais programas de servidor que compartilham seus recursos com clientes. Um

cliente não compartilha nenhum de seus recursos, mas solicita o conteúdo de um servidor ou função de serviço. Geralmente, os serviços oferecidos pelos servidores dependem de processamento específico que só eles podem fazer. O processo cliente, por sua vez, fica livre para realizar outros trabalhos. A interação entre os processos cliente e servidor é uma troca cooperativa, em que o cliente é o ativo e o servidor reativo, ou seja o cliente requisita uma operação, e neste ponto o servidor processa e responde ao cliente. Exemplos de aplicativos de computador que usam o modelo cliente-servidor são Email, impressão em rede e a World Wide Web.

A Figura 3.4 contém o diagrama que representa o modelo cliente-servidor básico. Neste projeto, o cliente se limita ao usuário de aparelhos móveis, como smartphone e tablet. A conexão é feita pela internet, porém exclusivamente para clientes numa mesma rede, para facilitar a demonstração do produto em âmbito acadêmico.

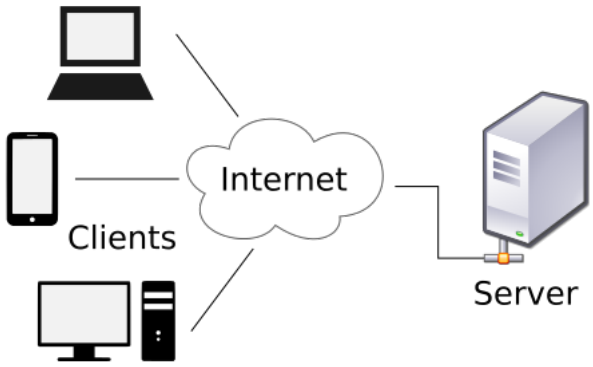


Figura 3.4 Diagrama do padrão de arquitetura cliente-servidor.

As especificidades do modelo cliente-servidor neste projeto residem na escolha das linguagens de programação para cada seção do trabalho, e suas conexões. Parte dessa definição será demontrada na seção seguinte, de implementação. Aqui, serão explicitadas as linguagens e um pouco da descrição de cada uma, bem como o motivo para as escolhas.

3.3.1 *Client-side*

Como já citado ao longo deste relatório, a linguagem de programação da aplicação do lado do cliente, ou seja, do software móvel é a Swift. Swift foi desenvolvida pela Apple para aplicações iOS, macOS, watchOS e tvOS e projetada para trabalhar com os frameworks Cocoa e Cocoa Touch. Ela é compatível com Objective-C, a linguagem anterior da Apple, mantendo algumas das características principais desta como o envio dinâmico e a programação extensível, e adicionando recursos de segurança como por exemplo o sistema de correção de ponteiros nulos e outros erros comuns.

A loja de aplicativos do iOS possui a maior quantidade de downloads cumulativos e também o maior volume de venda, quando comparada à loja de aplicativos Android. Essa é uma das razões pela qual o desenvolvimento móvel deste trabalho focou-se neste sistema operacional. A outra razão diz respeito a maior facilidade de acesso do autor ao hardware necessário para desenvolver uma aplicação iOS, em detrimento do Android.

3.3.2 *Server-side*

O servidor do projeto foi desenvolvido utilizando a linguagem Python. Python foi escolhida por ser uma linguagem de alto nível, interpretada, orientada a objetos e dinamicamente tipada; além de possuir uma biblioteca de módulos e frameworks de terceiros que aumenta bastante seus recursos. Estes requisitos que a tornam simples, direta e poderosa também a tornaram extremamente popular, o que facilita na procura de documentação relevante para o desenvolvimento de software. A possibilidade de conectar o código em Python com as outras linguagens utilizadas com facilidade também ajudou na escolha. Por conter muitas abstrações Python não é uma linguagem eficiente comparada a Java ou C, por exemplo. Mas para o serviço requerido, a potência pode ser deixada de lado em função da praticidade.

O lado do servidor ainda utiliza mais uma linguagem, MATLAB, independente do processo de conexão entre as partes, para cálculos pesados e manipulação de dados visuais. MATLAB denomina tanto a linguagem proprietária quanto seu ambiente de desenvolvimento. A robustez do ambiente para processamento e extração de dados de imagens, assim como classificação, treinamento de redes e reconhecimento de características a tornou a melhor escolha para o trabalho. Além disso, a funcionalidade embutida de rodar *scripts* do MATLAB dentro de outras linguagens, entre elas Python, foi de grande relevância.

3.4 Implementação

Implementação compreende as tecnicalidades do projeto até então descrito em alto nível. As subseções a seguir explicam com detalhes não só o que foi feito em cada parte da aplicação mas também o como, a partir do nível mais baixo, o banco de dados, até o aplicativo final do usuário.

3.4.1 Banco de dados de fontes

A revisão da bibliografia mostrou-se inconsistente no quesito da criação de banco de dados. [Chen et al., 2014] e [Bui and Collomosse, 2015] testaram sistemas com milhares de fontes, enquanto [Zhu et al., 1999], [Khoubyari and Hull, 1996] e [Sexton et al., 2000] possuem apenas algumas dezenas. Mas, à exceção de [Wang et al., 2015], do projeto DeepFont, nenhum deles especifica como obter esses dados, principalmente por haver discrepância no modo de avaliação de características (por palavra ou caractere, por exemplo). O DeepFont explica a geração de sua base de fontes e a disponibiliza para uso livre, porém sua distribuição em forma de múltiplas pastas no serviço Dropbox[2], com mais de 40 *gigabytes* de dados, a torna potencialmente impossível de ser acessada a não ser que o interessado possua este espaço disponivel em sua conta do Dropbox, o que não é o caso. Para este trabalho, como solução comercial viável, seria importante ter uma base ampla, mas construir tal base seria uma preocupação desproporcional aos prazos. Então, para efeitos de trabalho acadêmico, foi feita uma base de dados com doze fontes distintas. A escolha das fontes misturou nomes bastante conhecidos em textos formais com outras menos comuns. São elas:

[2]https://www.dropbox.com

- *Arial*
- *Avenir*
- *Calibri*
- *Comic Sans*
- *Courier*
- *Garamond*
- *Helvetica*
- *Impact*
- *Monotype Corsiva*
- *Nexa Light*
- *SimSun*
- *Times New Roman*

A criação da base foi feita em duas partes: a primeira em Python, utilizando os módulos `ImageFont` e `ImageDraw`. O código utiliza os arquivos de fonte para "desenhar" em uma imagem em branco quadrada padrão de 105x105 pixels os caracteres do alfabeto maíusculo e minúsculo, bem como os algarismos. O extrator de características, como será visto na próxima subseção, precisa de imagens com o tamanho padronizado para oferecer uma quantidade igual de *features*. Para cada caractere, são criadas 10 imagens, com tamanho da fonte 100. Um tratamento foi feito para centralizar o caractere na primeira imagem do caractere e modificar ligeiramente esta centralização para cada uma das nove cópias. A segunda parte utiliza uma imagem real retirada da câmera do celular, de todos os caracteres de cada fonte. Esta imagem é tratada da mesma maneira que as fotos tiradas pelo aplicativo (a subseção 3.4.3 explica os detalhes do processamento) e retorna um total de 504 caracteres por fonte. A figura 3.5 mostra alguns exemplos de caracteres da base, o contorno das figuras é apenas ilustrativo.

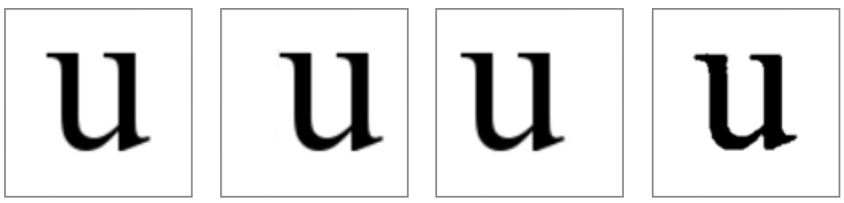


Figura 3.5 Três caracteres sintéticos da base de dados com diferentes alinhamentos e um caractere real.

Com 27 letras no alfabeto (contando o cedilha), multiplicadas por 2 pela capitalização, mais 10 algarismos, temos 64 caracteres. Multiplicando este valor por 10, número de cópias,

obtemos o total de 640 imagens sintéticas por fonte. Juntamente com as 504 imagens reais temos 1144 imagens por fonte no banco de dados. Ao fim as imagens são salvas divididas em pastas nomeadas por suas respectivas fontes. Essa formatação dos dados auxilia na leitura posterior pelo extrator de características e pelo classificador.

