

Universidade Paulista

Ciência da Computação

Atividade Prática Supervisionada

DESENVOLVIMENTO DE UM SISTEMA PARA RECONHECIMENTO DE ESCRITA A MÃO

Aleksander Rocha - R.A.: C630IH-0

Daniel Jançanti - R.A.: C630IG-2

Ingrid Oliveira - R.A.: C51791-7

Rafaela Aranas - R.A.: C29CII-0

Sumário

1	Objetivo e motivação do trabalho	3
2	Introdução	4
3	Sistemas de reconhecimento de caracteres	6
3.1	Origens e Evolução	6
3.2	Tipos de Sistemas de Reconhecimento de Caracteres	7
3.3	Descrição dos Sistemas OCR	7
3.3.1	Etapas de Processamento	8
3.3.2	Técnicas de Reconhecimento	9
3.3.3	Avaliação do Reconhecimento	9
3.4	Redes Neurais	10
3.4.1	Características e Aplicações das Redes Neurais	10
4	Plano de desenvolvimento da aplicação	12
5	Código fonte	14
6	Bibliografia	29

1 Objetivo e motivação do trabalho

Este trabalho apresenta um estudo sobre o reconhecimento visual de caracteres através da utilização das redes neurais. São abordados os assuntos referentes ao processamento digital de imagens, aos sistemas de reconhecimento de caracteres, e às redes neurais.

Em relação ao processamento digital de imagens são apresentados temas, abrangendo os assuntos referentes à aquisição de imagens, ao tratamento e reconhecimento de padrões.

2 Introdução

Segundo Osorio (1991), um sistema de reconhecimento de caracteres permite que o computador possa adquirir informações através da leitura de textos, ou seja, ao invés de entrar as informações pelo teclado, é possível implementar um sistema computadorizado, associado a um dispositivo ótico, para a aquisição automática de informações descritas sob a forma textual.

Os sistemas de reconhecimento de caracteres possuem uma grande importância junto ao processamento de dados. Isto é, devido ao fato da linguagem escrita ser a forma mais usual do ser humano armazenar e transmitir informações.

As **redes neurais** são um novo paradigma de desenvolvimento de sistemas, que tem se difundido muito na atualidade. As redes neurais tem se apresentado como uma solução muito adequada para sistemas de reconhecimento de padrões, como é o caso de um sistema de reconhecimento de caracteres, onde os padrões os para serem reconhecidos são os próprios caracteres. Esta característica, a ser demonstrada em maiores detalhes posteriormente, foi o que levou ao estudo e emprego das redes neurais junto a este trabalho.

O **processamento de imagens** é uma área de estudos da computação que tem crescido muito nos últimos anos. Seu grande crescimento se deve principalmente ao desenvolvimento de equipamentos, cada vez mais sofisticados e baratos, utilizados para a captura e tratamento de imagens digitais. O processamento gráfico engloba as área de processamento de imagem e computação gráfica, onde o processamento de imagens está relacionado ao tratamento e reconhecimento de padrões em imagens, e a computação gráfica está ligada à síntese de imagens e à geração de descrições (dados não pictóricos) utilizadas na obtenção dessas imagens.

O objeto de trabalho no processamento de imagens é a imagem digital, onde esta é definida como sendo uma matriz de $M \times N$ elementos (vetores com informações referentes aos pontos da imagem). Cada elemento da imagem digital é denominado de pixel, tendo associado a si uma informação referente à luminosidade e à cor. Esta informação pode ser um valor indicativo de intensidade luminosa. Esta apresentação através de uma matriz bidimensional de pontos é resultante da manipulação da imagem como sendo uma área de memória do computador. A cada pixel é associada uma ou mais posições de memória, as quais deve armazenar as diversas informações referentes a estes. Com isto pode-se armazenar e processar as informações referentes a uma imagem para posteriormente exibi-la.

O **processo de digitalização** consiste em realizar a aquisição de uma cena, a qual é passada para o computador em um formato adequado para que este possa manipulá-la. As informações visuais são convertidas em sinais elétricos por sensores óticos, e esses sinais são quantificados em valores binários e armazenados na memória do computador. No processo de digitalização, os sinais são amostrados espacialmente e quantificados em amplitude, de forma a obter a imagem digital.

No processo de digitalização, a imagem sofrerá uma amostragem e uma discretização da intensidade luminosa, que é denominada de quantização. No **processo de quantização**, uma imagem com tons contínuos é convertida em uma de tons discretos. Para o armazenamento e processamento por um computador, cada tonalidade (intensidade da luz refletida por cada ponto da imagem) é representada por um valor armazenado de forma binária. Cada ponto amostrado possuirá portanto um valor binário correspondente à intensidade luminosa da imagem naquele ponto. As imagens podem ser do tipo monocromáticas ou policromáticas. Nas imagens monocromáticas somente uma faixa de comprimentos de onda (uma cor) é analisada pelo sensor, determinando as intensidades de luminosidade para esta faixa. As intensidades de luminosidades descritas acima são denominadas de tonalidades ou níveis de intensidade de cor. Para as imagens policromáticas a digitalização é feita para diferentes faixas de comprimento de onda (diferentes cores).

3 Sistemas de reconhecimento de caracteres

Segundo Silva (2003), os sistemas de reconhecimento ótico de caracteres são sistemas desenvolvidos para, de certa forma, reproduzir a capacidade humana de ler textos.

3.1 Origens e Evolução

Mantas (1986) disse que as origens da tentativa de sumular a leitura humana datam de 1870, quando foi inventado por Carey, o scanner de retina. A partir da evolução dos dispositivos capazes de captar e traduzir imagens em sinais que podiam ser medidos, novos horizontes foram descobertos. Entretanto, a primeira tentativa bem sucedida de reconhecimento de caracteres foi realizada somente em 1990 pelo cientista russo Tyutin, que desenvolveu um sistema que visava o auxílio de deficientes visuais.

