PGCOMP

IVISION

Práticas Laboratoriais em Reconhecimento de Padrões em Imagens - versão 3.0

Disciplina: Visão Computacional e Reconhecimento de Padrões

2019

Sumário

Capítulo 1

Introdução

Este documento foi elaborado com a intenção de orientar os alunos nas aulas práticas de laboratório. O conteúdo a ser utilizado baseia-se no Planejamento da Disciplina Visão Computacional e Reconhecimento de Padrões. Para facilitar a compreensão, este documento foi subdivido em seções, de modo que cada seção contemple um Roteiro de aula prática.

1.1 Ambiente computacional de trabalho

Todas as práticas serão realizadas com o auxílio da plataforma Anaconda, que disponibiliza uma variedade de pacotes voltados para trabalhos com inteligência artificial e aprendizado de máquina.[1] A figura 1.1 ilustra a arquitetura de funcionamento da plataforma, de modo que cada trabalho pode ser isolado em um ambiente separado (ex: *conda env. 1* / *Analysis 1*). Já a figura 1.2 ilustra um possível ambiente de produção baseado no Container Docker.

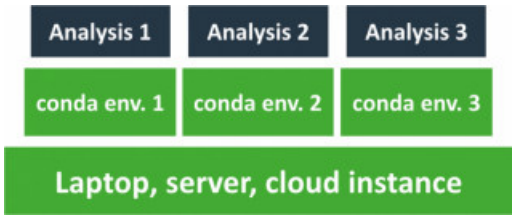

Figura 1.1 Data Science Development

Para a instalação da plataforma será utilizado um computador com os seguintes requisitos mínimos:

- Processador de 32 ou 64 bits;

[1] **Nota**: Não é necessario privilégios administrativos para instalar o Anaconda, se você selecionar uma pasta local (user-permissions).

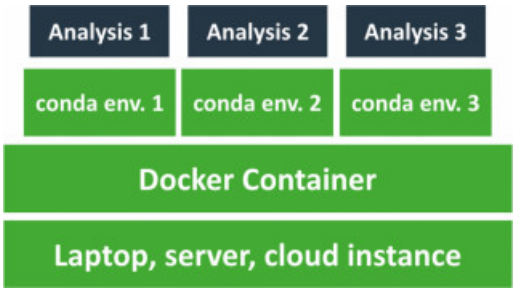

Figura 1.2 Data Science Deployment

- Para Miniconda - 400 MB de espaço em disco;
- 3GB de espaço em disco para Anaconda (download e instalação);
- Linux. Outras opções: Windows ou macOS;
- Python versão atual;
- Pycosat; (Boolean problem)
- PyYaml; (data serialization)
- Requests. (HTTP/1.1)

1.1.1 Instalação da plataforma Anaconda

Realizar os seguintes passos no sistema operacional Ubuntu 16.04 ou 18.04:

1. Abra um terminal na pasta Downloads;
2. Faça o Download do instalador do Anaconda. Para realizar essa tarefa, escolha a opção referente a versão mais recente da linguagem python:
   https://www.anaconda.com/download/
3. Realize a instalação do Anaconda.

1.1.2 Gerenciando o ambiente computacional

Por padrão, existe um *ambiente básico* gerado pela instalação. Contudo, o gerenciador Conda permite a criação de novos ambientes isolados contendo arquivos, pacotes e suas dependências. Essa característica permite ao usuário trabalhar em ambientes virtuais com diferentes perfis de configuração.

O gerenciamento do ambiente será realizado pela interface gráfica do Anaconda, que pode ser acessado pelo comando:

```
$ anaconda-navigator
```

1.1.3 Instalando pacotes no ambiente virtual

Para a realização das aulas práticas será necessária a instalação de alguns pacotes já disponíveis no gerenciador conda:

- Numpy;
- Scikit-image;
- Scikit-learn;
- Spyder;
- Jupyter;
- Matplotlib
- Pandas;
- Keras;
- Scipy.

Cada aluno deve realizar a instalação desses pacotes e verificar se os mesmos foram devidamente instalados (ex: listar os pacotes do ambiente).

Capítulo 2

Pré-processamento de imagem

2.1 Teoria

2.1.1 Espaços de cor

O objetivo de um espaço de cor é facilitar a especificação das cores em alguma forma padronizada. Essencialmente, um modelo de cores é uma especificação de um sistema de coordenadas e um subespaço dentro desse sistema no qual cada cor é representada por um único ponto [10].

A maioria dos sistemas de cores utilizados atualmente é orientada ou em direção ao hardware (como no caso de monitores e impressoras coloridas) ou em direção a aplicações envolvendo a manipulação de cores (como a criação de imagens coloridas para uma animação). Em termos de processamento digital de imagens, os modelos orientados para hardware mais utilizados na prática são o modelo RGB (red, green, blue – vermelho, verde, azul) para monitores coloridos e uma ampla classe de câmeras de vídeo em cores; o modelo CMY (cyan, magenta, yellow – ciano, magenta, amarelo) e o modelo CMYK (cyan, magenta, yellow, black – ciano, magenta, amarelo e preto) para impressão colorida; e o modelo HSI (hue, saturation, intensity – matiz, saturação, intensidade), que corresponde estreitamente à forma como os seres humanos descrevem e interpretam as cores.

2.1.2 Espaços de cor RGB

No modelo RGB, cada cor aparece em seus componentes espectrais primários de vermelho, verde e azul. Este modelo baseia-se em um sistema de coordenadas cartesianas. O subespaço de cores de interesse é o cubo, apresentado na Figura 2.1, no qual os valores RGB primários estão em três vértices, as cores secundárias ciano, magenta e amarelo estão em três vértices;

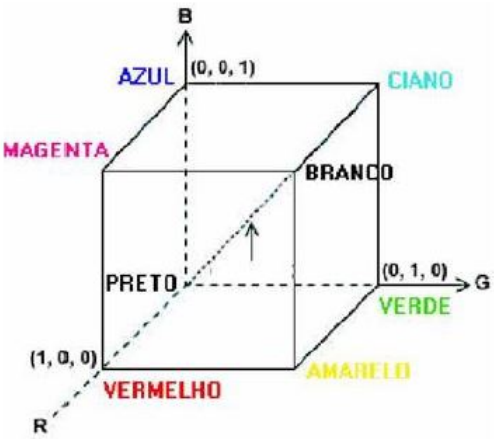

Figura 2.1 Esquema do cubo de cores RGB. Imagem retirada de [10].

o preto está na origem; e o branco está no vértice mais distante da origem. Nesse modelo, a escala de cinza (pontos de valores RGB iguais) estende-se do preto até o branco ao longo do segmento de reta que une esses dois pontos. As diferentes cores nesse modelo são pontos no cubo ou dentro dele e são definidas por vetores que se estendem a partir da origem. Por conveniência, assume-se que todos os valores de cor foram normalizados, de forma que o cubo mostrado na Figura 2.1 é o cubo unitário [10]. Isto é, assume-se que todos os valores de R, G e B estejam no intervalo [0, 1].

2.1.3 Correção gamma

O objetivo da correção gamma é apresentar uma imagem com precisão em uma tela. A correção gamma controla o nível de brilho de uma imagem que, se não apropriadamente corrigida, pode se apresentar muito embranquecida ou escurecida [4]. A luz emitida pela exibição de uma TV ou monitor é gerada por tubo de raios catódicos, no entanto, esta resposta é não linear, como pode ser visto na Figura 2.2. O sistema é não linear porque o perfil do feixe de elétrons é moldado por uma gaussiana limitando assim a resolução vertical eficaz. Em quase todo monitor, a resposta à intensidade da voltagem é na forma de uma curva gaussiana aproximadamente uma função exponencial de expoente 2,5. A correção gamma opera corrigindo a linha gaussiana para uma linear[4].

Chama-se fator gamma o critério que define o caráter não linear da intensidade luminosa de um elemento. Assim, a luminância de um monitor de computador não é linear na medida em que [3]:

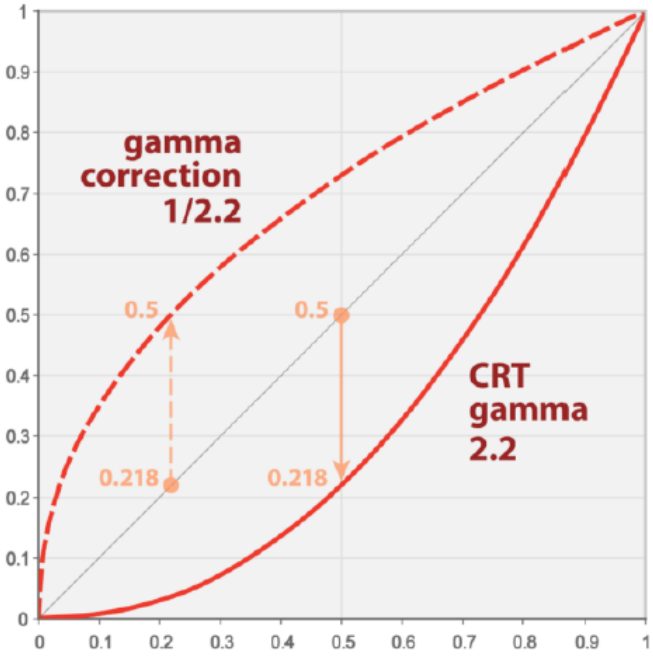

Figura 2.2 Correção gamma para uma linear. Imagem retirada de [4].

- A intensidade luminosa que emite não é linearmente proporcional à tensão aplicada, mas corresponde a uma curva função da gamma do monitor (geralmente compreendida entre 2,3 e 2,6): I ~ Vgamma;
- A intensidade luminosa percebida pelo olho não é proporcional à quantidade de luz efetivamente emitida.