3.4.2 Classificador

O classificador utilizado neste trabalho depende de três principais técnicas: A extração de histogramas de gradientes orientados (HOG) e a classificação de duas maneiras, com a intenção de utilizar a mais eficiente no trabalho. São elas, o algoritmo de k vizinhos mais próximos [Bui and Collomosse, 2015]; e uma rede neural de reconhecimento de padrões [Shiffman, 2000]. Antes de demonstrar os detalhes do treinamento e teste, esta subseção introduz uma familiarização conceitual com as técnicas.

3.4.2.1 Revisão teórica

O HOG é um descritor de recurso usado na visão computacional e processamento de imagem para fins de detecção de objetos. A técnica conta as ocorrências de orientação de gradiente em porções localizadas de uma imagem. O pensamento essencial por trás do descritor do histograma de gradientes orientados é que a aparência e forma do objeto local dentro de uma imagem pode ser descrita pela distribuição de gradientes de intensidade ou direções de borda. A imagem é dividida em pequenas regiões conectadas chamadas células, e para os pixels dentro de cada célula, um histograma de direções de gradiente é compilado. O descritor é a concatenação desses histogramas. Para melhorar a precisão, os histogramas locais podem ser normalizados por contraste, calculando uma medida da intensidade em uma região maior da imagem, chamada de bloco e, em seguida, usando esse valor para normalizar todas as células dentro do bloco. Essa normalização resulta em melhor invariância para mudanças na iluminação e sombreamento. A Figura 3.6 compara uma imagem real com os descritores extraídos dela.

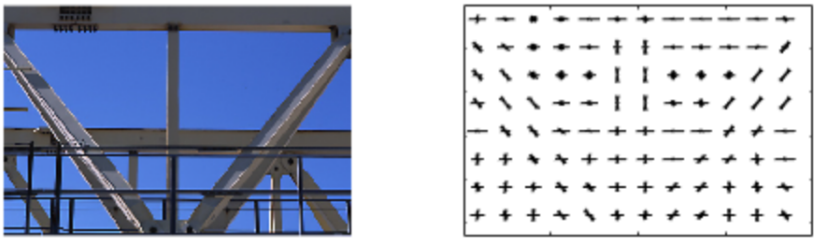


Figura 3.6 Imagem de entrada e suas características extraídas utilizando HOG. Fonte: MathWorks[3]

O algoritmo k vizinhos mais próximos (k-NN, do inglês k-Nearest Neighbors) é um método não-paramétrico utilizado para classificação e regressão. Em ambos os casos, a entrada consiste nos k exemplos de treinamento mais próximos no espaço de recursos. Para o caso de

[3]https://www.mathworks.com/help/vision/ref/extracthogfeatures.html

classificação, utilizado neste trabalho, a saída é uma associação de classe. Um objeto é classificado por um voto majoritário de seus vizinhos, sendo o objeto atribuído à classe mais comum entre seus k vizinhos mais próximos. Por exemplo, se $k = 1$, então o objeto é simplesmente atribuído à classe desse único vizinho mais próximo. Tanto para a classificação quanto para a regressão, pode ser útil atribuir peso às contribuições dos vizinhos, de modo que os vizinhos mais próximos contribuem mais para a média do que para os mais distantes. Por exemplo, um esquema de ponderação comum consiste em dar a cada vizinho um peso de *1/d*, em que *d* é a distância para o vizinho. Os vizinhos são retirados de um conjunto de objetos para os quais a classe é conhecida. Isto pode ser pensado como o conjunto de treinamento para o algoritmo, embora nenhuma etapa de treinamento explícita seja necessária[4].

Uma Rede Neural Artificial (RNA) é um paradigma de processamento de informação que é inspirado na estrutura neural de organismos inteligentes e que adquirem conhecimento através da experiência. O elemento-chave deste paradigma é a nova estrutura do sistema de processamento de informação. É composto por um grande número de elementos de processamento altamente interconectados (neurônios) que trabalham em uníssono para resolver problemas específicos. Uma grande rede neural artificial pode ter centenas ou milhares de unidades de processamento; já o cérebro de um mamífero pode ter muitos bilhões de neurônios. Uma RNA é configurada para uma aplicação específica, como reconhecimento de padrões ou classificação de dados, através de um processo de aprendizagem. Aprendizagem em sistemas biológicos envolve ajustes para as conexões sinápticas que existem entre os neurônios. Isto é verdadeiro para RNAs também.

3.4.2.2 Técnica

Como especificado acima, o algoritmo k-NN não necessita de uma fase de pré-processamento, com treinamento. Mas a rede neural sim. E ambos dependem de um conjunto de treinamento relativamente grande, onde se tenham padrões com classe conhecida suficientes para garantir a robustez do sistema. A base de dados gerada possui 12 fontes de texto distintas, que são as classes, e 1145 caracteres para cada fonte. As imagens geradas dos caracteres são inseridas em um `imageDatastore`, do MATLAB, que funciona como uma espécie de array, no sentido de criar uma coleção organizada dos elementos, mas estes não são alocados em espaços na memória e continuam sendo acessados de seus diretórios padrão. O que otimiza consideravelmente o gerenciamento desta coleção de imagens. A `imageDatastore` também rotula cada imagem com a classe a qual esta pertence (fonte de texto). A extração das características utilizando HOG multiplica drasticamente o número de elementos para classificação. Por exemplo, para uma extração utilizando células de 4x4 pixels, e blocos de 2x2 células, considerando a imagem normalizada para a resolução de 105x105 pixels, são recuperadas por volta de 21 mil características por imagem. Na base inteira, isso significa um conjunto de treinamento de quase 200 milhões de dados. Os classificadores com esta configuração de variáveis para descritores HOG, bem como algumas outras configurações, serão avaliados no capítulo 4, testados inicialmente com um conjunto de testes, um subconjunto da base de dados; e posteriormente com oito palavras para cada uma das fontes de texto pré-definidas. Os melhores resultados,

[4] http://www.saedsayad.com/k_nearest_neighbors.htm

considerando taxa de acertos e consumo de tempo e recursos, serão utilizados no projeto.

3.4.3 *Script* MATLAB

O *script* explicado nesta subseção é o código principal da aplicação, que é chamado pelo servidor e utiliza o classificador gerado para obter os resultados da análise da imagem enviada pelo usuário. Esta imagem usada como entrada passa por um tratamento antes da extração dos caracteres, para garantir uma maior confiabilidade dos dados. Inicialmente a imagem é passada de colorida para escala de cinza, utilizando a função `rgb2gray`. Em seguida a imagem resultante é tratada para suavização das arestas. A binarização da imagem acontece a seguir, utilizando o método de Otsu para calcular o limiar ótimo entre os pixels do primeiro plano e do plano de fundo. O resultado da aplicação deste método é inserido como parâmetro na função `imbinarize`. Para os próximos passos a imagem precisa ter os caracteres em branco, e o plano de fundo em preto, para isso é feita uma verificação, caso necessário, as cores da imagem são invertidas. Esta então passa por uma função chamada `bwlabel`. A função funciona da seguinte forma: a partir da matriz de "zeros" e "uns" que é a imagem binária, ela identifica agrupamentos de "uns". Cada agrupamento individual de "uns" é colocado numa nova matriz com o seu posicionamento relativo à imagem original. Ao fim, temos uma matriz com todas as chamadas *bolhas*, ou objetos da imagem; e uma variável numérica com a quantidade de objetos reconhecidos. A média da quantidade de pixels por bolha e o desvio padrão são calculados. As bolhas menores que a subtração destes dois valores são removidas da imagem, para eliminar o conteúdo que pode ser sido reconhecido como objeto mas não é caractere. Em seguida, é utilizada a função `regionprops` com a matriz filtrada, função que possui diversas opções para mensurar regiões de imagens. Nesse caso, o parâmetro requerido para a aplicação é *bounding box*, que retorna o menor retângulo possível que contorna todo o objeto (caractere). Ou um conjunto de retângulos, considerando a imagem completa. A Figura 3.7 mostra o resultado visual do passo a passo explicitado neste parágrafo.

As caixas retangulares ao redor de cada caractere definem os pontos de corte da imagem. Neste momento temos um conjunto de imagens que representam um caractere cada. Por questões de sobreposição de algumas fontes, é possível que haja em algumas das figuras geradas, ruídos de caracteres vizinhos. Novamente o tratamento de exclusão de elementos é aplicado, porém desta vez são eliminadas todas as bolhas que não sejam a maior, que assume-se com bastante certeza ser o caractere de fato. Uma vez com a garantia de que cada imagem representa apenas um caractere, o algoritmo então preenche o plano de fundo até que a figura fique com a proporção 1x1 (quadrada) pois o extrator de características precisa de todas entradas neste formato. Em seguida, são criadas 2 imagens para cada caractere, redimensionadas para 105 pixels de altura e largura, contendo o caractere em tamanhos levemente distintos, o primeiro exatamente igual ao da base e o segundo um pouco menor. Essas cópias auxiliarão na interpretação dos dados individuais retornados pelo clasificador, para obter um resultado geral mais eficiente.