Alguns dos motivos do desenvolvimento de um sistema de reconhecimento de caracteres são:

- Aquisição de dados numéricos comerciais;
- Compactação de dados;
- Leitura automática de formulários;
- Identificação de endereçamento postal;
- Reconhecimento de cheques bancários;
- Sistemas de aquisição de textos para a tradução automática;
- Interface homem x máquina mais natural e sem necessidade de recodificação dos dados (sujeitos a um menor número de erros, pois não é necessário redigitar os textos, introduzindo novos erros);
- Auxílio a deficientes visuais, na leitura de textos (tradução e impressão automática em braile)

3.2 Tipos de Sistemas de Reconhecimento de Caracteres

Os sistemas de reconhecimento de caracteres podem ser desenvolvidos utilizando-se diferentes procedimentos tanto de aquisição de dados como de processamento de informações além de serem orientados para diferentes tipos de aplicações.

Podem ser classificados primeiramente quanto ao tipo de mecanismo utilizado na aquisição dos textos a serem reconhecidos. Nesta categoria, encontram-se os sistemas magnéticos, sistemas mecânicos e sistemas óticos.

Dentro dos sistemas OCR, são encontradas duas classes importantes, aqueles que são baseados no processamento sequencial das informações e os baseados no processamento paralelo. Existem três categorias principais: sistema de reconhecimento online, sistema de reconhecimento de caracteres isolados e sistema de reconhecimento de escrito cursiva.

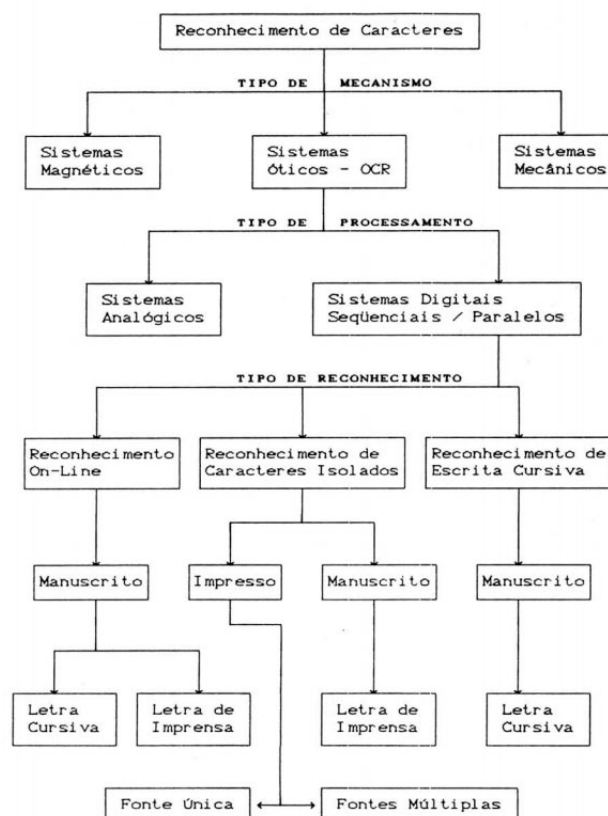


Figura 1: Organograma

3.3 Descrição dos Sistemas OCR

Existem diferentes etapas de processamento das informações visando o reconhecimento de textos por um sistema OCR. Estas etapas se estendem desde a cap-

tura da imagem do texto até a obtenção final de uma identificação deste. Em um sistema de COR não se deve considerar apenas a parte referente às técnicas empregadas na resolução do problema de classificação, mas um item de extrema importância é a medição e avaliação dos resultados obtidos por estes.

3.3.1 Etapas de Processamento

Em geral, os sistemas de OCR possuem implementadas as seguintes funções: aquisição da imagem do texto, tratamento da imagem, localização e separação dos caracteres, pré-processamento dos padrões, extração de atributos, reconhecimento/classificação dos padrões e pós-processamento.

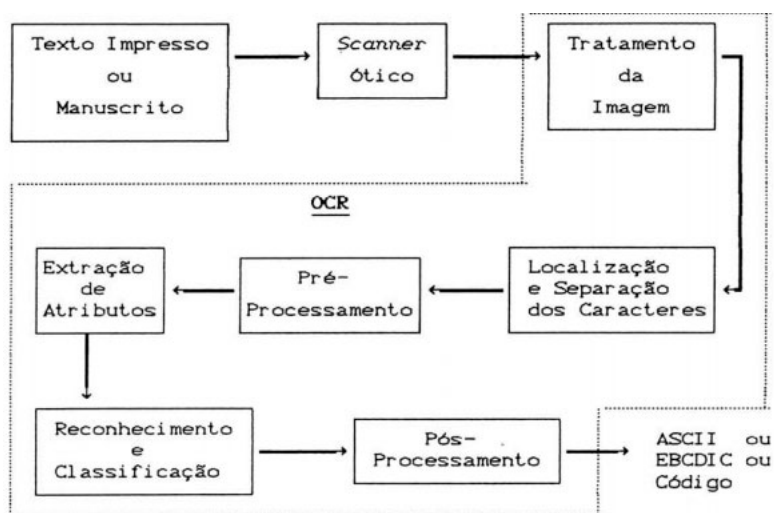


Figura 2: Etapas do processo de reconhecimento

Para a separação dos caracteres, o algoritmo tem que levar em consideração diversos fatores, entre eles: o espaçamento entre caracteres e tamanho do caractere (fixo ou variável), a utilização de uma grade auxiliar (no caso de formulários com espaçamento bem definidos), e o tipo de texto a ser reconhecido, onde podem haver caracteres que se tocam ou caracteres com descontinuidade (características muito importante para módulos de segmentação). Esta etapa de processamento é até hoje uma das etapas mais críticas, sendo um problema ainda não completamente solucionado.