Para remediar para esse efeito e obter uma reprodução satisfatória da intensidade luminosa, é possível compensar a luminância aplicando uma transformação chamada correção gamma. Assim, a cada periférico de afixação corresponde uma transformação gamma, podendo ela própria ser adaptado à percepção do utilizador [3].

2.1.4 Aguçamento de imagem

Os principais objetivos do aguçamento de imagens digitais são melhorar a qualidade visual geral da imagem digital, aumentando-se o contraste entre os elementos apresentados na imagem, e realçar *features* específicas relacionadas aos objetos de interesse. O contraste entre dois objetos pode ser definido como a razão entre os seus níveis de cinza médios.

Dentre as diversas técnicas de aguçamento de imagens, uma das mais comuns é a equalização de histogramas. Essa técnica tem por finalidade produzir uma imagem cujo histograma tenha um formato desejado. A equalização modifica o histograma da imagem

original de tal forma que a imagem transformada tenha um histograma (aproximadamente) uniforme, ou seja, todos os níveis de cinza devem aparecer na imagem com frequências semelhantes.

2.1.5 Equalização de histograma

A equalização de histograma é uma técnica a partir da qual se procura redistribuir os valores de tons de cinza dos pixels em uma imagem, de modo a obter um histograma uniforme, no qual o número (percentual) de pixels de qualquer nível de cinza é praticamente o mesmo. Para tanto, utiliza-se uma função auxiliar, denominada função de transformação. A forma mais usual de se equalizar um histograma é utilizar a função de distribuição acumulada (cdf - cumulative distribution function) da distribuição de probabilidades original, que pode ser expressa por [19]:

$$s_k = T(r_k) = \sum_{j=0}^{k} \frac{n_j}{n} = \sum_{j=0}^{k} p_r(r_j) \tag{2.1}$$

onde $0 \leq r_k \leq 1$ e k = 0, 1,..., L-1.

A Figura 2.3 traz um exemplo de equalização de histograma.

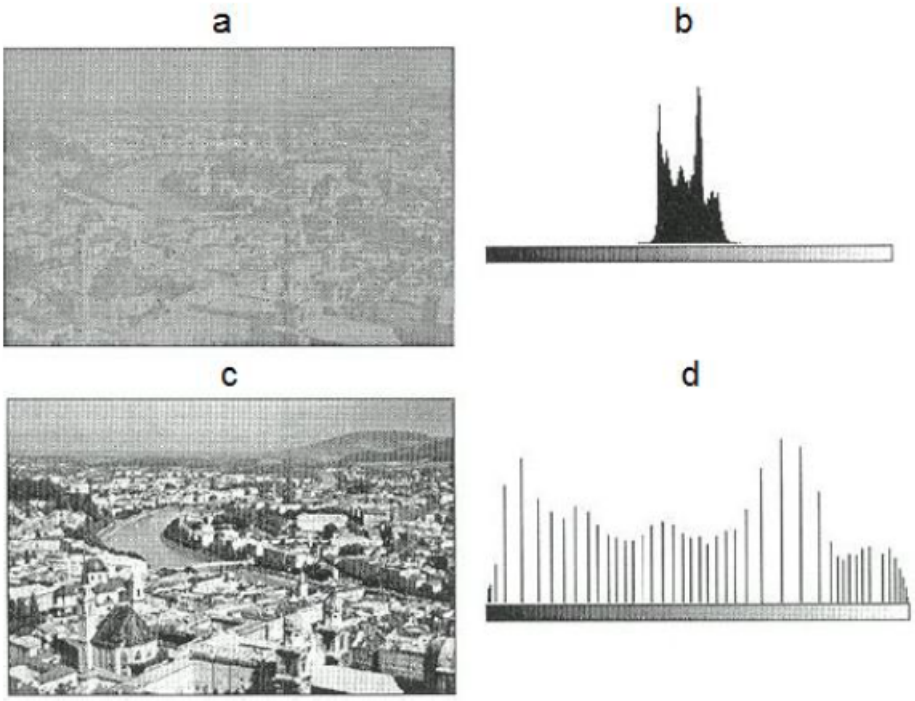

Figura 2.3 Equalização de Histograma. (a) Imagem original. (b) Histograma da imagem original. (c) Imagem após equalização do histograma. (d) Histograma equalizado. Imagem retirada de [19].

2.1.6 Suavização de imagem

Os filtros (máscaras) de suavização são usados para a redução do ruído e por este motivo diminuem a nitidez da imagem. Tais filtros também são chamados de filtros passa-baixas, por atenuarem as regiões de borda e detalhes finos na imagem (e.g., ruído) que correspondem aos componentes de alta frequência. Um dos filtros de suavização mais comumente usados em processamento de imagens são os filtros gaussianos.

O filtro gaussiano também é muito utilizado para suavização de imagens, com a diferença de não preservar as arestas, uma vez que não considera a diferença das intensidades. Ele possui dois parâmetros, a dimensão da janela e um valor para o desvio padrão máximo sigma. Seu comportamento é similar ao filtro passa-baixa, isto é, suavização de imagens. O quanto a imagem será suavizada está relacionado ao desvio padrão sigma, isto é, quanto maior o sigma, mais a imagem é suavizada, não dependendo muito do parâmetro referente a dimensão da janela. Quanto maior o sigma, maior o número de pixels cujo valor é diferente de zero, o que leva os pixels vizinhos a terem maior influência em cada ponto, realizando uma suavização maior na imagem [2].

O filtro Gaussiano é definido por:

$$GB[I]_p = \sum_{q \varepsilon S} G_\sigma(\|p-q\|) I_q \tag{2.2}$$

onde $0 \leq r_k \leq 1$ e k = 0, 1,..., L-1.

A Figura 2.4 mostra o resultado da aplicação do filtro gaussiano com diferentes σ.

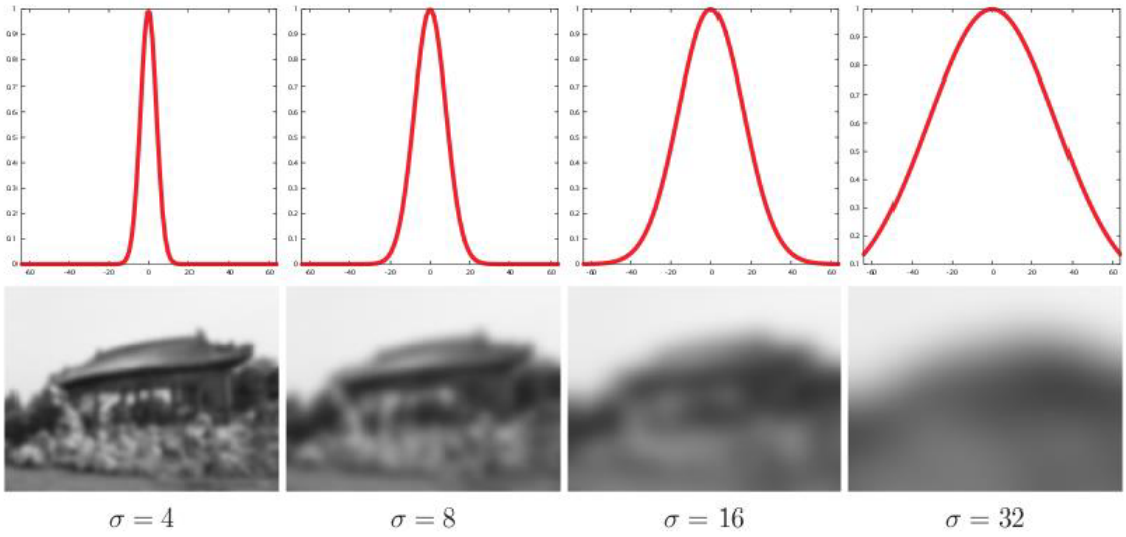

Figura 2.4 Exemplos de filtragem gaussiana para diferentes valores de σ. Imagem retirada de [2].

2.2 Prática

2.2.1 Objetivo

Nessa prática, algumas operações de pré-processamento de imagem são apresentadas. O objetivo é estudar os impactos dessas operações no desempenho do classificador, o qual será objeto de práticas posteriores. Com isso, espera-se motivar reflexões sobre a conveniência de se utilizar operações de pré-processamento de imagens em um sistema de Visão Computacional.

2.2.2 Ferramentas

Ao longo dessa prática os seguintes materiais são necessários:

- *Pets Dataset* contendo imagens anotadas de cães e gatos, num total de 10.000 imagens de treino e 10.000 imagens para teste. Para realizar o *Download* acesse o link aqui.
- **Nota:** Padronize todas as imagens do *Pets Dataset* com a mesma dimensão de linhas e colunas (N $\times$ N).
- **Skimage package (*image processing in python*):**
  - skimage.io.imread(filename) - Função para leitura de um arquivo de imagem;
  - skimage.io.imsave(img) - Função para salvar imagem para um arquivo;
  - skimage.color.rgb2gray(img) – Função para converter de RGB para escala de Cinza;
  - skimage.transform.resize(img, output_shape) – Função para redimensionar a imagem;
  - skimage.exposure.adjust_gamma(img, value) - Função para correção gamma;
  - skimage.filters.gaussian(img, sigma) - Função para filtro Gaussiano;
  - skimage.filters.laplace(img, ksize, mask) - Função para filtro Laplace;
  - skimage.exposure.equalize_hist(img, nbins) - Função para equalização de histograma.