Após a avaliação de todas as imagens geradas, 2 vezes o número de caracteres da palavra de entrada, um sistema de pesos é utilizado para quantificar o valor das identificações. Funciona da seguinte forma: cada agrupamento de 2 imagens da mesma letra tem os seus resultados listados. Se existem duas respostas diferentes, o peso deste caractere é três, a ser dividido na

Figura 3.7 Demonstração da imagem de entrada em cada passo do algoritmo.

sequência 2 - 1, considerando que o primeiro resultado tem mais chance de ter sido reconhecido corretamente, por ser do tamanho exato das imagens da base. Se a resposta das duas imagens é única, o peso é quatro para esta resposta, pois percebe-se que mesmo com uma mudança na imagem a fonte foi reconhecida de maneira igual, significando maior confiabilidade no resultado. Então, a resposta com maior ocorrência em cada caractere é inserida em uma estrutura de dados X vezes, sendo X o peso definido para este caractere. O resultado mais frequente nesta estrutura é o resultado final, a ser passado de volta para o usuário. Apesar da pouca probabilidade, caso haja empate, os dois melhores resultados são enviados.

3.4.4 Servidor

O Servidor Python consiste de uma classe que importa o módulo `BaseHTTPServer`, que possui recursos para a criação de um servidor web simples. Um servidor web é um programa de computador responsável por aceitar pedidos HTTP de clientes, geralmente os navegadores, e servi-los com respostas HTTP, incluindo opcionalmente dados, que geralmente são páginas web, tais como documentos HTML com objetos embutidos, como imagens, ou um computador que executa um programa que provê a funcionalidade descrita anteriormente.

Ao receber uma requisição do tipo POST, de envio de dados seguro, o servidor cria um arquivo de imagem na pasta do projeto. Esta imagem é nomeada de forma a ser reconhecida pelo *script* do MATLAB. Em seguida, este *script* é chamado a partir do módulo `matlab.engine` já citado anteriormente, que permite rodar uma função sem sair do ambiente Python, e utilizar em variáveis comuns o retorno desta função. Neste trabalho três variáveis são retornadas: O

nome da fonte que teve o maior número de combinações, ou das duas fontes, caso exista um empate no reconhecimento; a quantidade de caracteres reconhecidos com esta(s) fonte(s); a quantidade total de caracteres identificados.

Essas três informações são então concatenadas juntamente com o cabeçalho e retornadas ao cliente na mesma requisição, como um comprovante do sucesso da chamada.

3.4.5 Aplicativo

A aplicação móvel foi desenvolvida utilizando o ambiente Xcode em sua versão 8.1 e testado utilizando um aparelho com o sistema iOS 10.1 instalado. Como já foi especificado na seção 3.2, o aplicativo iOS possui um fluxo de duas telas. Esta subseção entra nos detalhes da implementação de cada uma das telas e a sua relação com as outras partes explicitadas anteriormente.

A tela inicial do aplicativo possui um elemento de visualização principal, que é a câmera. As formas de interação com esta tela giram em torno deste elemento. São estas: o botão *"capture"* que captura a foto e a envia para a próxima tela; e o toque em determinado ponto da imagem da câmera, que ajusta o foco da lente para este ponto.

Para ter acesso ao hardware da câmera e passar o vídeo capturado em tempo real para a visualização é preciso importar um pacote do Swift chamado `AVFoundation`. As classes deste pacote relevantes para o trabalho, e suas definições segundo a documentação do website de desenvolvedores da Apple[5] são:

- `AVCaptureDevice`: Um objeto `AVCaptureDevice` representa um dispositivo de captura física e as propriedades associadas a esse dispositivo. O dispositivo de captura é usado para configurar as propriedades do hardware subjacente. Um dispositivo de captura também fornece dados de entrada (como áudio ou vídeo) para um objeto `AVCaptureSession`. Utilizando os métodos da classe `AVCaptureDevice` pode-se enumerar os dispositivos disponíveis, consultar suas capacidades e ser informado sobre quando os dispositivos estão ou não ativos.

- `AVCaptureSession`: Para executar uma captura em tempo real ou off-line, é necessário instanciar um objeto `AVCaptureSession` e adicionar entradas apropriadas (como `AVCaptureDevice`) e saídas (como `AVCaptureStillImageOutput`).

- `AVCaptureVideoPreviewLayer`: Essa camada de visualização é utilizada em conjunto com uma `AVCaptureSession` para exibir vídeo enquanto este está sendo capturado por um dispositivo de entrada. As propriedades `videoGravity` e `videoOrientation` existem para definir a proporção visualização em relação ao elemento onde ela está sendo projetada e estabelecer a orientação correta do vídeo exibido em relação à tela, respectivamente.

- `AVCaptureStillImageOutput`: É uma classe utilizada para capturar imagens estáticas de alta resolução com metadados de acompanhamento. Orientação, configurações de cor e formato da imagem são definidos no objeto `AVCaptureStillImageOutput`.

[5] https://developer.apple.com

Os objetos referentes a estas classes são configurados no início do carregamento da tela. Em específico, `AVCaptureDevice` precisa especificar qual das câmeras do smartphone será utilizada (frontal ou traseira). Uma vez carregados, a imagem da câmera é visualizada normalmente. Como existe um delimitador retangular para encaixar a palavra a ser identificada, foi preciso combinar dois elementos à visualização da câmera. O primeiro é uma camada de coloração escura com transparência, passando por toda a extensão da *view* com exceção de um retângulo de proporção 5x2 no centro. O segundo é um objeto retangular que ocupe exatamente este espaço do centro sem interferir na visualização, mas que possa ser usado posteriormente como coordenada para o corte da imagem.

Ao clicar no botão de capturar foto, a sessão de captura chama a função `captureStillImageAsynchronously` que gera uma imagem do tamanho da visualização, e performa o seguimento para a próxima tela. Na preparação para a chamada da tela seguinte, a função `retrieveCroppedImage` utiliza as coordenadas do retângulo central para cortar a imagem original. Uma versão modificada da função customizada `fixedOrientation`[6] foi usada para auxiliar a captura dos pixels corretos dentro da imagem. A próxima tela então aparece e a imagem cortada é inicializada como um de seus parâmetros.

A tela seguinte oferece duas visualizações importantes. A primeira e única a aparecer inicialmente é a imagem cortada na proporção correta. Uma vez que o botão *"confirm"* é clicado e a chamada para o servidor realizada, a segunda visualização, com o nome da fonte identificada, se torna visível.

A conexão com o servidor é feita utilizando a API `NSURLSession` que suporta nativamente os esquemas de URL de dados, arquivos, ftp, http e https. Com `NSURLSession`, o aplicativo cria uma ou mais sessões, cada uma das quais coordena um grupo de tarefas de transferência de dados relacionadas. Por exemplo, escrevendo um navegador da Web, o aplicativo pode criar uma sessão por guia ou janela ou uma sessão para uso interativo e outra para downloads em segundo plano. Dentro de cada sessão, o aplicativo adiciona uma série de tarefas, cada uma das quais representa uma solicitação para um URL específico.

Estas tarefas podem, opcionalmente, carregar dados para um servidor e, em seguida, recuperar dados do servidor como um arquivo no disco ou como um ou mais objetos `NSData` na memória. `NSURLSession` fornece três tipos de tarefas: dados, upload e download; das quais apenas a de upload foi usada neste projeto. As tarefas de upload enviam dados (geralmente sob a forma de um arquivo) e suportam uploads em segundo plano enquanto o aplicativo não está sendo executado.

As classes da API `NSURLSession` utilizadas no projeto são:

- `URLSession`: Um objeto de sessão.
- `URLSessionConfiguration`: Um objeto de configuração usado ao inicializar a sessão.
- `URLSessionUploadTask`: Uma tarefa para carregar um arquivo e, em seguida, recuperar o conteúdo de um URL como um objeto `NSData`.

[6] https://gist.github.com/schickling/b5d86cb070130f80bb40

Além disso, `NSURLSession` requer o uso destes três protocolos:

- `URLSessionDelegate`: Define métodos encarregados de manipular eventos de nível de sessão.
- `URLSessionTaskDelegate`: Define métodos encarregados de manipular eventos de nível de tarefa comuns a todos os tipos de tarefa.
- `URLSessionDataDelegate`: Define métodos incumbidos de lidar com eventos no nível da tarefa específicos para tarefas de upload e dados.