Após terem sido isolados os caracteres, e preferencialmente, com a realização de um ajuste de tamanho e posição pré-definidos, pode-se então realizar uma etapa de extração de atributos. Esta etapada é opcional e dependerá exclusivamente do tipo de técnica empregada no reconhecimento e classificação dos padrões.

3.3.2 Técnicas de Reconhecimento

Existem diferentes técnicas empregadas no reconhecimento de padrões, mas estas podem ser classificadas em três grandes categorias, técnicas baseadas em:

Atributos globais: Nesta categoria, os atributos (features) são extraídos de cada ponto interior ao retângulo que circunscreve a matriz do caractere a qual também pode ser denominado de *frame*. Os atributos empregados não refletem nenhuma propriedade local, geométrica ou topológica do próprio traçado. Algumas dessas técnicas foram desenvolvidas originalmente para reconhecer apenas caracteres impressos.

Distribuição de pontos: É uma outra forma de reduzir a dimensionalidade dos conjuntos de atributos, onde estes são derivados a partir das distribuições estatísticas dos pontos. Diferentes tipos de distribuições tem sido utilizadas correspondendo a diferentes técnicas de reconhecimento.

Atributos geométricos e topológicos: Esta técnica é baseada na extração de atributos que descrevem a geometria ou topologia de interesse no traçado do caractere. Estes atributos podem representar propriedades locais. Esta é de longe a mais popular técnica estudada pelos pesquisadores.

3.3.3 Avaliação do Reconhecimento

Uma das formas de realizar a avaliação destes sistemas foi através da criação de bancos de dados padrões para análise de desempenho, contendo amostras variadas de pares a serem reconhecidos por um novo algoritmo submetido a avaliação de alguns destes bancos de amostras de pares de caracteres se tornaram muito difundidos e utilizados na avaliação de diferentes sistemas. Dois destes bancos de dados muito conhecidos são: o de Highleyman e o de Munson. Estes bancos de dados estão disponíveis junto ao IEEE e fornecem um conjunto de caracteres manuscritos em letra de impressora. Os dados de Munson possuem 12760 amostras de 46 tipos diferentes de caracteres, com uma resolução de 24 x 24 pixels, e os dados de Highleyman possuem 1800 amostras de letras e 500 de numerais, com um total de 36 tipos diferentes de caracteres, com uma resolução de 12 x 12 pixels. Esta resolução (12x12) infelizmente não é a mais adequada, uma vez que para o reconhecimento de caracteres manuscritos é aconselhada uma matriz com dimensões em torno de 20 x 25 pixels.

O maior problema referente a tais bancos de dados é a divulgação de sua existência e as formas de acesso aos mesmos. Infelizmente, para a realização deste trabalho não foi possível a utilização destas bases de dados de pares de caracteres, devido primeiramente ao fato de serem orientados a sistemas de reconhecimento de caracteres manuscritos e, em segundo, porque não foi possível até o momento ter acesso a eles.

Os principais parâmetros a serem tomados como referenda são os índices de avaliação de desempenho de sistemas de OCR. Estes índices consistem de três indicadores: taxa de caracteres reconhecidos corretamente, taxa de caracteres substituídos (classificações incorretas) e taxa de rejeição (caracteres não identificados).

3.4 Redes Neurais

As redes neurais foram desenvolvidas a partir de uma tentativa de criar um modelo que descrevesse a estrutura e o funcionamento dos neurônios do cérebro humano. Os modelos de redes neurais buscam definir novos computadores ou novos modelos de processamento de dados. Estes devem apresentar um comportamento baseado em modelos neurobiológicos ao invés de modelos baseados em "circuitos de silício"(portas lógicas, circuitos combinacionais, biestáveis, etc).

3.4.1 Características e Aplicações das Redes Neurais

As redes neurais possuem a característica de serem muito apropriadas ao reconhecimento de padrões. Como já foi visto, as redes neurais podem sofrer um aprendizado, modificando seu comportamento frente a um conjunto de estímulos de entrada (padrão de entrada). Portanto, a rede pode aprender a dar uma resposta específica para um determinado conjunto de estímulos fornecidos. Isto será obtido através da alteração dos pesos de atuação das entradas. Logo as redes neurais são muito adequadas para reconhecimento e classificação de pares, pois podem se adaptar para responder a um padrão específico. As redes neurais possuem também uma alta velocidade de processamento, devido ao seu maciço paralelismo interno, possibilitando o desenvolvimento de certos tipos de aplicações complexas que necessitam operar em tempo real. Devido às características inerentes às redes neurais, estas podem realizar alguns tipos de tarefas que não são executadas de uma forma satisfatória em sistemas computacionais tradicionais, mas que para a ser humano são tarefas triviais. Elas possuem a característica de se adequar perfeitamente às seguintes aplicações:

- Reconhecimento e síntese contínua da fala;
- Reconhecimento visual de padrões e padrões em geral
- Reconhecimento e classificação de imagens como, por exemplo: textos, assinaturas, impressões digitais, objetos, etc;
- Processamento adaptativo de sinais e eliminação de ruído dos;
- Aplicações onde os dados fornecidos são incompletos e os resultados produzidos são aproximados

4 Plano de desenvolvimento da aplicação

Implementamos e desenvolvemos essa aplicação durante nosso estudo com processamento de imagens e OCR (reconhecimento ótico de caracteres). Já está conseguindo reconhecer palavras completas com espaços e quebra de linhas. Assim conseguimos reconhecer qualquer letra ou palavras do nosso alfabeto, nosso projeto foi desenvolvido com o intuito de utilizar duas matérias muito importantes durante esse 6º módulo, que são Processamento de Imagem, e Sistema de Informação Inteligentes, dessas matérias retiramos conhecimento para o desenvolvimento dessa aplicação.