2.2.3 Roteiro I

1. Aplique a equalização de histograma nas imagens do dataset original (treino e teste). Em seguida, salve os resultados dos novos datasets;

2. Aplique correção gamma nas imagens do dataset original (treino e teste), utilizando três diferentes valores de gamma (ajuste os parâmetros a seu critério). Em seguida, salve os resultados dos novos datasets;

3. Suavize as imagens do dataset original(treino e teste) aplicando o filtro gaussiano (ajuste os parâmetros a seu critério). Em seguida, salve os resultados do novo dataset;

4. Realize o aguçamento das imagens do dataset original (treino e teste) aplicando o filtro laplace (ajuste os parâmetros a seu critério). Em seguida, salve os resultados do novo dataset.

2.2.4 Roteiro II

1. Implemente as transformações de cor do cabeçalho da Figura 2.5, utilizando 3 imagens do dataset original [1]. Além disso, realize a extração das seguintes *features*:

   (a) Histograma RGB

   (b) O_1, O_2

   (c) Transformada de Cor

	Light intensity change $\begin{pmatrix} a & 0 & 0 \\ 0 & a & 0 \\ 0 & 0 & a \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$	**Light intensity shift** $\begin{pmatrix} R \\ G \\ B \end{pmatrix} + \begin{pmatrix} o_1 \\ o_1 \\ o_1 \end{pmatrix}$	**Light intensity change and shift** $\begin{pmatrix} a & 0 & 0 \\ 0 & a & 0 \\ 0 & 0 & a \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} + \begin{pmatrix} o_1 \\ o_1 \\ o_1 \end{pmatrix}$
RGB Histogram	-	-	-
O_1, O_2	-	+	-
Transformed color	+	+	+

Figura 2.5 Invariância dos descritores contra tipos de mudanças no modelo de deslocamento diagonal e suas especializações. A invariância é indicada com "+", a falta de invariância é indicada por -". Imagem adaptada de [34].

2.2.5 Produtos

Ao final dessa prática, o aluno deverá ter em mãos os seguintes itens:

- Versões de datasets (treino e teste) com equalização de histograma, suavização, aguçamento e três diferentes parâmetros de correção gamma.

[1] Escolha aleatória

2.2.6 Relatório

A seguir, são apresentados alguns pontos que devem constar no relatório. Ressalva-se, entretanto, que o aluno não deve se sentir limitado a segui-los, sendo interessante expandir a discussão para além do que está sendo proposto. Segue alguns itens:

- Analisar a partir dos parâmetros utilizados para cada função de pré-processamento os aspectos visuais de cada procedimento ao longo dessa prática (equalização de histograma, suavização, aguçamento e correção gama);
- Avaliar e discutir a invariância de cor de cada feature de cor sobre o efeito das transformações realizadas.

Capítulo 3

Extração de características

3.1 Teoria

Extração de *features* ou característica é o processamento utilizado para identificar e selecionar possíveis informações relevantes em um conjunto de dados com o intuito de melhorar a representação da informação crua da imagem. Em um conceito mais direto: é transformar os dados (grande quantidade de informação) em uma representação útil.

A extração de *features* pode ser vista como um pré-processamento sobre o conjunto de dados, podendo crescer (descritor HOG, LBP, Haar-like features, etc.), diminuir (Análise de Componentes Principais, etc.) ou não alterar a dimensionalidade dos dados iniciais. A principal motivação deste passo é representar o fenômeno visual de interesse de modo a descrever cada uma das suas *features* possivelmente de forma única.

Abaixo é apresentada uma visão geral dos três descritores (LBP, GLCM e Sobel) utilizados nesta prática. Para maiores detalhes sobre extração de *features*, consulte [43].

3.1.1 LBP

Local binary patterns (LBP), foi inicialmente proposto por Ojala, Pietikäinen e Harwood (1996) ou qualquer de suas variações [23]. LBP é um extrator de textura que analisa o padrão de tons de cinza e rotula todos os pixels por meio da comparação com os seus vizinhos de acordo com o valor do pixel central - produz uma representação invariante a iluminação, apresentado na Figura 3.1. Em seguida, concatena se todos os valores atribuídos em relação ao pixel central em um número binário. Por fim, os rótulos dos pixels computados são organizados em histogramas para descrever a textura, o que pode ser feito para a imagem inteira ou partes da imagem [32]. A representação do cálculo LBP (ver as equações 3.1 e 3.2) é feita por:

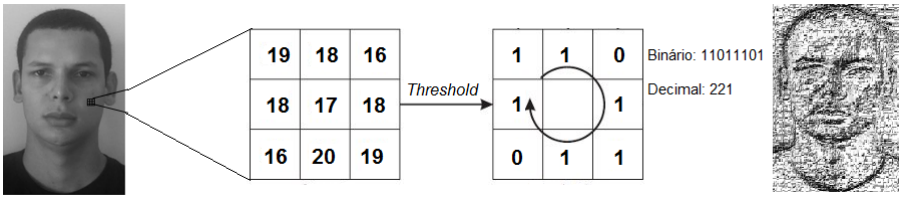

Figura 3.1 Processo de transformação do operador de análise de textura, onde é processada a imagem em tons de cinza por meio do operador LBP original. Imagem retirada de [32].

$$LBP_{P,R} = \sum_{p=0}^{P\text{-}1} S(g_p - g_c)2^p\,, \tag{3.1}$$

$$S(x) = \begin{cases} 1, & Se\ \ x \geq 0 \\ 0, & Se\ \ x < 0 \end{cases}\,, \tag{3.2}$$

onde P,R são utilizados para designar pixels vizinhos, P o número de amostras, e R um raio de vizinhança; g_c é o valor do pixel central em nível de cinza, g_p é o valor dos vizinhos, e S designa uma função de limiar.

3.1.2 GLCM

Grey-Level Co-occurrence Matrix (GLCM) é uma técnica utilizada dentro da área de análise de texturas, que foi desenvolvida na década de 70 pelo pesquisador Robert M. Haralick [13]. É um método estatístico para extração de *features* que usa estatísticas de segunda ordem, pois analisa as co-ocorrências existentes entre pares de pixels, ou seja, ela não analisa cada pixel individualmente mas sim, conjuntos de pixels relacionados através de algum padrão [29]. A Figura 3.2 ilustra um exemplo de uma matriz GLCM obtida a partir dos pixels de uma imagem. Observe que o relacionamento estabelecido é (1,1), ou seja, um pixel adjacente horizontalmente a uma distância unitária. O elemento na linha 1 e coluna 1 da matriz GLCM tem o valor 1, o que indica que existe apenas uma única situação na imagem onde um pixel de nível de intensidade 1 é adjacente horizontalmente a um outro pixel de nível de intensidade 1. O elemento na linha 1 e coluna 2 da matriz GLCM tem o valor 2, o que indica que existem duas ocorrências na imagem onde um pixel de nível de intensidade 1 é adjacente horizontalmente a um pixel de nível de intensidade 2. Esse processo é repetido até que a matriz GLCM esteja completa [29].

A partir da GLCM, pode-se extrair uma série de *features* proposto por Haralick, sendo que a quantidade de *features* utilizadas em um problema varia de acordo com as especificações do mesmo [13]. Nesta prática, será computada uma das principais *features* definidas por

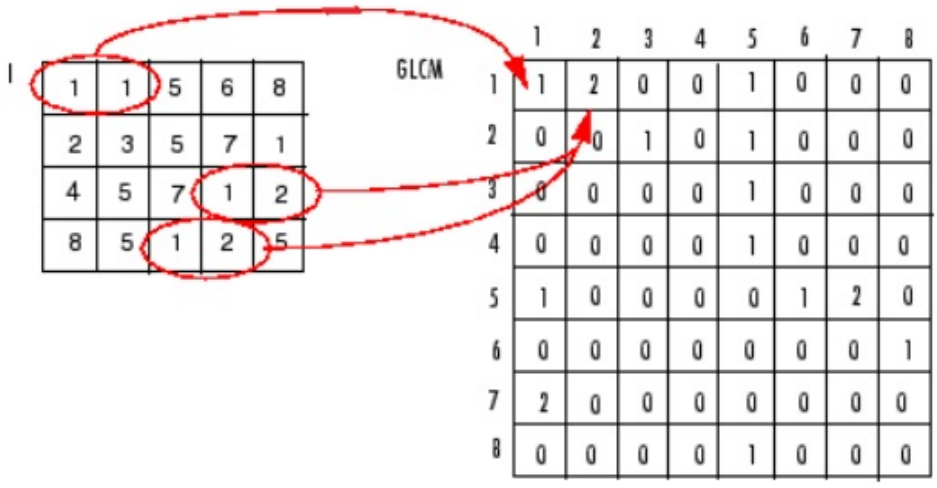

Figura 3.2 Processo de construção de uma matriz GLCM a partir da imagem. Imagem retirada de [21]

Haralick: Contraste, que reflete a quantidade de variação local de níveis de cinza em uma imagem. A representação matemática do contraste (ver equação 3.3) é obtida por:

$$\sum_{i,j=1}^{G} C_{ij}\,(i-j)^2\,, \tag{3.3}$$

onde G é a quantidade de níveis de cinza na imagem, e C_{ij} é a matriz GLCM na linha i e coluna j.

3.1.3 Sobel

O operador Sobel é usualmente utilizado em processamento de imagem para extração de bordas [31]. Essa extração é obtida a partir da convolução de dois kernels 3x3 com a imagem original - um para extrair "mudanças horizontais"e outra para extrair "mudanças verticais". A convolução (ver as operações 3.4 e 3.5) é demonstrada por:

$$\begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \begin{bmatrix} a_{11} & a_{12} & \dots \\ \vdots & \ddots & \\ a_{K1} & & a_{KK} \end{bmatrix} \tag{3.4}$$

$$\begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \begin{bmatrix} a_{11} & a_{12} & \dots \\ \vdots & \ddots & \\ a_{K1} & & a_{KK} \end{bmatrix} \tag{3.5}$$

O resultado da aplicação do filtro pode ser visto na Figura 3.3.