Por fim, algumas classes não específicas desta API, que também foram utilizadas:

- `NSURL`: Um objeto que contém um URL.
- `NSURLRequest`: encapsula metadados relacionados a uma solicitação de URL, incluindo o URL, o método de solicitação e assim por diante.
- `URLResponse`: Encapsula metadados relacionados à resposta de um servidor a uma solicitação, como o tipo e o comprimento do MIME de conteúdo.

3.5 Manual de Utilização

O manual de utilização do usuário, com instruções de uso simples e de fácil entendimento, está disponível no apêndice A.

CAPÍTULO 4

Experimentos e Resultados

Para a criação e treinamento dos classificadores, primeiramente é preciso definir valores para as variáveis que modificam o conjunto de características extraídas pelo histograma de gradientes orientados. As combinações dos seguintes elementos retornam uma quantidade diferente de características que lida de maneira direta com a eficiência da classificação: menos dados extraídos significa reconhecimento mais rápido, porém uma maior chance de erro, por falta de detalhe. Os valores avaliados foram:

- Tamanho da célula: definido em pixels, quanto maior, mais gerais as características capturadas. Valores definidos: 2x2; 4x4; 8x8.
- Tamanho do bloco: definido em células, menores blocos têm melhor desempenho suprimindo variações de contraste. Blocos maiores requerem menos processamento. Valores definidos: 2x2; 4x4.
- Número de caixas de histograma de orientação: escalar, quanto maior, mais detalhadas as informações de orientação. Valores definidos: 9; 18.

Ao rodar a aplicação utilizando o algoritmo *k*-NN, utilizando os valores padrão da função do MATLAB, com *k* igual a 1, o reconhecimento de fontes foi satisfatório, sem perceptíveis mudanças nos resultados, quando o tamanho da célula estava em 4x4 ou maior. Para 2x2, apesar do maior refinamento dos dados, houve sobreajuste. Tratando do desempenho temporal, célula com tamanho 4x4, blocos 2x2 e 18 caixas aumentam a quantidade de características para as dezenas de milhares, de forma que o resultado numa palavra de 10 letras por exemplo, pode chegar a demorar 30 segundos para ser analisado. Por essa razão a melhor escolha vem da combinação: tamanho de célula 8x8; tamanho do bloco 4x4; e número de caixas 9. Com um total de 3600 características por caractere, até palavras grandes são reconhecidas em tempo hábil e com uma taxa de acerto satisfatória.

Para a rede neural, a mesma combinação foi utilizada inicialmente, já sabendo de sua vantagem no outro algoritmo. Porém nesse caso, ainda devem ser definidas as variáveis de treinamento da própria rede. As que foram modificadas durante os testes são:

- Função de treinamento.
- Número de neurônios na(s) camada(s) escondida(s).
- Número máximo de épocas.
- Número máximo de falhas.

As duas primeiras variáveis mudam o comportamento da rede, enquanto as duas últimas funcionam como meios de parada do treinamento, quanto menores, maior a chance da rede não ser suficientemente treinada. Entre as funções de treinamento disponibilizadas pelo MATLAB para classificação, a que obteve os melhores resultados foi a função padrão `trainscg`, ou backpropagation de gradiente conjugado escalonado[1]. O número de neurônios no melhor resultado foi 75, distribuídos em apenas uma camada. Esses números foram alcançados em testes empíricos, com valores que variavam de 30 a 2000, em uma ou mais camadas. Por fim, as condiçoes de parada do treinamento se davam por um número máximo de 10000 épocas (ou interações) e 50 falhas seguidas. Foi percebido que aumentar a quantidade de características dos dados de entrada só tornava o processo de treinamento mais custoso, mas não interferiu fortemente nos resultados.

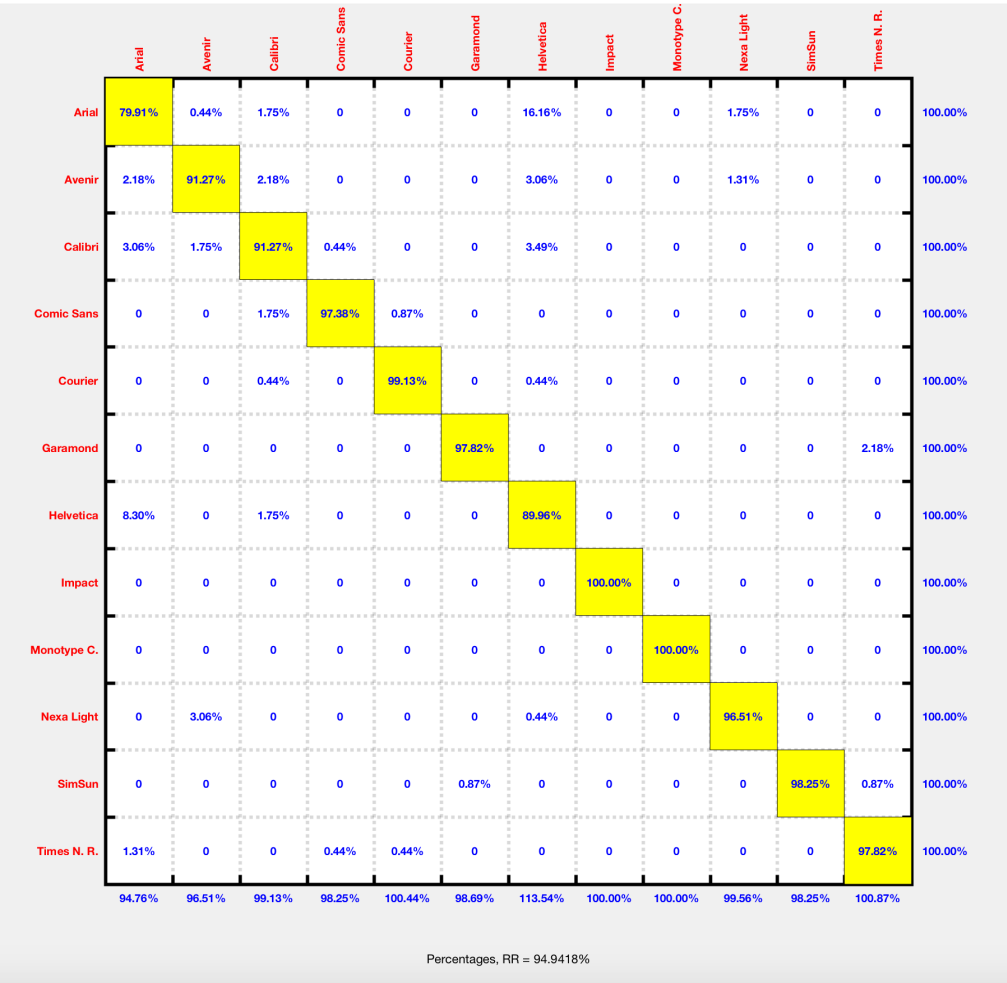


Figura 4.1 Matriz de confusão do algoritmo *k*-NN.

Para comparação entre os dois algoritmos, foi usada uma divisão aleatória em cada classe, de 80% para o treinamento e 20% teste. Utilizando a *toolbox* de redes neurais do MATLAB,

[1] http://www.ra.cs.uni-tuebingen.de/SNNS/UserManual/node241.html

com a função `patternnet`, a base (80%) é automaticamente dividida de forma aleatória com 70% dos dados para treinamento e 15% cada para validação e teste. O resultado do treinamento da rede é então usado com a outra base (20%). As figuras 4.1 e 4.2 mostram respectivamente a matriz de confusão do conjunto de teste do *k*-NN e a matriz de confusão do conjunto de teste da rede neural.