A ideia é fazer o sistema reconhecer letras e palavras do alfabeto escrita a mão, para que funcionasse perfeitamente treinamos uma imagem do nosso alfabeto escrito a mão. Para que a aplicação aprenda todas as letras, e reconheça as palavras que forem escritas a mão, toda vez que quisermos que o sistema entenda uma palavra diferente, precisamos treiná-la, para que o sistema aprenda e com isso reconheça a palavra na imagem. O processo para que a aplicação leia a palavra, começa quando recortamos as letras do alfabeto que o sistema já reconhece e formamos a palavra, importamos para o programa para treiná-la, após o treinamento, pedimos para o sistema reconhecer a palavra que está na imagem, ele reconhece a palavra corretamente. Qualquer letra ou palavra do alfabeto que for colocado no sistema para reconhecimento, será reconhecida, no log do projeto é visualizado letra por letra a porcentagem de aprendizado do programa, assim verificamos como se fosse um gráfico para o aprendizado de cada letra que o programa teve, podendo ter uma ideia da qualidade do sistema que foi desenvolvido, até o momento todas palavras e letras foram reconhecidas com êxito.

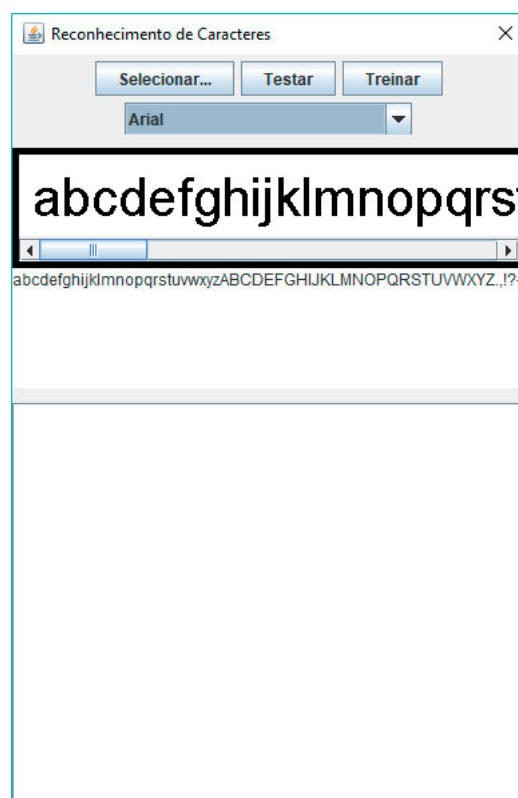


Figura 3: Reconhecimento de letras básicas

5 Código fonte

CharacterImage.java

```
import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.image.BufferedImage;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

public class CharacterImage implements Serializable {

    private static final long serialVersionUID = 1L;

    private List<Position> positions = new ArrayList<Position>();
    private Position first;
    private BufferedImage image;
    private int width, height;
    private int initWidth, initHeight;

    private double proximidade;

    private char character;

    private boolean isLineBreak;

    public char getCharacter() {
        return character;
    }

    public void setCharacter(char character) {
        this.character = character;
    }

    public CharacterImage(Position first) {
        this.first = first;
        this.initWidth = this.first.getX();
        this.initHeight = this.first.getY();
        this.add(first);
    }

    public boolean contains(Position position) {
        return positions.contains(position);
    }

    public void add(Position position) {
        if (width < position.getX()+1) {
            width = position.getX()+1;
        }
        if (height < position.getY()+1) {
            height = position.getY()+1;
        }
        if (initHeight > position.getY()) {
            initHeight = position.getY();
        }
        if (initWidth > position.getX()) {
            initWidth = position.getX();
        }
        positions.add(position);
    }

    public void add(CharacterImage character) {
        positions.addAll(character.getPositions());
        if (character.initWidth < initWidth) {
            initWidth = character.initWidth;
            first = character.first;
        }
        if (character.initHeight < initHeight) {
```

```

        initHeight = character.initHeight;
    }
    if (character.width > width) {
        width = character.width;
    }
    if (character.height > height) {
        height = character.height;
    }
}

public void addPosition(int x, int y) {
    add(new Position(x, y));
}

public boolean isCompl(CharacterImage character) {
    int space1 = initHeight - character.height;
    int space2 = character.initHeight - height;
    return (positions.size()/2) > character.positions.size()
        && character.initWidth >= initWidth && character.width <= width
        && ((space1 >= 0 && space1 < getHeight()/2)
            || (space2 >= 0 && space2 < getHeight()/2));
}

public boolean isSelf(CharacterImage character) {
    for (Position position: positions) {
        if (character.haveSibling(position)) {
            return true;
        }
    }
    return false;
}

public boolean haveSibling(Position olter) {
    for (Position position: positions) {
        if (position.isSibling(olter)) {
            return true;
        }
    }
    return false;
}

public boolean isValid(int x, int y) {
    int with = image.getWidth(), height = image.getHeight();
    return x >= 0 && y >= 0 && with > x && height > y;
}

public List<Position> getPositionScale() {
    List<Position> positionsScale = new ArrayList<Position>();
    for (int x = 0; x < image.getWidth(); x++) {
        for (int y = 0; y < image.getHeight(); y++) {
            if (image.getRGB(x, y) != Color.WHITE.getRGB()) {
                positionsScale.add(new Position(x, y));
            }
        }
    }
    return positionsScale;
}

public void analisarProximidade(CharacterImage outer) {
    proximidade = 0;
    if (outer.image == null) {
        outer.newImage();
    }
    newImageResize(outer.getWidth(), outer.getHeight());
    List<Position> outerPositions = outer.getPositionScale();
    List<Position> positionsScale = getPositionScale();
    for (Position pScale: positionsScale) {
        if (outerPositions.contains(pScale)) {
            proximidade++;
        }
    }
}