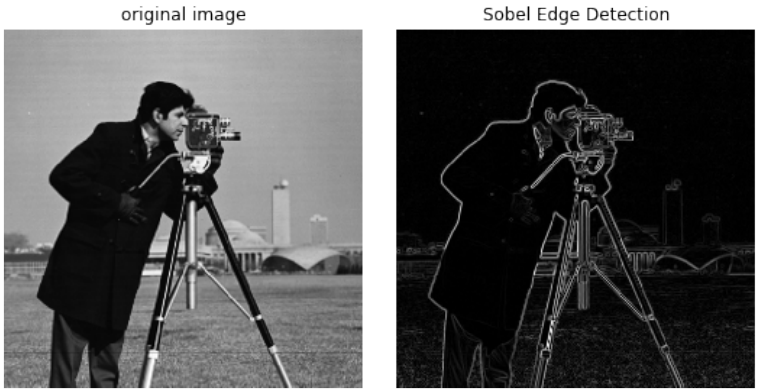

Figura 3.3 Exemplo da aplicação do filtro de Sobel. Imagem retirada de [28].

3.2 Prática

3.2.1 Objetivo

Nessa prática são apresentadas algumas técnicas de extração de *features*, com base nos descritores LBP, GLCM e Sobel. O objetivo é compreender o processo de extração e geração de um vetor de *features*, que será objeto de práticas posteriores.

3.2.2 Ferramentas

Ao longo dessa prática os seguintes materiais são necessários:

- Data sets de treino e teste gerados na prática anterior: equalização de histograma, correção gamma (3 versões), suavização e aguçamento.
- **Nota**: Para cada imagem do data set original, foram gerados 6 filtros na prática anterior (pré-processamento), os quais serão vetorizados pelos descritores LBP, GLCM (Constraste) e Sobel, e concatenados em um vetor de *features*. Caso ocorra "estouro"de memória, avalie a eventual necessidade de adotar estratégias para obter a extração das *features*.
- **Skimage package (*image processing in python*):**
  - skimage.feature.local_binary_pattern(image, n_points, radius, method) - Função para extração de *feature* via LBP.
  - skimage.feature.greycomatrix(image, distances, angles, levels, symmetric, normed) - Função para calcular a GLCM.

- skimage.feature.greycoprops(n_points, property) - Função para extração de *feature* de Haralick via GLCM.
- skimage.filters.sobel(image, mask) - Função para extração de *feature* via Sobel.

3.2.3 Roteiro I

1. Aplique o operador **LBP** nos data sets gerados anteriormente[1](treino e teste) para extrair as *features* de cada imagem. Em seguida, salve os vetores de *features* resultante em uma única matriz, de modo que o número de linhas da matriz corresponda ao número de imagens do data set, e as colunas sejam do tamanho do vetor de *features* da imagem.

2. Verifique a representação desta *feature* em formato de imagem. Para tanto, utilize 3 imagens do data set gerado na etapa de pré-processamento[2].

3.2.4 Roteiro II

1. Calcule a propriedade de **Contraste** proposto por **Haralick** por meio de **GLCM** nos data sets gerados anteriormente[1](treino e teste) para extrair as *features* de cada imagem. Em seguida, salve os vetores de *features* resultante em uma única matriz, de modo que o número de linhas da matriz corresponda ao número de imagens do data set, e as colunas sejam do tamanho do vetor de *features* da imagem;

2. Verifique a representação desta feature em formato de imagem. Para tanto, utilize 3 imagens do data set gerado na etapa de pré-processamento[2].

3.2.5 Roteiro III

1. Aplique o operador **Sobel** nos data sets gerados anteriormente[1](treino e teste) para extrair as *features* de cada imagem. Em seguida, salve os vetores de *features* resultante em uma única matriz, de modo que o número de linhas da matriz corresponda ao número de imagens do data set, e as colunas sejam do tamanho do vetor de *features* da imagem;

2. Verifique a representação desta feature em formato de imagem. Para tanto, utilize 3 imagens do data set gerado na etapa de pré-processamento[2].

[1] 10.000 imagens de treino e 10.000 imagens de teste para cada pré-processamento
[2] Escolha aleatória

3.2.6 Produtos

Ao final dessa prática, o aluno deverá ter em mãos os seguintes itens para a prática posterior:

- Salvar os vetores de *features* (treino e teste) referente à cada extração resultante dos métodos do LBP, GLCM (Contraste) e Sobel.

3.2.7 Relatório

A seguir, são apresentados alguns pontos que devem constar no relatório. Ressalva-se, entretanto, que o aluno não deve se sentir limitado a segui-los, sendo interessante expandir a discussão para além do que está sendo proposto. Seguem alguns itens:

- Analisar e discutir os aspectos visuais de cada procedimento de extração de *features* adotado ao longo dessa prática (LBP, GLCM e Sobel).

Capítulo 4

Classificação

4.1 Teoria

4.1.1 Classificação de padrões em imagem

Classificação de padrões (reconhecimento de padrões) é um problema de separação de espaço. O objetivo é separar os grupos de entradas similares de dados em categorias funcionais. Com relação a imagens, o problema é encontrar uma relação entre cada classe de interesse com os sub-conjuntos de pixeis da imagem que a representa.

Os sistemas de sensoriamento que utilizam sensores de visão são muito eficientes devido a alta densidade de informação que se pode extrair a partir de uma imagem. Entretanto, tal densidade também caracteriza a complexidade em se utilizar tais sistemas. Sob esse ponto de vista, para tais sistemas, um método adequado deve ser selecionado para a determinação dos padrões de imagens. Tal método deve ser escolhido a partir da análise do custo computacional e desempenho dos algoritmos envolvidos, bem como a tarefa a que se propõe.

A escolha do melhor método de classificação deve se baseada na determinação do padrão apropriado para a tarefa proposta e da análise dos métodos propostos para cada padrão. As seções a seguir discutem conceitos teóricos relativos a classificação de padrões em imagens e as abordagens mais utilizadas para essa classificação.

4.1.2 O que é um classificador?

Os classificadores são responsáveis pela separação do espaço de entrada. Esta separação, como apresentada na Figura 4.1, é realizada pelo classificador para determinar a classe de cada objeto a partir de suas *features*. O tipo de função que separa as entradas, que pode ser linear ou não-linear, é determinado normalmente por parâmetros do classificador. O número

de coordenadas no vetor de *features* é determinado pelos tipos extrator utilizado, devendo este número ser constante para todas as amostras de entrada

O processo de separação do espaço de entrada pode ser de dois tipos: supervisionado ou não supervisionado. Na separação supervisionada, durante o treinamento, as amostras de teste são acompanhadas de anotações indicando a classe real da amostra. Na separação não-supervisionada, o classificador deve inferir N divisões no grupo de dados a partir de relações entre *features* das amostras, sendo o número de divisões normalmente especificado pelo desenvolvedor.

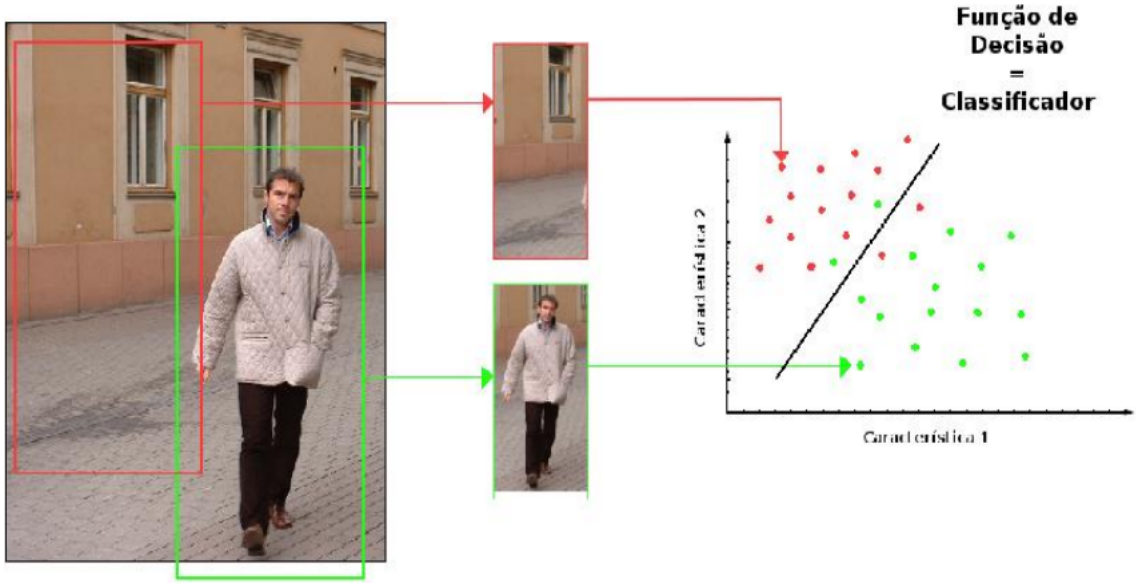

Figura 4.1 Exemplo de separação do espaço de entrada, onde uma função de decisão, também denominada de classificador, é utilizada para separar as classes de objetos, a partir dos valores das *features*. O classificador determinará a qual classe o objeto pertence. Vale notar que devido a imprecisão associada as *features* que foram extraídas, algumas entradas positivas foram marcadas como negativas e vice-versa.