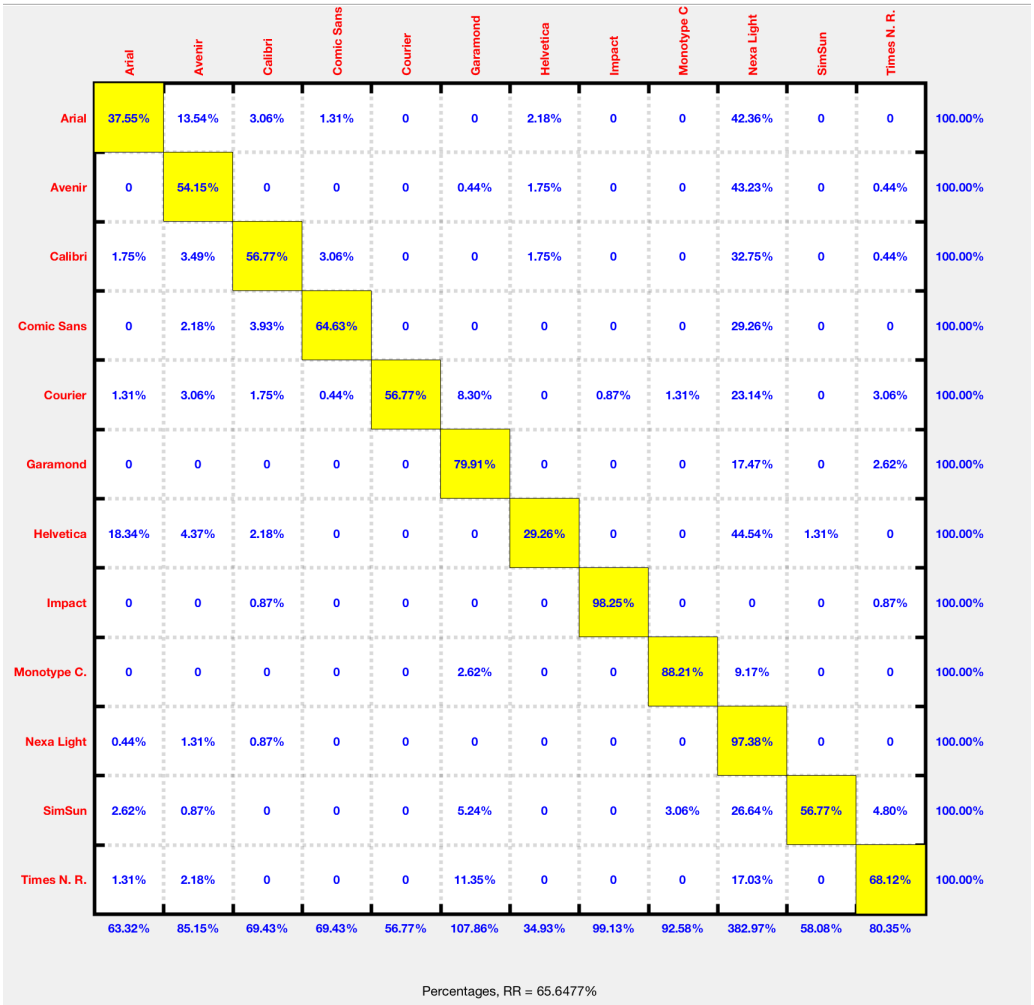


Figura 4.2 Matriz de confusão da rede neural.

Percebe-se que os dois classificadores possuem resultados bastante distintos. O classificador *k*-NN tem uma taxa de acerto de aproximadamente 95% contra 65% da rede neural. Porém a semelhança entre diversos elementos da base de dados pode ser o motivo principal desta taxa tão positiva do *k*-NN. Pois com a divisão aleatória dos dados entre treinamento e teste, pode acontecer de um determinado caractere usado no treinamento também esteja (com uma diferenciação mínima) no teste, apesar dos subconjuntos de treinamento e teste serem disjuntos. Então o próximo passo é testar ambas com exemplos do mundo real não utilizados na base de dados. Para isso, foram gerados conjuntos de imagens com 8 palavras cada, em situações de uso real. A Figura 4.3 e mostra o documento com textos referentes à fonte *SimSun* em diversos tama-

nhos, diferente capitalização (maíuscula e minúscula), colorações distintas de texto e plano de fundo além de um agrupamento de letras sem sentido conjunto, para que se avalie a independência do reconhecedor por caractere. Cada fonte possui um igual a este. A Figura 4.4 é uma fotografia do documento, tirada pelo mesmo aparelho onde a aplicação está sendo testada, que adiciona os valores de ruído e inclinações em todos os eixos. As 8 palavras pertencentes a este documento são então cortadas da foto real e colocadas como entrada para o classificador.

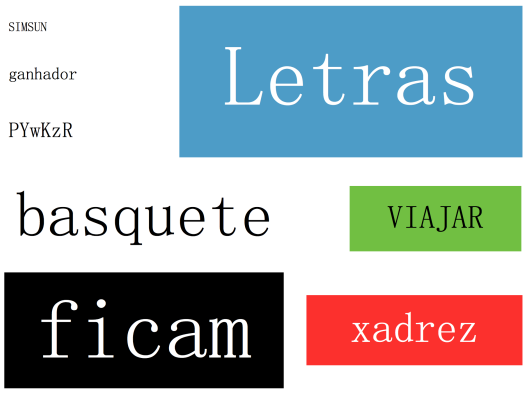


Figura 4.3 Documento com 8 palavras na fonte *SimSun*.

Figura 4.4 Fotografia utilizada como entrada para criação do conjunto de testes.

As 96 palavras (8 para cada uma das 12 fontes), numeradas adequadamente a partir do canto superior esquerdo, foram então usadas como entrada nos dois classificadores. As tabelas 4.1 e 4.2 a seguir mostram os resultados obtidos em ambos.

Imagem	**1**		**2**		**3**		**4**	
Fonte	k-NN	RN	k-NN	RN	k-NN	RN	k-NN	RN
Arial	✓/✗	✗	✓	✓	✓	✗	✓	✗
Avenir	✓	✓	✓	✓	✓	✗	✓	✓
Calibri	✓	✓	✓	✓	✓	✓	✓	✓/✗
Comic Sans	✓	✓	✓	✓	✓	✓	✓	✓
Courier	✓	✓	✓	✓	✓	✓	✓	✓
Garamond	✓	✓	✓	✓	✓	✗	✓	✓
Helvetica	✓	✓	✓	✗	✓	✓	✓	✓/✗
Impact	✓	✓	✓	✓	✓	✓	✓	✓
Monotype Corsiva	✓	✓	✓	✓	✓	✓	✓	✓
Nexa Light	✓	✓	✓	✓	✓	✓	✓	✓
SimSun	✓	✓/✗	✓	✗	✓	✓	✓	✗
Times New Roman	✓	✓	✓	✓	✓	✗	✓	✓

Tabela 4.1 Tabela de avaliação dos classificadores. Parte 1.

Imagem	**5**		**6**		**7**		**8**	
Fonte	k-NN	RN	k-NN	RN	k-NN	RN	k-NN	RN
Arial	✓	✗	✓	✗	✓	✗	✓	✓/✗
Avenir	✓	✓/✗	✓	✓/✗	✓	✓/✗	✓	✗
Calibri	✓	✗	✓	✓/✗	✓	✓/✗	✓	✓
Comic Sans	✓	✓	✓	✓	✓	✓	✓	✓
Courier	✓	✓	✓	✓	✓	✓	✓	✓
Garamond	✓	✓	✓	✓	✓	✓	✓	✓
Helvetica	✓	✗	✓	✗	✓	✓	✓	✗
Impact	✓	✓	✓	✓	✓	✓	✓	✓
Monotype Corsiva	✓	✓	✓	✓	✓	✓	✓	✓
Nexa Light	✓	✓	✓	✓	✓	✓	✓	✓
SimSun	✓	✗	✓	✗	✓	✓	✓	✗
Times New Roman	✓	✓/✗	✓	✗	✓	✓	✓	✓/✗

Tabela 4.2 Tabela de avaliação dos classificadores. Parte 2.

O símbolo ✓ significa que a predição foi correta, ✗ incorreta e ambos representam um empate onde uma das respostas está certa.

A partir dos resultados obtidos nesse ponto, com o *k*-NN obtendo apenas acertos à exceção de uma resposta parcialmente correta (empate), percebe-se que o desempenho dos classificadores em uso real foi comparável ao que havia sido determinado nas matrizes de confusão. Este resultado não satisfatório da rede neural pode se dever a alguns fatores, como a quantidade de elementos na base de dados e a quantidade de características por elemento. Em relação ao primeiro fator, é possível que a base ainda seja pequena para o treinamento efetivo de uma rede neural, ainda que tenha sido suficiente para o outro classificador. No segundo, a ideia é contrária: uma quantia menor de características, melhor definidas, pode ser mais eficiente para a rede neural, por potencialmente diminuir a ambiguidade na classificação. No geral, pela capacidade de reconhecimento satisfatória dada a organização dos dados de entrada deste projeto, o classificador *k*-NN foi escolhido para uso na aplicação.

4.1 Trabalhos Relacionados

Esta seção oferece um resumo do funcionamento do único software de reconhecimento de fontes de texto para dispositivos móveis achado durante a revisão bibliográfica, o WhatTheFont Mobile; e com o projeto DeepFont que possui o protótipo móvel, apesar da não disponibilização do mesmo até o momento. Após o resumo será feita a análise comparativa deste projeto com o seus “concorrentes”, ressaltando pontos positivos e negativos das soluções.

4.1.1 WhatTheFont Mobile

O aplicativo WhatTheFont Mobile é uma extensão do website `https://www.myfonts.com/WhatTheFont/` da empresa MyFonts. Ele está disponível para download para dispositivos com o sistema operacional iOS.

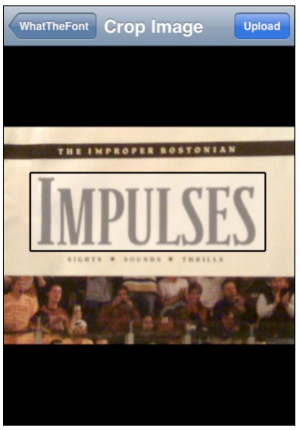
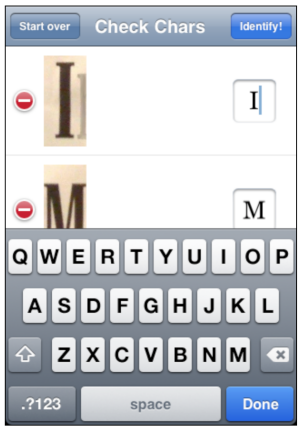


Figura 4.5 Telas do WhatTheFont Mobile. Parte 1. Fonte: MyFonts[3]



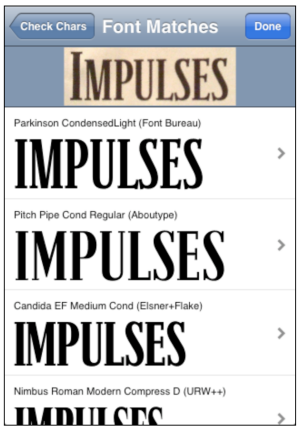


Figura 4.6 Telas do WhatTheFont Mobile. Parte 2. Fonte: MyFonts

Nas Figuras 4.5 e 4.6, disponibilizadas pela empresa na página do aplicativo, é possivel observar quatro telas. A primeira delas é para escolher imagens da galeria; a segunda uma tela de corte da imagem; a terceira serve para verificar se os caracteres reconhecidos pela aplicação batem com o que está escrito na imagem; por fim, a tela de reconhecimento de fontes. Além destas funcionalidades mostradas, o WhatTheFont Mobile também oferece a opção de capturar a foto dentro do *app* e uma tela com detalhamento das fontes, incluindo contexto histórico e curiosidades.

4.1.2 DeepFont

O DeepFont é uma ferramenta desenvolvida por [Wang et al., 2015] para a Adobe que reconhece fontes instantaneamente a partir de imagens. Na metade do ano 2016 a funcionalidade foi implementada na versão mais recente da aplicação Photoshop[4] para computadores. Um demo da versão para dispositivos móveis é mostrado em vídeo na página do criador da solução, Atlas Wang[5], mas não foi documentada nenhuma previsão do lançamento desta versão. As figuras 4.7 e 4.8, tiradas deste vídeo, trazem algumas imagens do uso da aplicação.

4.1.3 Análise Comparativa

Para possibilitar a comparação entre os três trabalhos, foram identificadas uma série de características e funcionalidades relevantes que destacam os diferenciais das soluções e oferecem

[3]https://www.myfonts.com/WhatTheFont/mobile/iphone.html
[4]http://www.bbc.com/news/technology-36276671
[5]http://www.atlaswang.com/deepfont.html

Figura 4.7 Telas do demo DeepFont Mobile. Parte 1. Fonte: Atlas Wang

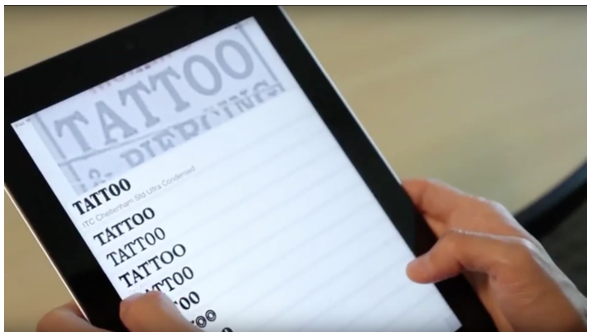


Figura 4.8 Telas do demo DeepFont Mobile. Parte 2. Fonte: Atlas Wang

uma visão mais objetiva em relação ao desempenho destas. Levando em consideração que as três operam o reconhecimento visual de fontes de maneira confiável, a comparação está nos métodos e extras que podem facilitar a vida do usuário final. As características definidas estão listadas a seguir:

1. Oferece opção de tirar fotos pelo *app*.
2. Oferece opção de buscar imagens na galeria.
3. Reconhece fontes sem precisar reconhecer caracteres.
4. Não requer confirmações do usuário além do envio de fotos.
5. Oferece grande quantidade de fontes em seu banco de dados.
6. Possui suporte para as versões mais recentes do sistema operacional.

A tabela 4.3 compara as funcionalidades disponíveis em cada um dos trabalhos.

Funcionalidade	Este trabalho	WhatTheFont Mobile	DeepFont
1.	Sim	Sim	Sim
2.	Não	Sim	Sim
3.	Sim	Não	Sim
4.	Sim	Não	Sim
5.	Não	Sim	Sim
6.	Sim	Não	Sim

Tabela 4.3 Tabela comparativa de funcionalidades.

A análise da tabela leva a algumas conclusões. A primeira delas é que a aplicação WhatTheFont Mobile está num patamar mais profissional, por oferecer por exemplo a funcionalidade de buscar imagens na galeria, além do óbvio, um banco de dados com uma quantidade de fontes de

texto relevante para oferecer um reconhecimento útil comercialmente. Por outro lado, a solução apresentada neste trabalho se destaca pela facilidade de uso, além de oferecer um aplicativo atual. Em comparação com ambas soluções o DeepFont leva vantagem, o que é esperado de uma solução desenvolvida por uma companhia de renome. Porém, como ainda não há maneira de testar a aplicação num ambiente móvel, toda a avaliação da funcionalidade fica restrita ao que os próprios criadores descrevem em seu trabalho.

Fazendo uma análise da classificação de imagens entre este trabalho e o WhatTheFont Mobile, utilizando as mesmas 96 palavras do teste anterior, precebe-se uma vantagem da solução apresentada aqui, com apenas um empate e 95 acertos. Enquanto o WhatTheFont Mobile deixou de reconhecer corretamente todas as 8 palavras da fonte *SimSun*, além de outros 15 erros, especialmente nas imagens com menor tamanho de fonte. No total foram 72 acertos e 1 empate. Com esta comparação confirma-se que além de atingir o objetivo de simplificar o processo de VFR e trazer melhor usabilidade, este trabalho também oferece uma classificação mais confiável do que o seu único “concorrente”.

CAPÍTULO 5

Considerações Finais

No último capítulo deste trabalho, é apresentado o resultado da implementação do software móvel para reconhecimento visual de fontes de texto, *Fontr*. O resultado é mostrado na forma de funcionalidades desenvolvidas, ou seja, contribuições. Em seguida, as dificuldades encontradas ao longo do projeto são explicitadas, bem como as soluções alternativas para contorná-los. Por fim, uma outra lista de funcionalidades, estas não implementadas, é inclusa na seção de trabalhos futuros oferecendo uma visão de possíveis melhorias para este projeto especialmente no sentido de escalar o trabalho acadêmico ao nível comercial inicialmente definido.

5.1 Contribuições

Na seção de objetivos, foram expostas as principais funcionalidades específicas para este trabalho. Em seguida, foram definidos os requisitos funcionais para satisfazer os objetivos. Esta seção faz um paralelo entre ambas as listas e explicita a obtenção (ou não) de cada uma das funcionalidades, ou seja, se as contribuições definidas no início foram devidamente implementadas.

1. Criação de uma aplicação para dispositivos móveis para captura e edição (corte) de fotos de texto:

   - O software móvel deve ser capaz de acessar a câmera do smartphone e capturar imagens.
   - O software móvel deve cortar a foto no formato da palavra de entrada.

   Como mostrado na subseção 3.4.5, este primeiro objetivo foi alcançado, com o uso da API `AVFoundation`. A Figura 5.1 ilustra a câmera em uma captura não relacionada ao projeto enquanto a 5.2 mostra o detalhe da imagem capturada cortada.

2. Implementação de um servidor que receba imagens e as use como entrada em um classificador, tendo como saída a fonte identificada:

   - O servidor deve receber a imagem e rodar um *script* que analisa-a e retorna a fonte da palavra de entrada.
   - O *script* deve processar a imagem, separar os caracteres, e rodá-los num classificador. O resultado final a ser retornado deve ser definido por uma heurística que leve em consideração a credibilidade de cada resultado individual.

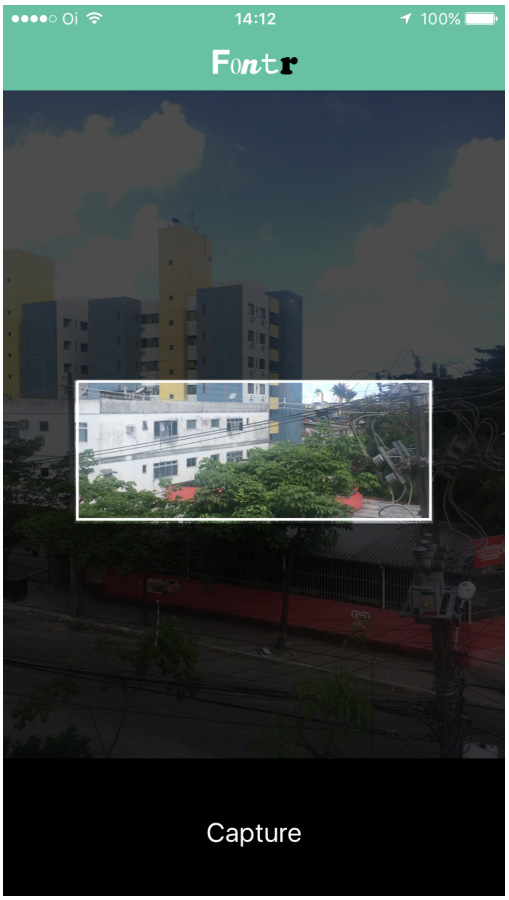


Figura 5.1 Captura da câmera na tela inicial.

Figura 5.2 Detalhe da imagem cortada.

O servidor, descrito na subseção 3.4.4, implementa os requisitos que validam o objetivo número dois. O código do servidor escrito em Python pode ser encontrado no Apêndice B.

3. Treinamento e verificação do classificador;

   - O classificador deve ser treinado utilizando-se um banco de dados de fontes de texto distintas.
   - O classificador deve obter um combinação satisfatória dos dados de entrada do usuário com os dados próprios, e retornar um resultado condizente.

   O classificador desenvolvido está mostrado na subseção 3.4.2 e se utiliza do banco de dados definido em 3.4.1. Os resultados mostrados ao longo da seção 3.4, em vários momentos, confirmam que a aplicação reconheceu os dados com uma taxa de acerto satisfatória e certa robustez no tratamento de entradas não convencionais. O *script* de treinamento do classificador pode ser visto no Apêndice C.

4. Validação e testes do sistema, além de comparação com os produtos existentes.

A seção 4.1 confronta o sistema com os trabalhos existentes, analisa e comapara suas funcionalidades, apesar de haver diferenças nos processos e alguma dificuldade em documentar a performance dos "concorrentes". O sistema foi testado extensivamente ao longo do desenvolvimento, mas não houve validação com potenciais usuários, ou testadores.

A seguir, temos os requisitos não-funcionais do projeto. Estes possuem uma característica subjetividade em termos de avaliação, portanto, esta lista compreende uma pequena descrição da opinião do autor em relação ao cumprimento ou não de cada requisito.

- **O software móvel deve ser de fácil utilização:** Este é talvez o quesito com menos abertura para interpretação. A aplicação foi desenvolvida tendo como príncipio o ato de capturar uma imagem e enviá-la ao classificador. Apenas dois cliques separam a abertura do software da resposta requerida. A única dificuldade que pode ser encontrada está em possíveis problemas de captura, como por exemplo: enquadramento incorreto, falta de foco, caracteres e plano de fundo com colorações similares.

- **O software móvel deve ter uma interface limpa e centrada no conteúdo:** Seguindo a descrição do item anterior, a interface do software é tão simples quanto a sua utilização. Por questões de gosto, não se pode garantir que todos os usuários se sentirão satisfeitos, mas ao meu ver a interface é limpa o suficiente.

- **O software móvel deve ser familiar para os usuários alvo:** Infelizmente não houve tempo para testes no ambiente de uso, não é possível verificar a familiaridade deste software para o possível usuário final.

- **O software móvel deve realizar suas funções em tempo satisfatório:** Entre o envio da imagem e o retorno da fonte identificada, o software demora em torno de 15 a 20 segundos. Isso se deve à conexão não otimizada entre dispositivos, visto que o *script* do MATLAB faz o reconhecimento em menos de 1 segundo. Neste caso acredito que este requisito não é atendido no momento, porém, uma implementação futura pode adquirir mais robustez com certa facilidade, considerando que o problema de desempenho não está no algoritmo principal de reconhecimento.

5.2 Dificuldades

Houveram várias dificuldades específicas durante o percurso deste trabalho, em especial questões de implementação. Nesta seção, foi preferível explicitar os principais focos de problema de uma maneira global ao invés de ater-se aos detalhes. Desta forma, foi possível chegar nos três desafios mais gerais, que são raízes de uma quantidade maior de dificuldades e estão dispostos na lista a seguir:

- **A escassez de referências sobre o tema:** O reconhecimento visual de fontes (VFR) utiliza conceitos conhecidos e amplamente estudados, então ao compreender a ordem dos

processos, não há problemas em desenvolver o que foi estipulado. O problema é chegar a essa definição, pois a literatura alcança este reconhecimento seguindo por caminhos distintos e uma boa parte do conteúdo disponivel tem entre 10 e 20 anos de idade. Em resumo, é preciso um pouco de autonomia para chegar a um objetivo definido, não e recomendável basear-se apenas da bibliografia acadêmica.

- **A falta de familiaridade as linguagens de programação:** As linguagens escolhidas foram as melhores opções possíveis dadas as necessidades e os recursos disponíveis. A implementação tornou-se mais direta utilizando MATLAB para criação da base, classificação e identificação. Enquanto Swift é a unica solução possível considerando os dispositivos móveis disponíveis para depuração e teste, além de possuir um ambiente de desenvolvimento completo e de alto nível, que auxilia bastante a programação. Porém, o fato de não ter experiência anterior com nenhuma destas linguagens levou a diversos momentos de atraso por pequenos detalhes e constantes buscas por ajuda em cada novidade que surgia, além da recorrente confusão entre as sintaxes, ao trabalhar com duas diferentes numa mesma tarefa.

- **A administração do tempo:** A dificuldade em conseguir prever a quantidade de trabalho requerida para cada tarefa tornou-se um ponto problemático nas últimas semanas do prazo de entrega. Outra questão relacionada ao tempo diz respeito à instalação e adequação de ferramentas para uso durante o projeto: foi preciso trocar a versão dos compiladores/ambientes de desenvolvimento da parte servidor do código em dois momentos; retroceder uma versão do sistema operacional móvel por imcompatibilidade; além do esforço requerido pela organização e familiarização do relatório escrito com a linguagem laTeX. A sessão seguinte mostra possíveis melhorias futuras no projeto. Uma parte delas poderia ter sido implementada ainda na versão atual caso o tempo do trabalho tivesse sido melhor gerenciado.

5.3 Trabalhos Futuros

A primeira melhoria para o futuro desta aplicação, num ambiente de uso real, seria a viabilização de uma base de dados com mais fontes de texto disponíveis. Possivelmente algo na faixa dos milhares. Considerando as repetições de caracteres para cada fonte (neste trabalho cada fonte possui por volta de 1000 caracteres na base) e a extração de dezenas de features por caractere, são centenas de milhões de dados. Nesse ponto, pode se dizer com alguma certeza que o usuário encontrará a fonte exata do texto, ou pelo menos uma boa aproximação.

Porém, para obter um desempenho satisfatório com uma base de dados tão grande, alguns outros elementos devem ser considerados. O primeiro deles é um classificador mais confiável, isto é, uma função que possua uma melhor e mais refinada avaliação das características distintas dentre os dados de entrada, pois a quantidade de caracteres similares tende a crescer a medida que a base recebe mais fontes. Métodos de redução de protótipos do k-NN podem ser testados. A otimização da classificação também acontece filtrando os melhores elementos, tanto para extração quanto para reconhecimento. Como citado na revisão de literatura, [Bui and

Collomosse, 2015] exclui caracteres que julga difíceis de distinguir entre fontes, por exemplo. Após decidir quais caracteres tem mais chance de serem distinguidos, também pode-se aplicar OCR, ou reconhecimento óptico de caracteres para filtrar apenas estes antes de enviá-los para a classificação.

Uma segunda melhoria, independente da implementação do crescimento da base, é a diminuição do tempo de carregamento entre o início do envio da foto até o retorno da fonte identificada. Como já citado anteriormente, a conexão entre o cliente e o servidor e/ou a chamada do *script* feita por um plugin proprietário, prejudicaram o desempenho da aplicação e na situação de uma base de tamanho comercial, por exemplo, é de extrema importância que o tempo de espera seja prioritariamente derivado da classificação e não das conexões.

Outra melhoria que deve ser aprimorada com frequência, o aumento de robustez na captura da imagem. Atualmente o sistema reconhece fontes em diversas combinações de cores desde que haja contraste suficiente entre o texto e o plano de fundo, reconhece também texto com inclinações leves e mesmo imagens com menor resolução. Porém, existem técnicas mais avançadas no pré-processamento de imagens que podem oferecer ainda mais confiança a partir de fotos capturadas com menos cuidado.

Por fim, a implementação do mesmo sistema para a plataforma Android é um trabalho futuro interessante, visto que o único concorrente comercial com certa visibilidade possui, assim como este projeto, apenas uma versão para o iOS.

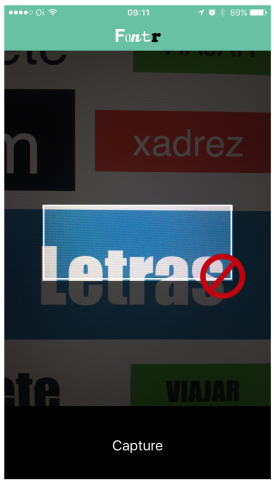
APÊNDICE A

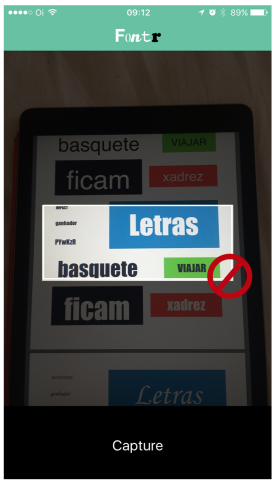
Manual do Usuário

Manual de utilização Fontr

Fontr é um reconhecedor visual de fontes para dispositivos iOS. Este manual guiará o uso da aplicação.

Ao entrar, o usuário tem acesso à câmera. As imagens a seguir mostram como deve ser feita a captura da imagem para otimizar os resultados.





As imagens à esquerda demonstram como NÃO fotografar o texto.

A palavra a ser capturada deve estar completamente dentro da caixa delimitadora.

É preferível focar em uma palavra por vez, apenas.

Capturar a palavra inclinada pode interferir no reconhecimento, mas o sistema é robusto o suficiente para acertar os resultados.

Esta é a maneira correta de tirar foto do texto.

Manual de utilização Fontr

Ao capturar a imagem, o usuário é levado a uma tela de confirmação para conferir o corte da figura antes do envio.

Tela de carregamento.

Chance da predição estar correta

Nome da fonte

Por fim, a aplicação apresenta ao usuário o resultado da identificação.

APÊNDICE B

Servidor Python

```
import BaseHTTPServer
import os, cgi
import cgitb; cgitb.enable()
import matlab.engine

class Handler(BaseHTTPServer.BaseHTTPRequestHandler):
    def do_GET(self, font, oc, total):
        self.send_response(200)
        self.send_header("Font", font)
        self.send_header("Ocurrences", oc)
        self.send_header("Total", total)
        self.end_headers()
    def do_POST(self):
        print(self.headers)
        form = cgi.FieldStorage(fp = self.rfile)
        print(self.headers['Content-Type'])
        ctype, pdict = cgi.parse_header(self.headers['Content-
            Type'])
        length = cgi.parse_header(self.headers['Content-Length
            '])
        print(length[0])
        if ctype == 'application/x-www-form-urlencoded':
            qs = self.rfile.read(int(length[0]))
            fout = open(os.path.join('/Users/Pasl/Documents/
                MATLAB/', 'image.jpg'), 'wb')
            fout.write (qs)
            fout.close()
            eng = matlab.engine.start_matlab()
            font, oc, total = eng.main(nargout=3)
        self.do_GET(font, oc, total)

if __name__ == '__main__':
    server = BaseHTTPServer.HTTPServer(("192.168.25.46", 9876)
        , Handler)
    server.serve_forever()
```

APÊNDICE C

Classificador MATLAB

```
trainDir = '/Users/pasl/Documents/MATLAB/FontDBGen/FontsNew/';

fontSet = imageDatastore(trainDir, 'IncludeSubfolders', true,
    'LabelSource', 'foldernames');
[trainingSet, testSet] = splitEachLabel(fontSet, 0.8, '
    randomized');
T = countEachLabel(trainingSet);

cellSize = [8 8];
blockSize = [4 4];
numBins = 9;
blockOverlap = ceil(blockSize/2);
img = readimage(trainingSet, 1);

BlocksPerImage = floor((size(img)./cellSize - blockSize)./(
    blockSize - blockOverlap) + 1);
N = prod([BlocksPerImage, blockSize, numBins]);

numImages = numel(trainingSet.Files);
trainingFeatures = zeros(numImages, N, 'single');

for k = 1: numImages
    img = readimage(trainingSet, k);
    if size(img, 3) == 3
        img = rgb2gray(img);
    end
    img = imbinarize(img);
    trainingFeatures(k,:) = extractHOGFeatures(img, 'CellSize
        ', cellSize, 'BlockSize', blockSize, 'numBins', numBins)
        ;
end

trainingLabels = trainingSet.Labels;

classifier = fitcknn(trainingFeatures, trainingLabels);
```

Referências Bibliográficas

[Bui and Collomosse, 2015] Bui, T. and Collomosse, J. (2015). Font finder: Visual recognition of typeface in printed documents. *2015 IEEE International Conference on Image Processing (ICIP)*. Disponível em: *http://personal.ee.surrey.ac.uk/Personal/J.Collomosse/pubs/Bui-ICIP-2015.pdf*.

[Chen et al., 2014] Chen, G., Yang, J., Jin, H., Brandt, J., Shechtman, E., Agarwala, A., and Han, T. X. (2014). Large-scale visual font recognition. *2014 IEEE Conference on Computer Vision and Pattern Recognition*. Disponível em: *http://www.ifp.illinois.edu/\textasciitidlejyang29/papers/CVPR14_Font.pdf*.

[Khoubyari and Hull, 1996] Khoubyari, S. and Hull, J. J. (1996). Font and function word identification in document recognition. *Computer Vision and Image Understanding*, 63(1):66–74. Disponível em: *http://jonathanjhull.com/content/pubs/khoubyari_cviu96.pdf*.

[Russell et al., 1995] Russell, S. J., Norvig, P., Canny, J. F., Malik, J. M., and Edwards, D. D. (1995). Artificial intelligence: A modern approach. Technical report. Disponível em: *http://www.cin.ufpe.br/\textasciitidletfl2/artificial-intelligence-modern-approach.9780131038059.25368.pdf*.

[Sexton et al., 2000] Sexton, A., Todman, A., and Woodward, K. (2000). Font recognition using shape-based quad-tree and kd-tree decomposition. Disponível em: *http://www.cs.bham.ac.uk/\textasciitidleaps/research/papers/pdf/SeToWo_CVPRIP00-FontRecognitionUsingShape-BasedQuad-TreeAndKD-TreeDecomposition.pdf*.

[Shiffman, 2000] Shiffman, D. (2000). The nature of code. Technical report. Disponível em: *http://natureofcode.com/book/chapter-10-neural-networks/*.

[Smola and Vishwanathan, 2008] Smola, A. and Vishwanathan, S. (2008). Introduction to machine learning. Technical report. Disponível em: *http://alex.smola.org/drafts/thebook.pdf*.

[Szeliski, 2010] Szeliski, R. (2010). Computer vision: Algorithms and applications. Technical report. Disponível em: *http://szeliski.org/Book/drafts/SzeliskiBook_20100903_draft.pdf*.

[Wang et al., 2015] Wang, Z., Yang, J., Jin, H., Shechtman, E., Agarwala, A., Brandt, J., and Huang, T. S. (2015). Deepfont: Identify your font from an image. Disponível em: *https://arxiv.org/pdf/1507.03196v1.pdf*.

[Zhu et al., 1999] Zhu, Y., Tan, T., and Wang, Y. (1999). Font recognition based on global texture analysis. *Proceedings of the Fifth International Conference on Document Analysis and Recognition. ICDAR '99 (Cat. No.PR00318)*. Disponível em: *http://www.cbsr.ia.ac.cn/publications/yzhu/Font\%20Recognition\%20Based\%20on\%20Global\%20Texture\%20Analysis.pdf*.

Este volume foi tipografado em LaTeX na classe UFPEThesis (www.cin.ufpe.br/~paguso/ufpethesis).