```

```

    }

    proximidade = ((proximidade*2)/(positionsScale.size()+outerPositions.size()))*100;

    image = null;
}

public void analisarProximidade2(CharacterImage outer) {
    proximidade = 0;
    newImage();
    outer.newImageResize(getWidth(), getHeight());
    List<Position> outerPositions = outer.getPositionScale();
    List<Position> positionsScale = getPositionScale();
    for (Position pScale: positionsScale) {
        if (outerPositions.contains(pScale)) {
            proximidade++;
        }
    }

    proximidade = ((proximidade*2)/(positionsScale.size()+outerPositions.size()))*100;

    image = null;
}

public BufferedImage newImageScale(int scale) {

    this.image = newImage();

    BufferedImage bi = new BufferedImage(scale * image.getWidth(null),
        scale * image.getHeight(null), BufferedImage.TYPE_INT_ARGB);

    Graphics2D grph = (Graphics2D) bi.getGraphics();
    grph.scale(scale, scale);

    grph.drawImage(image, 0, 0, null);
    grph.dispose();

    this.image = bi;
    return bi;
}

public BufferedImage newImageResize(int width, int height) {
    this.image = newImage();
    int type=0;
    type = image.getType() == 0? BufferedImage.TYPE_INT_ARGB : image.getType();
    BufferedImage resizedImage = new BufferedImage(width, height, type);
    Graphics2D g = resizedImage.createGraphics();
    g.drawImage(image, 0, 0, width, height, null);
    g.dispose();
    this.image = resizedImage;
    return resizedImage;
}

public BufferedImage newImage() {
    this.image = createImage(width-initWidth, height-initHeight);
    for (Position position: positions) {
        image.setRGB(position.getX() - initWidth, position.getY() - initHeight, Color.BLACK.getRGB());
    }
    return this.image;
}

private BufferedImage createImage(int width, int height) {
    BufferedImage newImage = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);

    Graphics2D g2d = (Graphics2D) newImage.getGraphics();
    g2d.setColor(Color.white);
    g2d.fillRect(0, 0, width, height);
    g2d.dispose();

    return newImage;
}

```



```

    }

    public List<Position> getPositions() {
        return positions;
    }

    public int getWidth() {
        return width-initWidth;
    }

    public int getHeight() {
        return height-initHeight;
    }

    public double getProximidade() {
        return proximidade;
    }

    public int getInitHeight() {
        return initHeight;
    }

    public int getInitWidth() {
        return initWidth;
    }

    public double calculateSpaceWidth(CharacterImage characterImage) {
        return characterImage.getInitWidth()-width;
    }

    public double calculateSpaceHeight(CharacterImage characterImage) {
        return characterImage.getInitHeight()-height;
    }

    public int getCenterY() {
        return height-(getHeight()/2);
    }

    public int getCenterX() {
        return width-(getWidth()/2);
    }

    public boolean isLineBreak() {
        return isLineBreak;
    }

    public void setLineBreak(boolean isLineBreak) {
        this.isLineBreak = isLineBreak;
    }
}

```

CharacterProcessing.java

```

import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import javax.imageio.ImageIO;

public class CharacterProcessing {

    private static BufferedImage grayscale, binarized;

    private double averageSpacingWidth;
    private double averageSpacingHeight;

```

```

public final static int WHITE = Color.WHITE.getRGB(), BLACK = Color.BLACK.getRGB();

public List<CharacterImage> discover(BufferedImage image) throws IOException {
    image = processing(image);

    List<CharacterImage> characters = new ArrayList<CharacterImage>();
    for (int x = 0; x < image.getWidth(); x++) {
        for (int y = 0; y < image.getHeight(); y++) {
            if (WHITE != image.getRGB(x, y)) {
                Position position = new Position(x, y);
                CharacterImage fist = null;
                List<CharacterImage> remove = new ArrayList<CharacterImage>();
                for (CharacterImage character: characters) {
                    if (character.haveSibling(position)) {
                        if (fist == null) {
                            character.add(position);
                            fist = character;
                        }
                        else {
                            fist.add(character);
                            remove.add(character);
                        }
                        //break;
                    }
                }
                characters.removeAll(remove);
                if (fist == null) {
                    characters.add(new CharacterImage(position));
                }
            }
        }
    }

    List<Integer> heightLines = new ArrayList<>();
    boolean init = false;
    for (int y = 0; y < image.getHeight(); y++) {
        boolean isLine = true;
        for (int x = 0; x < image.getWidth(); x++) {
            if (WHITE != image.getRGB(x, y)) {
                init = true;
                isLine = false;
                break;
            }
        }
        if (init && isLine) {
            init = false;
            heightLines.add(y);
        }
    }

    List<CharacterImage> charactersByLine = new ArrayList<>();
    CharacterImage last = null;
    Integer topHeightLine = null;
    for (Integer heightLine: heightLines) {
        for (CharacterImage character: characters) {
            if (character.getInitHeight()+character.getHeight() <= heightLine
                && (topHeightLine == null || character.getInitHeight() > topHeightLine)) {
                charactersByLine.add(character);
                last = character;
            }
        }
        last.setLineBreak(true);
        topHeightLine = heightLine;
    }
    last.setLineBreak(false);
    characters = charactersByLine;

    List<CharacterImage> remove = new ArrayList<CharacterImage>();
    for (CharacterImage character: characters) {
        for (CharacterImage characterOther: characters) {

```

```

        if (!character.equals(characterOther) && !remove.contains(characterOther)) {
            if (character.isCompl(characterOther)) {
                character.add(characterOther);
                remove.add(characterOther);
            }
            else if (characterOther.isCompl(character)) {
                characterOther.add(character);
                remove.add(character);
            }
        }
    }
}

characters.removeAll(remove);

CharacterImage previous = null;
double sumSpacesWidth = 0, amountSpacesWidth = 0;
double sumSpacesHeight = 0, amountSpacesHeight = 0;

for (CharacterImage character: characters) {
    if (previous != null) {
        double spaceWidth = previous.calculateSpaceWidth(character);
        if (spaceWidth > 0) {
            sumSpacesWidth+=spaceWidth;
            amountSpacesWidth++;
        }

        double spaceHeight = previous.calculateSpaceHeight(character);
        if (spaceHeight > 0) {
            sumSpacesHeight+=spaceHeight;
            amountSpacesHeight++;
        }
    }
    previous = character;
}

if (sumSpacesWidth > 0 && amountSpacesWidth > 0) {
    averageSpacingWidth = sumSpacesWidth / amountSpacesWidth;
}

if (sumSpacesHeight > 0 && amountSpacesHeight > 0) {
    averageSpacingHeight = sumSpacesHeight / amountSpacesHeight;
}

return characters;
}

public static BufferedImage processing(BufferedImage image) throws IOException {
    grayscale = toGray(image);
    binarized = binarize(grayscale);
    //writeImage("output_f");
    return binarized;
}

public static BufferedImage processing2(BufferedImage image) throws IOException {
    return toBinary(toGrayscale(image), 215);
}

private static BufferedImage toGrayscale(BufferedImage image)
    throws IOException {
    BufferedImage output = new BufferedImage(image.getWidth(),
        image.getHeight(), BufferedImage.TYPE_BYTE_GRAY);
    Graphics2D g2d = output.createGraphics();
    g2d.drawImage(image, 0, 0, null);
    g2d.dispose();
    return output;
}

private static BufferedImage toBinary(BufferedImage image, int t) {
    int BLACK = Color.BLACK.getRGB();
    int WHITE = Color.WHITE.getRGB();

    BufferedImage output = new BufferedImage(image.getWidth(),

```

```

        image.getHeight(), BufferedImage.TYPE_BYTE_GRAY);

    for (int y = 0; y < image.getHeight(); y++)
        for (int x = 0; x < image.getWidth(); x++) {
            Color pixel = new Color(image.getRGB(x, y));
            output.setRGB(x, y, pixel.getRed() < t ? BLACK : WHITE);
        }

    return output;
}

private static void writeImage(String output) throws IOException {
    File file = new File("test-resource/result-processing/"+output+".png");
    ImageIO.write(binarized, "jpg", file);
}

public static int[] imageHistogram(BufferedImage input) {
    int[] histogram = new int[256];

    for(int i=0; i<histogram.length; i++) histogram[i] = 0;

    for(int i=0; i<input.getWidth(); i++) {
        for(int j=0; j<input.getHeight(); j++) {
            int red = new Color(input.getRGB(i, j)).getRed();
            histogram[red]++;
        }
    }

    return histogram;
}

private static BufferedImage toGray(BufferedImage original) {
    int alpha, red, green, blue;
    int newPixel;

    BufferedImage lum = new BufferedImage(original.getWidth(), original.getHeight(), original.getType());

    for(int i=0; i<original.getWidth(); i++) {
        for(int j=0; j<original.getHeight(); j++) {

            alpha = new Color(original.getRGB(i, j)).getAlpha();
            red = new Color(original.getRGB(i, j)).getRed();
            green = new Color(original.getRGB(i, j)).getGreen();
            blue = new Color(original.getRGB(i, j)).getBlue();

            red = (int) (0.21 * red + 0.71 * green + 0.07 * blue);
            newPixel = colorToRGB(alpha, red, red, red);

            lum.setRGB(i, j, newPixel);
        }
    }

    return lum;
}

private static int otsuThreshold(BufferedImage original) {
    int[] histogram = imageHistogram(original);
    int total = original.getHeight() * original.getWidth();

    float sum = 0;
    for(int i=0; i<256; i++) sum += i * histogram[i];

    float sumB = 0;
    int wB = 0;
    int wF = 0;

    float varMax = 0;
    int threshold = 0;

```

```

        for(int i=0 ; i<256 ; i++) {
            wB += histogram[i];
            if(wB == 0) continue;
            wF = total - wB;

            if(wF == 0) break;

            sumB += (float) (i * histogram[i]);
            float mB = sumB / wB;
            float mF = (sum - sumB) / wF;

            float varBetween = (float) wB * (float) wF * (mB - mF) * (mB - mF);

            if(varBetween > varMax) {
                varMax = varBetween;
                threshold = i;
            }
        }

        return threshold;
    }

    private static BufferedImage binarize(BufferedImage original) {
        int red;
        int newPixel;

        int threshold = otsuThreshold(original);

        BufferedImage binarized = new BufferedImage(original.getWidth(), original.getHeight(), original.getType());

        for(int i=0; i<original.getWidth(); i++) {
            for(int j=0; j<original.getHeight(); j++) {

                red = new Color(original.getRGB(i, j)).getRed();
                int alpha = new Color(original.getRGB(i, j)).getAlpha();
                if(red > threshold) {
                    newPixel = 255;
                }
                else {
                    newPixel = 0;
                }
                newPixel = colorToRGB(alpha, newPixel, newPixel, newPixel);
                binarized.setRGB(i, j, newPixel);
            }
        }
        return binarized;
    }

    private static int colorToRGB(int alpha, int red, int green, int blue) {
        int newPixel = 0;
        newPixel += alpha;
        newPixel = newPixel << 8;
        newPixel += red; newPixel = newPixel << 8;
        newPixel += green; newPixel = newPixel << 8;
        newPixel += blue;

        return newPixel;
    }

    public boolean isBlankSpace(CharacterImage previus, CharacterImage next) {
        boolean blankSpace = false;
        if (previus != null && next != null) {
            double space = previus.calculateSpaceWidth(next);
            blankSpace = space > averageSpacingWidth*1.8;
        }
        return blankSpace;
    }
}

```

Classification.java

```
import java.util.List;

public interface Classification {
    CharacterImage analyze(CharacterImage character);
    List<CharacterImage> getCharacters();
}
```

OCRProcessing.java

```
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.ArrayList;
import java.util.List;

import javax.swing.JTextArea;

public class OCRProcessing {

    private static final String BASE_DIR = "test-resource";

    private JTextArea txtConsole;
    private Classification chassification;

    public OCRProcessing() {
        try {
            this.chassification = new PixelsPositionsClassification(loadData());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public OCRProcessing(JTextArea txtConsole) {
        this();
        this.txtConsole = txtConsole;
    }

    public void learn(String letras, BufferedImage image) throws IOException, ClassNotFoundException {
        letras = letras.replaceAll("_", "");

        CharacterProcessing processing = new CharacterProcessing();
        List<CharacterImage> characters = processing.discover(image);

        for (int index = 0; index < characters.size(); index++) {
            characters.get(index).setCharacter(letras.charAt(index));
        }

        save(characters);
    }

    public String recognizer(BufferedImage image) throws FileNotFoundException, ClassNotFoundException, IOException {
        CharacterProcessing processing = new CharacterProcessing();
        List<CharacterImage> characters = processing.discover(image);
        StringBuilder txt = new StringBuilder();

        CharacterImage previous = null;
        for (CharacterImage character: characters) {
            //ImageIO.write(character.newImage(), "png", new File(BASE_DIR+"/result-processing/"+Instant.now().toEpochMilli()+".png"));
            CharacterImage candidate = chassification.analyze(character);

            if (processing.isBlankSpace(previous, character)) {
                txt.append("_");
            }

            txt.append(candidate.getCharacter());
        }
    }
}
```

```

        if (character.isLineBreak()) {
            txt.append("\n");
        }

        print(String.format("Caractere:_%s_,_aprox:_%s\n", candidate.getCharacter(), candidate.getProximidade()));
        previus = character;
    }

    print(String.format("O_resultado_e:_%s\n", txt.toString()));

    return txt.toString();
}

private void save(List<CharacterImage> characters) throws IOException, ClassNotFoundException {
    File fileData = new File(BASE_DIR+"/data.ser");

    if (!fileData.getParentFile().exists()) {
        fileData.getParentFile().mkdirs();
    }

    chassification.getCharacters().addAll(characters);
    FileOutputStream fos = new FileOutputStream(fileData);
    ObjectOutputStream oos = new ObjectOutputStream(fos);
    oos.writeObject(chassification.getCharacters());
    oos.close();
    fos.close();
}

private List<CharacterImage> loadData() throws FileNotFoundException, IOException, ClassNotFoundException {
    File fileData = new File(BASE_DIR+"/data.ser");

    List<CharacterImage> characters = null;

    if (fileData.exists()) {
        FileInputStream fis = new FileInputStream(fileData);
        ObjectInputStream ois = new ObjectInputStream(fis);
        characters = (List<CharacterImage>) ois.readObject();
        fis.close();
        ois.close();
    }
    else {
        characters = new ArrayList<CharacterImage>();
    }

    return characters;
}

private void print(String text) {
    if (txtConsole == null) {
        //System.out.print(text);
    }
    else {
        txtConsole.append(text);
    }
}
}

```

OCRView.java

```

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.Graphics2D;
import java.awt.GraphicsEnvironment;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.font.FontRenderContext;
import java.awt.geom.AffineTransform;

```

```

import java.awt.geom.Rectangle2D;
import java.awt.image.BufferedImage;
import java.io.File;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import javax.imageio.ImageIO;
import javax.swing.BorderFactory;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JDialog;
import javax.swing.JFileChooser;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.filechooser.FileNameExtensionFilter;

import br.com.deployxtech.ocr.OCRProcessing;

public class OCRView extends JDialog {

    private static final long serialVersionUID = 1L;

    private JButton btnTest = new JButton("Testar");
    private JButton btnLearn = new JButton("Treinar");
    private JPanel pnlControl = new JPanel();

    private JLabel lblImage = new JLabel();
    private JComboBox<String> cmbFonts = new JComboBox<>();
    private JButton btnSearchImage = new JButton("Selecionar...");
    private JPanel pnlImage = new JPanel();
    private JTextArea txtResult = new JTextArea();
    private JTextArea txtConsole = new JTextArea();
    private JPanel pnlProcessing = new JPanel(new BorderLayout(10,10));

    private JFileChooser fc = new JFileChooser(".");

    private OCRProcessing processing = new OCRProcessing(txtConsole);

    private BufferedImage image;

    public OCRView() {
        setTitle("Reconhecimento_de_Caracteres");
        setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
        init();
    }

    private void init() {
        GraphicsEnvironment e = GraphicsEnvironment.getLocalGraphicsEnvironment();
        Font[] fonts = e.getAllFonts();
        for (Font font : fonts) {
            cmbFonts.addItem(font.getFontName());
        }

        setLayout(new BorderLayout(10,10));
        pnlControl.add(btnSearchImage);
        pnlControl.add(btnTest);
        pnlControl.add(btnLearn);
        pnlControl.add(cmbFonts);
        pnlControl.setPreferredSize(new Dimension(800, 60));
        getContentPane().add(pnlControl, BorderLayout.NORTH);
        pnlProcessing.setLayout(new GridLayout(2, 1));
        JScrollPane imageScroll = new JScrollPane(lblImage);
        imageScroll.setBorder(BorderFactory.createLineBorder(Color.BLACK,5));
        pnlProcessing.add(imageScroll, BorderLayout.NORTH);
        pnlProcessing.add(txtResult, BorderLayout.CENTER);
        getContentPane().add(pnlProcessing, BorderLayout.CENTER);
    }

```



```

txtConsole.setEditable(false);
JScrollPane consoleScroll = new JScrollPane(txtConsole);
consoleScroll.setPreferredSize(new Dimension(800, 300));
getContentPane().add(consoleScroll, BorderLayout.SOUTH);

btnSearchImage.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        try {
            FileNameExtensionFilter imageFilter = new FileNameExtensionFilter(
                "Image_files", ImageIO.getReaderFileSuffixes());
            fc.setFileFilter(imageFilter);
            fc.setAcceptAllFileFilterUsed(false);
            int returnVal = fc.showOpenDialog(OCRView.this);
            if (returnVal == JFileChooser.APPROVE_OPTION) {
                File fileImage = fc.getSelectedFile();
                image = ImageIO.read(fileImage);
                lblImage.setIcon(new ImageIcon(image));

                Pattern pattern = Pattern.compile("\\.(.*?\\.\\w+)$");
                Matcher matcher = pattern.matcher(fileImage.getName());
                if (matcher.find()) {
                    String letras = matcher.group(2);
                    txtResult.setText(letras);
                }
                else {
                    txtResult.setText("");
                }
            }
        }
        catch (Exception ex) {
            ex.printStackTrace();
        }
    }
});

btnTest.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        try {
            if (image == null) {
                JOptionPane.showMessageDialog(OCRView.this, "E_preciso_selecionar_uma_imagem.");
            }
            else if (!txtResult.getText().equals("")) {
                JOptionPane.showMessageDialog(OCRView.this, "E_preciso_apagar_a_caixa_de_texto_para_realizar_o_teste");
            }
            else {
                String result = processing.recognizer(image);
                txtResult.setText(result);
            }
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
});

btnLearn.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        try {
            if (image == null) {
                JOptionPane.showMessageDialog(OCRView.this, "E_preciso_selecionar_uma_imagem.");
            }
            else if (txtResult.getText().equals("")) {
                JOptionPane.showMessageDialog(OCRView.this, "E_preciso_escrever_as_letras_exatamente_como_esta_na_tela");
            }
            else {
                processing.learn(txtResult.getText(), image);
                txtResult.setText("");
                lblImage.setIcon(null);
                JOptionPane.showMessageDialog(OCRView.this, "Letras_aprendidas...");
            }
        }
        catch (Exception e) {

```

```

        e.printStackTrace();
    }
}

});

cmbFonts.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        String text = "abcdefghijklmnopqrstuvwxyz_ABCDEFGHIJKLMNOPQRSTUVWXYZ_!?-__0123456789";
        Font font = fonts[cmbFonts.getSelectedIndex()];
        font = new Font(font.getFontName(), Font.PLAIN, 40);

        Rectangle2D rectagleText = font.getStringBounds(text, new FontRenderContext(new AffineTransform(), true, true));

        int width = new Double(rectagleText.getWidth()).intValue()+20, height = new Double(rectagleText.getHeight()).intValue();

        BufferedImage newImage = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);

        Graphics2D g2d = (Graphics2D) newImage.getGraphics();
        g2d.setColor(Color.white);
        g2d.fillRect(0, 0, width, height);
        g2d.setColor(Color.black);
        g2d.setFont(font);
        g2d.drawString(text, 10, height-15);
        g2d.dispose();

        lblImage.setIcon(new ImageIcon(newImage));
        txtResult.setText(text.replace("_", ""));
        image = newImage;
    }
});
}

/**
 * @param args
 */
public static void main(String[] args) {
    OCRView view = new OCRView();
    view.setSize(400, 600);
    view.setLocationByPlatform(true);
    view.setVisible(true);
}
}

```

PixelsPositionsClassification.java

```

import java.util.List;

public class PixelsPositionsClassification implements Classification {

    private List<CharacterImage> characters;
    private double averageProximity = 100d;
    private int averageQt = 0;

    public PixelsPositionsClassification(List<CharacterImage> characters) {
        this.characters = characters;
    }

    @Override
    public CharacterImage analyze(CharacterImage character) {
        int qt = 0;
        int qtCandidate = 0;
        CharacterImage bestCandidate = null;
        for (CharacterImage charAnalyze: characters) {
            qt++;
            charAnalyze.analisarProximidade2(character);
            if (bestCandidate == null || (bestCandidate.getProximidade() < charAnalyze.getProximidade())) {
                bestCandidate = charAnalyze;
                qtCandidate = qt;
            }
        }
    }
}

```

```

        }

        if (bestCandidate.getProximidade() > averageProximity && qt > averageQt) {
            break;
        }
    }

    averageProximity = (averageProximity+bestCandidate.getProximidade())/2;
    averageQt = (averageQt+qtCandidate)/2;

    if (this.characters.remove(bestCandidate)) {
        this.characters.add(0,bestCandidate);
    }

    return bestCandidate;
}

@Override
public List<CharacterImage> getCharacters() {
    return characters;
}
}

```

Position.java

```

import java.io.Serializable;

public class Position implements Serializable {

    private static final long serialVersionUID = 1L;

    private int x;
    private int y;

    public Position(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public int getX() {
        return x;
    }

    public void setX(int x) {
        this.x = x;
    }

    public int getY() {
        return y;
    }

    public void setY(int y) {
        this.y = y;
    }

    @Override
    public boolean equals(Object obj) {
        if (obj instanceof Position) {
            Position outer = (Position)obj;
            return x == outer.x && y == outer.y;
        }
        else {
            return false;
        }
    }

    public boolean isSibling(Position position) {
        return (x+1 == position.x && y+1 == position.y)
            || (x+1 == position.x && y == position.y)
            || (x+1 == position.x && y-1 == position.y)
            || (x == position.x && y+1 == position.y)
    }
}

```

```

        || (x == position.x && y-1 == position.y)
        || (x-1 == position.x && y+1 == position.y)
        || (x-1 == position.x && y == position.y)
        || (x-1 == position.x && y-1 == position.y);
    }

    @Override
    public String toString() {
        return String.format("%s_-%s", x, y);
    }
}

```

6 Bibliografia

Referências

Mantas, J. (1986). An overview of character recognition methodologies.

Osorio, F. S. (1991). Um estudo sobre reconhecimento visual de caracteres através de redes neurais.

Silva, Eugênio e Thomé, A. (2003). Reconhecimento de caracteres manuscritos utilizando time de redes neurais. In *Congresso da Sociedade Brasileira de Computação, XXIII*, pages 1–8.