Os classificadores aqui abordados fazem parte dos tipos supervisionados, descrevendo um tipo determinístico de classificação. A escolha de *features* relevantes para classificação é fundamental para permitir a correta separação dos dados. Além da determinação das *features*, as amostras de treinamento devem representar corretamente o objeto que se deseja modelar. Amostras com padrões de *features* muito dispersas dificultam a generalização e podem comprometer o treinamento. A generalização permite a separação de um espaço de entrada e melhor adaptação a situações não vistas na fase de treino.

Na separação do espaço de entrada, como na Figura 4.1, o classificador determina uma pontuação de classificação (score) entre as duas classes de interesse, sendo zero o limiar inferior padrão para a escolha de classes positivas. Cada amostra, ao ser avaliada então é

associada a uma pontuação, essa pontuação normalmente varia de $-\infty$ até $+\infty$, e a partir disso o tipo da classe amostrada é definido.

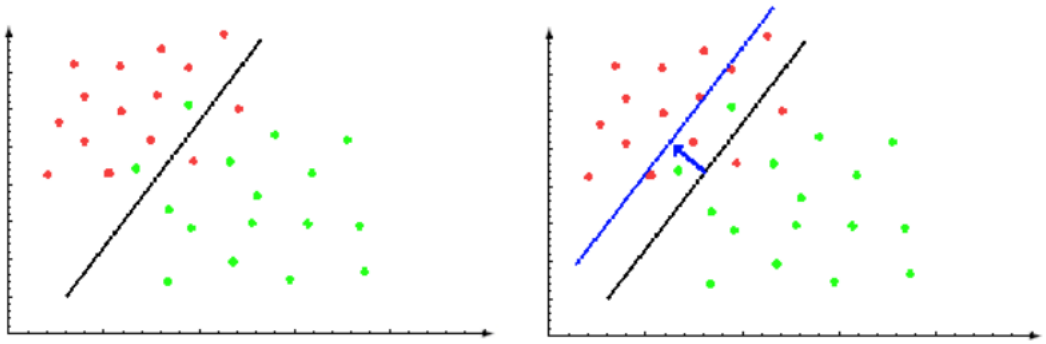

Figura 4.2 A figura demonstra o deslocamento do limiar de classificação de modo a modificar o balanço entre número de objetos detectados e os classificados incorretamente como positivo ou negativo. Isso pode tanto tornar o classificador mais conservador ou mais agressivo no momento de classificar os elementos de entrada como positivos ou negativos, a depender do limiar utilizado. Imagens retiradas e adaptadas de [7]

Com um valor definido para o limiar padrão, podemos alterá-lo para modificar o balanço entre o número de amostras positivas ou negativas encontradas e o número de objetos que são classificados incorretamente como parte de uma classe. Essa situação é demonstrada na Figura 4.2, onde o limiar de classificação é deslocado, por exemplo, de zero até quinze, e todas as amostras positivas passam a ser encontradas, ao custo de um maior número de amostras classificadas incorretamente como positivas (falso positivo).

4.1.3 RNA

Uma Rede Neural Artificial (RNA) é um classificador que possui um conjunto de unidades computacionais conectadas conforme uma arquitetura específica. Cada unidade consiste de um número de camadas de entrada, uma função de ativação e de um número de camadas de saída. A organizaçãoo das camadas de uma RNA define a forma como os neurônios da rede estão organizados, o que, por sua vez, define a arquitetura da rede. Uma RNA, em sua arquitetura básica, é formada por 2 camadas[1] : Uma camada oculta e outra camada de saída. Cada camada é composta por um conjunto de N neurônios definidos por funções de ativação. Quando as funções de ativação da camada oculta são não-lineares, os resultados produzidos são novamente combinados na últimda camada, de modo a produzir uma saída linear. Desta forma, a RNA funciona como um aproximador universal. A figura 4.3 ilustra a arquitetura de uma RNA com 3 camadas, sendo 2 camadas ocultas e 1 de saída.

[1]Segundo [14], o vetor de entrada não é considerado uma camada, pois não possui qualquer processo computacional

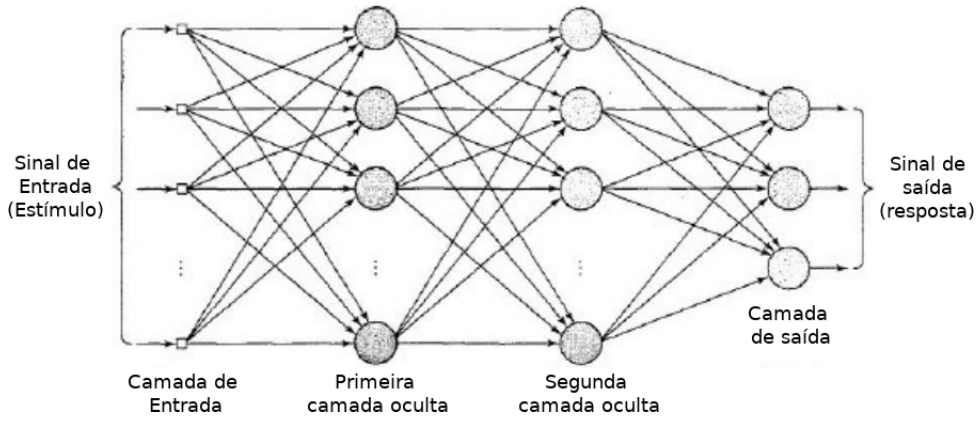

Figura 4.3 Exemplo de uma arquitetura de rede neural multicamada

Em geral, pode-se identificar três tipos de arquiteturas básicas para uma RNA: redes alimentadas adiante[2] com camada única, redes alimentadas adiante com múltiplas camadas (*Multi-layer Perceptron* - MLP) e redes recorrentes.

Algumas vantagens tornam a RNA, em suas vários formas, importantes no campo de detecção de padrões. Entre as *features* positivas, a aprendizagem ser feita a partir das amostras dos dados, ao invés de conhecimento especialista. Outro ponto benéfico é a capacidade de aproximar qualquer função não linear, a robustez a ruídos na amostra de dados e sua estrutura naturalmente paralela. Enquanto as principais desvantagens das RNAs são os longos tempos de treinamentos quando existem muitas amostras e a, susceptibilidade a overfitting.

4.1.4 SVM

Mesmo com as *features* positivas da RNA, o Máquina de Vetor de Suporte (*Support Vector Machine* - SVM), que por questões de escopo não será abordado com detalhes, é considerada por alguns autores [5][16] como um método superior de classificação. Essa aparente vantagem deriva da idéia que a SVM compartilha as vantagens do RNA, reduzindo, entretanto, as chances de ocorrência das desvantagens. Para separar o espaço de entrada, o SVM mapeia os elementos em uma dimensão maior, isso é feito com o intuito de encontrar um melhor plano de separação entre as classes.

As *features* da SVM são provenientes de sua origem híbrida, mistura entre uma abordagem estatística e determinística. Isto significa que, para encontrar as melhores hipóteses de classificação, a SVM não utiliza uma distribuição de probabilidade conhecida a priori, mas determina uma função de probabilidade a partir do espaço de entrada. o classificador SVM avalia a função de kernel $K(x_i, x_j)$, onde K, como uma função que recebe dois pontos x_i e x_j

[2]Do Inglês, *feed forward*

Tabela 4.1 Funções de kernel aplicadas no classificador SVM

Tipo de kernel	**Função** $K(x_i, x_j)$
Linear	${x_i}^T x_j$
Polynomial	$(\gamma({x_i}^T x_i) + r)^d$, onde γ denota gamma, r por coefθ e d por degree.
RBF	$exp(-\gamma\|\|x_i - x_j\|\|^2)$, onde γ refere gamma.
Sigmoid	$tanh(\gamma({x_i}^T x_j) + r)$, onde γ denota gamma e r é especificada por coefθ.

no espaço de entrada e computa o produto escalar desses dados usando o espaço de *features* multidimensional. Esta função no SVM aceita os seguintes kernels: linear, polinomial, RBF e sigmoid, listados na Tabela 4.1.

4.2 Prática

4.2.1 Objetivo

Nesta etapa, serão introduzidas as duas fases da classificação: treino e inferência. O objetivo é observar qual o desempenho da inferência do classificador após o treino com determinados parâmetros. Assim, espera-se reflexões sobre quais condições de treino o classificador apresentou melhor resultado, avaliando os aspectos referidos nas práticas anteriores.

4.2.2 Ferramentas

Estão disponibilizados para esta etapa funções para serem aplicadas nos classificadores SVM e MLP, bem como as métricas para serem utilizadas. Para cada um deles há parâmetros que podem ser alterados e podem gerar resultados diferentes (ajuste os parâmetros a seu critério). Ao longo dessa prática os seguintes materiais são necessários:

- Vetor de *features* dos data sets de treino e teste (junção) gerados na prática anterior;
- **Sklearn package (*machine learning in python*):**

- sklearn.svm.SVC(kernel, degree, gamma, coef0, C, epsilon) - Função para definir os parâmetros de cada kernel do classificador SVM.
- sklearn.neural_network.MLPClassifier(hidden_layer_sizes, activation, solver, alpha, batch_size, learning_rate, max_iter, momentum) - Função para definir os parâmetros do classificador MLP.
- sklearn.neural_network.MLPClassifier.score(X,y[, $sample_weight$]) - Função para calcular a acurácia do classificador MLP.
- sklearn.svm.SVC.fit(X,y) - Função para ajustar o modelo SVM de acordo com os dados de treinamento fornecidos.
- sklearn.svm.SVC.predict(X) - Função para realizar a classificação nas amostras.
- sklearn.svm.SVC.score(X, y) - Função para retornar a precisão média nos dados e rótulos de teste fornecidos.

4.2.3 Roteiro I

1. Realize o treinamento no classificador **SVM** com 3 modelos diferentes de classificação[3], sobre os vetores de *features* resultantes da prática anterior.
2. Com base nos 3 modelos de classificação, realize o procedimento de predição para cada classificador;
3. Com base no resultado das predições geradas no passo 2, calcule a acurácia de cada classificador. Note que para esse procedimento será utilizado o data set de teste.

4.2.4 Roteiro II

1. Realize o treinamento no classificador **RNA-MLP**[4] sobre os vetores de *features* resultantes da prática anterior;
2. Realize o procedimento de predição para classificação após o treinamento do passo 1, e calcule a acurácia de cada rede. Note que para esse procedimento será utilizado o data set de teste.

[3] A escolha dos parâmetros para cada kernel é de livre escolha
[4] A escolha do número de neurônios em cada camada é de livre escolha.

4.2.5 Produtos

Ao final dessa prática, o aluno deverá ter em mãos os seguintes itens (tanto para SVM quanto para RNA-MLP):

- Modelos de treinamentos de classificação;
- Resultados de inferências dos classificadores;
- Avaliação da inferência dos classificadores.

4.2.6 Relatório

A seguir, são apresentados alguns pontos que devem constar no relatório. Ressalva-se, entretanto, que o aluno não deve se sentir limitado a segui-los, sendo interessante expandir a discussão para além do que está sendo proposto. Seguem alguns itens:

- Analisar e discutir os resultados obtidos para cada classificador (tanto para SVM quanto para RNA-MLP), considerando o impacto dos ajustes de parâmetros durante o treinamento.

Capítulo 5

Análise de resultados

5.1 Teoria

Avaliar o desempenho de um classificador requer conhecimento das técnicas de avaliação e os benefícios obtidos com a utilização de cada uma delas. Essas técnicas são necessárias pois, um único número é incapaz de representar as capacidades de um sistema quando existe um balanço entre diferentes erros de classificação [20], isto é: qual o balanço entre o número de objetos encontrados que pertencem a classe buscada e o número de objetos que serão incorretamente classificados como parte desta classe?

O entendimento das técnicas depende da definição de certos conceitos básicos. Nesse sentido, seis variáveis são comumente usadas como base para definição dos métodos de análise de resultados, estas são:

- Positivos (**P**) - representa o conjunto de objetos que pertence a classe de interesse;
- Negativos (**N**) - representa o conjunto de objetos que não pertence a classe de interesse;
- Verdadeiro Positivo (**VP**) - representa uma classificação correta de um objeto que pertence a classe de interesse;
- Falso Positivo (**FP**) - representa uma classificação incorreta, onde um objeto que não pertence a classe de interesse foi marcado como pertencente;
- Verdadeiro Negativo (**VN**) – foi corretamente determinado que um objeto não pertencia a classe de interesse;
- Falso Negativo (**FN**) um objeto que pertence a classe de interesse não foi corretamente classificado como tal.

Neste documento, quando apresentadas em expressões matemáticas, as variáveis VP, FP, VN e FN representarão o número absoluto de classificações que atendem a definição da variável em questão. Quando os valores absolutos das variáveis VP, FP, VN, e FN são somados, temos o total de classificações realizadas. Além disso, as classificações corretas podem ser obtidas através de $VP+VN$ e as incorretas $FP+FN$. Podemos inferir o valor do total de amostras que pertencem a classe de interesse através de $P = VP+FN$, da mesma forma os que não pertencem são definidos tal que $N = FP+VN$.

A partir dos conceitos apresentados acima, alguns dos métodos de análise existentes na área de visão podem ser definidos. A partir destes métodos, espera- se facilitar a comparação de trabalhos com outros da área e a determinação da aplicabilidade em problemas reais.

5.1.1 Exatidão

Muitos trabalhos utilizam a medida de exatidão ("accuracy") para determinar o desempenho do classificador [25]. A exatidão define a relação entre o total de classificações corretas sobre o conjunto total de erros e acertos. A exatidão pode ser definida, matematicamente, como:

$$Exatidão = \frac{VP+VN}{VP+VN+FP+FN} \tag{5.1}$$

Mesmo quando a exatidão do classificador é a única preocupação, a utilização dela como medida única do desempenho pode tornar a comparação com outras soluções pouco significativas [25]. Dentre os problema inerentes a esta medida estão:

1. Muitos problemas possuem custos diferentes para erros e acertos, como em sistemas de empréstimos, onde conceder um empréstimo a um cliente inadimplente pode causar prejuízos financeiros, enquanto o contrário apenas não gera lucros;

2. A exatidão não leva em conta as possíveis diferenças de proporção entre positivos e negativos no conjunto de treinamento em relação ao conjunto real.

Alguns trabalhos indicam que o classificador com melhor exatidão, em geral, será o melhor classificador em todos os casos descritos acima. Entretanto, em [25], testes empíricos em diferentes datasets mostraram que, na maior parte dos casos, não houve um classificador que dominasse em desempenho ao longo de diferentes compromissos entre taxa de acerto e falsos positivos.

A partir dos argumentos apresentados, dado um problema onde não exista um classificador dominante, não existe uma métrica de apenas um número que determine com precisão o

melhor classificador. Assim, as técnicas apresentadas podem ajudar na representação do desempenho real de classificação em diferentes situações.

5.1.2 Receiver Operating Characteristic - ROC

A curva Receiver Operating Characteristic (ROC) determina a relação que existe entre a taxa de acerto (*hit-rate*) e falso alarme (*false positive*) [8]. Isso permite visualizar, organizar e selecionar classificadores quanto a seus respectivos desempenhos. A figura 5.1 (lado direito) exibe um gráfico da curva ROC, onde os pontos da curva são plotados na cor azul e provenientes das equações de taxa de acerto (5.2) e de taxa de falso alarme (5.3), são obtidas por:

$$taxa\ acerto = \frac{VP}{P} = \frac{VP}{VP+FN} \tag{5.2}$$

$$falso\ alarme = \frac{FP}{F} = \frac{FP}{FP+VN} \tag{5.3}$$

Além das características de pontos específicos, também podemos descrever o comportamento geral da curva. Nesse sentido, caso os valores produzidos por um classificador indiquem y = x, então pode-se inferir que para um problema de classificação binária, o classificador em questão produz resultados equivalentes a escolha aleatória de uma classe. Por outro lado, caso os valores produzidos indiquem a formação de uma curva convexa (y>x), então pode-se inferir que o classificador produz resultados melhores que uma escolha aleatória. Por último, se a curva traçada for concava (y<x), então pode-se inferir que o classificador possui um desempenho inferior a uma escolha aleatória.

Apesar das vantagens da comparação gráfica do desempenho, a possibilidade de representar o desempenho expresso pela curva ROC de uma maneira resumida facilita, por exemplo, a obtenção do desempenho geral de um classificador. Um dos métodos de resumir a curva ROC é dado a partir do cálculo da área abaixo da curva (AAC[1]). Em [12] o método é descrito como a probabilidade de que, a partir de uma amostra positiva e uma negativa, escolhidas aleatoriamente, a entrada positiva obtenha uma pontuação maior que a negativa.

O AAC, em [37], foi usada para determinar qual o melhor limite de confiança para avançar ao próximo nível de uma cascata de classificadores. A partir do resumo da curva ROC, o gráfico formado passou a ter duas dimensões, facilitando a compreensão da escolha do valor para o limite de confiança. Outra aplicação, também com o propósito de reduzir o

[1] Do Inglês, ***Area Under Roc***.

número de dimensões, foi feita por [36], para comparar o desempenho do classificador com diferentes proporções de imagens negativas e positivas durante o treino.

Além dos benefícios abordados acima, segundo [8], a curva ROC e os dados calculados a partir dela, diferente de medidas como exatidão e F-Score, são independentes dos seguintes fatores: distribuição entre positivos e negativos e mudanças relativas ao custo diferenciado para erros e acertos. Isso permite que variações nesses valores ao longo do tempo, isto é, não afetem a representação da curva de desempenho.

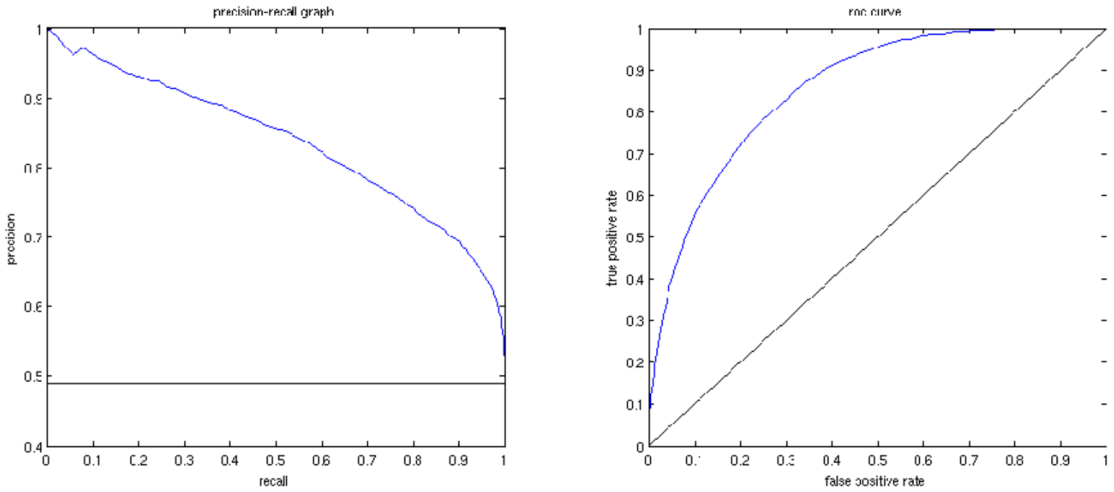

Figura 5.1 Comparação entre curva ROC e PR para um problema de classificação.

Apesar dos benefícios descritos, alguns trabalhos modificaram ou evitaram a utilização da curva ROC. Entre as modificações, a utilização do número absoluto de falso alarme por janela[2] no eixo x , como feito em [35], [24], [18], é realizada para facilitar a comparação de soluções diferentes. Outro problema, citado por [27], é a dificuldade de estender a curva para adaptá-la a problemas multi-classe.

5.1.3 Precision-Recall e F-Score

Em casos onde existe um enviesamento muito grande na distribuição de positivos e negativos, ou seja, uma predominância de elementos de uma certa classe, a curva ROC pode apresentar uma visão muito otimista do resultado [6]. Nesses casos, o Precision-Recall (PR) pode representar de maneira mais clara as diferenças de desempenho entre os algoritmos. Isso decorre, em parte, do PR ter se originado da área de processamento de informações. Nesta área, naturalmente, a quantidade de informações que não pertencem a classe de interesse é muito maior, como em uma busca por palavras na internet.

[2] O termo FPPW usado para sintetizar falso alarme por janelas deriva de False Positive Per Window (FPPW)

Para formar a curva PR, são utilizadas duas medidas distintas. Situada no eixo vertical, representando a taxa relativa ao número de objetos relevantes encontrados numa imagem (ou em um conjunto delas) dentre o total, a precisão:

$$precision = \frac{VP}{VP + FP} \tag{5.4}$$

No eixo horizontal, representando o conjunto de objetos que são relevantes numa imagem (ou em um conjunto de imagens) e que foram encontrados corretamente, o Recall (ou Taxa de acerto):

$$recall = \frac{VP}{P} = \frac{VP}{VP + FN} \tag{5.5}$$

Diferente de curvas ROC, os parâmetros que compõem a curva tem o melhor caso para o PR quando a curva tende ao eixo superior direito. As diferenças entre os algoritmos ROC e PR podem ser percebidas a partir da comparação apresentada na Figura 5.1. No exemplo apresentado, a curva ROC indica que o desempenho dos classificadores são próximos e ambos aceitáveis, enquanto na PR a impressão é que ainda existe muito espaço para melhoramentos. Além disso, os melhores resultados do Classificador 1 (“Algorithm 1”) em relação ao 2 (“Algorithm 2”) são mais claros no PR. Essas *features* são decorrentes da comparação do PR entre falso alarme e taxa de acerto, que melhor captura o efeito de um número muito superior de negativos [6].

A curva de Precision-Recall permite visualizar os diferentes balanços entre a precisão e a taxa de acerto. Entretanto, para determinar um valor escalar que representa a pontuação relativa a cada ponto do gráfico, a medida *F-score (F1-score)* pode ser utilizada. Essa medida representa uma média harmônica entre a precisão e a taxa de acerto, demonstrada por [26]:

$$F1 - score = 2 \cdot \frac{precision \cdot recall}{\cdot precision + recall} \tag{5.6}$$

Esta equação representa a média ponderada de *precision* e *recall*. Nesse caso, quanto maior a pontuação *F1-score* maior será as taxas de precision e recall.

5.2 Prática

5.2.1 Objetivo

A prática deve possibilitar um entendimento sobre as análises de resultados e suas formas de representação. Nesta etapa serão introduzidas as duas métricas de avaliação de resultado: ROC e *Precision-Recall*. Ao fim desta prática, espera-se reflexões sobre as análises e resultados obtidos.

5.2.2 Ferramentas

Para esta prática serão usados os seguintes recursos:

- Predição dos classificadores da prática anterior;
- sklearn.metrics.roc_curve(y_true, y_score) - Função para gerar a curva ROC;
- sklearn.metrics.precision_recall_curve(y_true, probas_pred) - Função para calcular a métrica *precision-recall*;
- sklearn.metrics.f1_score(y_true, y_pred) - Função para calcular a métrica F_1-*score*;
- sklearn.metrics.confusion_matrix(y_true, y_pred,) - Função para construir a Matriz de Confusão.

5.2.3 Roteiro I

1. A partir das informações de predição dos classificadores da prática anterior, calcule os valores da curva ROC;
2. Em seguida, gere um gráfico da curva ROC para cada classificador e salve as imagens. Estas imagens devem ser utilizadas no procedimento de análise no relatório;

5.2.4 Roteiro II

1. A partir das informações de predição dos classificadores da prática anterior, calcule os valores de *Precision* e *Recall* para cada classificador;
2. Em seguida, gere um gráfico de *Precision-Recall* para cada classificador e salve as imagens. Estas imagens devem ser utilizadas no procedimento de análise no relatório.

5.2.5 Roteiro-III

1. A partir das informações de predição dos classificadores da prática anterior, calcule os valores da métrica F_1-*score* para cada classificador.

5.2.6 Produtos

Ao final dessa prática, o aluno deverá ter em mãos os seguintes itens:

- Resultados das métricas ROC, *Precision-Recall* e F_1-*score* gerados.
- Imagens dos gráficos ROC e *Precision-Recall*.

5.2.7 Relatório

A seguir, são apresentados alguns pontos que devem constar no relatório. Ressalva-se, entretanto, que o aluno não deve se sentir limitado a segui-los, sendo interessante expandir a discussão para além do que está sendo proposto.

- Analisar os resultados obtidos pelas métricas;
- Discutir os resultados das classificações observando os gráficos;
- Avaliar e discutir eventuais impactos das etapas anteriores (pré-processamento e extração de features) na classificação a partir dos dados obtidos.

Capítulo 6

Redes de convolução

6.1 Teoria

Atualmente, as Redes Neurais Convolutivas (CNNs[1]) são amplamente utilizadas dentro das arquiteturas de redes neurais profundas, também conhecidas pelo termo *Deep learning* ou *deep network*. O interesse nas CNNs tomou impulso a partir da publicação da AlexNet em 2012 [17]. Desde então, diversos outros trabalhos foram publicados com base no conceito de uma CNN [38], [30], [33], [15], [9]. A principal vantagem do uso de uma CNN advém de sua capacidade intrínsica de detectar *features* importantes sem uma supervisão humana. Por exemplo, ao processar o dataset de gato e cachorro a rede pode aprender *features* distintivas de cada classe de forma direta, ou seja, sem necessitar dos diversos procedimentos vistos nas práticas anteriores (e.g. Pré-processamento e Extração de *features*). Além disso, ainda é considerada computacionalmente eficiente para realizar essa tarefa pois recorre às técnicas específicas para reduzir o espaço de entrada de dados a computar (e.g. convolução, *pooling*, compartilhamento de parâmetros). A figura 6.1 ilustra a arquitetura de uma CNN.

Uma arquitetura CNN é estruturada em camadas, cada uma aplicando uma execução específica e compartilhando *features* ao longo da rede. Essas camadas consistem em: convolução (*convolutional*), sub-amostragem(*subsampling*) e totalmente conectada (*fully connected*), como blocos de construção básicos [11]. Além disso, existem funções de ativação [2] aplicadas na arquitetura CNN, como Unidade Linear Retificada (ReLU), Sigmoid, Softmax, exponencial e etc. A arquitetura CNN pode ser particionada em duas etapas: **extração de *features*** e **classificação**. A etapa de extração de *features* é inicidada com o processamento de uma imagem bidimensional como entrada da rede, de modo que nas camadas posteriores é de fato realizado o procedimento gradativo de extração das *features*, através da convolução por

[1] Convolutional Neural Networks
[2] https://keras.io/activations/

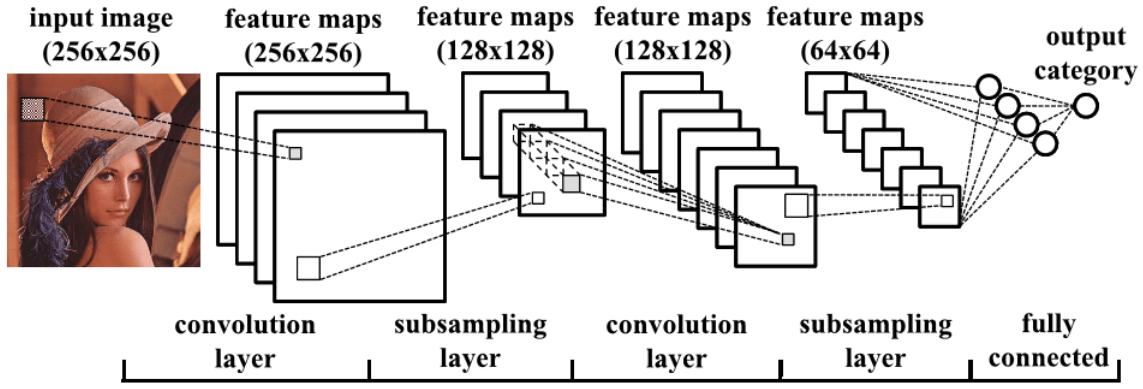

Figura 6.1 Exemplo de uma arquitetura CNN. Da esquerda para a direita: **input image** de Lena no espaço de cor RGB com dimensão 256 $\times$ 256 pixels é aplicada duas camadas de *convolutional* para extrair mapas de *features*, duas camadas de *subsampling*, e no final uma camada de *fully connected* contendo o vetor de *features* é classificado por um MLP. Imagem retirada de [1].

diferentes *kernels*. Cada *kernel* é definido como uma janela quadrada com valores de peso fixos, que serão convoluídos por toda a imagem de entrada. Ao final desse procedimento haverá uma nova matriz com a mesma dimensão da imagem processada - *feature map*. O número de mapas produzidos irá depender da quantidade de *kernels* utilizados para a extração das *features*. Em seguida, uma função não-linear é aplicada em cada um dos mapas de *features* produzidos nas camadas de convolução. Uma das funções não lineares chama-se ReLu, que evita problemas como a saturação de gradiente durante o treinamento. Após a função ReLu, é aplicada a camada *subsampling* para operações de amostragem, que também são realizadas para obter certo tipo de invariância de translação e de escala, e calcular os mapa de *features* em uma menor representação. Por fim, na etapa de classificação é utilizada uma ou mais camadas *fully connected*. Esta camada realiza uma classificação usando as *features* extraídas nas camadas anteriores. Isso permite que a rede profunda aprenda uma função que mapeia essa camada final com altos níveis de mapas de *features* para classificação binária ou multiclasse.

6.2 Prática

6.2.1 Objetivo

Nesta prática, será introduzida uma arquitetura CNN e suas variações de representação. O objetivo é avaliar o desempenho da arquitetura CNN em relação à arquitetura tradicional (pré-processamento, extração de feature e classificação) realizada nas práticas anteriores.

6.2.2 Ferramentas

Para esta prática serão usados os seguintes recursos:

- *Pets Dataset* contendo imagens anotadas de cães e gatos, num total de 10.000 imagens de treino e 10.000 imagens para teste.
- *MNIST database of handwritten digits* contendo 60.000 imagens com dimensão 28 $\times$ 28 em escala de cinza dos 10 dígitos, junto com um conjunto de teste de 10.000 imagens.
- **Keras package (*deep learning in python*):**
- (x_train, y_train), (x_test, y_test) = mnist.load_data() - Função para carregar o data set MNIST de dígitos manuscritos;
- keras.utils.to_categorical(y, num_classes=None, dtype='float32') - Função para converter um vetor de classe (inteiros) em matriz de classe binária;
- model = Sequential() - Função para gerar o modelo sequencial linear das camadas;
- model = add(layers, activation, input_shape) - Função para adicionar os parâmetros no modelo sequencial;
- model.compile(loss='type', optimizer='type', metrics='type') - Função de perda contendo parâmetros para compilar o modelo da rede;
- model.fit(x_train, y_train, batch_size= num, epochs= num, verbose=1, validation_data=(x_test, y_test)) - Função de ajuste para treinar o modelo da rede para evitar *overfitting*;
- score = model.evaluate(x_test, y_test, batch_size= num) - Função de avaliação sobre o modelo treinado para obter a pontuação de perda e precisão.
- preds = model.predict(x) - Função para gerar predições de saída para as amostras de entrada.

6.2.3 Roteiro I

1. Implemente uma arquitetura CNN usando o classificador MLP[1] sobre o *MNIST database of handwritten digits*, e calcule a média da acurácia;

[1] A escolha do número de camadas, função de ativação, otimizador, *batch size*, épocas e outros parâmetros são de livre escolha.

2. A partir da arquitetura CNN realizada no passo 1, plote o gráfico de acurácia (eixo y) no data set de treino e validação sobre épocas de treinamento (eixo x), e inclusive um gráfico de perda (eixo y) no data set de treino e validação sobre épocas (eixo x) de treinamento.

3. Repita o passo 2 para o data set de teste.

6.2.4 Roteiro II

1. Implemente uma arquitetura CNN usando o classificador MLP[1] sobre o data set *Pets Dog* (original), e calcule a média da acurácia, f1-score e precision;

2. A partir da arquitetura CNN realizada no passo 1, varie o número de *batch_size* e épocas, e execute a arquitetura CNN modificada calculando a média da acurácia, f1-score e precision;

3. Após a execução das duas arquiteturas CNN nos passos 1 e 2, opte pela arquitetura CNN vantajosa com base nas métricas avaliadas, e plote o gráfico de acurácia (eixo y) no data set de treino e validação sobre épocas de treinamento (eixo x), e inclusive um gráfico de perda (eixo y) no data set de treino e validação sobre épocas (eixo x) de treinamento.

4. Repita o passo 3 para o data set de teste.

6.2.5 Produtos

Ao final dessa prática, o aluno deverá ter em mãos os seguintes itens:

- Resultados das médias da acurácia, f1-score e precision;

- Imagens dos gráficos de acurácias e perdas com épocas para os data sets *MNIST e Pets Dog* para treino e teste.

6.2.6 Relatório

A seguir, são apresentados alguns pontos que devem constar no relatório. Ressalva-se, entretanto, que o aluno não deve se sentir limitado a segui-los, sendo interessante expandir a discussão para além do que está sendo proposto.

- Discutir os gráficos gerados de acurácias e perdas com épocas, nos roteiros I e II, utilizando os data sets *MNIST e Pets Dog* para treino e teste;

- Analisar criticamente os resultados obtidos e os parâmetros atribuídos nas arquiteruras CNN realizadas no Roteiro II nos passos 1 e 2.
- Exibir as duas arquiteturas CNN criadas nesta prática, sendo uma no roteiro I e outra no roteiro II (arquitetura CNN escolhida).

Referências Bibliográficas

[1] Academy, D. S. (2017). O que É visÃo computacional? [Online; accessed 27-setembro-2018].

[2] Alves, A. C. (2009). Joint bilateral upsample. [Online; accessed 30-agosto-2018].

[3] CCM (2017). A representação informática da cor. [Online; accessed 30-agosto-2018].

[4] CENTENO, M. D. (2009). Fundamentos de processamento de imagens. [Online; accessed 15-outubro-2011].

[5] Cristianini, N. and Shawe-Taylor, J. (2003). *An Introduction to Support Vector Machine an Other Kernel-based Learning Algorithms.* Cambridge Univ Press.

[6] Davis, J. and Goadrich, M. (2006). The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240. ACM.

[7] Duda, R., Hart, P., and Stork, D. (2001). *Pattern classification*, volume 2. Citeseer.

[8] Fawcett, T. (2006). An introduction to roc analysis. *Pattern Recogn. Lett.*, 27(8):861–874.

[9] Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587.

[10] Gonzalez, R. C., Woods, R. E., et al. (2002). Digital image processing.

[11] Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, G., Cai, J., and Chen, T. (2018). Recent advances in convolutional neural networks. *Pattern Recognition*, 77:354 – 377.

[12] Hanley, J. A., Barbara, P. D., McNeil, and Ph., D. I. M. M. B. (2005). The meaning and use of the area under a receiver operating characteristic (roc) curve.

[13] Haralick, R. M., Shanmugam, K., et al. (1973). Textural features for image classification. *IEEE Transactions on systems, man, and cybernetics*, (6):610–621.

[14] Haykin, S. (2001). *Redes Neurais: Princípios e Práticas*. Bookman, second edition.

[15] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[16] Kecman, V. (2001). *Learning and Soft Computing*. The MIT press.

[17] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

[18] Kruppa, H. and Schiele, B. (2003). Using local context to improve face detection. In *BMVC*, pages 1–10.

[19] Marques Filho, O. and Neto, H. V. (1999). *Processamento digital de imagens*. Brasport.

[20] Martin, A., Doddington, G., Kamm, T., Ordowski, M., and Przybocki, M. (1997). The det curve in assessment of detection task performance. pages 1895–1898.

[21] Mathworks, M. (1994-2018). Create gray-level co-occurrence matrix from image. [Online; accessed 01-setembro-2018].

[22] Ojala, T., Pietikäinen, M., and Harwood, D. (1996). A comparative study of texture measures with classification based on featured distributions. *Pattern recognition*, 29(1):51–59.

[23] Ojala, T., Pietikäinen, M., and Mäenpää, T. (2002). Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(7):971–987.

[24] Papageorgiou, C. and Poggio, T. (2000). A trainable system for object detection. *International journal of computer vision*, 38(1):15–33.

[25] Provost, F. J., Fawcett, T., and Kohavi, R. (1998). The case against accuracy estimation for comparing induction algorithms. In *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, pages 445–453, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[26] Rijsbergen, C. J. V. (1979). *Information Retrieval*. Butterworth-Heinemann, Newton, MA, USA, 2nd edition.

[27] Saffari, A., Leistner, C., and Bischof, H. (2009). Regularized multi-class semi-supervised boosting. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 967–974. IEEE.

[28] Shokrolah Shirazi, M. and Morris, B. (2015). Fast edge detection architecture using different levels of parallelism on a fpga. 113:1–8.

[29] Silva, M. (2013). Uso de matrizes de co-ocorrências para classificação automática de imagens de cana-de-açúcar. *Monografia de Graduação em Ciência da Computação, Universidade Federal de Lavras, Minas Gerais, Brasil*.

[30] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

[31] Sobel, I. (2014). An isotropic 3x3 image gradient operator.

[32] Souza, L. O. d. O. (2016). Um estudo sistemático sobre detecção de impostor facial.

[33] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9.

[34] van de Sande, K. E. A., Gevers, T., and Snoek, C. G. M. (2010). Evaluating color descriptors for object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1582–1596.

[35] Viola, P. and Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–I. IEEE.

[36] Wang, G., Hoiem, D., and Forsyth, D. (2009). Building text features for object image classification. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 1367–1374. IEEE.

[37] Wolf, L. and Bileschi, S. (2006). A critical view of context. *International Journal of Computer Vision*, 69(2):251–261.

[38] Zeiler, M. D. and Fergus, R. (2014). Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer.