UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA ELÉTRICA
ENGENHARIA ELETRÔNICA E DE TELECOMUNICAÇÕES
CAMPUS PATOS DE MINAS

FÁBIO CAMPOS FERREIRA

APLICATIVO CAPAZ DE RECONHECER CÉDULAS MONETÁRIAS PARA DEFICIENTES VISUAIS

Patos de Minas - MG

2023

FÁBIO CAMPOS FERREIRA

APLICATIVO CAPAZ DE RECONHECER CÉDULAS MONETÁRIAS PARA DEFICIENTES VISUAIS

Trabalho de conclusão de curso apresentado à Faculdade de Engenharia Elétrica, da Universidade Federal de Uberlândia, Campus Patos de Minas, Minas Gerais, como requisito parcial à obtenção da graduação em Engenharia Eletrônica e de Telecomunicações.
Orientador: Prof.ª Dr.ª Eliana Pantaleão

Patos de Minas - MG
2023

FÁBIO CAMPOS FERREIRA

APLICATIVO CAPAZ DE RECONHECER CÉDULAS MONETÁRIAS PARA DEFICIENTES VISUAIS

Trabalho de conclusão de curso apresentado à Faculdade de Engenharia Elétrica, da Universidade Federal de Uberlândia, Campus Patos de Minas, Minas Gerais, como requisito parcial à obtenção da graduação em Engenharia Eletrônica e de Telecomunicações.

Patos de Minas - MG, 06 de fevereiro de 2023:

Prof.ª Dr.ª Eliana Pantaleão
Orientador

Prof. Dr. Pedro Luiz Lima Bertarini – FEELT/UFU
Examinador

Prof. Dr. Júlio Cezar Coelho – FEELT/UFU
Examinador

Dedico este trabalho a minha família, em especial a minha mãe Maria José Campos Ferreira, e as amizades que criei na universidade.

Agradecimentos

Agradeço primeiramente a Deus, por ter me guiado pelos caminhos certos desde o início da minha vida. A minha família, em especial minha mãe, por sempre ter me incentivado aos estudos.

A todo o corpo docente que contribuiu diretamente na minha formação profissional, por todo conhecimento doado, motivação e desafios que ajudaram no meu desenvolvimento.

E por fim agradeço também a Universidade Federal de Uberlândia, por toda a estrutura dada a minha educação e formação, como também por fazer o intermédio de diversas amizades, que acredito que levarei para vida inteira.

"Por isso não tema, pois estou com você; não tenha medo, pois sou o seu Deus. Eu o fortalecerei e o ajudarei; Eu o segurarei com a minha mão direita vitoriosa."

Isaías 41:10

Resumo

Este trabalho descreve o desenvolvimento de um aplicativo para facilitar a vida de uma pessoa que sofre de deficiência visual. O aplicativo consegue reconhecer o valor de uma cédula monetária e informá-lo ao usuário, dando-lhe maior liberdade e confiança nas suas transações econômicas. São usadas técnicas de reconhecimento de objetos na imagem proveniente câmera do aparelho celular, identificando o valor da nota. O aplicativo tem uma *interface* sonora, descrevendo de forma audível o valor da nota que ele tem em mãos. A sua construção se deve principalmente à utilização das ferramentas Python, OpenCV e Kivy. Foram realizados diversos testes comparativos com os classificadores SVM, KNN e MLP para escolher a melhor técnica a se usada no produto final. Dos resultados obtidos pelo aplicativo se viu uma boa acurácia de 87,62%, permitindo a sua utilização para diferenciar as notas.

Palavras-chave: Cédulas monetárias. Python. Deficiente visual. Reconhecimento de imagens. Aplicativo.

Abstract

This paper describes the development of an application to facilitate the life of a visually impaired person. The application can recognize the value of a banknote and inform the user about it, giving him more freedom and confidence in his economic transactions. Object recognition techniques are used in the image provided by the cell phone camera, identifying the value of the banknote. The application has a sound *interface*, audibly describing the value of the banknote he is holding. Its construction is mainly due to the use of Python, OpenCV and Kivy tools. Several comparative tests were performed with SVM, KNN and MLP classifiers to choose the best technique to be used in the final product. From the results obtained by the application a good accuracy of 87.62% was seen, allowing its use to differentiate the notes.

Keywords: Banknote. Python. Visually impaired. Image recognition. Application.

Lista de ilustrações

Lista de códigos

Lista de quadros

Lista de tabelas

Lista de abreviaturas e siglas

ANN	Rede Neural Artificial (*Artificial Neural Network*)
DAG	Grafo Direcionado Acíclico (*Directed Acyclic Graph*)
KNN	k-Vizinhos mais próximos (*k-Nearest Neighbors*)
ML	Aprendizado de Máquina (*Machine Learning*)
MLP	Redes Neurais Multicamadas (*Multi-layer Perceptron*)
RBF	Função de base radial (*Radial Basis Function*)
SVM	Máquina de Vetores de Suporte (*Support Vector Machine*)
SIFT	Transformação de Características Invariantes à Escala (*Scale-Invariant Feature Transform*)

Lista de símbolos

$f\,(x,y)$	Imagem bidimensional onde (x,y) são as coordenadas de um *pixel*.
R	Intensidade da fração vermelha de uma cor no padrão RGB
G	Intensidade da fração verde de uma cor no padrão RGB
B	Intensidade da fração azul de uma cor no padrão RGB
H	Intensidade da matiz de uma cor no padrão HSV
S	Intensidade da saturação de uma cor no padrão HSV
V	Intensidade da luminância de uma cor no padrão HSV
$g\,(x,y)$	Filtro Gaussiana
σ	Desvio padrão entre os pontos distantes de um raio predefinido, usado no filtro Gaussiano
$M\,(x,y)$	Filtro mediana
$L\,(x,y)$	Função de limiarização
l	Limiar da função de limiarização
G_c	Magnitude do gradiente do filtro Canny
G_θ	Ângulo do gradiente do filtro Canny
$p_k\,(f)$	Função de histograma de uma imagem f
$h'\,(k')$	Função da equalização de histograma
$S_\theta\,(x,y)$	Ângulo dos pontos detectados pelo SIFT
$S\,(x,y)$	Magnitude dos pontos detectados pelo SIFT
$\{\omega_1,\omega_2,\omega_n\}$	Conjuntos de classes presentes na fase de treinamento
$\{X_1,X_2,\ldots,X_j\}$	Entrada dos vetores de características
$\{Y_1,Y_2,\ldots,Y_j\}$	Classes de indicação de cada vetor de características

$\boldsymbol{a}$	Vetor de parâmetro da região de fronteira do SVM
$\boldsymbol{a}_o$	Melhor valor para o vetor $\boldsymbol{a}$
b	Parâmetro da região de fronteira do SVM
b_o	Melhor valor para o parâmetro b
Δ	Distância entre os vetores das extremidades de cada classes do SVM binário
C	Parâmetro de regulação do SVM de classificação linear de margem suave
α	Multiplicador lagrangiano usado no SVM
$K\,(X_i,X)$	Função *kernel*
ξ	Variável de folga não negativa usada no SVM de classificação linear de margem suave
ϕ	Função de transformação não linear usada no SVM de classificação não linear
c	Parâmetro da função *kernel* Polinomial
$degree$	Parâmetro da função *kernel* Polinomial
γ	Parâmetro das funções *kernel* Polinomial, RBF, Sigmóide e CHI2 do SVM
ρ	Peso a ser aplicado a cada característica no ANN
α	Parâmetro das funções de ativação Sigmoid Simétrica, Gaussiana e Leaky ReLU
β	Parâmetro das funções de ativação Sigmoid Simétrica e Gaussiana

Sumário

1 Introdução

A visão é o sentido mais avançado do ser humano. Ela possibilita a compreensão de imagens que contêm a representação de objetos, processos e várias outras informações. A ação de projetar computadores capazes de converter as imagens em informações pretende simular a visão humana. Isso possibilita à máquina realizar um aprendizado para fazer inferências e agir com base nas informações visuais. Este processo pertence ao ramo de estudo da Ciência da Computação denotada como inteligência artificial (1). A importância do estudo das técnicas de processamento digital de imagens pode ser vista na grande contribuição desta área de pesquisa para resolver problemas na medicina, biologia, automação industrial, sensoriamento remoto, astronomia, microscopia, artes, área militar, arqueologia, segurança e vigilância (2).

A área de reconhecimento de padrões (ou objetos) possibilita que o computador consiga reconhecer várias categorias de objetos do mundo físico e abstrato, como o rosto de uma pessoa, uma mesa ou uma nota musical, por exemplo. O reconhecimento é possível mesmo se estes objetos não estiverem em um estado ideal, ou seja, apresentarem-se distorcidos ou incompletos. As técnicas de reconhecimento de padrões são divididas em duas partes: a primeira é o aprendizado que a máquina tem de realizar para definir um objeto; a segunda é a decisão que ela tem de realizar para classificar um tipo de objeto sobre o qual ela já adquiriu conhecimento na primeira parte. Estes conhecimentos são usados nos campos do reconhecimento de fala automática, da biomedicina, da impressão digital, da agricultura, da geologia, na melhora de processos industriais, na área militar, da robótica e da inteligência artificial (3).

1.1 Tema do Projeto

Este projeto se baseia no objetivo de facilitar a vida de uma pessoa com deficiência visual, permitindo-lhe usar um aplicativo apto para reconhecer o valor de uma cédula monetária e informá-lo ao deficiente visual, dando ao mesmo maior liberdade e confiança nas suas transações econômicas.

A identificação das cédulas monetárias é uma tarefa que deve ser realizada por algoritmos dedicados à área de reconhecimento de objetos, denominados classificadores. Entre esses, podemos citar a Rede Neural Artificial (ANN), o k-Vizinhos Mais Próximos

(KNN) e a Máquina de Vetores de Suporte (SVM). Esses classificadores apresentam-se já implementados em diversas linguagens de programação como Python, C++, R, entre outras. Isso permite o desenvolvimento de um algoritmo aprimorado para o reconhecimento de cédulas monetárias e a sua incorporação dentro de um aplicativo para celulares.

O processo de desenvolvimento deste aplicativo envolve várias etapas de estudos e processos. O aplicativo tem capacidade de utilizar uma tecnologia baseada no reconhecimento de objetos aplicada à imagem proveniente da câmera do aparelho celular, reconhecendo o valor da nota. É utilizada uma biblioteca do Python que converte texto para fala permitindo assim, ao deficiente visual, saber exatamente qual o valor da nota que ele tem em mãos.

1.2 Problematização

O sentido mais usado pelo ser humano é a visão. Quando começamos o dia, a primeira coisa que fazemos é abrir os olhos para nos localizarmos no ambiente e notar quais objetos que o compõem. Assim, podemos obter rapidamente objetos presentes no ambiente que estamos necessitando ou simplesmente encontrar a saída do local, indo em direção ao que queremos. Este processo é realizado em nossa casa, no meio público ou no trabalho (4). A capacidade de visualizar o que nos cerca nos dá, também, uma capacidade autônoma de realizarmos tarefas pertencentes a nossa vida em sociedade, como trabalhar, gerir os bens pessoais, requisitar serviços de outras pessoas e muitas outras. Porém, os deficientes visuais são submetidos a uma dificuldade extra na realização destas tarefas. Muitas vezes eles têm que recorrer a um familiar ou amigo para acompanhá-lo na sua jornada por um mundo que não foi feito para eles.

As revoluções tecnológicas como computadores, *notebooks* e celulares a princípio parecem ser mais uma parte da vida de um ser humano com visão à qual os deficientes visuais não teriam acesso. Contudo, muitas pessoas e instituições tentam superar as limitações físicas dos deficientes e criar ferramentas para que os mesmos consigam acessar essas tecnologias e sejam incluídos na sociedade. Uma dessas instituições é a Google, que criou uma ferramenta que possibilita pessoas com pouca visão, ou nenhuma, usarem aparelhos celulares. O TalkBack é um leitor de tela da Google para dispositivos Android. Ele informa ao usuário todas as informações contidas na tela, para possibilitá-lo a usar o aparelho, mesmo sem conseguir ver as informações gráficas (5). Em aplicativos, se os mesmos tiverem compatibilidade, o aplicativo da Google também consegue interpretar o

seu ambiente para o usuário (6).

Visando auxiliar nas atividades cotidianas de pessoas com deficiência visual, pensou-se na elaboração de um aplicativo que irá ajudar na identificação de cédulas monetárias para deficientes visuais. Já existem no mercado os aplicativos Cash Reader e LetSeeApp com a função de identificar as notas monetárias de vários países, incluindo o Real brasileiro. Contudo, eles não são gratuitos. O Cash Reader apresenta uma versão de teste que permite somente classificar notas de R$ 2,00 e R$ 5,00. O desenvolvedor deste aplicativo descreve que na construção do Cash Reader foram utilizadas ferramentas de reconhecimento de objetos, como o Tensorflow (7). Já o LetSeeApp disponibiliza gratuitamente somente a identificação de notas de três países. As outras, como o Real, devem ser compradas. O diferencial deste aplicativo é a presença de um sensor de luz que ajuda o deficiente a encontrar lampadas e janelas, visando melhorar a leitura das notas. Ele também disponibiliza um sistema de cadastramento de cartão bancário, permitindo ao usuário identificar o cartão correto no momento do pagamento. O desenvolvedor do LetSeeApp comenta que, para utilizar seu aplicativo, é necessário desenvolver uma prática e que os desenvolvedores não assumem a responsabilidade por erros que causem prejuízo ao usuário (8). Pode-se citar também outros aplicativos com função semelhante, como o Qatari Money Reader que reconhece a moeda do Catar, MCT Money Reader que não apresenta o real brasileiro e o Banknote Identifier que não realiza leitura dinâmica, não é adaptado para uso de deficientes visuais, necessita de uma conexão a *internet* e não tem o Real (9) (10) (11).

O Banco Central do Brasil disponibiliza o aplicativo Dinheiro Brasileiro que reconhece notas de Real e identifica itens de segurança das cédulas. No entanto, não é um aplicativo com recursos de acessibilidade que permitam o uso por deficientes visuais. Sua *interface* apresenta botões pequenos, não reproduz em forma sonora o valor da nota e só reconhece o lado que contém o animal (12).

Também foi encontrado um Trabalho de Conclusão de Curso realizado pelo aluno Leonardo Maffei da PUC-Campinas com a mesma temática. Neste trabalho foi criado o aplicativo Blind para reconhecer notas de Real, projetado para dispositivos *IOS* (13). No entanto, no início de 2017, o iPhone participava de 4,7% do mercado de celulares móveis no Brasil. Já o Android dominava 90,4% do mercado (14). Ou seja, esta opção é inacessível a grande parte da população.

1.3 Hipóteses

As hipóteses deste trabalho são:

- Com a criação de um aplicativo para celulares Android utilizando a linguagem de programação Python é possível utilizar um classificador que irá receber uma foto capturada pela câmera e retornar o resultado da classificação. O usuário obterá este resultado por meio de voz voz reproduzida pelo celular;
- A análise de diversos testes, modificando os classificadores, características usadas e opções de tratamento da imagem conduzirão à melhor montagem possível do sistema de classificação para a diferenciação das cédulas de Real;
- As características extraídas da imagens irão gerar um classificador com percentuais de acertos que garantam uma boa utilização do aplicativo.

1.4 Objetivos

1.4.1 Objetivos Gerais

O objetivo deste projeto de pesquisa é desenvolver um aplicativo com a função de identificar cédulas da moeda brasileira, usando algoritmos para reconhecimento de objetos com a linguagem de programação Python.

1.4.2 Objetivos Específicos

- Conseguir identificar e extrair as principais características das cédulas monetárias para garantir uma boa classificação;
- Realizar a segmentação da nota, em primeiro plano, do fundo da imagem;
- Realizar diversos experimentos com configurações variadas, para conseguir determinar qual o melhor classificador (ANN, KNN ou SVM) para esta aplicação;
- Criar um aplicativo contendo o código do classificador treinado;
- Usar uma imagem da cédula, capturada pela câmera do celular, para testar o aplicativo.

1.5 Justificativas

As cédulas de real já possuem algumas características que facilitam a seu manuseio pelos deficientes visuais, como os tamanhos diversificados para cada valor, que permitem o seu reconhecimento só pelo tato. Entretanto, precisa-se de certa habilidade e treino dos deficientes visuais para conseguir fazer esse reconhecimento de forma eficiente (15). A criação deste aplicativo, que consiga realizar a identificação dos valores da cédula, quase sem custo para o deficiente, pode se tornar uma ferramenta de acessibilidade muito útil, possibilitando um processo de reconhecimento mais rápido, independente e seguro, o que daria ao usuário um conforto nas suas negociações comerciais. As tecnologias em que o aplicativo se baseia estão basicamente nas áreas de processamento digital de imagens e reconhecimento de objetos. Essas áreas de pesquisa são de grande importância na computação, devido às suas características de adaptação a uma infinidade de aplicações e aos excelentes resultados, que não podem ser atingidos usando outras técnicas.

1.6 Organização do Trabalho

Este trabalho está organizado da seguinte forma. O Capítulo 2 contém o referencial teórico utilizado como base para a elaboração deste projeto, apresentando os recursos de programação como a linguagem Python, usada para a criação dos algoritmos de treinamento e classificação, incorporados no aplicativo, também construído usando esta mesma linguagem. Também apresenta uma visão descritiva sobre as principais técnicas de reconhecimento de objetos utilizadas no projeto, o SVM, KNN e ANN, com a definição das principais métricas usadas para pontuá-los.

O Capítulo 3 apresenta a metodologia com a descrição das principais etapas do projeto: o estudo na área de aprendizado de máquina, das técnicas de processamento de imagem, dos algoritmos de reconhecimento de padrões, e a realização dos testes comparativos dos classificadores. Também apresenta uma descrição dos recursos utilizados na criação do aplicativo.

No Capítulo 4 estão descritas as etapas do projeto que foram realizadas, como: a construção do banco de dados, o estudo de vários tipos de processamento de imagem aplicados nas notas monetárias e a descrições dos testes realizados com a combinação dos processamentos de imagem e formas de extrair características das imagens.

O Capítulo 5 apresenta uma visão dos resultados obtidos para cada configuração e característica testada em cada um dos três classificadores propostos. Para o melhor

classificador, foram gerados resultados da variação dos seus parâmetros de configuração. Por último, são expostos os resultados gerados diretamente pelo aplicativo apresentando a melhor configuração de processamento, melhor classificador com seus melhores parâmetros.

O Capítulo 6 contém as considerações finais sobre o aplicativo desenvolvido e a utilização das técnicas de reconhecimento de objetos, discussão sobre os resultados finais, bem como propostas para a continuidade do trabalho e melhorias no aplicativo.

1.7 Considerações Finais

Deficientes visuais apresentam várias dificuldades na sua vida. A manipulação do dinheiro é uma delas, pois não ter a confiança de saber qual o valor da nota que esta entregando, ou se o troco está certo pode gerar grandes preocupações e reduzir a sensação de independência financeira e social.

Existem algumas tecnologias que ajudam a deficientes visuais a identificar o valor das notas, como a diferença de tamanho. Contudo, não é um processo fácil de se realizar, é necessário uma certa prática e ter gravado em sua memória qual a diferença de tamanho entre uma nota de R$ 2,00 e R$ 5,00, por exemplo.

Com o objetivo de ajudar estas pessoas, foi proposta a criação de um aplicativo para celulares Android que consiga capturar uma foto de qualquer cédula, identificá-la e retornar seu valor para o usuário. Assim, os deficientes visuais poderão ter a certeza do valor da notas que estão entregando ou recebendo durante suas transações econômicas.

2 Fundamentação Teórica

As áreas de processamento de imagens e inteligência artificial estão diretamente relacionadas e são de grande importância nas diversas atividades humanas, tornando-as mais eficientes e acessíveis. No processo de desenvolvimento do aplicativo, elas representam a base de seu funcionamento, possibilitando interpretar a imagem da cédula e definir o seu valor.

Para que seja possível desenvolver a aplicação que identifica cédulas monetárias, foi necessário entender conceitos e fundamentos por trás de cada uma das ferramentas utilizadas no projeto. Dentre os recursos utilizados, os principais se concentram na linguagem de programação utilizada, nos métodos de manipulação de imagens digitais, procedimentos de obtenção de atributos responsáveis por diferenciar os grupos de imagens entre si, métodos pertencentes ao estudo referente ao reconhecimento de objetos e finalmente a possibilidade de incorporar todos os passos anteriores em um sistema Android.

A escolha da linguagem de programação foi muito importante durante o início do projeto, visto que é fundamental para a utilização dos outros recursos, como bibliotecas presentes na linguagem e que foram construídas em cima da teoria estudada. Por conseguinte, Python foi a escolha feita devido a sua potencialidade em executar de maneira satisfatória as principais técnicas estudadas nesse projeto.

Partindo da linguagem Python, o estudo e escolha dos principais mecanismos presentes no tema de processamento digital de imagens utilizados se reúnem nas propriedades da imagem. A cor é normalmente analisada utilizando um histograma, ou seja, um gráfico distributivo de cores. A textura é outra característica muito utilizada e efetiva na diferenciação de imagens. Adicionalmente, ainda se tem a possibilidade de utilizar o contorno de objetos contidos nas imagens. Na etapa de processamento da imagem, a escolha de qual propriedade utilizar, e quais técnicas serão utilizadas, pode ser decisiva para o processo de classificação (16).

Desta forma, antes de realizar a classificação, é necessário obter da imagem estas características. Todavia, extrair estas informações da imagem sem nenhum tratamento precedente pode ocasionar alguns erros, ou pouca caracterização dos valores esperados. Para evitar estes problemas e melhorar a resposta obtida, a utilização de filtros é indicada, pois eles auxiliam a remover ruídos e aumentar a distância dos valores que representam as características desejadas. Dentro dos tipos de filtros de imagens utilizados ou estudados

durante este projeto, pode-se citar o Gaussiano, Gabor e o Blur.

2.1 Processamento de Imagens

Antes de realizar o reconhecimento das cédulas, primeiramente, é necessário extrair uma informação da imagem que seja coerente com o esperado, tendo pouca variação entre objetos do mesmo tipo, neste caso, as notas de mesmo valor, para minimizar o erro da classificação. Desta forma, é requerido um pré-processamento da imagem capturada diretamente da câmera do celular, que vise minimizar interferências externas como a iluminação, qualidade de preservação do objeto, dimensionamentos e formatos diferentes gerados por dispositivos diversos.

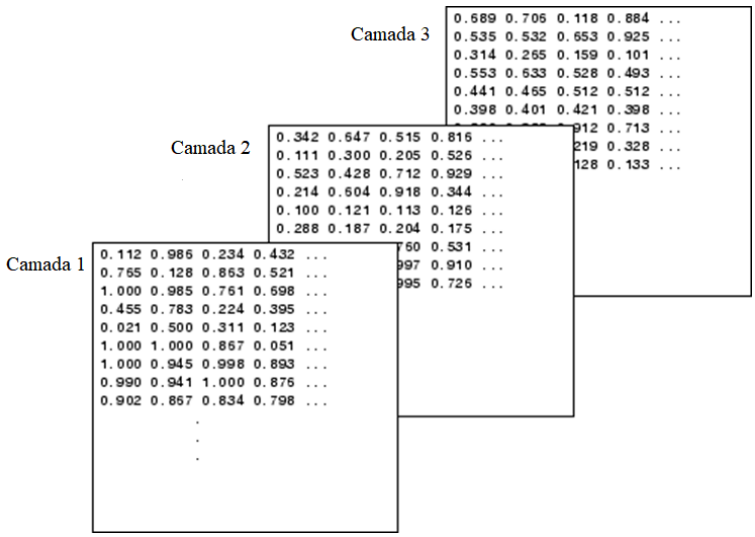
Uma imagem $f(x,y)$ é representada matematicamente como uma matriz de comprimento x e largura y, onde cada coordenada representa um *pixel* de valor que pode variar de 0 a 255 de intensidade para imagens baseadas em um sistema de 8 bits de armazenamento. A redução dos níveis de intensidade ou das dimensões (x,y), resolução, provoca uma perda de detalhamento e qualidade. O ponto (0,0) da matriz é localizado na parte superior esquerda da imagem (2).

O olho humano é capaz de reconhecer milhares de tons e intensidades de cores. Porém, para reproduzi-las em vídeo, são utilizadas combinações das três cores primárias: o verde, o vermelho e o azul. Para distinguir uma cor de outra são utilizadas 3 características: a intensidade luminosa ou brilho; a matiz, que é a graduação ou nuança da cor; e a saturação, também chamada de crominância.

Os diferentes formatos de imagens digitais tendem a diferenciar-se no modelo de quantizar as variáveis que formam uma imagem. Os principais tendem a representar cada *pixel* como um valor tridimensional, com sua cor final definida pela combinação destes três valores. A Figura 2.1 é a exemplificação da distribuição dos valores de uma cor com três camadas. Os formatos de imagem utilizados neste trabalho foram o RGB (*red*, *green*, *blue*), Tons de Cinza e HSV (*hue*, *saturation*, *value*).

Dada uma imagem, é comum aplicar algum tipo de filtragem ou processamento, tendo o propósito de modificá-la, minimizando ruídos e realçando certas características, para obter uma melhor resposta visando a meta final. Neste projeto foi utilizado para processar a imagem os filtros e processamento o Gaussiano, Mediana, Bilateral, Equalização de Histograma, K-Means, Limiarização, Canny e Morfológico de Fechamento.

Figura 2.1 – Representação matricial de uma imagem de três camadas.

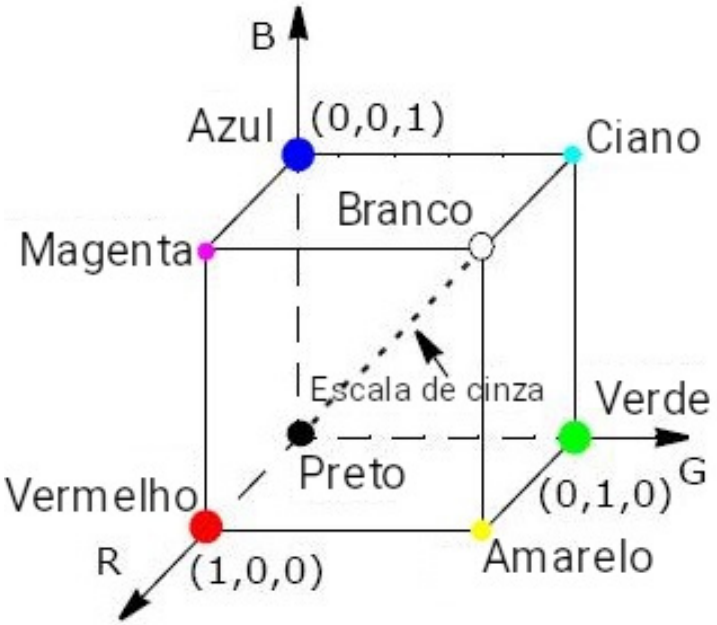


Fonte: adaptado de (17).

2.1.1 Modelo RGB

O modelo RGB é um sistema tridimensional, em que o primeiro valor refere-se à intensidade do vermelho, o segundo do verde e o terceiro, da cor azul. A combinação destas três intensidades formam a cor final do *pixel*. Quando um *pixel* apresenta o mesmo valor nas três camadas, temos a resposta de uma escala em cinza, cujo valor 0 resulta na cor preta e 255, na branca. Por isso, imagens em cinza podem ser trabalhadas digitalmente pela manipulação de somente de uma única camada, facilitando as operações e o seu entendimento (2). A Figura 2.2 representa o modelo RGB, onde cada eixo tridimensional apresenta a intensidade de cada cor e a diagonal o nível cinza, em escala unitária.

Figura 2.2 – Modelo RGB.



Fonte: adaptado de (17).

2.1.2 Modelo Tons de Cinza

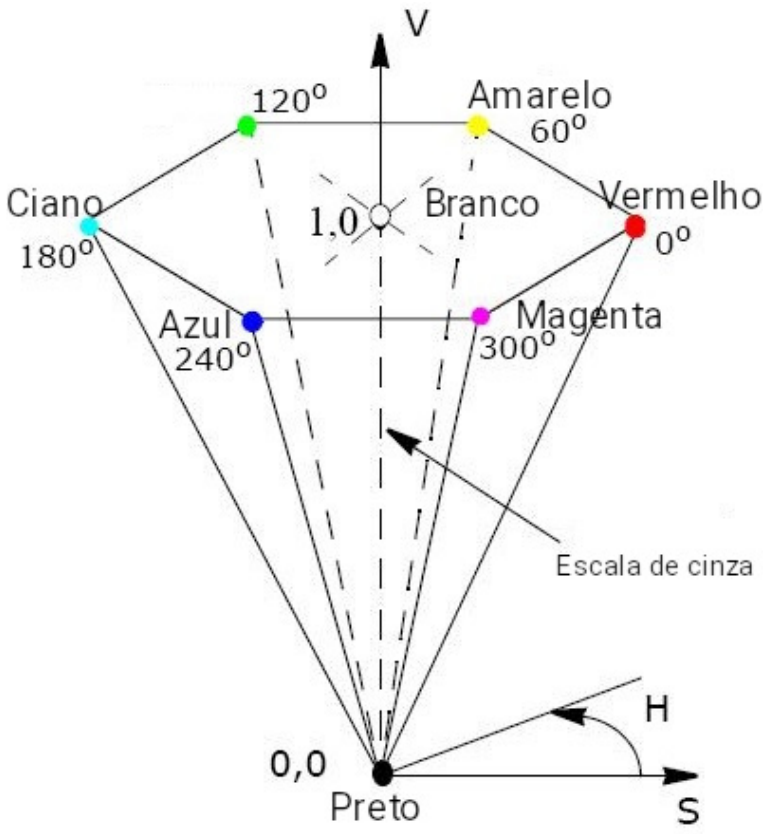
Imagens podem ser representadas em tons de cinza, o que gera a perda da informação de cor, mas mantém informações de forma e textura dos objetos da imagem. Cada *pixel* de uma imagem cinza é representado por um único valor de intensidade permitindo a imagem ser representada por uma única camada, requisito necessário para a aplicação de diversos processamentos de imagem (18). A intensidade de $Cinza$ de um *pixel* é uma soma ponderada das intensidades de vermelho (R), verde (G) e azul (B) dada pela Equação 2.1 (19) .

$$Cinza = 0{,}2989R + 0{,}5870G + 0{,}1140B \tag{2.1}$$

2.1.3 Modelo HSV

O modelo HSV apresenta a camada H (*hue*), que representa a matiz, a saturação na camada S (*saturation*) e a camada V (*value*) como a luminância. A Figura 2.3 apresenta o diagrama do modelo HSV. Este modelo é baseado no RGB, assim é possível fazer uma conversão entre estes formatos de imagem. A Equação 2.2 apresenta os cálculos a serem realizados para esta conversão (2).

Figura 2.3 – Modelo HSV.



Fonte: adaptado de (17).

$$
\begin{aligned}
H &= \begin{cases} 60\dfrac{(G-B)}{(M-m)}, & \text{se } M = R \\ 60\dfrac{(B-R)}{(M-m)}, & \text{se } M = G \\ 60\dfrac{(R-G)}{(M-m)}, & \text{se } M = B \end{cases} \\
S &= \begin{cases} \dfrac{(M-m)}{(M)}, & \text{se } M \neq R \\ 0, & \text{caso contrário} \end{cases} \\
V &= M
\end{aligned}
\tag{2.2}
$$

em que:

- M é o maior valor entre os três canais do modelo RGB, variando de 0 a 255;
- m é o menor valor entre os três canais do modelo RGB, variando de 0 a 255.

É recomendado utilizar o HSV quanto se deseja trabalhar com detecção de cores, pois a camada H já é a informação de cor da imagem (20).

2.1.4 Filtro Gaussiano

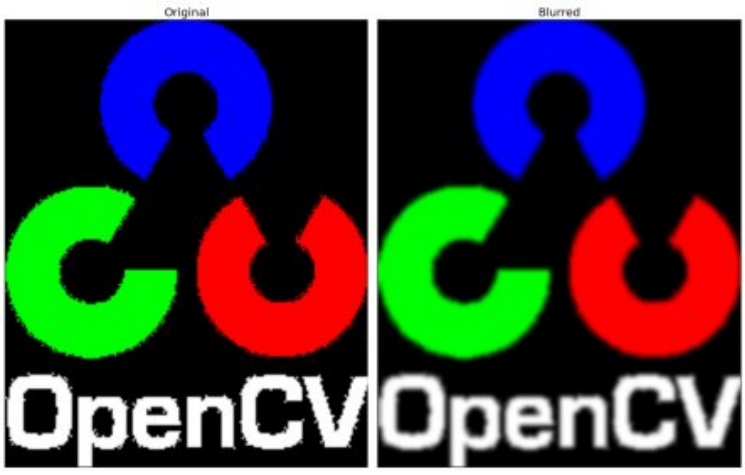
O filtro Gaussiano calcula um novo valor para cada *pixel* tendo em vista os valores dos *pixels* vizinhos. A Equação 2.3 apresenta a função Gaussiana para um determinado *pixel* na posição (x,y), onde o novo valor será a convolução entre o valor antigo com a resposta da função. Este filtro pode ser entendido também como uma média ponderada dos *pixels* vizinhos, onde os mais próximos têm pesos maiores. Isto é justificado pois os *pixels* mais próximos vão ter valores semelhantes, excetuando os contornos das imagens, ou seja, este filtro retira informação de contorno da imagem (21). Ele é um dos mais usados para suavizar e remover ruídos, mas ele não é um algoritmo rápido (22) (23) (24). Em imagens coloridas, este processo é realizado para cada camada de forma independente (24). A Figura 2.4 apresenta o resultado do filtro Gaussiano de uma imagem.

$$g(x,y) = \frac{1}{2\pi\sigma^2} \cdot e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.3}$$

em que:

- σ é o desvio padrão entre os pontos distantes de um raio predefinido.

Figura 2.4 – Imagem original à esquerda e o resultado depois do filtro Gaussiano à direita.



Fonte: ver referência (24).

2.1.5 Filtro Mediana

O filtro mediana consegue reduzir os ruídos de uma imagem com a vantagem de preservar seus contornos. O valor do *pixel* é definido simplesmente pegando a mediana

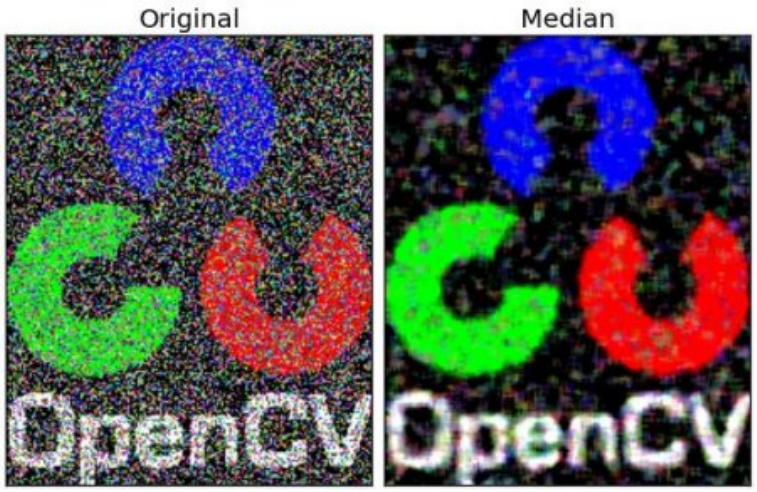
dos valores de seus *pixels* vizinhos, Equação 2.4 (25). Assim como o Gaussiano, o filtro mediana é calculado em cada camada de forma independente (24). A Figura 2.5 apresenta o resultado do filtro mediana em uma imagem.

$$M(x,y) = mediana(v_1,v_2,...,v_n) \tag{2.4}$$

em que:

- $v_1,v_2,...,v_n$ são as intensidades dos n vizinhos mais próximos do *pixel* na posição (x,y).

Figura 2.5 – Imagem original à esquerda e o resultado depois do filtro Mediana à direita.

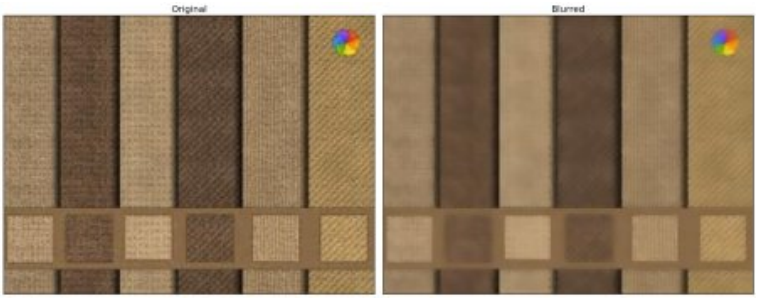


Fonte: ver referência (24).

2.1.6 Filtro Bilateral

O filtro Bilateral tem como objetivo manter a redução de ruídos do filtro Gaussiano nas regiões de baixa variação ao mesmo tempo que preserva os contornos presentes nas regiões de grande variação (21). A Figura 2.6 apresenta o resultado do filtro Bilateral de uma imagem.

Figura 2.6 – Imagem original à esquerda e o resultado depois do filtro Bilateral à direita.



Fonte: ver referência (24).

2.1.7 Limiarização

Uma das principais utilidades da limiarização é possibilitar a segmentação da imagem, separando o primeiro plano do plano de fundo quando possuem valores de intensidades ou cores diferentes. A imagem após o processo de limiarização é dada pela Equação 2.5 (26) (3). A Figura 2.7 apresenta o resultado de limiarização de uma imagem.

$$L(x,y) = \begin{cases} 255, \text{ se } f(x,y) > l \\ 0, \text{ caso contrario} \end{cases} \tag{2.5}$$

em que:

- l é um valor limiar que pode ser escolhido arbitrariamente ou pode ser calculado para uma melhor segmentação.

Figura 2.7 – Imagem original à esquerda e o resultado depois da limiarização à direita.

Fonte: ver referência (1).

2.1.8 Canny

O filtro Canny foi desenvolvido por John F. Canny para identificar o contorno das imagens. Ele é um algoritmo de múltiplas etapas. A primeira etapa é reduzir o ruído

usando o filtro Gaussiano. A segunda etapa é aplicar o *kernel* Sobel que tem a função de identificar as mudanças de intensidades dos *pixels* na horizontal G_x e vertical G_y, Equações 2.6 e 2.7, e obter o gradiente G_c com seu ângulo G_θ que é sempre perpendicular ao contorno, Equações 2.8 2.9.

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * f(x,y) \tag{2.6}$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * f(x,y) \tag{2.7}$$

$$G_c = \sqrt{{G_y}^2 + {G_x}^2} \tag{2.8}$$

$$G_\theta = \tanh^{-1}\left(\frac{G_y}{G_x}\right) \tag{2.9}$$

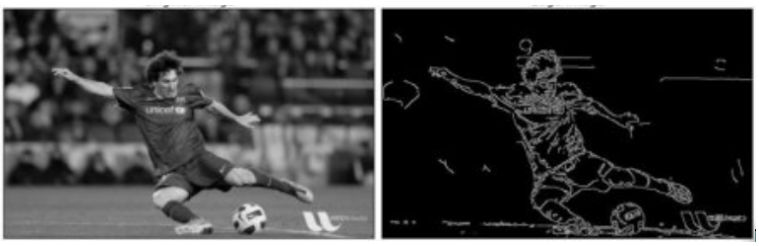
em que:

- *f(x,y)* é a imagem que será convolucionada pelo *kernel*;

O terceiro passo é remover todos os *pixels* que não fazem parte de um contorno. O valor gradiente do *pixel* é comparado com o de seus vizinhos. Se ele for um máximo local, ele terá valor 255 na imagem resultante. Caso contrário, ele será suprimido, assumindo o valor 0. A imagem resultante é uma imagem binária, apresentando somente dois valores de intensidade.

A quarta etapa e final é verificar se os *pixels* filtrados pela terceira etapa realmente fazem parte de um contorno. São definidos um limiar de máximo e outro de mínimo. Se a intensidade do *pixel* é maior que o limiar máximo, então é certeza que é um contorno. Se é menor que o limiar minimo, então é certeza que não é um contorno. As intensidade intermediárias são classificadas verificando sua conectividade com os *pixels* já definidos como certeza de contorno. Nesta etapa, são removidos os *pixels* que são ruídos que formam contornos pequenos. A Figura 2.8 apresenta o resultado de detecção de contornos usando Canny em uma imagem (27).

Figura 2.8 – Imagem original à esquerda e o resultado depois da detecção de Contornos de Canny à direita.



Fonte: ver referência (27).

2.1.9 Equalização de histograma

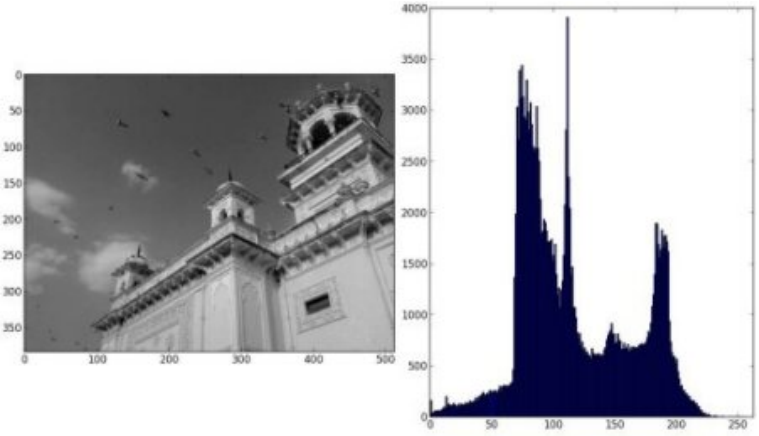
O histograma de uma imagem é um gráfico de intensidades, ou níveis de cinza, por quantidade de *pixels*. O histograma é obtido de uma matriz de duas dimensões, podendo ser uma imagem cinza ou uma camada da imagem colorida (2). A Equação 2.10 apresenta o cálculo que deve ser realizado para obter o histograma de uma matriz bidimensional. A Figura 2.9 apresenta o histograma de uma imagem.

$$p_k(f) = \frac{h(k)}{MN} \tag{2.10}$$

em que:

- f é uma imagem de $M \times N$ *pixels*;
- $h(k)$ é o número de *pixels* que possuem o nível cinza k.

Figura 2.9 – Imagem original e o seu histograma.



Fonte: ver referência (28).

A equalização de histograma modifica o valor dos *pixels* para reduzir transições bruscas no gráfico. Este processamento consegue normalizar a luminosidade e aumentar o contraste da imagem, permitindo visualizar detalhes anteriormente escondidos.

A equalização realiza o mapeamento das intensidades do histograma original para novos valores com base na função distribuição acumulada, gerando um novo gráfico mais espalhado e com todo o intervalo de 0 a 255 preenchido, Equação 2.11 (1).

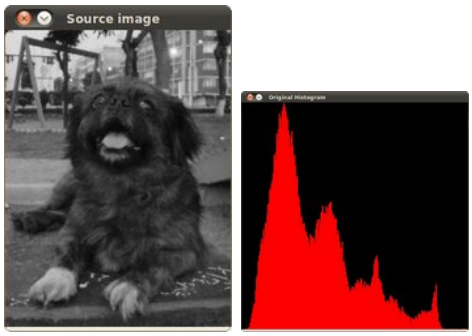
$$h'(k') = \frac{255}{MN} \sum_{j=0}^{k'} h(j) \tag{2.11}$$

em que:

- $h'(k')$ é a quantidade de *pixels* de intensidade k' do histograma equalizado.

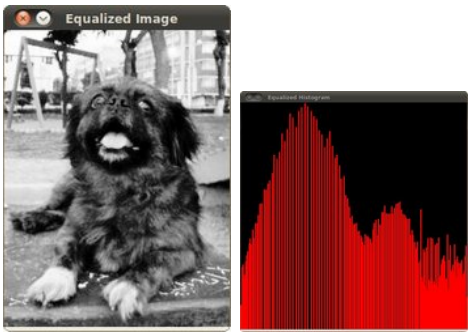
A Figura 2.10 apresenta uma imagem em tons de cinza com baixo contraste e o seu histograma com as intensidades mais concentradas. A Figura 2.11 é o resultado após passar pelo processo de equalização de histograma com seu histograma mais espaçado e a imagem com maior contraste.

Figura 2.10 – Imagem original e seu histograma.



Fonte: ver referência (29).

Figura 2.11 – Resultado da equalização de histograma.



Fonte: ver referência (29).

2.1.10 K-Means

O K-Means é um classificador não supervisionado que pode ser usado para processar imagens. Este classificador recebe as intensidades dos *pixels* como entrada e agrupa os valores mais próximos. Para cada grupo é criado um valor de intensidade média que representará todos os *pixels* daquele grupo. Suas principais utilizações são a segmentação de imagem e compressão de dados (30).

O primeiro passo do algoritmo do K-Means é definir a quantidade k de grupos que serão criados. O segundo é escolher os centroides iniciais que irão representar cada grupo, podem ser os primeiros valores dos dados de entrada ou escolhidos aleatoriamente. No terceiro passo, cada valor de entrada será colocado no grupo do centroide de menor distância geométrica. No quarto, para cada grupo criado é calculado um novo centroide que representa a média dos valores. Os passos três e quatro ficam se repetindo até não haver mais a modificação dos centroides. Assim, todos os pontos em um grupo serão similares. O último passo é substituir todos os valores do grupo pelo valor do centroide (30).

Logo, ao utilizar o K-Means em uma imagem é possível reduzir a quantidade de intensidade de cores para um número fixado (30). A Figura 2.12 apresenta a imagem original e o resultado após a utilização do filtro K-Means para diferentes valores de K.

Figura 2.12 – Imagem original e o resultado do K-Means para K=2, K=4 e K=8.

Fonte: ver referência (31).

2.1.11 Filtro Morfológico de Fechamento

O filtro morfológico de fechamento é útil para preencher os objetos. Este filtro é a combinação do processo de dilatação seguido da erosão, usando o mesmo *kernel*. Supondo uma imagem $f(x,y)$ e B o *kernel* escolhido, a dilatação é calculada realizando a $f(x,y) \oplus B$ em que $f(x,y)$ é a matriz da imagem original. Ou seja, se pelo menos um elemento da imagem $f(x,y)$ coincidir com B, então o *pixel* terá o valor do objeto. Caso contrário, o valor é igual ao do plano de fundo. A erosão é o inverso da dilatação sendo $f(x,y) \ominus B$,

ou seja, o valor do *pixel* só será igual ao do objeto se todos os elementos do *kernel* forem iguais aos da imagem. Caso contrário, é atribuído o valor do plano de fundo (32) (1).

A Figura 2.13 apresenta o resultado do filtro morfológico de fechamento em uma imagem. Pode-se ver a letra jota com alguns buracos e o efeito do filtro é preencher ou fechar estes buracos que estão presentes dentro do contorno (32).

Figura 2.13 – Imagem original à esquerda e o resultado depois do filtro morfológico de fechamento à direita.

Fonte: ver referência (32).

2.2 Reconhecimento de Objetos

Pode-se dizer que o universo é formado por um grande número de elementos. Esses elementos podem sofrer sucessivas transformações, existindo ainda o surgimento de novos elementos constantemente. A capacidade do ser humano e de outros seres vivos de distingui-los no ambiente é consequência de um aprendizado constante na vida do ser vivente onde este entra em contato com vários objetos. Não é possível conhecer todos eles pois são numerosos e estão espalhados pelo universo. Entretanto, o ser humano tem a capacidade de identificar um com o qual nunca teve contato antes, pois este pode apresentar características semelhantes a de um outro objeto conhecido, a partir das quais pode-se dizer que tipo de objeto é este (1).

2.2.1 Conceitos Básicos

A ideia do aprendizado de máquina tem como base o aprendizado que uma pessoa ou animal realiza durante sua vida. Quando se observa pela primeira vez um objeto, pode-se dizer que ele é um celular, por exemplo. Isso porque provavelmente já foram vistos vários celulares que apresentam certas semelhanças entre si, como o tamanho, a presença de tela, botões externos ou câmeras, que ajudam a definir esse objeto, nunca visto, como sendo um celular. Ou seja, para se poder identificar objetos nunca vistos antes, deve-se

ter um conhecimento anterior de outros objetos que se assemelham ao novo objeto, sendo um aprendizado por memorização, o qual usa uma generalização das características em comum dos objetos aprendidos.

Assim, uma pessoa que já conheceu muitos celulares sabe que as principais características que o descrevem visualmente são: o tamanho, o formato, o número de câmeras e botões. Então é apresentado um *tablet* a esta pessoa, que não tinha conhecimento deste tipo de objeto. Ela verá que o formato, câmeras e botões são semelhantes ao celular, porém o tamanho é muito desigual. Assim, ela pode deduzir que este objeto é desconhecido, ou se tiver que aproximá-lo de um já conhecido dirá que é um celular.

A seguir, o objeto é apresentado à pessoa como sendo um novo tipo de objeto, um *tablet*. Então, é mostrado um novo modelo de celular com quatro câmeras ao invés de uma e é pedido para ela dizer se este é um celular ou um *tablet*. Os botões e o formato são semelhantes a ambas as classes, porém o número de câmeras é igualmente diferente. Então, nenhuma destas características é útil para definir o tipo deste objeto. Porém, o tamanho é uma informação útil, pois sendo mais próximo do celular que do *tablet* a pessoa pode concluir que este objeto é um celular.

Na área de reconhecimento de objetos, é chamado de característica qualquer informação que pode ser extraída de um objeto. No exemplo anterior, podemos ter como características o posicionamento e número dos botões e câmeras, bem como o tamanho, formato, cor, textura entre outros, sendo que para comparar dois ou mais objetos deve-se utilizar as mesmas características em todos. Normalmente elas são representadas por valores numéricos, porém também é possível utilizar características nominais.

2.2.2 Vetor de características

Um conjunto de características que será usado para identificar um objeto é chamado de padrão. O padrão no exemplo do novo modelo de celular poderia ser: tamanho pequeno, sem botões, quatro câmeras e formato quadrado. Também existe a definição de classe, em que cada uma tem seu conjunto de padrões que ajudam a agrupar objetos semelhantes (1). Pode existir uma classe chamada *tablet* e outra de nome celular, a última pode ter dois padrões, o padrão do novo modelo de celular já descrito e o padrão do celular tradicional sendo tamanho pequeno, sem botões, duas câmeras, e formato quadrado.

O padrão que descreve um elemento pode ser chamado também de vetor de características, pois os algoritmos de classificação normalmente utilizam vetores de n elementos para representarem um padrão. Cada elemento representa a grandeza de uma

característica, portanto temos n características. Por ser um vetor, é possível representar um padrão por meio de um gráfico de n dimensões (1).

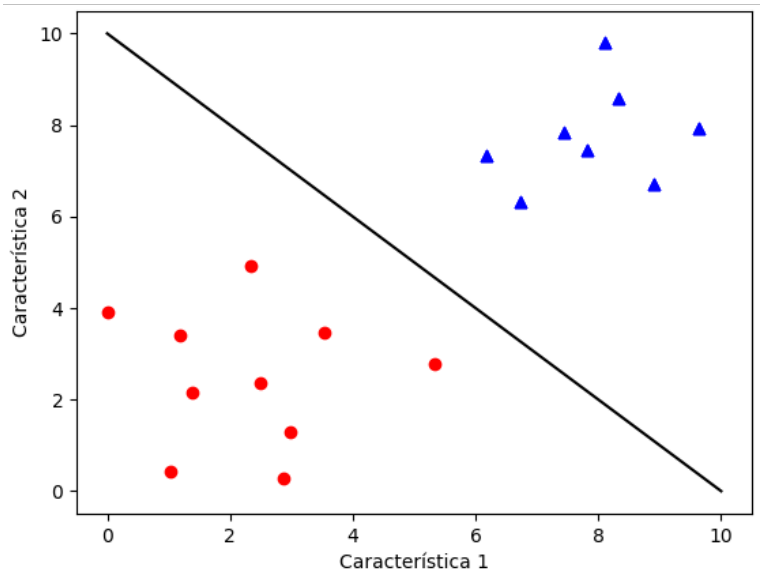
Tomando o exemplo do celular e do *tablet*, pode-se escolher inicialmente as características do formato, tamanho e o número de câmeras. Ao observar estas características graficamente, fica visível que tanto o número de câmeras quanto o formato fazem com que os dois pontos, cada um representando um padrão, estejam muito próximos. Isso torna a utilização dessas características desnecessária e em muitos casos podem dificultar a separação destes pontos, pois quanto mais afastados eles estão, mais fácil é identificar a classe de um novo ponto. Existem várias técnicas para identificar um novo objeto, mas as mais comuns usam o princípio de proximidade entre os pontos. Assim, se uma classe possui vetores que geram pontos próximos de outra classe, a probabilidade de acerto da classe de um novo objeto pode cair rapidamente (1).

No gráfico formado por vários padrões é recorrente ter vários agrupamentos de pontos. Estes existem devido à semelhança compartilhada das características usadas, ou seja, cada agrupamento representa um conjunto de objetos semelhantes que normalmente são incluídos em uma mesma classe. Pode-se dizer que cada classe possui uma zona do gráfico que inclui todos os padrões desta classe, assim cada parte do gráfico pertence a uma classe. Consequentemente, para definir a classe de um objeto desconhecido basta ver em qual região o seu respectivo vetor de características irá se posicionar, então esta provavelmente será a região da classe à qual ele pertence (1).

A Figura 2.14 apresenta um gráfico contendo vinte padrões genéricos, em que os marcados como vermelho pertencem à mesma classe, a classe dos círculos, e todos os azuis pertencem à classe dos triângulos. Todos os vetores possuem os mesmos elementos, característica 1 e característica 2, ambas também genéricas. Portanto, são vetores de dois elementos que resultam em um gráfico com duas dimensões. A linha traçada no gráfico é uma possibilidade de separação do gráfico em duas regiões, cada uma pertencente a uma classe. Nesse caso uma simples linha reta consegue separar completamente todos os pontos corretamente.

Se um novo vetor de características de classe desconhecida for adicionado, para saber qual classe este novo padrão se encaixa basta analisar em qual região ele irá se posicionar no gráfico. A Figura 2.15 utiliza os mesmos padrões e classes da Figura 2.14, porém apresenta ponto quadrado, representando um padrão novo, de classe desconhecida. É possível ver pelo gráfico que, apesar de muito próximo da fronteira de decisão, ele está na região da classe dos círculos, portanto essa é sua classe mais provável. Os algoritmos de

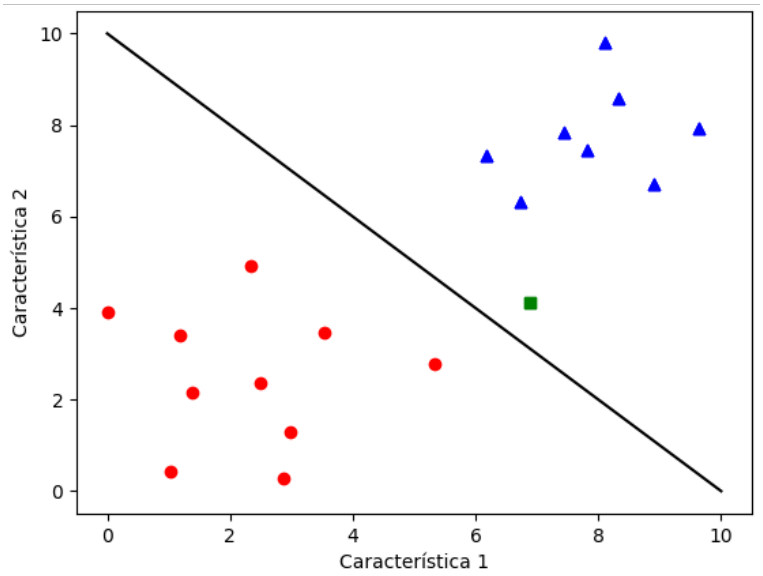
Figura 2.14 – Divisão do espaço bidimensional gerada pelo classificador para duas classes de vetores.



Fonte: O autor.

reconhecimento de objetos normalmente não guardam a informação de todos os padrões, pois em muitos casos estes são usados em grandes quantidades e isso causaria lentidão e alto consumo de memória. Assim, é mais viável, por exemplo, armazenar simplesmente a equação da reta que descreve a fronteira de decisão.

Figura 2.15 – Predição do classificador para um vetor desconhecido.



Fonte: O autor.

A divisão em regiões do gráfico não é uma tarefa simples, pois o posicionamento dos vetores em geral não se aproximam de um caso ideal. Este seria o qual a inclusão

de padrões da classe geraria pontos com o módulo da distância entre eles muito menor do que o módulo da distância entre quaisquer outros dois pontos pertencentes a classes diferentes. Além da possibilidade de um espalhamento de pontos da mesma classe, outras situações piores podem ocorrer como, por exemplo, a ocorrência de um ponto de uma classe no meio de outra, podendo ser um pequeno grupo próximo cercado por pontos de outra classe ou uma mistura entre os pontos das extremidades de cada agrupamento. Nesses casos, é improvável que exista uma forma eficaz de agrupar estes pontos em uma região que apresente somente pontos que compartilhem a mesma classe. Esses dois últimos casos podem gerar um erro de classificação, em que a melhor solução é repensar nas características utilizadas e verificar se a obtenção de seus valores sofreu algum tipo de interferência (1).

Assim, devido às inúmeras possibilidades de organização dos padrões, existem inúmeras técnicas de classificação de objetos, onde cada uma foi criada para realizar um tipo tarefa específica de classificação, apresentando um ótimo aproveitamento para esta, porém para outras o resultado pode não ser tão bom. Estas técnicas são um conjunto de passos que utilizam vários conhecimentos matemáticos utilizados para construir essa fronteira entre regiões (1).

2.2.3 Extração de Características

Para se obter um padrão de uma imagem, deve-se conseguir extrair informações da imagem convertendo esta informação para uma sequência numérica. Uma forma simples de realizar esta função é transformar a matriz que forma a imagem em um vetor, usando as intensidades dos *pixels* como característica.

No entanto, uma imagem cinza com resolução de 480×360 geraria um vetor de 172.800 valores, o que provocaria um grande custo computacional pelo elevado número de dimensões do padrão. Uma solução para este problema é usar o histograma da imagem, que tem dimensão de 256, facilitando o processamento.

Mas, em alguns casos, utilizar o valor de intensidade dos pixels pode não ser suficiente ou a melhor escolha. Por isso existem métodos como o SIFT (*Scale-Invariant Feature Transform* - Transformação de Características Invariante na Escala) que tem como objetivo usar os contornos presentes na imagem como característica. As dimensões dos contornos podem variar quando a imagem é ampliada ou rotacionada, mas o SIFT consegue transformar a informação do contorno em valores que são invariantes na escala (33) (34).

O algoritmo do SIFT apresenta quatro etapas principais. A primeira é detectar pontos que fazem parte dos contornos que sejam invariantes a escala usando a diferença da função Gaussiana (DoG), Equação 2.12, onde são comparadores dois valores de α permitindo eliminar os pontos ruidosos e manter os que apresentam uma forte informação quando realiza a convolução com a imagem, Equação 2.13. Um *pixel* de extremo é detectado quando seu valor DoG é máximo quando comparado com seus 8 vizinhos na escala atual e os 9 pixels da escala menor anterior e a próxima maior (33) (34).

$$DoG = G(x,y,k\alpha) - G(x,y,\alpha) \tag{2.12}$$

em que k é uma parâmetro que modifica o valor de α.

$$L(x,y,k\alpha) = G(x,y,k\alpha) * f(x,y) \tag{2.13}$$

A segunda etapa é determinar a localização exata deste pontos usando a expansão da série de Taylor. Se a intensidade da localização é menor que um limiar, ela é rejeitada, assim todos os pontos de baixo contraste e de borda são rejeitados (33) (34).

A terceira etapa é determinar uma orientação, Equação 2.14, e uma magnitude, Equação 2.15, do gradiente para cada ponto filtrado pela segunda etapa usando as diferenças entre os *pixels*. A determinação da orientação permite que a característica seja invariante à rotação da imagem. O histograma de orientações apresenta 36 valores possíveis cobrindo de 0º até 360º (33) (34).

$$S_\theta(x,y) = \tan^{-1}\left(\frac{L(x+1,y,\alpha) - L(x-1,y,k\alpha)}{L(x,y+1,\alpha) - L(x-1,y-1,\alpha)}\right) \tag{2.14}$$

$$S(x,y) = \sqrt{(L(x+1,y,\alpha) - L(x-1,y,k\alpha))^2 + (L(x,y+1,\alpha) - L(x-1,y-1,\alpha))^2} \tag{2.15}$$

A última etapa é criar o vetor descritor para cada ponto que irá descrevê-lo de forma invariante a escala e rotação. Ele é obtido criando uma janela 16x16 contendo os *pixels* vizinhos, que é subdividida em 16 janelas 4x4. Para cada janela, são gerados 8 pares de orientações e magnitudes. Assim, o vetor descritivo irá conter 128 valores. Desse modo, este vetor pode ser usado como um padrão que irá descrever o contorno da imagem de forma invariante (33) (34).

2.2.4 Técnicas de Classificação

O principal objetivo dos estudos na área de reconhecimento de objetos é a criação de um sistema de classificação. Assim, a máquina pode usar este sistema para reconhecer os objetos, fazendo a rotulação de cada um. Ou seja, com o conhecimento das principais características que identificam um determinado grupo de objetos, isto é, uma classe, a máquina conseguirá rotular os objetos que pertencem a uma classe com o rótulo da mesma (2).

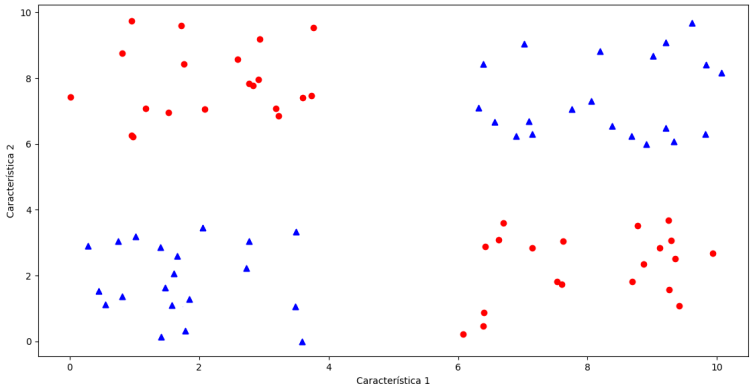
Os rótulos podem representar uma classe de objetos. Em um único sistema de classificação pode ser utilizado um grande número de classes. Assim, alguns autores usam a letra "ω" para identificar classes genéricas ao facilitar a sua referência no decorrer do texto. Por exemplo, a primeira classe é ω_1, a segunda ω_2, a assim até a última classe que será ω_n, em que "n" é o número total de classes (2).

Para que uma máquina tenha a capacidade de identificar e organizar os objetos corretamente em grupos, é necessário que a mesma tenha um sistema de instruções baseadas em alguma lógica matemática a qual deverá ser seguida, transformando-as em comandos de linguagem de máquina. Este sistema de instruções varia de linguagem para linguagem e depende da técnica de classificação escolhida para ser implementada. É comum atribuir o nome "algoritmo de classificação" ou "classificador" para esses sistemas (2).

A Figura 2.16 apresenta um caso em que pontos de uma mesma classe estão agrupados em duas zonas diferentes. Neste caso, não é possível utilizar uma simples reta para realizar a divisão entre as duas classes. Portanto é necessário aumentar a complexidade da fronteira de maneira a esta tomar uma forma que gere uma separação efetiva entre todos os vetores de características. Uma possível solução para o caso apresentado pode ser visto na Figura 2.17, onde a região mais escura representa a região da classe dos triângulos e a clara a região dos círculos. Esta última foi dividida em duas regiões, porém se um novo vetor se posicionar em qualquer uma ele será reconhecido como sendo da classe dos triângulos. Quanto mais próximo um vetor está da região de decisão, menor será a certeza do resultado de classificação final.

O objetivo das técnicas de classificação é conseguir definir a fronteira entre as classes de forma a gerar a melhor separação possível, agrupando todas os padrões de uma classe na mesma região. Assim, tanto os objetos que foram usados para construir esta região, bem como os que não foram usados mas pertencem a esta classe, possuem um padrão que se posiciona dentro da região de sua classe. Para poder obter a fronteira de

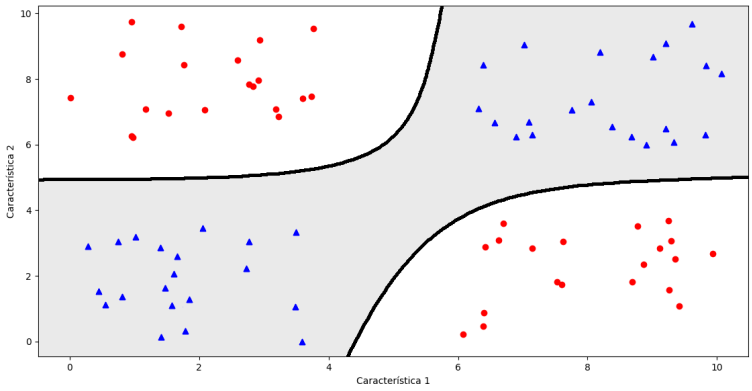
Figura 2.16 – Vetores de características da mesma classe em regiões diferentes.



Fonte: O autor.

regiões que cumpra com este objetivo é preciso analisar a complexidade da distribuição dos padrões de objetos que se quer realizar a predição para definir as possíveis técnicas que podem retornar uma resposta aceitável. Essas técnicas irão utilizar diversos conhecimentos matemáticos e aplicá-los em algoritmos para construir a melhor fronteira de decisão possível para os vetores de características disponibilizados (1).

Figura 2.17 – Divisão do espaço bidimensional usando um modelo de classificador mais complexo.



Fonte: O autor.

Quando se constrói um sistema de classificação, a principal característica de avaliação de sua qualidade é a taxa de acertos que este irá conseguir obter após utilizar o sistema para classificar um grupo de objetos de classe desconhecida. O classificador irá retornar a classe na qual o vetor de características melhor se encaixa, então este resultado é comparado com o valor verdadeiro da classe. A porcentagem de acertos de um classificador está relacionada tanto com a escolha correta do classificador, como com a qualidade do estudo desempenhado na etapa de escolha dos vetores de características usados no

treinamento (1).

A extração dos valores da imagem pode proporcionar o sucesso ou fracasso do classificador. Os valores representam características do objeto retratado, se algum tipo de distorção ou um erro não previsto estiver contido em algum valor, significa que uma característica verdadeira foi substituída por uma falsa, assim, no treinamento, o mecanismo de demarcação de classe irá se adaptar para receber um objeto com esta característica errada, podendo aumentar a complexidade do classificador, aumentando a probabilidade de erro do sistema como um todo (2).

Na construção de um sistema de classificação que apresenta baixa probabilidade de acertos, o programador tende a pensar em soluções que ajudem a aumentar a qualidade de seu projeto. Uma possibilidade é aumentar o número de características de cada objeto, ou seja aumentar a dimensionalidade do espaço de características buscando um distanciamento maior entre pontos de classes diferentes em outras dimensões. Contudo, esta ação nem sempre se mostra eficaz. Aumentar o número de características sem conhecer a distribuição atual dos padrões no espaço torna o aumento da dimensionalidade ineficaz onde esta nova dimensão não consegue distanciar os vetores de classes diferentes que provocavam um agrupamento destes (2).

Na tentativa de aumentar a taxa de acertos é comum tentar aumentar o número padrões utilizados no treinamento, porém esta ação pode não surtir efeito ou, na pior da hipóteses, pode reduzir a qualidade do classificador. Quando o classificador erra a classe de um padrão, duas situações podem ter ocorrido. Na primeira, as características do padrão recebido podem ter sido geradas de forma errada, com a sua extração do objeto original apresentando algum tipo de distorção fora da realidade do grupo de treinamento, portanto este ponto pode ter se posicionado graficamente entre pontos de uma outra classe. Neste caso, aumentar o número de dados usado no treinamento não resultará na melhora do sistema de classificação (1). O outro evento que pode ocorrer é este novo ponto se posicionar mais próximo do agrupamento de uma outra classe que de sua própria, porém é possível que os dados extraídos de seu objeto estejam corretos, sendo que possivelmente na fase de treinamento não foram usados padrões que atenderiam a possibilidade de classificar corretamente esta variação que elementos desta classe podem incluir. Neste caso, aumentar os dados utilizados no treinamento resultaria em uma melhor definição da fronteira de decisão, englobando todas as possibilidades de padrões que uma classe pode a vir ter. O ideal é que nenhum dos vetores de características usados no treinamento esteja mesclado com outros vetores pertencentes a outras classes (1).

A principal informação que os algoritmos de classificadores retornam, dado um vetor de características, é um valor que representa uma única classe. Entretanto é possível obter valores probabilísticos para cada uma das classes existentes, onde a classe que tiver o maior probabilidade é considerada como sendo a resposta final do classificador. Utilizando a análise do gráfico pode-se dizer que um ponto mais afastado da fronteira de decisão tem uma probabilidade maior que um próximo à mesma. Assim, este índice é um dado importante para avaliar a qualidade de uma classificação específica bem como entender as limitações do sistema atual como um todo (1).

Para que possa ocorrer o reconhecimento de um objeto, que será feito por uma máquina, é necessário primeiramente que a mesma passe por um processo de aprendizado, onde ela entrará em contato com objetos semelhantes, os quais ela deverá reconhecer no futuro por si mesma. Esta etapa de constituição do sistema de classificação é chamada de treinamento ou aprendizagem. Nesta etapa, a principal ação a ser realizada é a utilização de padrões para construir o sistema de classificação. Mais especificamente, o classificador irá aplicar os vetores de características, com sua classe especificada, em um determinado processo matemático tendo como resultado a base do sistema de classificação, podendo ser um conjunto equações, bem como uma estrutura matemática. Tal variação é causada pela existência de classificadores com metodologias únicas. Quando o sistema de classificação for requisitado para a classificação de um objeto desconhecido, ele irá utilizar o recurso criado na fase de treinamento para dizer a qual classe este novo objeto pertence (1).

Para que o classificador possa classificar um objeto, é necessário que ele tenha conhecimento das características que definem todas as possíveis classes dos elementos a serem classificados. A estipulação deste conhecimento deve considerar que este é suficiente para que o classificador possua uma porcentagem de acertos aceitável. A etapa referente a esta operação é chamada de "treinamento", onde são utilizados as características de vários objetos para cada classe. O classificador os usa para mapear as regiões do gráfico referentes a cada classe. Posteriormente esta informação é armazenada, sendo usada quando o classificador se deparar com um objeto de classe desconhecido, para assim poder definir a sua respectiva classe (2).

Após o posicionamento de todos os elementos utilizados no treinamento no espaço de características na forma de vetores, o classificador irá definir regiões para cada classe presente no treinamento, onde cada vetor que possa a vir existir em uma determinada região pertencerá à mesma classe que esta região pertence (2).

Contudo, existem métodos que não se utilizam de um treinamento anterior a

classificação. Eles não possuem qualquer informação sobre as classes e nem quantas serão utilizadas. Neste tipo de metologia, o classificador recebe um grande número de vetores de características de todas as classes de forma desordenada. A partir deste ponto, a sua função se baseia em analisar e agrupar os vetores que possuem características muito próximas. Então, cada agrupamento é definido como uma classe de nome genérico, portanto este tipo de classificador pode retornar o número de classes que ele identificou e os elementos que constituem cada uma. Como neste método o classificador tem maior liberdade para realizar a classificação, ele é comumente chamado de "classificação não-supervisionada". Para os métodos que se utilizam de uma fase de treinamento, ou seja, é definido manualmente o número de classes bem como os objetos pertencentes a cada uma, são definidos como “classificação supervisionada” (2).

Para este projeto de classificação de cédulas monetárias, onde o número de classes é fixo e os elementos que pertencem a cada uma delas não sofrem grande variação, foram usados somente métodos de classificação supervisionada. Assim, os métodos aqui utilizados são Redes Neurais Artificiais (*Artificial Neural Network* - ANN), Regra dos k-vizinhos mais próximos (*k-Nearest Neighbors* - KNN) e Máquina de Vetores de Suporte (*Support Vector Machine* - SVM) (2).

As categorias dos métodos de classificação não se resumem apenas à presença de uma etapa de treinamento. Também é possível classificá-los em “abordagem sintática” ou “estrutural” e “abordagem estatística” (2).

Classificadores que se enquadram na abordagem sintática usam uma sequência de símbolos gramaticais para descrever as características de um objeto. Neste sistema o classificador irá verificar qual a classe, cada uma possuindo uma gramática própria, melhor corresponde à gramática descrita pela sequência de símbolos pertencentes ao objeto (2).

Já a abordagem estatística utiliza-se da representação de características usando grandezas numéricas. Cada objeto pode ser representado por um conjunto de características ou de números, assim como um vetor. Uma classe é representada como um conjunto de infinitos vetores, porém, estes estão delimitados por certos valores limites de forma a englobar todos os objetos da classe e somente eles. O classificador tem a função de criar regras ou expressões matemáticas que consigam fazer esta delimitação de vetores possíveis para cada classe. Assim, quando o classificador é utilizado para classificar um novo vetor, ele se utiliza destas regras para determinar a sua respectiva classe (2).

Contudo, o simples uso de um classificador complexo para todos os casos não é uma boa prática, pois este apresenta pontos negativos como o alto uso de recursos

computacionais, devido ao aumento de operações matemáticas, ocasionando também um maior tempo de execução. Portando, se um classificador mais simples e rápido já apresenta resultados aceitáveis, o seu uso deve ser priorizado (2).

As técnicas de classificação não se resumem somente a essa delimitação de regiões, se aplicando para outras possibilidades lógicas de reconhecer padrões e agrupar com outros semelhantes. Exitem diversos algoritmos escritos em várias linguagens de programação que seguem a teoria destas técnicas e muitos deles disponibilizam-se para uso da comunidade na forma de bibliotecas. Portanto não faz parte do objetivo do projeto desenvolver o próprio algoritmo de classificação. É mais do que recomendado utilizar essas bibliotecas, pois foram otimizadas e testadas de forma profissional por quem as disponibilizou tendo como retorno de qualidade de vários indivíduos da comunidade (1).

2.2.5 Máquina de Vetores de Suporte

O classificador SVM é um modelo desenvolvido com o objetivo de atuar em problemas com espaço de características com um número grande de dimensões e gerar uma separação das classes por uma grande margem (35). Pode ser considerada uma grande margem de separação das classes quando, além de todos os vetores de treinamento com o mesmo lado estarem na mesma região, a distância entre a região de fronteira e o vetor mais próximo é máxima (3) (35).

A utilização de um algoritmo SVM para gerar uma separação no espaço de características com uma grande margem pode proporcionar uma complexidade pequena na análise das amostras de treinamento, mesmo quando o número de características usado tende ao infinito (35).

Como existem diferentes distribuições dos vetores no plano, o SVM apresenta variações que podem se adaptar e gerar uma ótima separação. Dentre estas, podemos citar três: classificador linear de margem máxima, classificador linear de margem suave e classificador não linear (3).

2.2.5.1 Classificação Linear de Margem Máxima

É recomendado para situações onde os vetores podem ser separados por uma fronteira linear. Neste caso, a fronteira pode ser construída por uma função linear, como apresentada na Equação 2.16. A etapa de treinamento do SVM tem o objetivo de encontrar

os melhores valores para os parâmetros a e b (3).

$$\boldsymbol{a} \cdot \boldsymbol{X} + b = 0 \tag{2.16}$$

Supondo que o conjunto da solução gerada pelo SVM seja $\{\boldsymbol{a}_o, b_o\}$, a classificação de um vetor não rotulado $\boldsymbol{X}_j$ pode ser realizada aplicando na função sinal da Equação 2.17. Assim, o resultado será dado entre duas classes $\omega = \{1, -1\}$. Se $\boldsymbol{X}_j$ estiver contido na região ω_1, o resultado será $\omega = 1$. Do mesmo modo, se $\boldsymbol{X}_j$ estiver contido na região de ω_2, então $\omega = -1$ (3).

$$\omega_j = sgn\left(\boldsymbol{a_o} \cdot \boldsymbol{X_j} + b_o\right) \tag{2.17}$$

Para todo $\boldsymbol{X}_i$ que pertence a uma classe ω_i aplicado na Equação 2.16, o resultado da mesma deverá ser igual a sua classe (Equações 2.18 e 2.19) (3).

$$\boldsymbol{a} \cdot \boldsymbol{X_i} + b \geq 1 \qquad \text{se } \omega_i = 1 \tag{2.18}$$

$$\boldsymbol{a} \cdot \boldsymbol{X}_i + b \leq -1 \qquad \text{se } \omega_i = -1 \tag{2.19}$$

A Figura 2.18 ilustra esta modelagem. A linha H é a fronteira de decisão desejada. H_1 e H_2 são hiperplanos dados pelas Equações 2.20 e 2.21. O Δ é a distância entre as extremidades das duas classes, sendo dada pela Equação 2.22 (3).

$$H_1 : \boldsymbol{a} \cdot \boldsymbol{X} + b = 1 \tag{2.20}$$

$$H_2 : \boldsymbol{a} \cdot \boldsymbol{X} + b = -1 \tag{2.21}$$

$$\Delta = \frac{2}{\|\boldsymbol{a}\|} \tag{2.22}$$

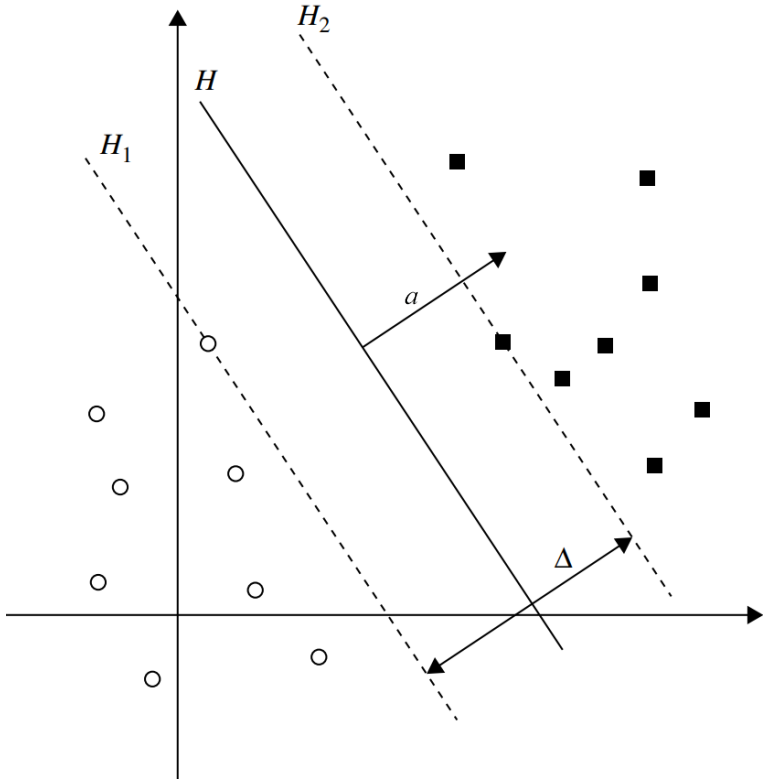
Tendo como objetivo maximizar o valor Δ, a Equação 2.23 deve ser minimizada (3).

$$\text{Minimizar} : \frac{1}{2}\boldsymbol{a} \cdot \boldsymbol{a} \tag{2.23}$$

Pode-se usar a função de lagrangiana para resolver este problema de otimização não linear. A Equação 2.24 apresenta a formulação desta função (3).

$$L\left(\boldsymbol{a}, b, \alpha\right) = \frac{1}{2}\boldsymbol{a} \cdot \boldsymbol{a} - \sum_{i=1}^{K} \alpha_i \left(\omega_i \left(\boldsymbol{a} \cdot X_i + b\right) - 1\right) \tag{2.24}$$

Figura 2.18 – O modelo SVM.



Fonte: adaptado de (3).

em que $\alpha_i \geq 0$ são os multiplicadores de Lagrange.

Na Equação 2.24, a minimização de $L(\boldsymbol{a},b,\alpha)$ em relação a $\boldsymbol{a}$ e b, provoca a maximação de α_i. Este desenvolvimento, chamado de problema de programação quadrático, é apresentado a seguir (3).

$$\left.\frac{\partial L(\boldsymbol{a},b,\alpha)}{\partial \boldsymbol{a}}\right|_{\boldsymbol{a}=\boldsymbol{a}_o} = \boldsymbol{a}_o - \sum_{i=1}^{K} \alpha_i \omega_i \boldsymbol{X}_i = 0 \tag{2.25}$$

$$\left.\frac{\partial L(\boldsymbol{a},b,\alpha)}{\partial b}\right|_{b=b_o} = \sum_{i=1}^{K} \alpha_i \omega_i = 0 \tag{2.26}$$

Resolvendo este problema duplo, são obtidos os valores de b_o e $\boldsymbol{a}_o$, como apresentado abaixo (3).

$$\boldsymbol{a}_o = \sum_{i=1}^{K} \alpha_i \omega_i \boldsymbol{X}_i \tag{2.27}$$

$$b_o = \omega_i - \boldsymbol{a}_o \cdot \boldsymbol{X}_i \tag{2.28}$$

Os vetores $\boldsymbol{X}_i$ com o atributo $\alpha > 0$ são chamados de vetores de suporte, sendo os padrões usados na fase de treinamento. Logo, um treinamento ótimo é determinado somente pelos vetores de suporte, e não por todos os padrões disponíveis (3).

2.2.5.2 Classificação Linear de Margem Suave

Os casos em que é possível a separação linear dos vetores são raros. Por isso, a classificação linear de margem suave tem como objetivo realizar o treinamentos de padrões que não são linearmente separáveis, ou apresentam sobreposição das regiões das classes. Para gerar um treinamento com o menor erro possível, é necessário reescrever as Equações 2.18 e 2.19 adicionando uma variável de folga não negativa ξ (Equações 2.29 e 2.30). (3)

$$\boldsymbol{a} \cdot \boldsymbol{X}_i + b \geq 1 - \xi \qquad \text{se } \omega_i = 1 \tag{2.29}$$

$$\boldsymbol{a} \cdot \boldsymbol{X}_i + b \leq -1 + \xi \qquad \text{se } \omega_i = -1 \tag{2.30}$$

Neste caso, para obter a fronteira de decisão que tenha uma melhor otimização na classificação, a Equação 2.31 deve ser minimizada, assim como foi realizado com a Equação 2.23 (3).

$$\text{Minimizar} : \frac{1}{2}\boldsymbol{a} \cdot \boldsymbol{a} + C\left(\sum_{i=1}^{K} \xi_i\right) \tag{2.31}$$

em que C é um parâmetro de penalidade ou regulação, que pode ser escolhido de forma experimental (3).

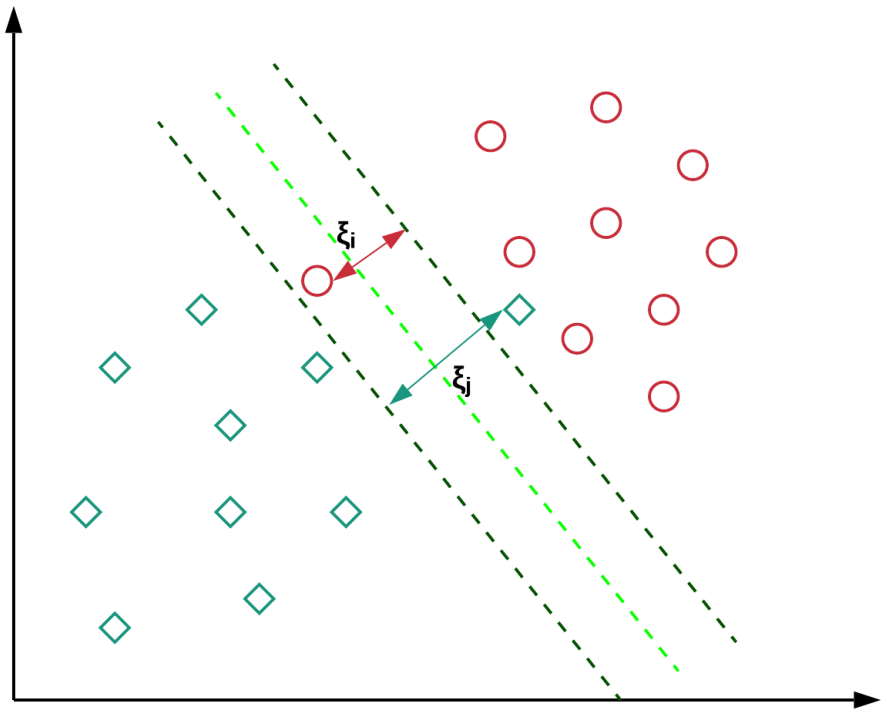
A função Lagrange estará em função de a, b e ξ_i. Utilizando os mesmos procedimentos da subseção 2.2.5.1, chega-se nas equações

$$\boldsymbol{a}_o = \sum_{i=1}^{K} \alpha_i \omega_i \boldsymbol{X}_i \tag{2.32}$$

$$b_o = \omega_i - \boldsymbol{a}_o \cdot \boldsymbol{X}_i \tag{2.33}$$

A região de decisão será construída utilizando todos os $\boldsymbol{X}_i$ com $\alpha_i > 0$. Estes, ainda, podem ser divididos em dois grupos. O primeiro é chamado de vetores de margem, possuem $\alpha_i < C$, resultando em $\xi_i = 0$, o que gera uma distância de separação $1/\|\boldsymbol{a}\|$ ótima. Já no segundo grupo, tem-se $\alpha_i = C$, onde os vetores corretamente separados estão a uma distância menor que $1/\|\boldsymbol{a}\|$ se $0 < \xi_i \leq C$. Para ξ_i, os vetores estão contidos fora da região pertencente a sua classe (3). A Figura 2.19 apresenta padrões que invadem a região da outra classe ou ficam muito próximos da fronteira de decisão. Para estes pontos é destacado o valor de ξ.

Figura 2.19 – Distribuições de padrões não linearmente separáveis pela presença de um $\xi > 0$.



Fonte: retirado de (36).

2.2.5.3 Classificação Não Linear

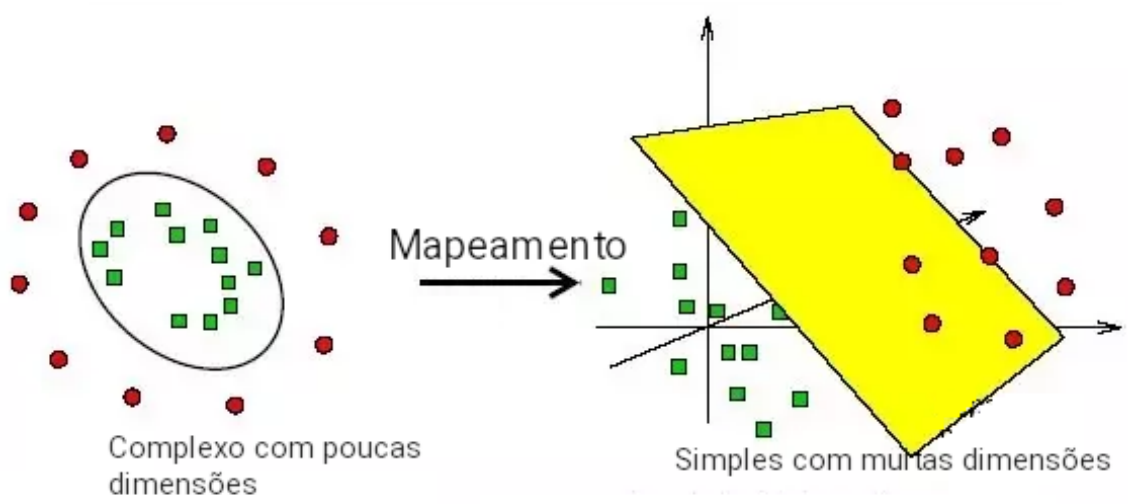
Usada quando os vetores de características não podem ser separados usando uma fronteira linear, mas podem apresentar uma boa separação das sobreposições com a utilização de fronteiras não lineares (3). O seu principio é usar uma transformação não linear (ϕ) para remapear os padrões para um novo espaço com dimensões maiores, que irá permitir a separação dos vetores de forma linear. Assim, é possível utilizar as técnicas discutidas nas duas subseções anteriores (3). A representação gráfica desta técnica está presente na Figura 2.20.

A Equação 2.16 é reescrita, para o caso da classificação não linear, como

$$\boldsymbol{a}_o \cdot \phi(\boldsymbol{X}) + b_o = \left(\sum_{i=1}^{K} \alpha_i \omega_i \phi(\boldsymbol{X}_i) \right) \cdot \phi(\boldsymbol{X}) + b_o \tag{2.34}$$

Na Equação 2.34, pode-se substituir $\phi(\boldsymbol{X}_i) \cdot \phi(\boldsymbol{X})$ por uma função *kernel* $K(\boldsymbol{X}_i, \boldsymbol{X}) = \phi(\boldsymbol{X}_i) \cdot \phi(\boldsymbol{X})$. Alguns exemplos de funções *kernel* estão na Quadro 2.1, em que γ, *degree*, c são parâmetros de configuração dos *kernels* que podem ser ajustados com o objetivo de melhorar o classificador para um aplicação especifica (37).

Figura 2.20 – Facilitação da separação dos vetores pelo aumento do número de dimensões.



Fonte: adaptado de (36).

Quadro 2.1 – *Kernels* do SVM.

Kernel	**Equação**
Linear	$K(\boldsymbol{X}_i,\boldsymbol{X}) = \boldsymbol{X}_i^T\boldsymbol{X}$
Polinomial	$K(\boldsymbol{X}_i,\boldsymbol{X}) = \left(\gamma \boldsymbol{X}_i^T\boldsymbol{X} + c\right)^{degree}$, $\gamma > 0$
Função de base radial (*radial basis function* - RBF)	$K(\boldsymbol{X}_i,\boldsymbol{X}) = e^{-\gamma\|\boldsymbol{X}_i,\boldsymbol{X}\|^2}$, $\gamma > 0$
Sigmóide	$K(\boldsymbol{X}_i,\boldsymbol{X}) = \tanh\left(\gamma \boldsymbol{X}_i^T\boldsymbol{X} + c\right)$
Exponencial Chi2	$K(\boldsymbol{X}_i,\boldsymbol{X}) = e^{-\gamma\chi^2(\boldsymbol{X}_i,\boldsymbol{X})}$, $\gamma > 0$ em que $\chi^2(\boldsymbol{X}_i,\boldsymbol{X}) = \frac{(X_i - X)^2}{(X_i + X)}$, com $\gamma > 0$
Interseção de Histograma	$K(\boldsymbol{X}_i,\boldsymbol{X}) = \min(\boldsymbol{X}_i,\boldsymbol{X})$

Fonte – adaptado de (37)(3)(37).

A região de decisão $f(\boldsymbol{X})$ será construída pela equação

$$f(\boldsymbol{X}) = sgn\left(\sum_{i=1}^{K} \alpha_i \omega_i K(\boldsymbol{X}_i,\boldsymbol{X}) + b_o\right) \tag{2.35}$$

2.2.5.4 Redes SVM

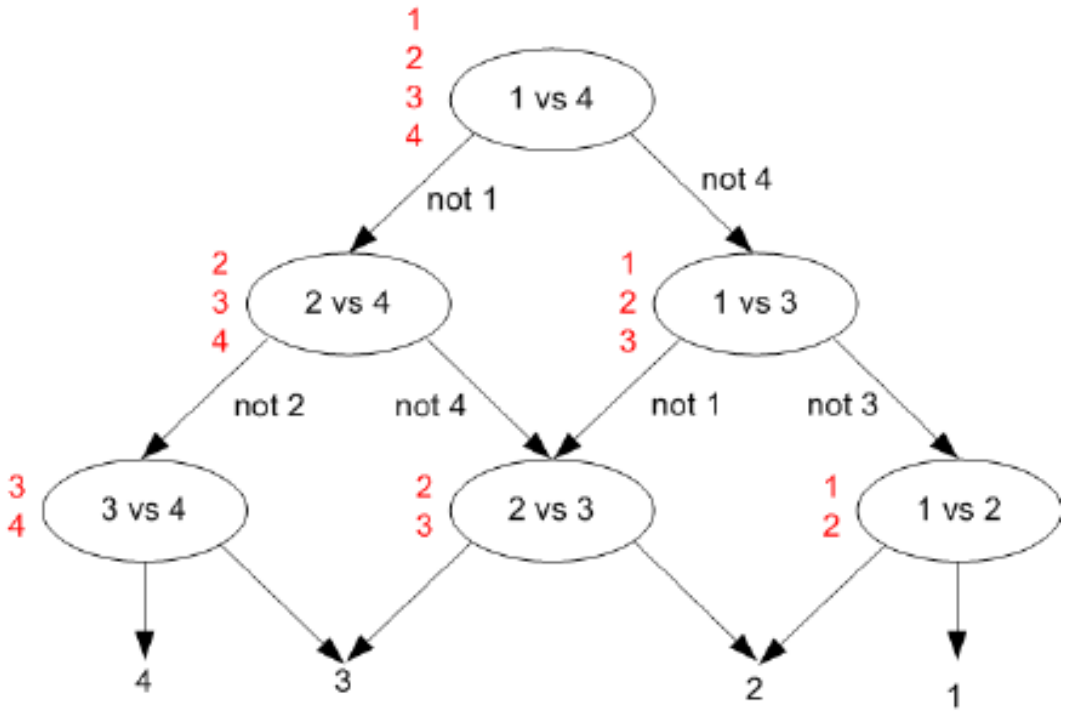
O SVM foi inicialmente idealizado para separar os vetores de características em dois grupos. Para realizar classificações utilizando mais de duas classes tem-se que criar uma rede SVM. Ou seja, é necessário aplicar várias vezes o algoritmo SVM (3).

Alguns tipos de redes são: um-contra-todos (*one-against-all*), um-contra-um (*one-against-one*) ou Grafo direcionado acíclico (*directed acyclic graph* - DAG) (3). A rede SVM usando um-contra-todos utiliza k classificadores SVM binários para classificar k classes.

Cada SVM binário irá identificar todos os padrões de uma dada classe e os outros são agrupados como sendo os que não são pertencentes a esta classe. O resultado ótimo é quando somente um SVM retorna valor positivo para sua respectiva classe. Caso mais de um retorne positivo, pode ser usado algum critério de confiabilidade da classificação (3). O método um-conta-um para classificar k classes utiliza $\frac{k(k-1)}{2}$ SVMs de resultados binários. No treinamento, cada classificador fica responsável por separar um par de classes, os padrões de classes diferentes são ignorados. Na classificação, o padrão não rotulado passa por todos os SVMs e a classe escolhida é aquela que tiver mais indicações do classificador (3).

O DAG é semelhante ao método um-contra-um, também apresentando $\frac{k(k-1)}{2}$ SVMs a principal diferença é a sua estruturação em árvore, a classe escolhida por cada nó irá determinar o caminho de classificação de um vetor desconhecido. O resultado será dado pelo último nó da árvore. A vantagem do DAG é a redução no número de nós que precisam ser usados, sendo somente k nós (3). A Figura 2.21 apresenta o funcionamento do DAG.

Figura 2.21 – Árvore do método DAG.



Fonte: retirado de (38).

Contudo, nestes métodos, cada SVM trabalha de forma isolada. Assim, a qualidade do sistema total irá depender da qualidade de cada SVM. Caso um produza algum

erro, este pode ser propagado, resultando em uma classificação incorreta (3).

2.2.6 Vizinho Mais Próximo

O método de classificação KNN é um classificador supervisionado. Contudo, apresenta etapa de treinamento com uma abordagem alternativa. Nesta etapa, não são realizados procedimentos matemáticos para determinar uma função, que irá criar as regiões de decisão. A etapa de treinamento é simplesmente o armazenamento dos padrões rotulados, que serão utilizados na etapa de classificação com os novos vetores de características a serem classificados. Esta ideia traz a vantagem de reduzir os custos computacionais na classificação (2).

Outra vantagem do KNN é a possibilidade da criação das regiões de decisão complexas. Ou seja, as classes não precisão estar bem separadas em grupos isolados. Por exemplo, este método consegue classificar uma classe espalhada pelo plano em vários pequenos grupos. Isso permite uma maior adaptação a distribuição dos padrões (2).

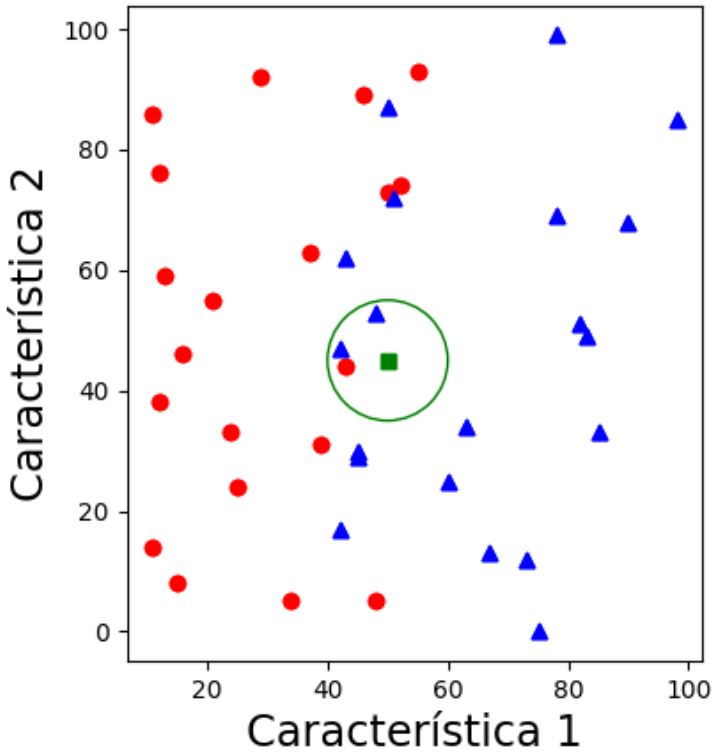
Já na etapa de classificação, o novo vetor de características não rotulado é comparado a todos os outros padrões rotulados na fase de treinamento. O objetivo é encontrar as k amostras mais próximas do vetor não rotulado. A classe que o classificador irá indicar para este novo padrão será a que possuir maior ocorrência dentro deste conjunto de k amostras (2).

Esta distância deve ser calculada entre todos os padrões rotulados até o não rotulado. Para o KNN, ao utilizar um grande número de vetores, é necessário um aumento no recurso computacional para realizar estes cálculos, podendo tornar a utilização deste classificador inviável ou pouco eficiente com base no tempo de execução do algoritmo (2).

Esta característica diverge do ANN e SVM. Nestes, o aumento dos vetores de características utilizados no treinamento, aumenta a precisão das fronteiras de decisão (2).

A Figura 2.22 apresenta um exemplo de classificação utilizando o KNN, para classificar o vetor de características representado pelo ponto quadrado. Utiliza-se $k = 3$, então são analisados os 3 vetores de características mais próximos, destacados na imagem, estando na circunferência centrada no padrão representado pelo símbolo quadrado. Analisando a maior frequência das classes presente nesse conjunto de três pontos, é notada a classe representada pelo símbolo triângulo. Logo, o classificador irá definir a classe do vetor de característica como sendo igual à dos padrões representados pelo triângulo.

Figura 2.22 – Exemplo de classificação KNN utilizando k=3.



Fonte: O autor.

2.2.7 Rede Neural Artificial

O modelo de aprendizado por ANN utiliza o conceito de computação não linear, que tenta criar soluções para problemas algébricos não lineares. Este modelo apresenta uma estrutura onde os dados passam por redes de operações matemáticas, sendo um processo muito similar ao realizado pelos neurônios no cérebro humano (1). A ANN leva vantagem para resolver problemas descontínuos e não lineares com resultados de alta acurácia e adaptabilidade (39).

2.2.7.1 Perceptron

O modelo ANN mais básico é o Perceptron, desenvolvido por Rosenblatt em 1958. Este modelo consegue diferenciar duas classes por uma função linear (1).

O processo de classificação se inicia com a entrada sendo um vetor de características de tamanho n. Cada elemento deste vetor é multiplicado por um peso e os resultados são somados. Estes pesos são definidos no processo de treinamento. Este cálculo é equivalente a uma soma ponderada, apresentada na Equação 2.36 (1).

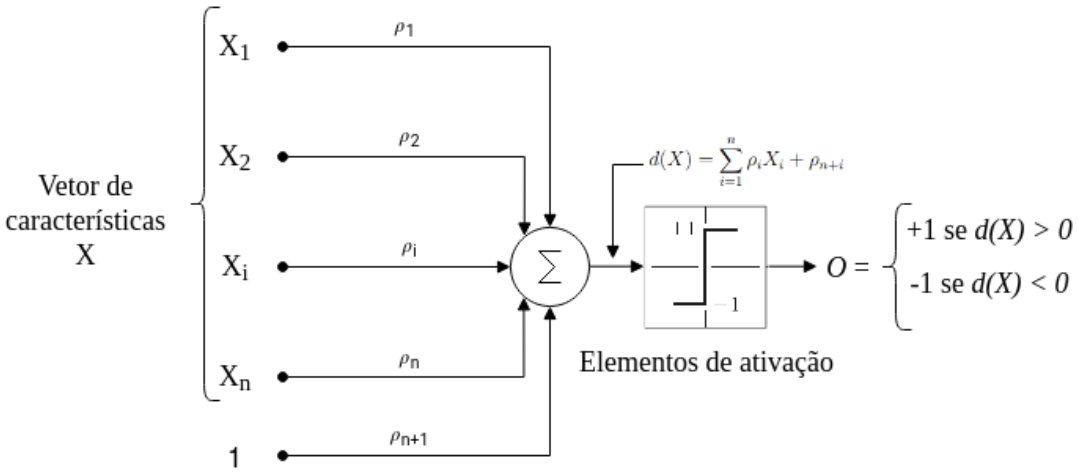
$$d(X) = \sum_{i=1}^{n} \rho_i X_i + \rho_{n+1} \tag{2.36}$$

em que:

- $\{X_1, X_2, ..., X_n\}$ é o vetor com n características;
- ρ é o peso a ser aplicado a cada característica.

O resultado da somatória passa por um processo de limiarização, ou ativação, limitando a saída a duas soluções possíveis. Se o valor do somatório for positivo, a limiarização resulta em +1, caso contrário, este valor é definido como -1. No treinamento, são utilizadas duas classes e cada resultado do Perceptron corresponde a uma classe. A Figura 2.23 é uma possível representação do processamento do Perceptron na classificação (1).

Figura 2.23 – Representação do processo de classificação realizado pelo Perceptron.



Fonte: adaptado de (1).

2.2.7.2 Redes Neurais Multicamadas

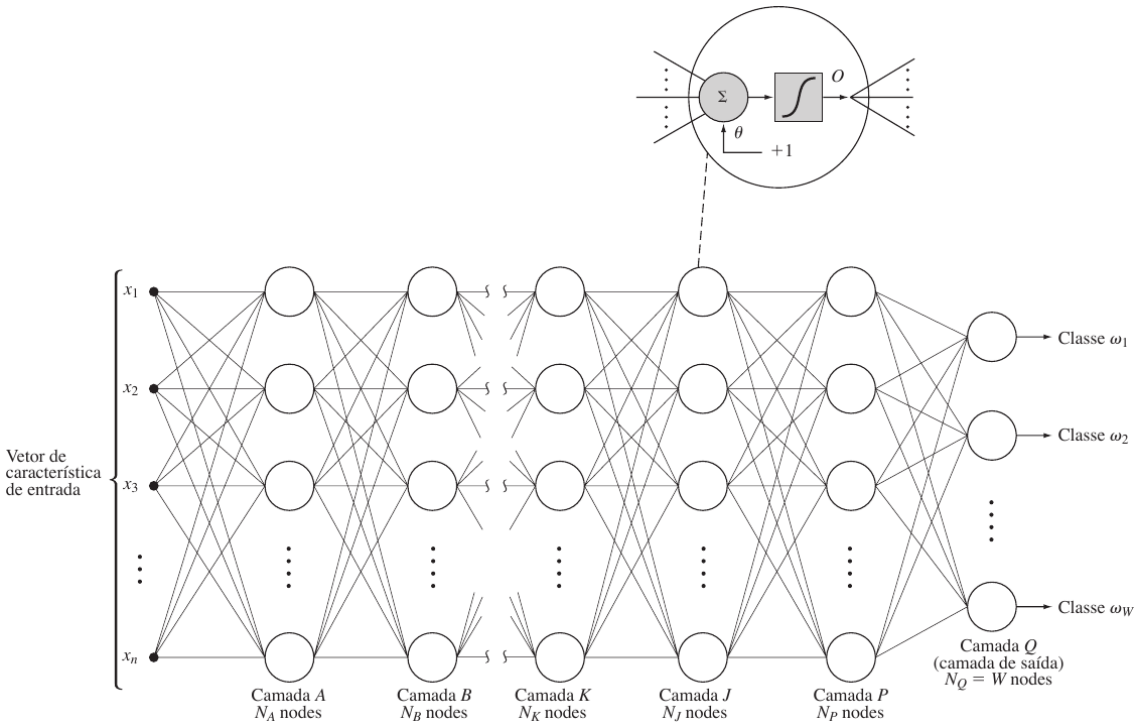
O classificador de redes neurais multicamadas (*multi-layer perceptron* - MLP) é a evolução do Perceptron. Sua superioridade é destacada na possibilidade de classificar um vetor em várias classes, ao invés de somente duas, permitindo a criar uma região de decisão não linear. Isso é possível pela sua concepção de reproduzir várias vezes o modelo do Perceptron (1).

As redes neurais multicamadas, como o nome sugere, utilizam L camadas que possuem N_i neurônios cada, com $i \in \{1, 2, \ldots, L\}$. $N_1 = n$, em que n é o número de

elementos do vetor de entrada. Cada neurônio recebe todos os elementos do vetor e realiza uma soma ponderada que é limiarizada para se tornar a saída do neurônio. Portanto, cada camada produz N_i resultados, ou seja, cada um dos N_i neurônios realiza um processo semelhante ao realizado pelo Perceptron (1).

Os n resultados da primeira camada irão alimentar a entrada da segunda camada. Este processo se repete até chegar na última camada, a qual deve possuir o número de neurônios igual ao número de classes. Ou seja, na última camada cada neurônio representa uma classe e sua função é apresentar um resultado binário para a pergunta "Este vetor de característica pertence à sua classe?" (1). A resposta esperada dos neurônios na última camada a esta pergunta deve conter somente uma afirmação, indicando que o vetor pertence a classe correspondente ao neurônio. A Figura 2.24 apresenta a estrutura da redes neurais multicamadas. Nesta também é apresentada a estrutura interna de cada neurônio (1).

Figura 2.24 – Estrutura de redes neurais multicamadas.



Fonte: adaptado de (1).

Existem funções que podem ser aplicadas no resultado de cada neurônio na etapa de classificação que podem ajudar a melhorar a adaptação da rede para cada tipo de aplicação. A Quadro 2.2 apresenta algumas destas funções.

Quadro 2.2 – Funções de ativação.

Ativação	**Equação**
Identidade	$f(x) = x$
Sigmoid Simétrica	$f(x) = \frac{\beta \times (1 - e^{-\alpha x})}{(1 + e^{-\alpha x})}$
Gaussiana	$f(x) = \beta e^{-\alpha x \times x}$
ReLU	$f(x) = \max(0,x)$
Leaky ReLU	para $x > 0$ $f(x) = x$, e para $x \leq 0$ $f(x) = \alpha x$

Fonte – ver referência (40).

2.2.7.3 Treinamento do MLP

Redes neurais multicamadas apresentam um treinamento cujo objetivo é definir o valor dos pesos. A qualidade deste classificador é orientada pela competência destes pesos em conseguir criar as regiões de fronteira bem definidas, conseguindo agrupar todos os objetos de uma mesma classe em uma mesma região. Este classificador pode utilizar uma grande quantidade de pesos, no entanto é verificado que os pesos anteriores à última camada são os que apresentam o ajuste mais árduo.

A etapa de treinamento consiste em definir valores de pesos que irão cada vez mais reduzir o erro de classificação dos vetores de treinamento. Quanto maior for a quantidade de vetores de características utilizados no treinamento, menor vai ser o erro de classificação e os pesos irão convergir para valores cada vez mais precisos. O erro de treinamento é obtido analisando os resultados dos neurônios na última camada. A classe de um objeto é definida pegando-se o neurônio que gerou o maior valor na última camada. Então, espera-se que exista somente um neurônio de valor alto. Pode-se considerar um valor $\geq 0{,}95$ como alto e um valor $\leq 0{,}05$ como baixo. Estes valores são utilizados para definir os pesos, com o objetivo de deixar o valor da classe correta cada vez mais alto ao mesmo tempo que conduz os outros a valores mais baixos.

Os pesos só serão modificados na fase de treinamento. Terminada esta fase, estes valores serão utilizados na etapa de classificação de padrões desconhecido. Caso algum resultado nesta fase apresente mais de um valor alto na saída, denota-se uma dúvida do classificador a qual este padrão deve pertencer. Nesta caso, existe duas possibilidades, na primeira, o classificador pode declarar um erro na classificação, na segunda, ele pode simplesmente escolher a classe que obteve o valor mais alto.

Na literatura, não existe uma regra clara para a definição do número de neurônios que a rede deve possuir. A primeira camada tem número de neurônios igual ao tamanho

do vetor de características e a última camada tem a quantidade de neurônios igual a quantidade de classes. No entanto, as camadas intermediárias não podem possuir qualquer quantidade de neurônios. O melhor número por camada irá gerar a maior quantidade de acertos, sendo possível obtê-lo depois da realização de sucessivos testes.

A etapa de treinamento das redes neurais tem como objetivo definir o valor dos ρ de cada neurônio. O algoritmo que define estes pesos é chamado de *backpropagation*, pois ele tem como entrada a saída da rede $\hat{y}$, usando este valor para atualizar a camada anterior (41).

2.3 Técnicas para análise dos resultados

A saída do classificador é a classe identificada, e este resultado pode estar certo ou errado. Para avaliar um classificador deve-se realizar várias classificações com diferentes padrões e comparar o resultado predito com o seu rótulo verdadeiro. Esta comparação pode ser melhor visualizada em uma matriz de confusão. Esta é uma tabela que apresenta as predições obtidas, permitindo ver a quantidade de padrões classificados erroneamente e quais classes são confundidas. A Quadro 2.3 apresenta um exemplo de matriz de confusão para as seis classes deste trabalho. Nesta tabela podemos ver que a classe R$ 2,00 é a que apresenta a menor quantidade de erros e a classe R$ 50,00 está se confundindo com a classe R$ 20,00.

Tabela 2.1 – Exemplo de matriz de confusão para seis classes e 96 padrões.

Classe Real	**Classe Prevista**					
	R$ 2,00	R$ 5,00	R$ 10,00	R$ 20,00	R$ 50,00	R$ 100,00
R$ 2,00	15	0	0	0	0	1
R$ 5,00	0	14	1	1	0	0
R$ 10,00	0	5	10	0	0	1
R$ 20,00	2	0	2	12	0	0
R$ 50,00	1	0	1	7	7	0
R$ 100,00	2	0	0	0	2	12

Fonte – Produzido pelo autor.

2.3.1 Mensurando desempenho dos classificadores

A qualidade de um classificador pode ser expressa na forma de métricas como acurácia, precisão e sensibilidade. Mas antes é importante definir o conjunto do espaço amostral resultante do classificador, sendo

- Verdadeiros positivos (VP): são os padrões com rótulos preditos iguais aos reais, ou seja, as predições corretas;
- Verdadeiros negativos (VN): tendo uma classe como referência, são os padrões que não foram preditos para esta classe e seu rótulo real não é igual a esta classe;
- Falsos positivos (FP): tendo uma classe como referência, são os padrões que são classificados para esta classe, mas não são realmente pertencente a ela;
- Falsos negativos (FN): tendo uma classe como referência, são os padrões que realmente pertencem a esta classe, mas foram rotulados como sendo pertencentes a outra.

A Quadro 2.3 mostra quais rótulos vão pertencer ao conjunto VP, VN, FP e FN, tendo como referência a classe R$ 5,00. Para esta classe tem-se um total de VP=14, VN=75, FP=5 e FN=2 (42) (43).

Quadro 2.3 – Definindo VP, VN, FP e FN, tendo como referência a classe R$ 5,00.

Classe Real	**Classe Prevista**					
	R$ 2,00	R$ 5,00	R$ 10,00	R$ 20,00	R$ 50,00	R$ 100,00
R$ 2,00	VN	FP	VN	VN	VN	VN
R$ 5,00	FN	VP	FN	FN	FN	FN
R$ 10,00	VN	FP	VN	VN	VN	VN
R$ 20,00	VN	FP	VN	VN	VN	VN
R$ 50,00	VN	FP	VN	VN	VN	VN
R$ 100,00	VN	FP	VN	VN	VN	VN

Fonte – Produzido pelo autor.

2.3.1.1 Acurácia

A forma mais simples de definir uma pontuação para o classificador é ver a porcentagem de acertos gerados. Esta métrica é chamada de acurácia e é definida pela Equação 2.37 (44). O valor da acurácia do exemplo é 0,73.

$$acuracia(y,\hat{y}) = \frac{1}{n_{\text{amostras}}} \sum_{i=0}^{n_{\text{amostras}}-1} 1(y_i = \hat{y}_i) \tag{2.37}$$

em que:

- n_{amostras} é o total de padrões classificados;
- y é a classe real do padrão;
- $\hat{y}$ é a classe predita pelo classificador.

2.3.1.2 Precisão

A precisão é uma medida obtida para cada classe que mede o percentual das predições corretas para esta classe. Ou seja, quanto maior a precisão, menor a quantidade de padrões que foram rotulados para esta classe e que pertencem na realidade a outra classe (41).

$$precisao = \frac{\text{VP}}{\text{VP} + \text{FP}} \tag{2.38}$$

A precisão para as classes do exemplo são 0,94 para R$ 2,00; 0,88 para R$ 5,00; 0,62 para R$ 10,00; 0,75 para R$ 20,00; 0,44 para R$ 50,00 e 0,75 para R$ 100,00. Para obter uma precisão geral, deve ser feita uma média ponderada. No exemplo, os pesos são o valor do rótulo das classes. Assim a precisão do classificador do exemplo é 0,76 (44).

2.3.1.3 Sensibilidade

A sensibilidade ou *recall* é uma medida obtida para cada classe dada pela Equação 2.39 que mensura a quantidade de acertos em relação à quantidade de padrões da classe de referência mas que foram rotulados para outras classes (41).

$$sensibilidade = \frac{\text{VP}}{\text{VP} + \text{FN}} \tag{2.39}$$

A sensibilidade para as classes do exemplo são 0,75 para R$ 2,00; 0,74 para R$ 5,00; 0,71 para R$ 10,00; 0,6 para R$ 20,00; 0,78 para R$ 50,00 e 0,86 para R$ 100,00. Para obter uma sensibilidade geral, deve ser feita uma média ponderada. No exemplo, os pesos são o valor do rótulo das classes. Assim a sensibilidade do classificador do exemplo é 0,73 (44).

2.3.2 Validação cruzada

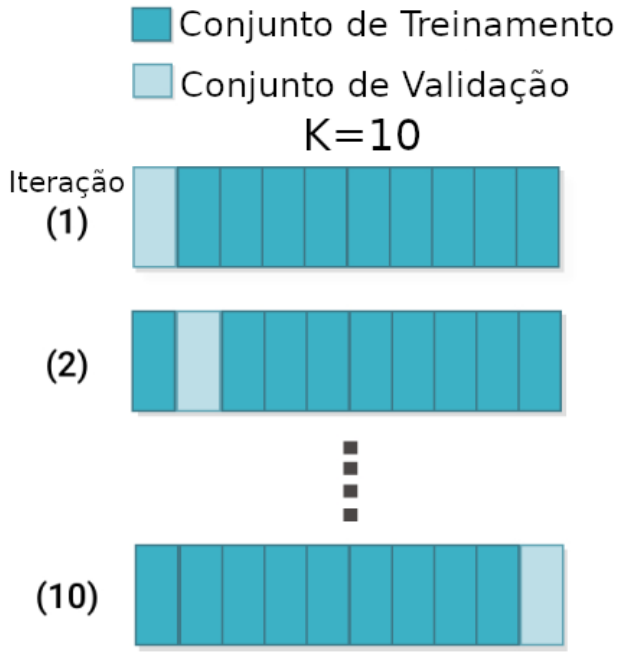
Para avaliar um classificador é necessário mandá-lo classificar vetores de características com rótulos conhecidos para comparar com a saída do classificador. E estes vetores não podem ter sido usados na fase de treinamento. Desta forma, deve-se dividir o banco de dados em dois grupos, um reservado para a fase de treinamento e outro para a fase de classificação.

Contudo, se sempre usarmos os mesmos conjuntos para treinar e classificar, pode ocorrer dos resultados não serem confiáveis, pois o resultado pode ficar dependente dos conjuntos de treinamento e classificação, e se estes são mudados o resultado também pode mudar drasticamente. Por isso é recomendado primeiramente usar um conjunto para treinar e outro para classificar e depois inverter estes conjuntos, somando os resultados destas etapas. Assim cada padrão será usado em um momento no treinamento e em outro na classificação. Este método é chamado de validação cruzada (*cross-validation*) (41).

Existem algumas variações para a validação cruzada. O *cross-validation k-fold* divide o banco de dados em k conjuntos e realiza k iterações. A cada iteração, um conjunto é treinado e os $k-1$ restantes são usados na classificação, Figura 2.25. Um dos valores de k mais utilizado é 5, para poder treinar 80% dos dados e classificar 20% a cada iteração (41).

Outro *cross-validation* também utilizado é o *leave-one-out* que a cada iteração treina todos os pontos, excluindo somente um, que será usado na classificação, ou seja, é um *cross-validation* com k igual à quantidade de vetores de características. A vantagem deste processo é usar quase todos os pontos para o treinamento, ajudando o classificador a melhorar a separação entre as classes (41).

Figura 2.25 – Cross-validation *k-fold* com k=10.



Fonte: adaptado de (45).

2.4 Considerações Finais

A construção do aplicativo está intimamente ligada com a qualidade esperada do percentual de acertos que o mesmo terá na sua utilização. Desta forma, ter amplo conhecimento dos fundamentos das áreas de processamento de imagem, e reconhecimento de padrões é fundamental para garantir o bom funcionamento do aplicativo.

O processamento de imagem é utilizado para melhorar a qualidade da imagem, removendo ruídos e outras características indesejadas, com o objetivo de torná-la mais bonita. No entanto, quando usada junto com o reconhecimento de objetos, a sua aparência visual deixa de ser importante. Neste caso, exaltar certas caraterísticas e tornar a informação contida na imagem mais útil torna-se o objetivo. Com isso, foram apresentados os princípios do processamento digital de imagens, bem como algumas transformações que podem ser necessárias, como a filtragem gaussiana.

Também foram apresentadas algumas opções de classificadores. O classificador SVM apresenta uma construção matemática para separar os padrões. Apresenta a opção de construção linear, linear de margem suave ou não linear. O KNN tem uma modelagem mais simples, onde a classificação analisa somente os padrões mais próximos (vizinhos)

do vetor a ser classificado. Seu treinamento consiste apenas em uma armazenagem dos vetores de treinamento, não realizando nenhum procedimento matemático para construção da fronteira de decisão. A classificação por ANN é composta por uma soma ponderada das características dos vetores utilizando pesos ρ_i. Cada somatória é chamada de neurônio. O ANN é composto por uma grande quantidade de neurônios. Os valores dos pesos são decididos e refinados na fase de treinamento.

A construção do aplicativo tem como requisito utilizar o melhor classificador, que deve ser escolhido usando métricas que validem a confiabilidade das classificações. Estas podem ser a acurácia, sensibilidade e a precisão.

Para poder aplicar a teoria descrita neste capítulo, é necessário a criação de algoritmos de processamento de imagem, treinamento, e classificação dos três métodos discutidos (SVM, KNN e ANN) em uma linguagem de programação. Ainda, deve existir a possibilidade de inserir estes códigos em um aplicativo Android. A solução para estas questões é discutida no próximo capítulo.

3 Materiais e Métodos

3.1 Método

A pesquisa necessária para que o sistema seja desenvolvido envolve várias áreas da computação, principalmente o processamento de imagens. A primeira etapa é a obtenção e análise de um banco de dados contendo as imagens que serão utilizadas na etapa de treinamento dos classificadores. Estas imagens devem apresentar o melhor processamento possível, que facilite a extração de suas características, facilitando o trabalho dos classificadores e aumentando a porcentagem de acertos. Para isso, é necessário aplicar uma fase de testes a fim de comprovar a melhor combinação de técnicas de processamento de imagens.

A validação do resultado final passa pela elaboração do algoritmo classificador que será incorporado ao aplicativo. Desta forma, uma etapa de teste deve ser realizada em primeira instância para avaliar o melhor classificador, desde já limitando-se ao teste de somente três (SVM, ANN e KNN). A escolha destes se deve a sua conhecida e esperada geração de resultados com grande índices de acertos, principalmente no caso específico do ANN, este apresentando ainda uma maior adaptação a qualquer tipo de aplicação, tornando-o bastante utilizado dentro da comunidade científica (46). Já o SVM e o KNN se destacam pelo baixo uso de recursos computacionais, porém mantendo bons resultados. Após a escolha do melhor classificador, existe outra etapa de testes referentes a configurações e variações pertencentes aos próprios classificadores (47).

O processo de desenvolvimento do aplicativo pode ser dividido em cinco estágios de produção. No primeiro, serão feitos os principais tratamentos da imagem capturada pela câmera, como a segmentação e aplicação de filtros para melhorar e exaltar certas caraterísticas da imagem contendo a cédula.

No segundo estágio, as características que estão relacionadas com cor, textura ou forma são selecionadas e extraídas das imagens presentes no banco de dados. Para uma boa classificação, neste estágio deve-se ainda verificar se as características escolhidas apresentam valores únicos para cada uma das seis classes.

No terceiro estágio, já possuindo as características extraídas do segundo estágio, são construídas diferentes opções de vetores de características. Para cada opção de construção dos vetores, é realizado seu treinamento e classificação, usando os três classifi-

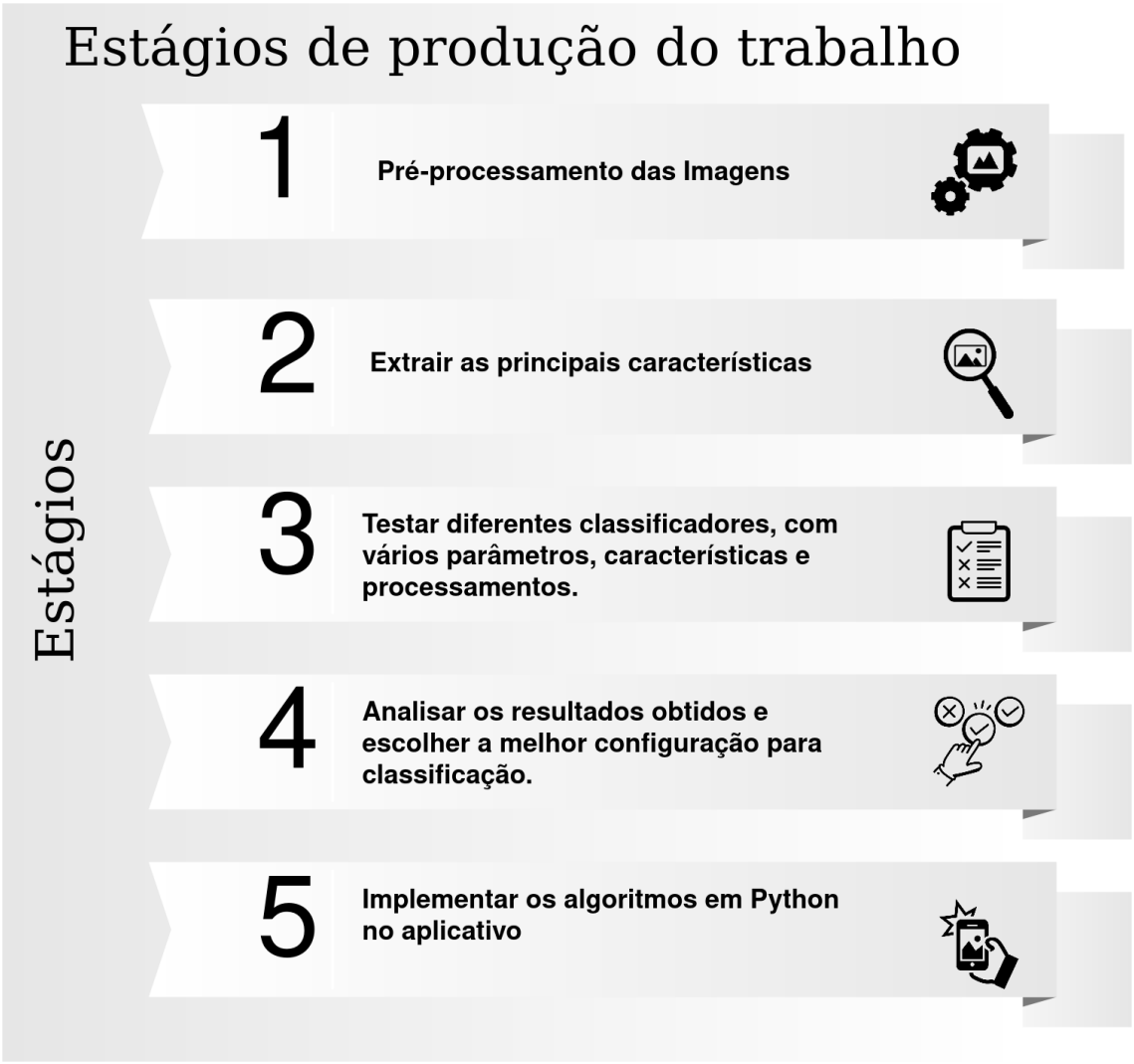
cadores: SVM, KNN e ANN. Também é necessário realizar testes para validar os melhores parâmetros dos classificadores.

No quarto estágio, os resultados obtidos no terceiro estágio são analisados e é feita a escolha das melhores características, do melhor classificador e a sua melhor configuração.

No quinto estágio, o classificador final é treinado e inserido no aplicativo para realizar a classificação das cédulas fotografadas pelo aparelho celular.

A Figura 3.1 apresenta os cinco estágios do processo de criação do aplicativo.

Figura 3.1 – Estágios de produção do trabalho.

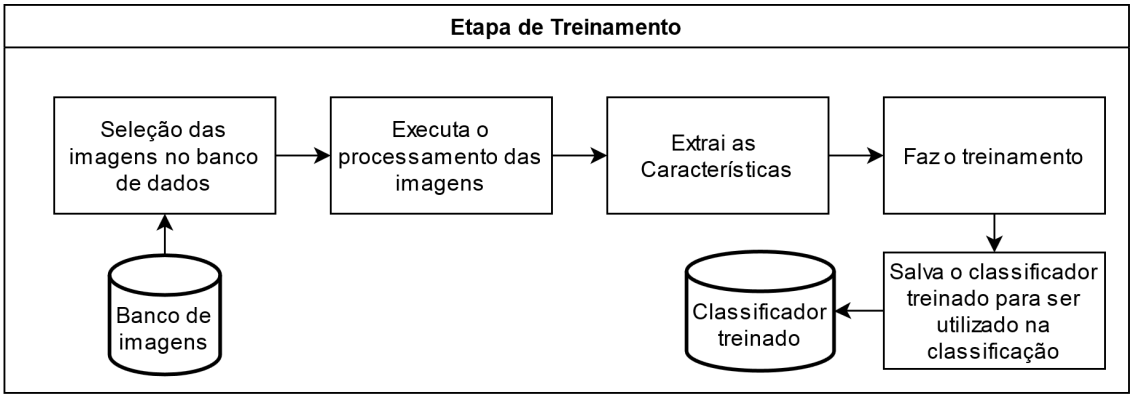


Fonte: o autor.

A Figura 3.2 apresenta o processo realizado na etapa de treinamento. Primeiro, são selecionadas as imagens do banco de dados, elas passam por um processamento de imagem que tem como objetivo acentuar as características. Estas são extraídas e utilizadas no treinamento. O classificador treinado é salvo em um arquivo para pode ser utilizado

posteriormente na etapa de classificação.

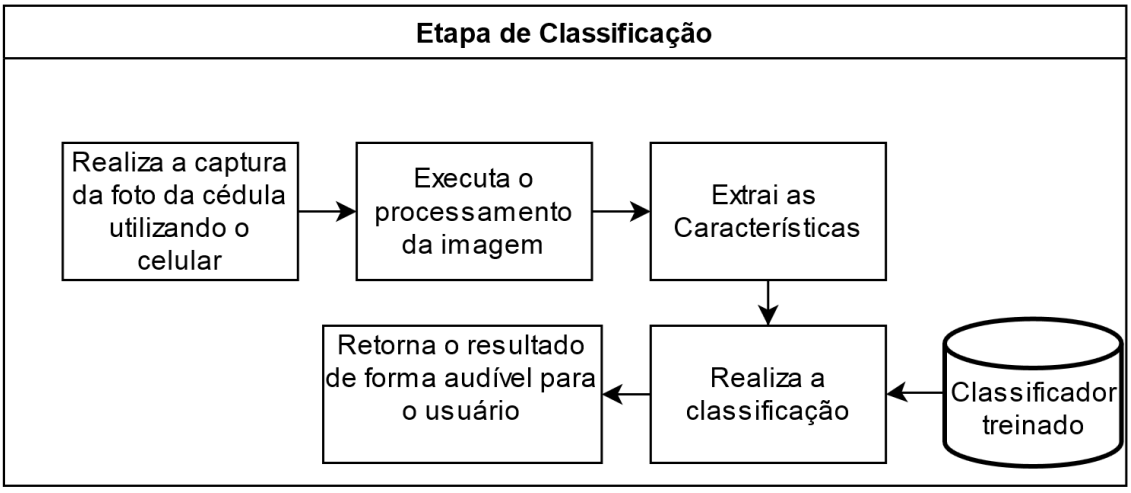
Figura 3.2 – Diagrama da etapa de treinamento.



Fonte: o autor.

A Figura 3.3 apresenta o processo realizado durante a etapa de classificação. Primeiramente, a foto da cédula é capturada pelo celular e são aplicados os mesmos processamentos de imagem e extração de características da etapa de treinamento. A classificação é realizada utilizando o classificador treinado. O último passo é retornar para o usuário, de forma sonora, o valor da cédula que o classificador identificou.

Figura 3.3 – Diagrama da etapa de classificação.



Fonte: o autor.

3.2 Materiais

O aplicativo foi desenvolvido utilizando a linguagem de programação Python, pois essa linguagem passa na frente quando o tema é bibliotecas para análise de imagens

(48)(49). Entretanto, é necessário usar uma ferramenta de conversão *bytecode* para converter o programa compilado em Python para Java, com o fim de conseguir usá-lo em uma aplicação Android. Para esta função, existe uma biblioteca do Python chamada Kivy que permite a utilização desta linguagem para a criação de aplicações móveis baseadas nesta linguagem (50).

A escolha da linguagem de programação foi muito importante durante o início do projeto, visto que para a utilização dos outros recursos, se resumem em encontrar bibliotecas presentes na linguagem construídas em cima da teoria estudada. Por conseguinte, Python foi a escolha feita devido a sua potencialidade em executar de maneira satisfatória as principais técnicas estudadas neste projeto.

Partindo da linguagem Python, o estudo e escolha dos principais mecanismos presentes no tema de processamento digital de imagens que foram utilizados se reúnem em primeiro nas propriedades da imagem como a cor, a qual é normalmente analisada utilizando um histograma sendo um gráfico distributivo de cores. A textura é outra característica muito utilizada e efetivo na diferenciação de imagens. Adicionalmente, ainda se tem a possibilidade de utilizar o contorno de objetos contidos nas imagens (16).

Pode ser observado que algumas dessas características, mesmo sendo de interesse para a classificação, quando são extraídas fornecem uma informação com uma grande margem de erro ou com pouca caracterização de valores esperados. Isto é um sinal para antes de utilizar estes valores, é necessário aplicar um ou mais filtros, com a finalidade de remover ruídos e aumentar distância dos valores que representam as características desejadas. Dentro dos tipos de filtros de imagens utilizados ou estudados durante este projeto, pode-se citar o Gaussiano, Mediana e Bilateral.

A linguagem Python possui bibliotecas que podem lidar com imagens, possuindo funções que ajudam a aplicar vários filtros bem como na extração de várias características da imagem. A biblioteca escolhida para este projeto é a OpenCV-Python (51).

Ainda se deve atentar ao fato da execução deste código Python em aplicativos Android. Felizmente, existem várias ferramentas disponibilizadas com capacidade de executar essa etapa do desenvolvimento, como o Kivy (52).

3.2.1 Python

Python é uma linguagem de programação que teve seu início na década de 90, figurando hoje entre as três linguagens de programação mais usadas (53). É uma linguagem interpretada, ou seja, necessita de um programa chamado interpretador para analisar o

código fonte que é facilmente compreendido pelo programador e interpretar de maneira que a CPU possa executar as ações descritas no código. No que diz respeito ao tratamento de dados, a verificação dos tipos de variáveis é feita de forma dinâmica, enquanto o programa está em execução, com tipificação forte, ou seja, para que o interpretador possa atribuir um valor a uma variável, este valor tem que corresponder ao mesmo tipo predefinido da variável (54).

A escolha da linguagem Python para a criação de projetos pode ser baseada nas suas características de otimização no desenvolvimento do código, mantendo-o limpo, retirando caracteres de referência de linha ou subconjunto desnecessários e substituindo-os por outros como ":" e adicionando um verificador de alinhamento vertical. Um ponto fundamental da sua característica é apresentar extensa documentação. O fato de estar entre as mais usadas indica uma grande comunidade que sempre está a compartilhar informações e códigos na *internet*. Esse fato também gera interesse de aptos conhecedores da linguagem na criação de materiais de estudo para programadores iniciantes ou experientes (55) (56) (57).

Python apresenta uma grande variedades de bibliotecas de suporte que atendem várias áreas de atuação. Dentre esta variedade, os pacotes usados neste projeto incluem Numpy, para a manipulação matemática de vetores (*array*). OpenCV é talvez a biblioteca mais conhecida tanto em Python como em Java para a manipulação de imagens, sendo responsável pelo reconhecimento de características e modificações de imagens. Entretanto, o pacote fundamental para qualquer projeto de reconhecimento de objetos em Python é o Scikit-learn, possuindo funções baseadas nos principais métodos de reconhecimento e várias funções auxiliares tanto para obter uma melhor visão dos resultados como para tratar os dados de entrada extraídos da imagem (58) (44) (37).

3.2.1.1 Biblioteca Scikit-learn

No que diz respeito a Python e aprendizado de máquina, a biblioteca Scikit-learn é a mais popular e acessível, sendo *open-source*. Esta biblioteca é a mais usada no meio acadêmico e no industrial. Ela é uma *interface* de algoritmos otimizados, que ajudam tanto na melhora dos resultados de classificação como na facilidade de implementação destes códigos (59). A Scikit-learn é um projeto que começou em 2010, com o objetivo de implementar algoritmos de ML, como classificadores, modelos de avaliação da informação usada na classificação, validação cruzada dos resultados para treinar os classificadores usando várias rodadas para aperfeiçoar os parâmetros utilizados, entre outros (44).

3.2.2 Biblioteca OpenCV

A biblioteca OpenCV apresenta uma grande quantidade de ferramentas de processamento de imagem, extração de características e reconhecimento de objetos, com vários exemplos compartilhados pelo OpenCV e pela comunidade. Ela também possui algoritmos dos classificadores SVM, KNN e MLP.

O SVM do OpenCV implementa o algoritmo SVC, que permite opções de configuração do algoritmo como *kernel* (Linear, Polinomial, Exponencial CHI2, RBF, Sigmóide e Interseção de histograma) e valores do C, γ e *degree*. O KNN do OpenCV também permite configurações, sendo a principal a escolha do k. As configurações do MLP são referentes à escolha da função de ativação (Linear, Gaussiana, Sigmóide, ReLU e Leaky ReLU), metodos de treinamento (Backpropagation, RPROP e Recozimento simulado), e valores do número de camadas interiores e quantidade de neurônios de cada uma, de α e β.

Algumas vantagens do OpenCV é a execução de códigos compilados em C, o que aumenta a velocidade de execução e o tratamento da informação (imagens, vetores de características) utilizando algoritmos de processamento que usam teorias de álgebra matricial para aumentar a velocidade de processamento. Também salva os classificadores treinados em formato xml, permitindo conferir a teoria sendo aplicada na prática. Por exemplo, é possível ver que o arquivo xml do KNN é formado por todos os vetores inseridos no treinamento.

3.2.3 Biblioteca Kivy

A biblioteca Kivy é um *framework* que permite criar aplicativos para computadores (Windows, macOs e Linux) e para dispositivos móveis (iPad, iPhone, *tablets* e sistemas Android) utilizando a linguagem Python (50). Assim, a ferramenta Kivy é mais do que suficiente para a realização deste projeto.

Outro motivo para a utilização do Kivy foi possuir um grande suporte e conteúdo *online* para o desenvolvimento de projetos que o empregaram. O próprio projeto Kivy recomenda algumas ferramentas compatíveis com a sua *interface* que conseguem fazer esta conversão. Essas são o Kivy Laucher, aplicativo disponível na GooglePlay capaz de abrir projetos Kivy e executá-los instantaneamente no dispositivo, o python-for-android, sendo um pacote de ferramentas para aplicativos Python no Android, e o Buildozer, que cria pacotes de aplicativos de maneira fácil usando como suporte o projeto python-for-

android e Android SDK (60)(61).

Uma desafio na utilização do python-for-android é a conversão de bibliotecas Python, pois o python-for-android necessita de um receita especifica de cada biblioteca. O projeto já disponibiliza uma grande quantidade de receitas como a do Kivy, OpenCV e Pillow, mas não existe para o Scikit-learn. Isso impossibilita a utilização desta biblioteca dentro do aplicativo (62). No entanto, os classificadores do Scikit-learn e OpenCV foram testados e comparados e seus resultados se mostraram equivalentes, o que indica que a utilização dos classificadores da biblioteca OpenCV não irá gerar uma perda na qualidade dos resultados.

3.3 Considerações Finais

O processo da construção do aplicativo se inicializa com a obtenção e processamento das imagens das cédulas monetárias. Depois, é necessário extrair características relevantes para a diferenciação das cédulas, que são usadas no treinamento do classificador. Com o classificador treinado, é realizada a classificação de uma grande quantidade de imagens para avaliar a confiabilidade do classificador.

Para realizar todos estes procedimentos, foram utilizadas ferramentas de programação e compilação como Python, Kivy, Buildozer, OpenCV, e Scikit-learn, além do dispositivo móvel Android para testar o aplicativo.

Contudo, existem diversas opções de técnicas e configurações que podem ser utilizadas para melhorar a classificação. Assim, é necessário realizar uma grande bateria de testes para averiguar as opções que produzam o melhor resultado possível. Os resultados já obtidos de uma primeira bateria de testes bem como os resultados esperados do classificador final são apresentados no próximo capítulo.

4 Desenvolvimento

Neste capítulo são apresentadas as etapas realizadas durante o projeto até chegar na construção final do aplicativo. São apresentados a forma em que foi construído banco de dados com as imagens das notas e os processamentos de imagens testados com o objetivo de extrair a melhor característica da imagem. Também são descritos os códigos utilizados para extrair as de cor utilizando histograma, informações de contorno usando SIFT e usando a imagem completa como vetor de características.

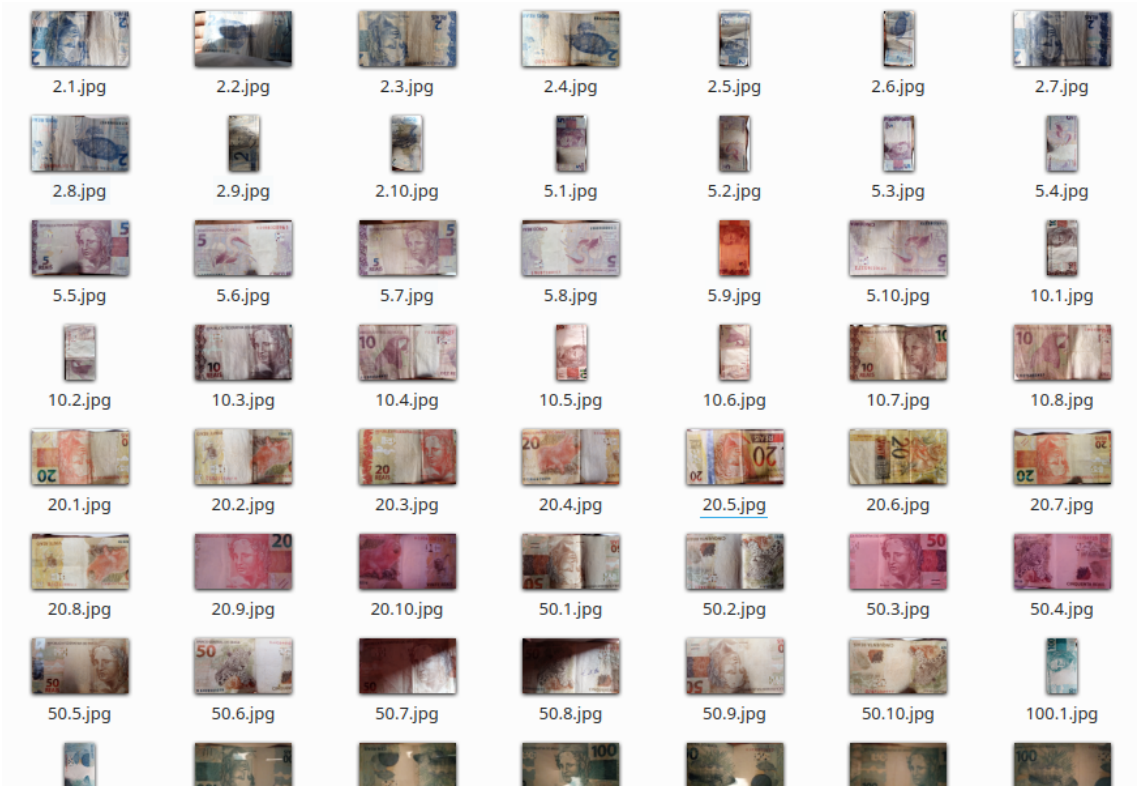
4.1 Pré processamento das Imagens

Foram utilizadas 300 imagens para construir o banco de dados para realizar o treinamento dos classificadores e para analisar sua eficiência. As imagens das cédulas de R$ 2,00; R$ 5,00; R$ 10,00; R$ 20,00; R$ 50,00 e R$ 100,00 foram geradas em quantidades iguais. A grande quantidade de imagens tem como objetivo incluir mudanças sutis que possar vir a ser presenciadas no uso do aplicativo, como a iluminação do ambiente, orientação da nota e defeitos presentes em notas antigas. A Figura 4.1 apresenta algumas imagens deste banco. As notas de referência para testar as opções de processamento são vistas na Figura 4.2, onde é apresentada uma nota para cada valor presente no banco de dados.

Cada celular pode possuir câmeras que possam a vir gerar imagem de dimensões diferentes, o que atrapalharia por exemplo a utilização do histograma como característica, pois imagens maiores teriam um número de *pixels* maior que outras. Por isso é realizado um redimensionamento da imagem para a dimensão 854x480, sendo proporcional ao tamanho das notas.

Também se viu a necessidade de rotacionar as imagens de forma a garantir que as notas estejam estejam na posição correta. No uso do aplicativo também é possível que sejam capturadas imagens de notas de viradas de cabeça para baixo, por isso a cada imagem do banco de dados são gerados duas em que uma está rotacionada 180° em relação à outra. Este rotacionamento também ajuda a duplicar o tamanho do banco de dados.

Figura 4.1 – Algumas imagens que estão presentes no banco criado pelo autor.



Fonte: o autor.

Figura 4.2 – Notas de R$ 2,00;R$ 5,00;R$ 10,00;R$ 20,00;R$ 50,00 e R$ 100,00 de referência para realizar os processamentos de imagem.

Fonte: o autor.

4.1.1 Convertendo para HSV

A transformação da imagem RGB para HSV permite separar as características de cores da imagem da saturação e da luminosidade, reduzir as influências indesejadas da mesma. As Figuras 4.3, 4.4 e 4.5 mostram os canais *hue*, saturação e luminosidade

extraído das notas usando o Código 4.1.

Código 4.1 – Código para transformar uma imagem RGB em HSV com OpenCV.

```
1  import cv2 as cv
2  arq = "2.png"
3  im = cv.imread(arq)
4  im = cv.cvtColor(im, cv.COLOR_BGR2HSV_FULL)
5  im = im[:, :, 0]
6  cv.imwrite("2 final H.png",im)
7  im = im[:, :, 1]
8  cv.imwrite("2 final S.png",im)
9  im = im[:, :, 2]
10 cv.imwrite("2 final V.png",im)
```

Figura 4.3 – Canal H das notas da Figura 4.2.

Fonte: o autor.

Figura 4.4 – Canal S das notas da Figura 4.2.

Fonte: o autor.

Figura 4.5 – Canal V das notas da Figura 4.2.

Fonte: o autor.

4.1.2 Imagem em tons de Cinza

A Figura 4.6 mostra os tons de cinza das notas obtido usando o Código 4.2. As imagens cinza mantiveram a forma dos objetos e sua textura, mesmo na nota de R$ 20,00, que apresenta baixa luminosidade ou na de R$ 10,00, que apresenta uma região sombreada.

Código 4.2 – Código para transformar uma imagem colorida em tons de cinza com OpenCV.

```
1 import cv2 as cv
2 arq = "2.png"
3 im = cv.imread(arq)
4 im = cv.cvtColor(im, cv.COLOR_BGR2GRAY)
5 cv.imwrite("2 cinza.png",im)
```

Figura 4.6 – Imagens em tons de cinza para as notas da Figura 4.2

Fonte: o autor.

4.1.3 Filtro Gaussiano

A Figura 4.7 mostra o efeito do filtro Gaussiano com *kernel* de 25x25 aplicado nas notas usando o Código 4.3. Ao aplicar estes filtros as notas ficaram menos nítidas e perderam informação de textura e contorno. Os ruídos como amassos nas notas apresentaram pouca redução.

Código 4.3 – Código para usar o filtro Gaussiano do OpenCV.

```
1 import cv2 as cv
2 arq = "2.png"
3 im = cv.imread(arq)
4 im = cv.GaussianBlur(im, (25, 25), 0)
5 cv.imwrite("2 gaussiano.png",im)
```

Figura 4.7 – Imagens com filtro Gaussiano para as notas da Figura 4.2

Fonte: o autor.

4.1.4 Filtro Mediana

A Figura 4.8 mostra o efeito do filtro Mediana com *kernel* de 25x25 aplicado nas notas usando o Código 4.4. As imagens apresentaram um borramento e uma suavização que arrendondou a forma do objetos. Os contornos mais fortes como os dos números se mantiveram bem definidos.

Código 4.4 – Código para usar o filtro mediana do OpenCV.

```
1 import cv2 as cv
2 arq = "2.png"
3 im = cv.imread(arq)
4 im = cv.medianBlur(im, 25)
5 cv.imwrite("2 mediana.png",im)
```

Figura 4.8 – Imagens com filtro Mediana para as notas da Figura 4.2

Fonte: o autor.

4.1.5 Filtro Bilateral

A Figura 4.9 mostra o efeito do filtro Bilateral com *kernel* de 25x25 aplicado nas notas usando o Código 4.5. Este filtro conseguiu remover ruídos em todas as imagens, como pequenos amassados nas notas, mantendo a forma dos objetos, mas perdendo informação das texturas.

Código 4.5 – Código para usar o filtro bilateral do OpenCV.

```
1 import cv2 as cv
2 arq = "2.png"
3 im = cv.imread(arq)
4 im = cv.bilateralFilter(im, 25, 50, 13)
5 cv.imwrite("2 bilateral.png",im)
```

Figura 4.9 – Imagens com filtro Bilateral para as notas da Figura 4.2

Fonte: o autor.

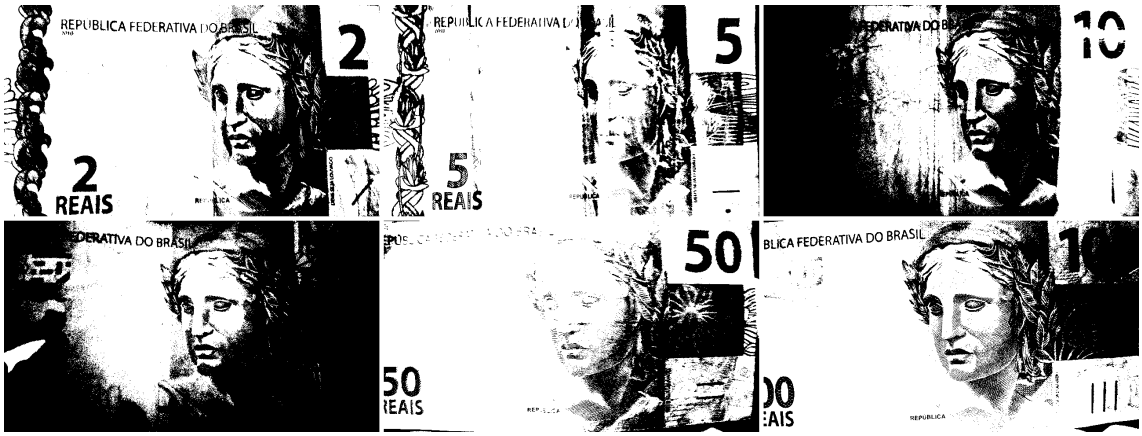
4.1.6 Limiarização

A Figura 4.10 mostra o efeito de limiarização com limiar de 127 aplicado nas notas usando o Código 4.6. Ao utilizar este processamento foi possível destacar alguns objetos da nota como os número e elementos que apresentam a cor principal da nota. Mas este efeito não foi atingido em região sombreadas da nota, como visto nas notas de R$10,00 e R$20,00.

Código 4.6 – Código para usar o filtro de limiarização do OpenCV.

```
1 import cv2 as cv
2 arq = "2.png"
3 im = cv.imread(arq)
4 im = cv.cvtColor(im, cv.COLOR_BGR2GRAY)
5 im = cv.threshold(im, 127, 255, 0)[1]
6 cv.imwrite("2 limiar.png",im)
```

Figura 4.10 – Imagens com limiarização para as notas da Figura 4.6



Fonte: o autor.

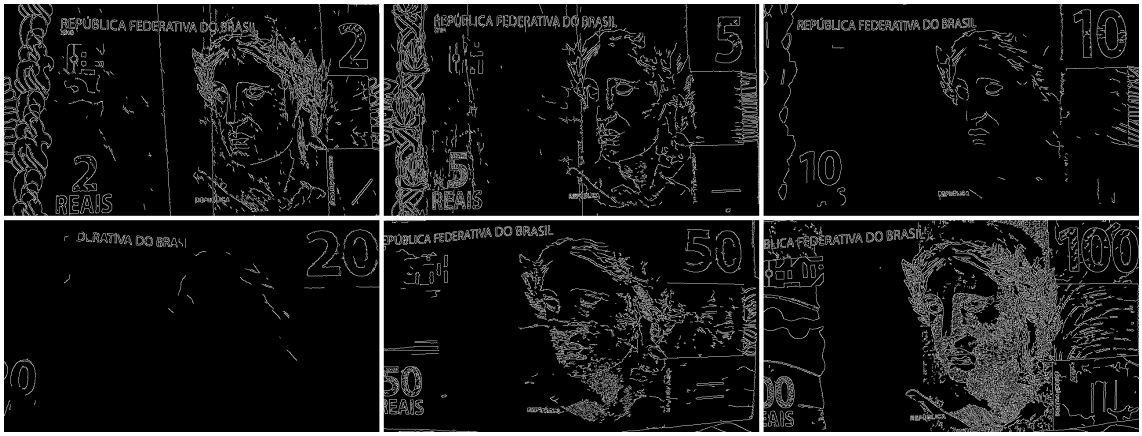
4.1.7 Canny

A Figura 4.11 mostra a detecção de contornos de Canny com limiares 100 e 200 aplicado nas notas usando o Código 4.7. Este processamento conseguiu identificar os contornos de alguns números e outros elementos como o rosto da nota.

Código 4.7 – Código para usar Canny do OpenCV.

```
1 import cv2 as cv
2 arq = "2.png"
3 im = cv.Canny(im, 100, 200)
4 cv.imwrite("2 canny.png",im)
```

Figura 4.11 – Imagens após a detecção de objetos de Canny para as notas da Figura 4.2



Fonte: o autor.

4.1.8 Equalização de histograma

A Figura 4.12 mostra a equalização de histograma aplicado nas notas usando o Código 4.8. As imagens após este processamento apresentaram uma melhora no contraste e uma definição melhor dos contornos dos objetos. Contudo, algumas regiões sombreadas ficaram ainda mais escuras, como a parte esquerda da nota de R$ 10,00, mostrando que este filtro não é bom em imagem com sombreamento parcial.

Código 4.8 – Código para usar a equalização de histograma do OpenCV em uma imagem em tons de cinza.

```
1 import cv2 as cv
2 arq = "2.png"
3 im = cv.cvtColor(im, cv.COLOR_BGR2GRAY)
4 im = cv.equalizeHist(im)
5 cv.imwrite("2 equalizacao.png",im)
```

Figura 4.12 – Imagens após equalização de histograma para as notas da Figura 4.6

Fonte: o autor.

4.1.9 K-Means

A Figura 4.13 mostra o efeito K-Means com $k = 3$ aplicado nas notas usando o Código 4.9. Os resultados deste processamento permitiu manter a forma do objetos com uma intensidade de cor predominantemente diferentes para as notas, excetuando a de R$ 20,00 e R$ 10,00.

Código 4.9 – Código para usar o k-Means do OpenCV.

```
1 import cv2 as cv
2 arq = "2.png"
3  _, label, center = cv.kmeans(np.float32(im.reshape((-1, 3))),
4                               3,
5                               None, (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 10,
                                      1.0),
6                               10, cv.KMEANS_RANDOM_CENTERS)
7 im = center[label.flatten()].reshape((im.shape))
8 cv.imwrite("2 equalizacao.png",im)
```

Figura 4.13 – Imagens após k-Means com k=3 para as notas da Figura 4.2

Fonte: o autor.

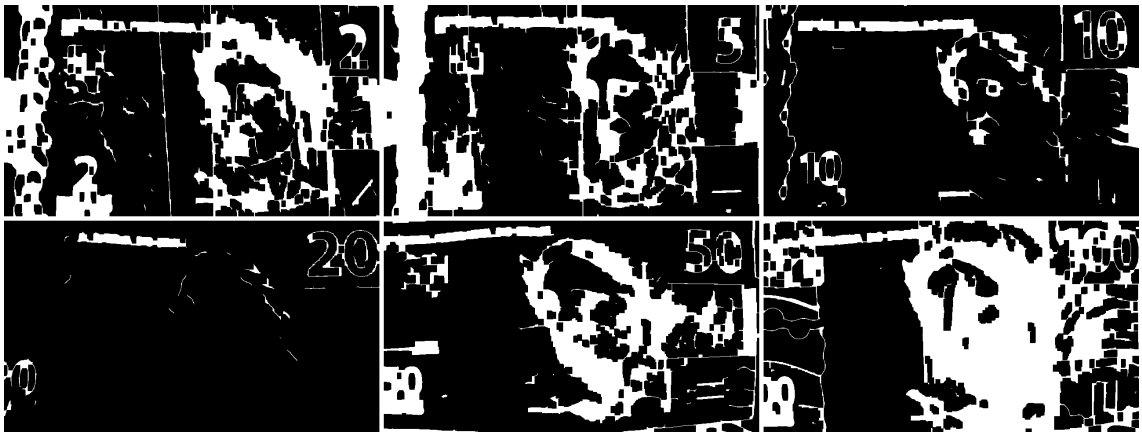
4.1.10 Filtro Morfológico de Fechamento

A Figura 4.14 mostra do morfológico aplicado nas notas usando o Código 4.10. O filtro morfológico é usado para preencher o interior dos contornos detectados pelo Canny, permitindo segmentar regiões da imagem, como por exemplo o número da nota, como é possível ver na nota de R$ 50,00.

Código 4.10 – Abrindo a imagem 2.png, aplicando Canny e o filtro morfológco de fechamento.

```
1 import cv2 as cv
2 arq = "2.png"
3 im = cv.Canny(im, 100, 200)
4 cv.morphologyEx(src=im, op=cv.MORPH_CLOSE,
5                 kernel=cv.getStructuringElement(shape=cv.MORPH_RECT, ksize=(10, 15)),
                      dst=im)
6 cv.imwrite("2 equalizacao.png",im)
```

Figura 4.14 – Imagens após o filtro morfológico de fechamento para as notas da Figura 4.11



Fonte: o autor.

4.1.11 *Patches*

Ao buscar objetos nas imagens, uma opção interessante de processamento é percorrer e analisar pequenas partes da imagem chamadas de *patches*. Assim, cada pedaço da imagem pode ser analisado separadamente. O resultado da classificação pode ser uma votação para determinar o valor da nota. O Código 4.11 mostra como foi feito este processamento.

Código 4.11 – Abrindo a imagem 2.png, e dividindo ela em *patches* de tamanho 4000 *pixels*.

```
patches = []
patches_len = 4000
step = int(patches_len**(1/2))
left, upper, right, lower = 0, 0, step, step
count = 0
h, l = im.shape[:2]
count = 0
while right < l:
    while lower < h:
        a = im[upper:lower, left:right]
        patches.append(a)
        count += 1
        left, upper, right, lower = left, upper+step, right, lower+step
    left, upper, right, lower = left+step, 0, right+step, step
```

4.1.12 Segmentação de Elementos relevantes

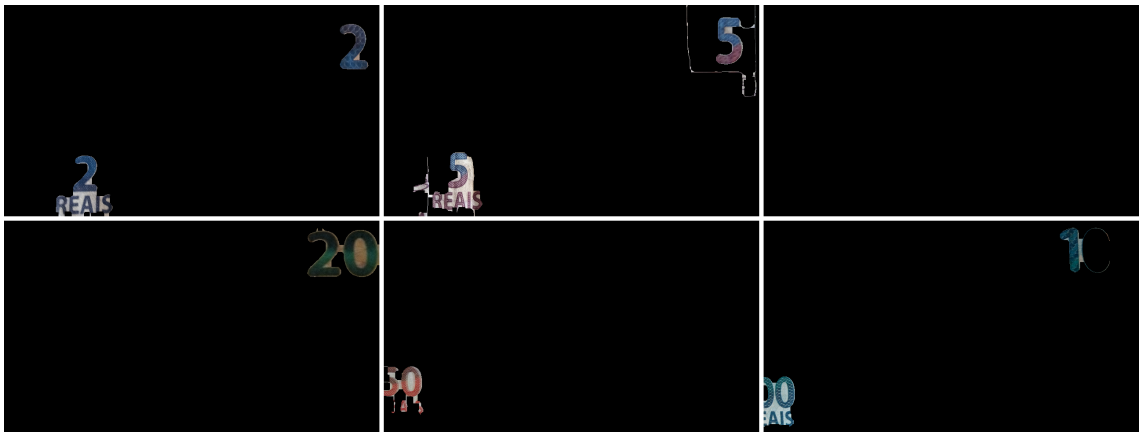
A segmentação pretende remover partes desnecessárias da imagem, deixando somente elementos com informações relevantes. Assim, este processamento visa criar uma

segmentação que destaque o número ou a cor principal das notas. O Código 4.11 mostra como foi feita esta segmentação e a Figura 4.15 apresenta o resultado nas imagens das notas. A segmentação utiliza a uma imagem *im2* apresentada na Figura 4.2 e *im* é usada para detectar os contornos e é o resultado da sequência de processamentos Tons de Cinza, Equalização de Histograma, Mediana, Canny, e Morfológico de Fechamento. Os contornos detectados são usados como uma máscara que irá permitir que somente os *pixels* de *im2* que estejam sobrepostos aos contornos encontrados sejam utilizados. O resultado pode ser visto na Figura 4.15. Foi possível destacar os números das notas de R$ 2,00; R$ 5,00; R$ 20,00 e R$ 50,00. Mas na nota de R$ 100,00 não foram destacados todos os zeros e nada foi encontrado na nota de R$ 10,00. A sequência de processamentos utilizada foi escolhida depois de alguns testes, sendo a que melhor destacou os números.

Código 4.12 – Encontrado objetos na imagem e segmentando-os na imagem original.

```
1  import cv2 as cv
2  returns = []
3  contours, _ = cv.findContours(im, mode=cv.RETR_EXTERNAL, method=cv.CHAIN_APPROX_NONE)
4  if len(contours):
5      for index, contour in enumerate(contours):
6          if cv.contourArea(contour) > constants.AREA_MIN:
7              mask = np.zeros(im.shape, dtype="uint8")
8              cv.drawContours(mask, [contour], -1, 255, -1)
9              temp = np.array(im2)
10             temp[mask == 0] = 0
11             returns.append(temp)
```

Figura 4.15 – Imagens após a segmentação da Figura 4.2



Fonte: o autor.

4.2 Implementando classificadores

A implementação dos algoritmos classificadores SVM, KNN e MLP foi realizada usando a biblioteca OpenCV. O Código 4.13 cria o classificador SVM com valores de $C = 1, \gamma = 1, degree = 1$ e usando o *kernel* linear. A matriz X é o conjunto de treinamento onde cada linha é um padrão e y é o vetor de rótulos, indicando a classe de cada linha da matriz. A matriz $X\_desconhecido$ é uma matriz de uma linha contendo um padrão com rótulo desconhecido que será usado na etapa de classificação na linha 15 que retorna a classe predita na variável $y\_predito$.

Código 4.13 – Código para criação, treinamento e classificação do SVM

```
1  import cv2 as cv
2
3  svm = cv.ml.SVM_create()
4  svm.setType(cv.ml.SVM_C_SVC)
5  svm.setKernel(opencv_kernels[kernel])
6  svm.setC(C)
7  svm.setGamma(gamma)
8  svm.setDegree(degree)
9
10 X = np.matrix(X, dtype=np.float32)
11 y = np.array(y)
12 svm.train(X, cv.ml.ROW_SAMPLE, y)
13
14 X_desconhecido = np.matrix(X_desconhecido, dtype=np.float32)
15 y_predito = np.array(svm.predict(X_desconhecido)[1], dtype=int)
```

A implementação do KNN, Código 4.14, segue a mesma lógica com o parâmetro de configuração $k = 3$.

Código 4.14 – Código para criação, treinamento e classificação do KNN.

```
1  import cv2 as cv
2
3  knn = cv.ml.KNearest_create()
4  knn.setDefaultK(3)
5
6  X = np.matrix(X, dtype=np.float32)
7  y = np.array(y)
8  knn.train(X, cv.ml.ROW_SAMPLE, y)
9
10 X_desconhecido = np.matrix(X_desconhecido, dtype=np.float32)
11 y_predito = np.array(knn.predict(X_desconhecido)[1], dtype=int)
```

Já a implementação do MLP, Código 4.15, para mais de uma classe apresenta uma etapa a mais, onde é necessário realizar um processamento chamado *one-hot-encoder*, que tem a função de transformar a representação numérica das classes 2, 5, 10, 20, 50 e 100

nos vetores [1,0,0,0,0,0], [0,1,0,0,0,0], [0,0,1,0,0,0], [0,0,0,1,0,0], [0,0,0,0,1,0] e [0,0,0,0,0,1], respectivamente. Este processo deve ser realizado pois na última camada do MLP existem 6 neurônios que irão gerar como resposta um vetor de 0s e 1s que representa uma classe (63). O *one-hot-encoder* é realizado usando uma função da biblioteca scikit-learn. No código abaixo é criado um rede neural de três camadas com 255, 10 e 6 neurônios, função de ativação sigmoid simétrica com valores de $\alpha = 1$ e $\beta = 1$ e usando o *backpropagation* como método de treinamento.

Código 4.15 – Código para criação, treinamento e classificação do MLP.

```
from sklearn.preprocessing import OneHotEncoder
import cv2 as cv

mlp = cv.ml.ANN_MLP_create()
mlp.setLayerSizes([256,10,6])
mlp.setActivationFunction(cv.ml.ANN_MLP_SIGMOID_SYM, 1, 1)
mlp.setTrainMethod(cv.ml.ANN_MLP_BACKPROP)
mlp.setTermCriteria((cv.TERM_CRITERIA_MAX_ITER + cv.TERM_CRITERIA_EPS, 300, 0.01))

X = np.matrix(X, dtype=np.float32)
y = np.array(y)
y = OneHotEncoder(sparse=False, dtype=np.float32).fit_transform(y.reshape(-1, 1))

mlp.train(X, cv.ml.ROW_SAMPLE, y)

X_desconhecido = np.matrix(X_desconhecido, dtype=np.float32)
temp  = sorted(set(y))
y_index = np.array(mlp.predict(X_desconhecido)[0], dtype=int)
y_predict = temp[y_index]
```

O processo de validação cruzada escolhido foi o *leave-one-out*, sendo mais simples de ser implementado, apesar de ter maior tempo de execução. Nos primeiros testes foi verificado uma ocorrência com resultados de acurácia de 100% para os classificadores SVM e KNN, ou seja, os classificadores estão acertando todas as notas. Mas depois foi verificado que o procedimento de gerar imagens rotacionadas em 180° estava gerando pontos duplicados de histogramas, o que permitia o classificador sempre acertar a nota. Por isso, no processo de validação cruzada, além da nota que é classificada também é retirada do grupo de treinamento a imagem invertida desta.

Como visto, cada classificador apresenta parâmetros de configuração que devem ser modificados com o objetivo de aumentar a acurácia dos classificadores. Essa procura dos melhores parâmetros é chamada de *grid search*. Constrói-se um conjunto de valores para cada parâmetro e são feitas combinações destes. É preciso considerar a quantidade de opções para cada parâmetro. Se, por exemplo, for realizado o *grid search* para o MLP com

5 funções de ativação, 10 opções das quantidades de neurônios presentes nas camadas, 10 valores de α e 10 de β, teria-se 5.000 combinações. Se o processo de *cross-validation* para cada combinação demorasse 5 minutos, então o *grid search* seria concluído em 18 dias. Adicionando mais uma opção de valor para o *alpha* o tempo do *grid search* aumentaria em 2 dias. Uma opção para reduzir este tempo é usar poucos valores que estejam distantes entre si, escolher o melhor e repetir o processo com valores mais próximos do que apresentou melhor resultado (41).

4.3 Características extraídas

As principais características de imagem são sua cor, textura e forma. Como é possível notar, cada nota possui uma cor principal diferente, sendo este um bom ponto de partida para extrair características. Assim, os processos de construção das características extraídas com base na cor que foram testados são identificados como Histograma Completo, Histograma Filtrado e Histograma Reduzido. Já a característica extraída com base na forma dos objetos contidos nas notas é o Histograma SIFT.

4.3.1 Histograma Completo

Esta característica é o vetor histograma contendo a intensidade dos *pixels* de um canal. Seu objetivo é converter a imagem para o formato HSV e gerar o histograma da camada H. O vetor de características contém 256 valores, cada um representando uma intensidade de cores. Cada nota apresenta uma distribuição de cores diferentes. O Código 4.16 é utilizado para gerar este vetor de característica para uma imagem *im* que já passou pelas etapas de processamento e apresenta somente um canal . O vetor é normalizado para que o maior valor seja 100.

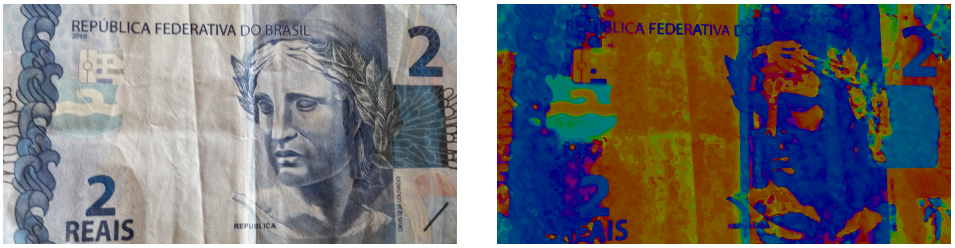
Código 4.16 – Código para gerar a característica Histograma Completo para uma imagem já processada.

```
1 import numpy as np
2 import cv2 as cv
3
4 X = np.squeeze(cv.calcHist([im], [0], None, [256], [0, 256])).tolist()
```

4.3.2 Histograma Filtrado

A representação da imagem no formato HSV consegue separar a cor da imagem na camada H. Entretanto, como pode ser visto na Figura 2.3, um *pixel* branco ou escuro que não apresenta uma cor assume, ainda assim, uma cor no modelo HSV. A Figura 4.16 apresenta uma imagem original de uma nota e outra com todo os valores de luminosidade em 125 e saturação em 255 para destacar as cores assumidas pelo canal H. É possível ver a cor atribuída a esta parte branca na camada H que descaracteriza a cor principal da imagem. O mesmo acontece quando a saturação do *pixel* é muito baixa, a cor pode não ser registrada corretamente.

Figura 4.16 – (a) Imagem original e (b) a camada H desta imagem.



(a) (b)

Fonte: o autor.

Percebendo este efeito foi pensado em excluir do histograma de cores os *pixels* com luminosidades extremas ou com baixa saturação. O Código 4.17 apresenta a forma como isso foi realizado. As constantes *SATURATION_TOLERANCE* e *VALUE_TOLERANCE* variam de 0 a 1, onde 1 indica que nenhum *pixel* é filtrado. Desse modo, seriam excluídos *pixels* que não apresentam uma cor em destaque.

Código 4.17 – Código para gerar a característica Histograma Filtrado para uma imagem já processada.

```
1 import numpy as np
2
3 im = im[(im[:, :, 1] > 255 - 255 * constants.SATURATION_TOLERANCE)
4         & (im[:, :, 2] > 255 / 2 - 255 / 2 * constants.VALUE_TOLERANCE)
5         & (im[:, :, 2] < 255 / 2 + 255 / 2 * constants.VALUE_TOLERANCE)]
6 X =  np.histogram(im[:, 0], bins=range(256+1))[0]
```

4.3.3 Histograma Reduzido

Como as cores principais das notas são bem distintas, é possível agrupar intensidades de *pixels* próximas para compensar pequenas variações que notas envelhecidas e

gastas possam ter em comparação com as novas. Outra vantagem é reduzir o tamanho do vetor de características agrupando características redundantes, o que pode ajudar o classificador a construir a fronteira de decisão. O Código 4.18 apresenta esta função.

Código 4.18 – Código para gerar a característica Histograma Reduzido para uma imagem já processada.

```
import numpy as np
import cv2 as cv

hist = np.squeeze(cv.calcHist([im], [0], None, [256], [0, 256])).tolist()
step = int(256/n_features)
new_hist = []
for index in range(n_features):
    new_hist += [sum(hist[step*index:(step*index)+step])]
X = new_hist
```

4.3.4 Imagem Completa

Uma forma simples de extrair uma característica da imagem é usar os valores de intensidade da matriz da imagem como vetor. Desse modo, o vetor de características seria a imagem inteira. Esta é uma opção que necessita de grande poder computacional para o treinamento de vetores de característica grandes, por isso a imagem foi reduzida em suas dimensões. O Código 4.19 mostra como isto foi realizado.

Código 4.19 – Código para gerar a caracteristica Imagem Completa para uma imagem já processada.

```
import numpy as np

X = np.ravel(im)
```

4.3.5 Histograma SIFT

A utilização do SIFT tem o objetivo de utilizar a forma dos objetos das notas. Para cada imagem ele gera um grande número de pontos, e para cada um, um vetor descritivo de 125 elementos representando 125 ângulos e cada elemento do vetor representa a intensidade do vetor para aquele angulo. Não seria viável treinar todos estes vetores descritivos no classificador. Por isso é utilizado K-Means para agrupar e reduzir esta infinidade de vetores. O K-Means tem a função de agrupar formas semelhantes. Os centroides do K-Means são usados para treinar um classificador KNN. Para cada imagem são obtidos os vetores descritivos e é usado o KNN para rotular cada vetor a um centroide. O vetor de

características da imagem é a quantidades de vetores classificados para cada centroide. O Código 4.20 apresenta a obtenção desta característica, em que *ims* é uma lista de imagens processadas, k é a quantidade de centroides e o tamanho de vetor de características.

Código 4.20 – Código para gerar a característica Histograma *Sift* para uma imagem já processada.

```
import numpy as np
import cv2 as cv

des_list = []
sift = cv.SIFT_create()
for im in ims:
    des_list += sift.detectAndCompute(im, None)
array = np.array(des_list)
 _, _, center = cv.kmeans(array, k,
                          None, (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER,
                                 10, 1.0),
                          10, cv.KMEANS_RANDOM_CENTERS)

knn = cv.ml.KNearest_create()
knn.setDefaultK(1)
knn.train(center, cv.ml.ROW_SAMPLE, range(len(center)))

# Classificando uma imagem im com rotulo desconhecido
X = np.zeros(n_features)
_, des = sift.detectAndCompute(im, None)
des_length = len(des)
for d in des:
    index = np.array(knn.predict(np.matrix(d, dtype=np.float32))[1], dtype=int)[0, 0]
    X[index] += 1/des_length
```

4.4 Configurações Testadas

O Quadro 4.1 apresenta as configurações testadas para gerar vetores de características que consigam uma boa separação entre as classes. Para cada configuração são apresentados o resultado do processamento de imagem para uma nota de R$ 2,00 e um gráfico da média do histograma de características para as notas de R$ 2,00 do banco de dados. Foram testados vários parâmetros de configuração para o tamanho do *kernel* dos filtros e outros parâmetros para os outros processamentos e a melhor opção encontrada foi a utilizada. O objetivo dos testes de Id 1, 2, 3, 4, 5, 6, 7, 8, 9 e 10 foi utilizar o parâmetro de cor da imagem usando o canal H. Já o teste 11 usa imagem em tons de cinza como característica, transformando a matriz 854x480 em um vetor de 409920 elementos. O teste 12 usa a informação da forma dos objetos da nota como característica usando o SIFT.

Para os testes que usam filtros, ou seja, 2, 3 e 4, foram testadas as dimensões do *kernel* 5, 15, 30 e 55. O melhor resultado de classificação foi obtido para o *kernel* 5 para todos os tipos, assim foi escolhido este valor. Para o teste 5 foram testados os valores de k 6, 60, 100 e 200, e o melhor resultado foi obtido com $k = 200$.

No teste 6 foi verificada uma dificuldade computacional ao utilizar muitos *patches*, o que resultava em um grande número de vetores de características. Isso provocou um grande tempo de execução e travamento da execução do código ao executar o *cross-validation*. Uma possível solução que poderia ser testada, além de aumentar o poder computacional da máquina, é usar o *cross-validation k-fold*, que realizaria apenas k etapas de treinamento e classificação, ao invés do *leave-one-out* que apresenta um número de etapas igual ao número de vetores de características. A solução escolhida foi reduzir as dimensões da imagem de 854x480 para 85x48.

O objetivo do teste 7 foi segmentar elementos de interesse das notas como os números e os animais da nota. Para isso foi usado o Canny para detectar os contornos, o filtro morfológico para preenchê-los e a segmentação para extraí-los e obter a cor destes elementos. O teste 8 tem o mesmo objetivo, porém é usado o k-Means para realizar a segmentação.

Analisando os gráficos é possível ver que a cor vermelha, que ocorre no início e final do eixo horizontal, apresenta grande quantidade de *pixels*, podendo provocar uma confusão dos classificadores para os testes de 1 a 10. Para este teste é possível observar que não houve grande diferença na separação das classes, pois em todos é possível ver que existe um pico da nota de R$ 100,00 em 125 aproximadamente, depois da nota R$ 2,00 em 150 e o último pico da nota de R$ 5,00 em 245 aproximadamente. A única diferença é uma mudança na suavidade da curva por uma possível perda de informação causada pelos diferentes processamentos.

A análise dos histogramas nos testes 11 e 12 são inconclusivos, a princípio não apresentam nenhuma separação entre as classes. Contudo, para realizar uma análise concreta dos testes é necessário verificar a acurácia dos classificadores ao usar cada forma de características.

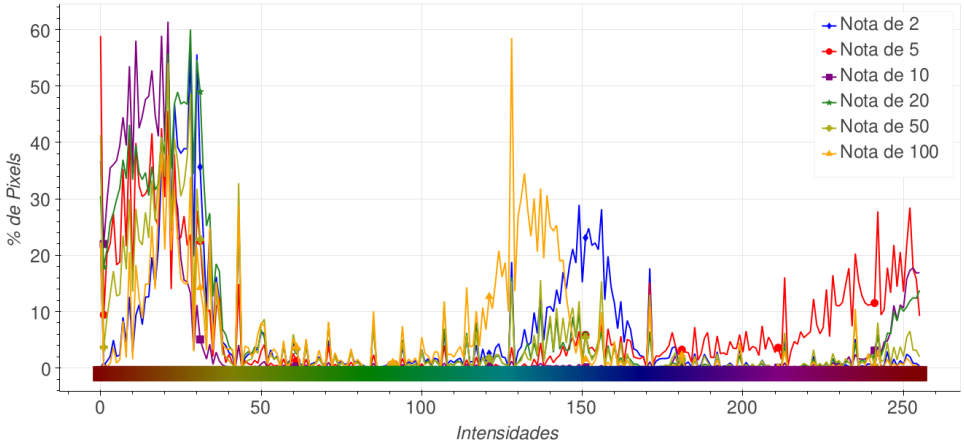
Quadro 4.1 – Combinações testadas para gerar os vetores de características.

Id	**Processamentos**	**Imagens Processadas**	**Característica**	**Gráfico**
1	HSV	Figura 4.17 (a)	Histograma Completo	Figura 4.17 (b)
2	Gaussiano, kernel=5 HSV	Figura 4.17 (c)	Histograma Completo	Figura 4.17 (d)
3	Mediana, *kernel*=5 HSV	Figura 4.17 (e)	Histograma Completo	Figura 4.17 (f)
4	Bilateral, *kernel*=5 HSV	Figura 4.17 (g)	Histograma Completo	Figura 4.17 (h)
5	K-Means com k=200 HSV	Figura 4.17 (i)	Histograma Completo	Figura 4.17 (j)
6	HSV *Patches* 85x10	Figura 4.17 (k)	Histograma Completo	Figura 4.17 (l)
7	Tons de Cinza, Equalização de Histograma, Mediana Canny, Morfológico de Fechamento, HSV, Segmentação	Figura 4.17 (m)	Histograma Completo	Figura 4.17 (n)
8	K-Means com k=2, HSV, Limiar, Segmentação	Figura 4.17 (o)	Histograma Completo	Figura 4.17 (p)
9	HSV	Figura 4.17 (q)	Histograma Reduzido	Figura 4.17 (r)
10	HSV	Figura 4.17 (s)	Histograma Filtrado	Figura 4.17 (t)
11	Tons de Cinza Mediana	Figura 4.17 (u)	Imagem Completa	Figura 4.17 (v)
12	Tons de Cinza Gaussiano, *kernel*=5	Figura 4.17 (x)	Histograma SIFT	Figura 4.17 (x)

Fonte – Produzido pelo autor.

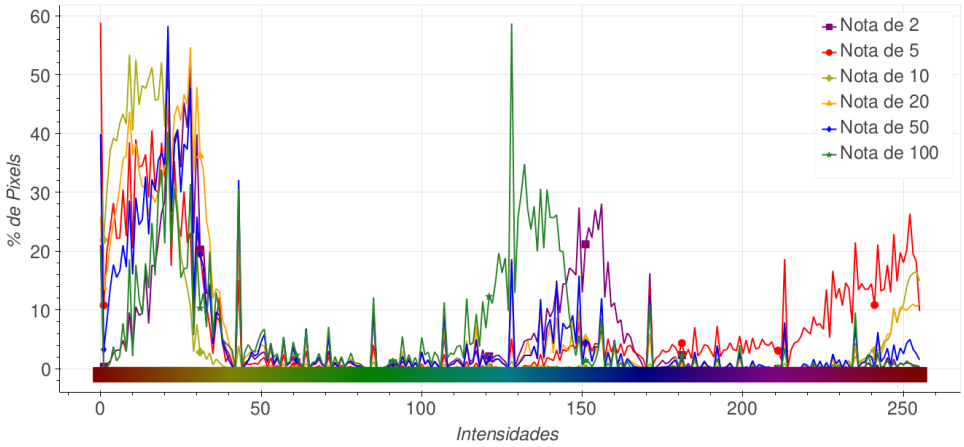
Figura 4.17 – Resultados das combinações do Quadro 4.1.

(a)



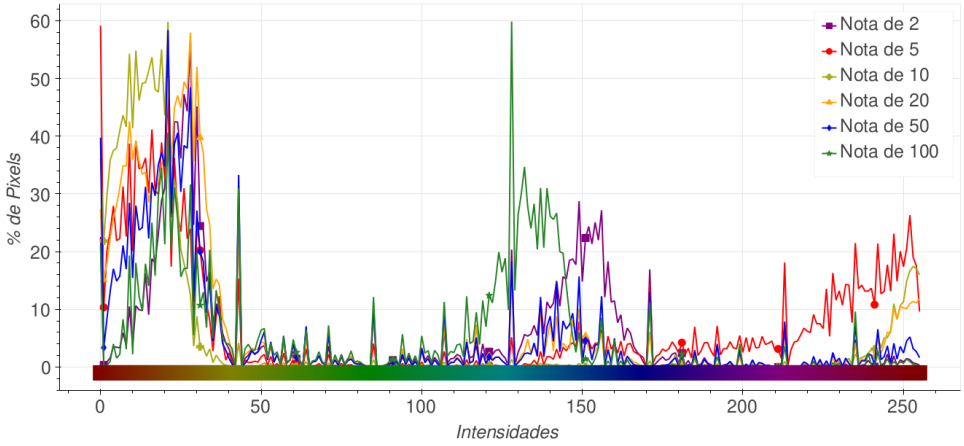
(b)

(c)



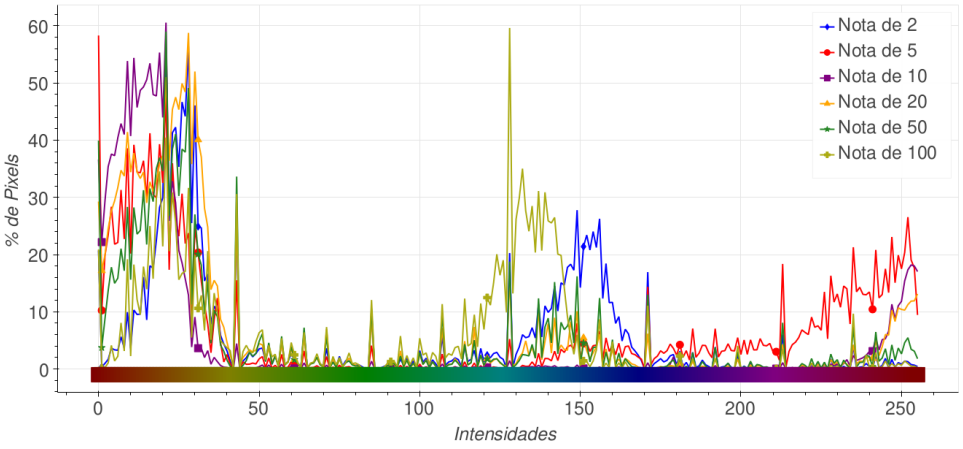
(d)

(e)



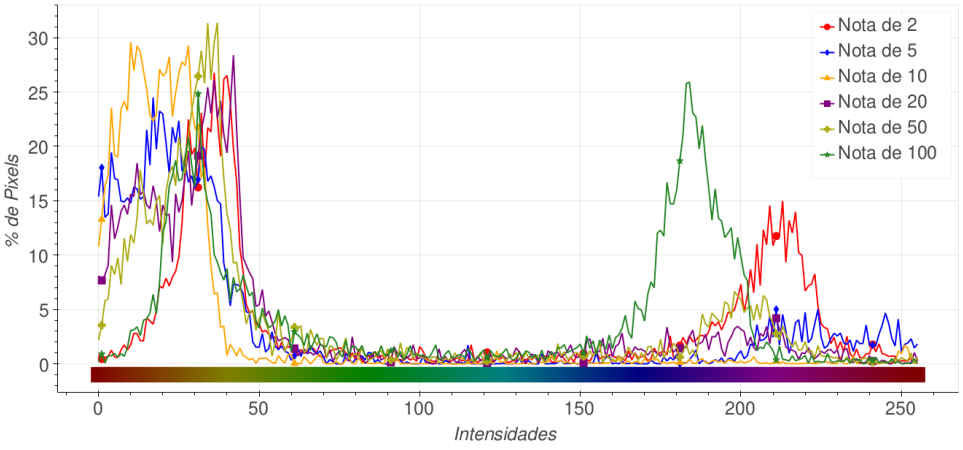
(f)

(g)


Nota de 2
Nota de 5
Nota de 10
Nota de 20
Nota de 50
Nota de 100
% de Pixels
Intensidades
0
10
20
30
40
50
60
0
50
100
150
200
250

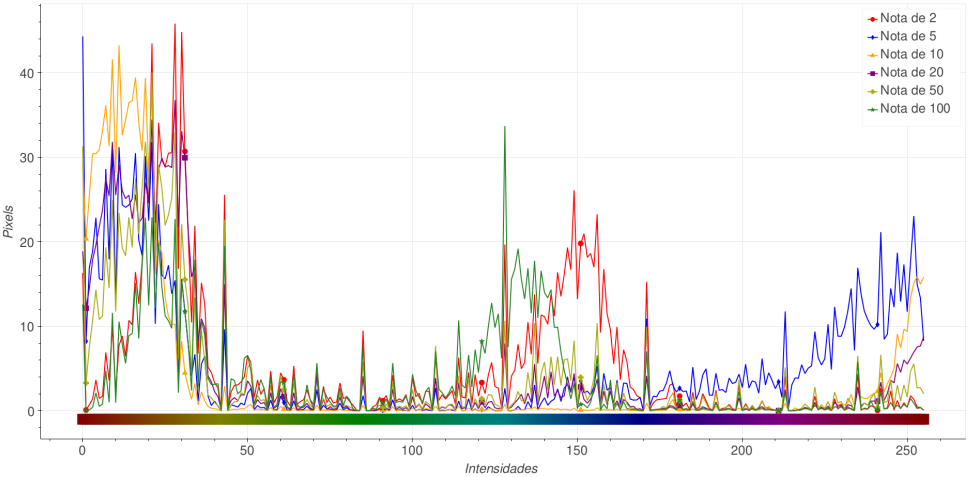
(h)

(i)


Nota de 2
Nota de 5
Nota de 10
Nota de 20
Nota de 50
Nota de 100
% de Pixels
Intensidades
0
5
10
15
20
25
30
0
50
100
150
200
250

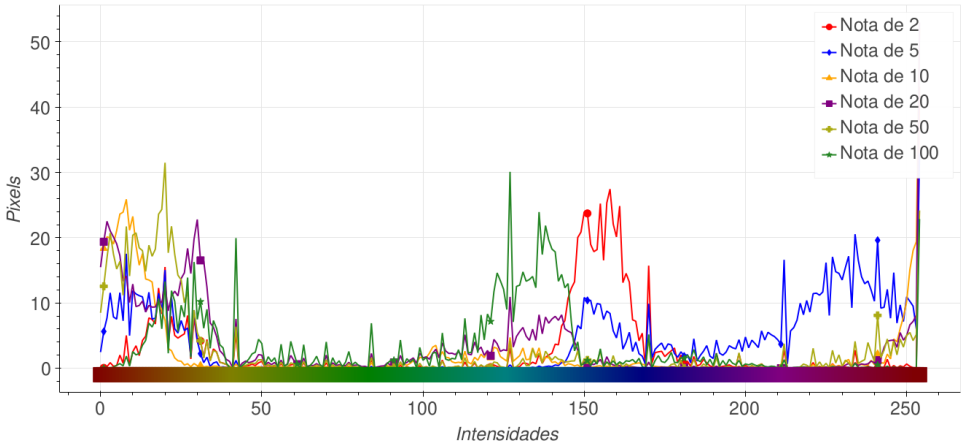
(j)

(k)


Nota de 2
Nota de 5
Nota de 10
Nota de 20
Nota de 50
Nota de 100
Pixels
Intensidades
0
10
20
30
40
0
50
100
150
200
250

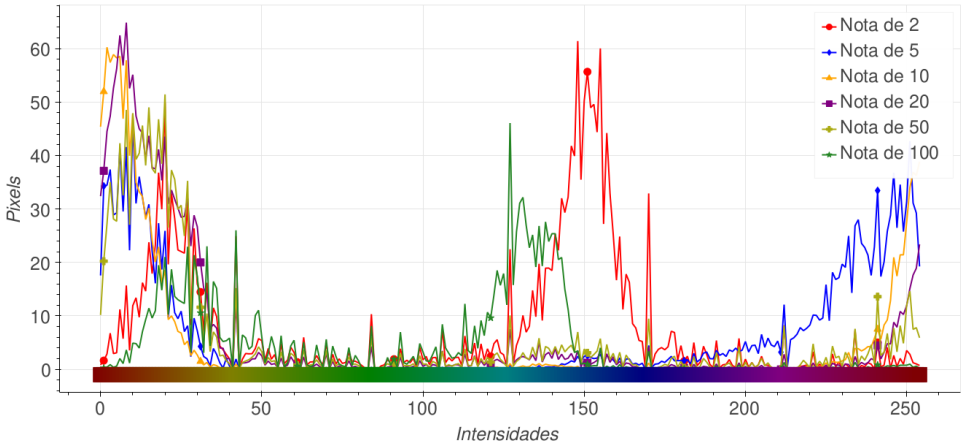
(l)

(m)


Nota de 2
Nota de 5
Nota de 10
Nota de 20
Nota de 50
Nota de 100
Pixels
Intensidades
0
10
20
30
40
50
0
50
100
150
200
250

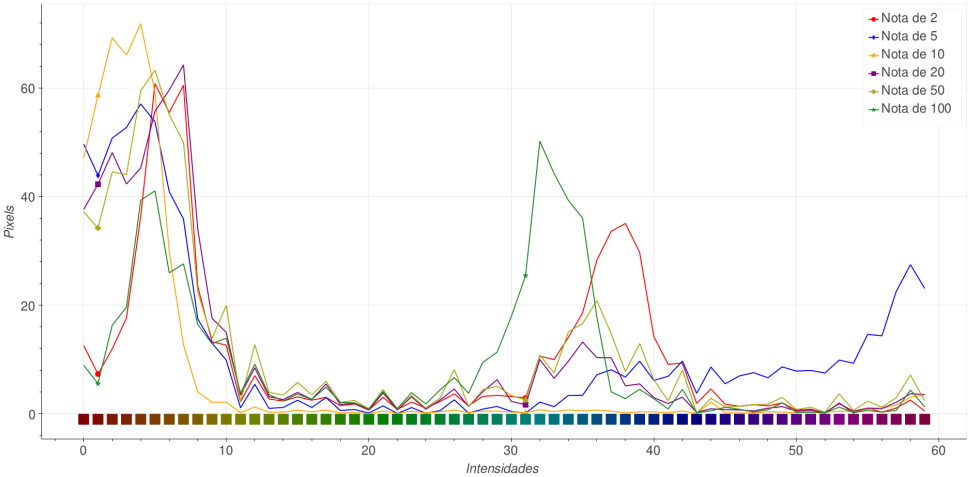
(n)

(o)


Nota de 2
Nota de 5
Nota de 10
Nota de 20
Nota de 50
Nota de 100
Pixels
Intensidades
0
10
20
30
40
50
60
0
50
100
150
200
250

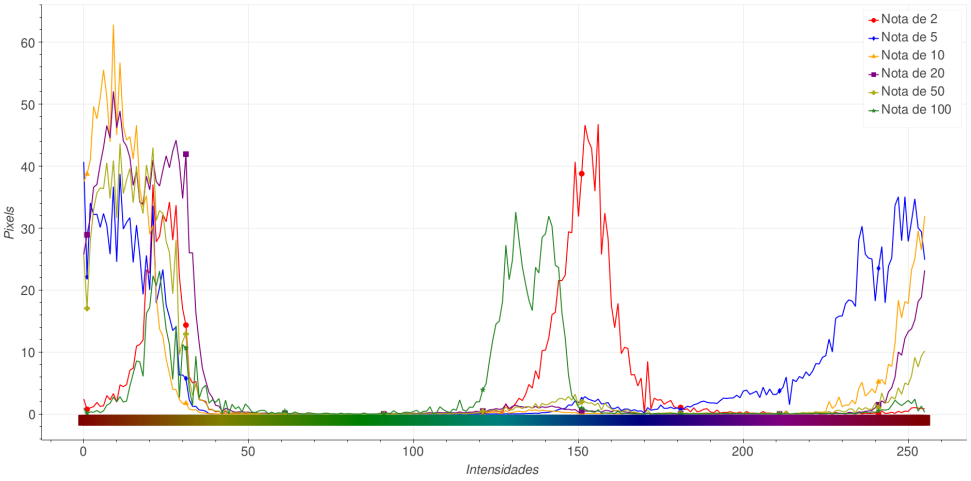
(p)

(q)


Nota de 2
Nota de 5
Nota de 10
Nota de 20
Nota de 50
Nota de 100
Pixels
Intensidades
0
20
40
60
0
10
20
30
40
50
60

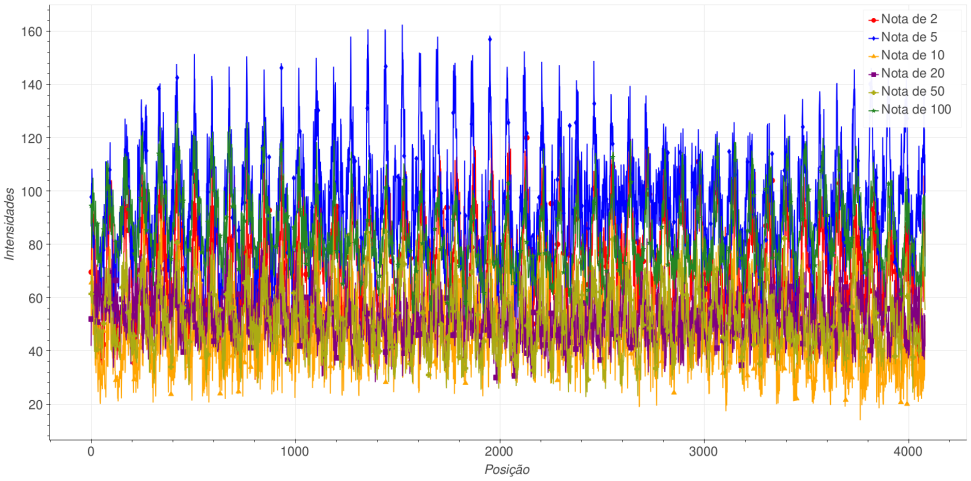
(r)

(s)



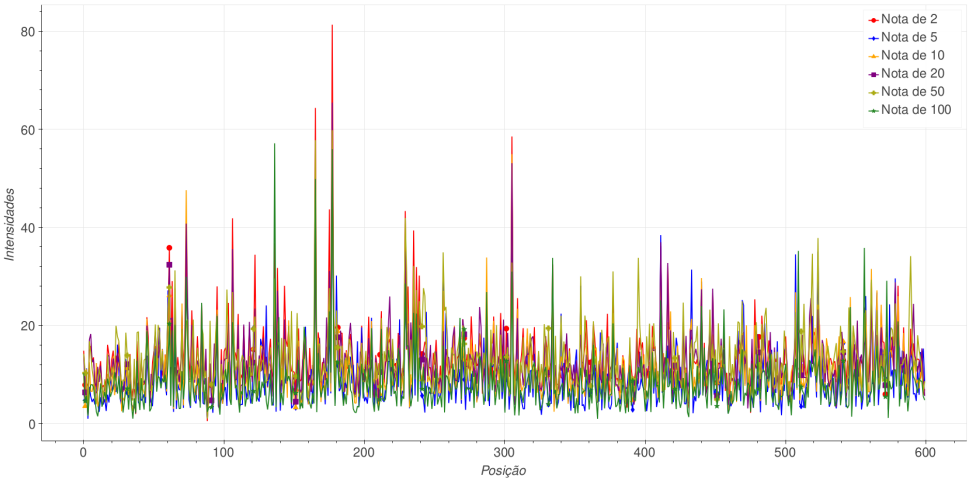
(t)

(u)



(v)

(w)



(x)

Fonte: o autor.

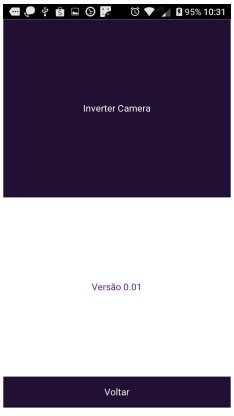
4.5 Construindo aplicativo

O aplicativo foi construído em Python utilizando a biblioteca Kivy permitindo sua execução em celulares Android. Analisando os aplicativos Cash Reader, Blind e LetSeeApp foi pensado na criação de uma *interface* simples onde a tela inicial, Figura 4.18, apresenta a imagem da câmera com um botão de configuração que leva para a tela de configurações. Esta tela de configurações apresenta apenas a função de trocar as câmeras do celular frontal e traseira, Figura 4.19.

Figura 4.18 – Tela inicial do aplicativo.

Fonte: o autor.

Figura 4.19 – Tela de configurações do aplicativo.



Fonte: o autor.

Ao iniciar o aplicativo, a imagem da câmera é apresentada na tela e uma função é chamada a cada intervalo de tempo. A cada chamada, a função realiza a captura da imagem atual gerada pela câmera, realiza o processamento de imagem, extrai a característica, utiliza o classificador treinado para gerar um rótulo para esta característica que aparece

em destaque em cima da imagem da câmera. Também é utilizada uma biblioteca que reproduz em voz o valor da nota. O Código 4.21 apresenta a implementação destas funções. A função *image_processing.process_texture* converte a textura em imagem e retorna a imagem após realizados os processamentos escolhidos. O comando *clsf.classify(im)* extrai a característica escolhida da imagem e retorna o rótulo gerado. A função *tss* pertence à biblioteca *plyer* que transforma texto em voz, sendo usada para reproduzir o valor da nota em som. O valor da nota aparece em amarelo no *label* com id "valor".

Código 4.21 – Código chamado pelo aplicativo para classificar a imagem atual da camera.

```
1  def classify_image(self, *largs):
2          """Get and classify image
3          """
4          camera = self.ids['camera']
5          texture = camera.export_as_image().texture
6          im = image_processing.process_texture(texture)
7
8          y = clsf.classify(im)
9          valor = self.ids['valor']
10         if y != None:
11             valor.text = str(int(y))
12             valor.color = (1, 1, 0, 1)
13             if platform == "android":
14                 tts.speak("".join((str(int(y)), " reais")))
15         else:
16             valor.color = (1, 1, 0, 0)
```

No início do desenvolvimento do aplicativo, viu-se uma dificuldade para realizar capturas utilizando a câmera do celular. A solução encontrada foi realizar uma captura indireta. O Kivy constrói um *widget* com textura gerada pelo vídeo da câmera, e é essa textura que é extraída e convertida em uma matriz para ser processada pelas funções do OpenCV. Um problema desse método é uma possível perda de qualidade da imagem, já que a imagem da qual será extraída a característica não vem diretamente da câmera, pois é como se ela fosse uma captura da tela do celular. Se, por exemplo, o *widget* da câmera não assumir uma resposta responsiva a algum modelo de celular ou à mudança de orientação do celular, poderia gerar uma imagem muito pequena ou cortada, o que ocasionaria na perda de informações.

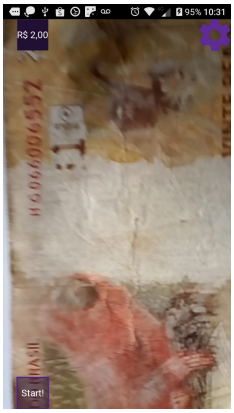
O acesso da câmera gerou outro problema de instabilidade ao aplicativo, sendo a permissão de uso desta pelo sistema Android. Quando instalado o APP pela primeira vez, o mesmo não pede autorização automaticamente para acessar a câmera e ao carregar ele fecha automaticamente. Então é necessário entrar nas configurações do mesmo e ativar a permissão de câmera manualmente.

A ideia inicial do projeto era utilizar os algoritmos de classificação da biblioteca scikit-learn no aplicativo, mas no momento de sua criação foi visto uma incompatibilidade desta biblioteca com o compilador python-for-android. Este compilador apresenta em sua descrição as bibliotecas que são compatíveis, mas infelizmente as bibliotecas Python de *machine learning* scikit-learn, Pytorch e TensorFlow não estão inclusas. Este foi um momento decisivo na construção do aplicativo, quando se cogitou abandonar o Python e o Kivy na tarefa de construir o aplicativo e usar o Android Studio. Felizmente descobriu-se que a biblioteca OpenCV, mais conhecida por algoritmos de processamento de imagem, também possui os principais algoritmos de reconhecimentos de objetos e é compatível com o python-for-android. No entanto, a OpenCV apresenta algumas limitações se comparada com a scikit-learn, como documentação confusa, menos exemplo disponibilizados pela comunidade, falta de funções específicas como o *grid search*. Contudo, o OpenCV possibilitou continuação do uso do Python para construir o aplicativo.

Os três classificadores treinados podem ser salvos em um arquivo xml, permitindo seu uso de forma repetida em várias execuções do código para classificar padrões desconhecidos. O arquivo do classificador escolhido é copiado para a pasta raiz do aplicativo para que o compilador entenda que ele dever ser incluído na conversão para Java. Com isso, o aplicativo abre o arquivo xml e o OpenCV carrega o classificador treinado, permitindo a realização da etapa de classificação do aplicativo.

Foi criada outra versão do aplicativo para testar a acurácia em situações reais. Esta versão adiciona dois novos botões à tela inicial. O botão *start/stop* inicia ou finaliza o processo de classificação da imagem, que irá apresentar duas novas etapas: o salvamento automático das imagens geradas e o resultado de classificação na pasta *downloads* do celular. Além de permitir a análise e verificação posterior do resultado encontrado pelo aplicativo, também é possível usar as imagens para aumentar o banco de dados, Figura 4.20. O resultado é salvo em um arquivo .csv que grava a quantidade de acertos e erros, permitindo gerar as métricas de desempenho do aplicativo para imagens fora do banco de dados. O segundo botão adicionado tem a função de selecionar o tipo de nota que o aplicativo está lendo. Esta opção não influencia no resultado do classificador, é somente uma referência para poder construir o arquivo .csv.

Figura 4.20 – Tela inicial do aplicativo na versão desenvolvedor.



Fonte: o autor.

4.6 Considerações Finais

Antes da construção do aplicativo, foi necessário realizar vários testes e estudos para diferentes tipos de processamento de imagem, como filtros de redução de ruído e de segmentação, visando aumentar sua qualidade. Também foram testados códigos distintos para a extração das caraterísticas a fim de encontrar o melhor vetor de características, que permitiria a melhor separação das classes de notas. Para cada processamento, foi apresentado uma imagem gerada para cada classe e o código utilizado para gera-lá, bem como o gráfico de histograma que permite visualizar a separação das classes.

Foram criadas doze combinações diferentes de processamento de imagens e extração de características para poderem ser testadas usando os classificadores. São apresentados os códigos utilizados em cada uma destas combinações, onde os parâmetros utilizados, como o *kernel* dos filtros de redução de ruído e o k do K-Means, foram variados e o que apresentou melhor resultado de acurácia nos classificadores foi o escolhido para ser usado nas combinações.

Por fim, foi construído o aplicativo usando a ferramenta Kivy para criar a *interface* e o Buildozer para fazer o arquivo instalável no Android. Foram criadas duas versões do aplicativo, uma para o usuário final e outra para o desenvolvedor poder testar a acurácia do classificador diretamente no aplicativo, usando imagens fora do banco de dados. Os códigos utilizados no projeto e o arquivo ".apk" foram salvos no GitHub e podem ser acessados pelo link https://github.com/FabioCamposFerreira/TCC2, que contém também o banco de dados com as imagens das notas.

Com a realização destas etapas no desenvolvimento do projeto, foi possível criar a *interface* do aplicativo contendo os algoritmos de classificação e processamento de imagem,

permitindo reconhecer as notas monetárias no celular Android.

5 Resultados e Discussões

Neste capítulo são apresentados os resultados obtidos para cada uma das combinações escolhidas em cada um dos três classificadores. Os resultados apresentados são a acurácia, a precisão e a sensibilidade. Também é apresentada a matriz de confusão para cada combinação com o intuito de visualizar quais as classes estão com a menor taxa de acertos.

Tendo em mãos estes resultados é possível escolher o melhor classificador e característica que deve ser utilizada na construção final do aplicativo. Para a combinação selecionada, são apresentados os resultados do *grid search*, para confirmar a utilização dos melhores parâmetros de configuração do método de classificação escolhido. E por fim é feita a análise dos resultados obtidos diretamente do aplicativo.

5.1 Resultados dos testes

Para que fosse possível obter os resultados foram definidos valores iniciais dos parâmetros de configuração dos classificadores testados, apresentados no Quadro 5.1. Estes parâmetros foram escolhidos usando um *grid search* preliminar para algumas combinações testadas. As Tabelas 5.1, 5.2 e 5.3 apresentam os resultados para cada configuração testada para os classificadores SVM, KNN e MLP, respectivamente. Nestas tabelas, a coluna **Id** identifica a característica testada conforme o Quadro 4.1. Os resultados foram obtidos usando o banco de dados construído e usando *cross validation leave-one-out*. Para cada teste são apresentadas a acurácia, precisão, sensibilidade e a matriz de confusão, com a mesma estrutura do Quadro 2.3.

Quadro 5.1 – Parâmetros de configuração usados para obter os dados das Tabelas 5.1, 5.2 e 5.3.

Classificador	**Parâmetros**
SVM	C=1, *degree*=1, *kernel*=Interseção de Histograma
KNN	k=3
MLP	camadas=[256,10,6], ativação=Sigmoid Simétrica, $\alpha = 0{,}5$, $\beta = 1$

Fonte – Produzido pelo autor.

Tabela 5.1 – Resultados dos testes para cada configuração usando SVM.

(continua)

Id	**Acurácia**	**Precisão**	**Sensibilidade**	**Matriz de confusão**
1	90,33%	88,49%	89,02%	$\begin{bmatrix} 98 & 0 & 0 & 0 & 2 & 0 \\ 2 & 88 & 6 & 0 & 4 & 0 \\ 0 & 6 & 86 & 6 & 2 & 0 \\ 0 & 2 & 8 & 82 & 8 & 0 \\ 2 & 4 & 4 & 2 & 88 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \end{bmatrix}$
2	90,0%	89,53%	88,05%	$\begin{bmatrix} 98 & 0 & 0 & 0 & 2 & 0 \\ 0 & 88 & 8 & 0 & 4 & 0 \\ 2 & 4 & 86 & 8 & 0 & 0 \\ 0 & 2 & 6 & 84 & 8 & 0 \\ 4 & 4 & 4 & 4 & 84 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \end{bmatrix}$
3	89,67%	87,60%	86,98%	$\begin{bmatrix} 98 & 0 & 0 & 0 & 2 & 0 \\ 0 & 86 & 10 & 0 & 4 & 0 \\ 2 & 6 & 86 & 6 & 0 & 0 \\ 0 & 4 & 4 & 84 & 8 & 0 \\ 4 & 4 & 4 & 4 & 84 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \end{bmatrix}$
4	90,33%	89,23%	87,57%	$\begin{bmatrix} 98 & 0 & 0 & 0 & 2 & 0 \\ 0 & 86 & 10 & 0 & 4 & 0 \\ 0 & 4 & 88 & 6 & 2 & 0 \\ 0 & 2 & 6 & 84 & 8 & 0 \\ 4 & 4 & 4 & 2 & 86 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \end{bmatrix}$

Tabela 5.1 – Resultados dos testes para cada configuração usando SVM.

(continuação)

Id	**Acurácia**	**Precisão**	**Sensibilidade**	**Matriz de confusão**
5	66,83%	57,18%	55,52%	$\begin{bmatrix} 89 & 1 & 1 & 0 & 9 & 0 \\ 7 & 50 & 22 & 14 & 7 & 0 \\ 0 & 22 & 67 & 7 & 4 & 0 \\ 4 & 17 & 10 & 50 & 13 & 6 \\ 9 & 11 & 11 & 17 & 49 & 3 \\ 1 & 0 & 0 & 0 & 3 & 96 \end{bmatrix}$
6	87,33%	84,78%	82,65%	$\begin{bmatrix} 98 & 0 & 0 & 0 & 2 & 0 \\ 0 & 80 & 14 & 2 & 2 & 2 \\ 0 & 10 & 80 & 2 & 8 & 0 \\ 0 & 0 & 4 & 86 & 10 & 0 \\ 4 & 4 & 4 & 8 & 80 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \end{bmatrix}$
7	51,67%	70,41%	47,49%	$\begin{bmatrix} 40 & 2 & 2 & 50 & 4 & 2 \\ 2 & 44 & 4 & 44 & 6 & 0 \\ 0 & 0 & 40 & 48 & 8 & 4 \\ 0 & 0 & 2 & 84 & 8 & 6 \\ 2 & 12 & 10 & 32 & 44 & 0 \\ 0 & 0 & 6 & 36 & 0 & 58 \end{bmatrix}$
8	86,67%	78,22%	85,02%	$\begin{bmatrix} 92 & 4 & 0 & 0 & 4 & 0 \\ 0 & 82 & 8 & 0 & 10 & 0 \\ 0 & 13 & 79 & 7 & 1 & 0 \\ 0 & 9 & 4 & 83 & 4 & 0 \\ 1 & 6 & 4 & 2 & 87 & 0 \\ 0 & 3 & 0 & 0 & 0 & 97 \end{bmatrix}$

Tabela 5.1 – Resultados dos testes para cada configuração usando SVM.

(conclusão)

Id	**Acurácia**	**Precisão**	**Sensibilidade**	**Matriz de confusão**
9	84,0%	84,15%	78,21%	$\begin{bmatrix} 98 & 2 & 0 & 0 & 0 & 0 \\ 0 & 76 & 14 & 8 & 2 & 0 \\ 2 & 8 & 82 & 6 & 2 & 0 \\ 4 & 0 & 10 & 76 & 10 & 0 \\ 4 & 6 & 4 & 10 & 74 & 2 \\ 2 & 0 & 0 & 0 & 0 & 98 \end{bmatrix}$
10	87,67%	84,45%	82,31%	$\begin{bmatrix} 96 & 0 & 0 & 0 & 4 & 0 \\ 0 & 80 & 8 & 4 & 8 & 0 \\ 0 & 10 & 84 & 2 & 4 & 0 \\ 0 & 2 & 6 & 88 & 4 & 0 \\ 4 & 4 & 10 & 4 & 78 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \end{bmatrix}$
11	85,33%	78,98%	79,96%	$\begin{bmatrix} 86 & 0 & 2 & 0 & 12 & 0 \\ 2 & 76 & 10 & 4 & 8 & 0 \\ 0 & 10 & 88 & 0 & 2 & 0 \\ 0 & 10 & 0 & 84 & 4 & 2 \\ 4 & 4 & 6 & 6 & 78 & 2 \\ 0 & 0 & 0 & 0 & 0 & 100 \end{bmatrix}$
12	83,0%	76,64%	81,75%	$\begin{bmatrix} 84 & 10 & 0 & 4 & 2 & 0 \\ 12 & 78 & 0 & 2 & 2 & 6 \\ 6 & 16 & 72 & 2 & 2 & 2 \\ 0 & 10 & 0 & 84 & 2 & 4 \\ 2 & 2 & 4 & 8 & 84 & 0 \\ 0 & 2 & 0 & 0 & 2 & 96 \end{bmatrix}$

Fonte: o autor.

O classificador SVM apontou o teste 7 como pior tendo 51,67% de acurácia; este utilizava a cor dos objetos dentro da imagem como característica. A melhor acurácia foi vista nos testes 1 e 4, ambos com de 90,33%. As tabelas de confusão dos teste 1 e 4 para

o SVM mostram que houve uma grande taxa de acertos para as notas de R$ 100,00 e R$ 2,00, e a que apresentou maior confusão foi a nota de R$ 20,00.

Tabela 5.2 – Resultados dos testes para cada configuração usando KNN.

(continua)

Id	**Acurácia**	**Precisão**	**Sensibilidade**	**Matriz de confusão**
1	85,33%	82,22%	81,13%	$\begin{bmatrix} 96 & 0 & 2 & 0 & 2 & 0 \\ 2 & 76 & 10 & 4 & 8 & 0 \\ 0 & 14 & 76 & 6 & 4 & 0 \\ 2 & 0 & 12 & 80 & 6 & 0 \\ 4 & 6 & 4 & 2 & 84 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \end{bmatrix}$
2	87,0%	86,40%	82,42%	$\begin{bmatrix} 96 & 2 & 0 & 0 & 2 & 0 \\ 2 & 76 & 12 & 2 & 8 & 0 \\ 0 & 10 & 80 & 6 & 4 & 0 \\ 2 & 0 & 10 & 82 & 6 & 0 \\ 4 & 0 & 6 & 2 & 88 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \end{bmatrix}$
3	87,33%	86,69%	83,57%	$\begin{bmatrix} 94 & 2 & 2 & 0 & 2 & 0 \\ 4 & 78 & 10 & 0 & 8 & 0 \\ 0 & 10 & 80 & 6 & 4 & 0 \\ 2 & 0 & 8 & 84 & 6 & 0 \\ 4 & 0 & 6 & 2 & 88 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \end{bmatrix}$
4	87,67%	87,75%	83,54%	$\begin{bmatrix} 96 & 2 & 0 & 0 & 2 & 0 \\ 4 & 78 & 10 & 0 & 8 & 0 \\ 0 & 8 & 82 & 6 & 4 & 0 \\ 2 & 0 & 10 & 82 & 6 & 0 \\ 4 & 0 & 6 & 2 & 88 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \end{bmatrix}$

Tabela 5.2 – Resultados dos testes para cada configuração usando KNN.

(continuação)

Id	**Acurácia**	**Precisão**	**Sensibilidade**	**Matriz de confusão**
5	51,17%	43,67%	40,02%	$\begin{bmatrix} 69 & 6 & 1 & 9 & 15 & 0 \\ 11 & 34 & 26 & 9 & 16 & 4 \\ 7 & 17 & 55 & 19 & 2 & 0 \\ 19 & 20 & 13 & 36 & 9 & 3 \\ 20 & 17 & 19 & 7 & 35 & 2 \\ 9 & 1 & 0 & 2 & 10 & 78 \end{bmatrix}$
6	88,0%	86,36%	83,94%	$\begin{bmatrix} 98 & 0 & 2 & 0 & 0 & 0 \\ 0 & 82 & 16 & 0 & 2 & 0 \\ 0 & 12 & 76 & 6 & 6 & 0 \\ 0 & 0 & 0 & 92 & 8 & 0 \\ 4 & 2 & 6 & 8 & 80 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \end{bmatrix}$
7	49,67%	65,35%	46,45%	$\begin{bmatrix} 88 & 6 & 0 & 0 & 4 & 2 \\ 42 & 44 & 4 & 4 & 6 & 0 \\ 44 & 8 & 30 & 4 & 14 & 0 \\ 54 & 0 & 4 & 32 & 2 & 8 \\ 32 & 8 & 8 & 0 & 52 & 0 \\ 36 & 6 & 2 & 4 & 0 & 52 \end{bmatrix}$
8	79,33%	79,57%	71,33%	$\begin{bmatrix} 94 & 2 & 0 & 2 & 2 & 0 \\ 16 & 62 & 13 & 0 & 9 & 0 \\ 6 & 6 & 68 & 16 & 4 & 0 \\ 5 & 4 & 10 & 77 & 4 & 0 \\ 6 & 8 & 8 & 0 & 78 & 0 \\ 3 & 0 & 0 & 0 & 0 & 97 \end{bmatrix}$

Tabela 5.2 – Resultados dos testes para cada configuração usando KNN.

(conclusão)

Id	**Acurácia**	**Precisão**	**Sensibilidade**	**Matriz de confusão**
9	82,67%	79,46%	73,73%	$\begin{bmatrix} 94 & 0 & 0 & 0 & 6 & 0 \\ 0 & 68 & 12 & 10 & 8 & 2 \\ 0 & 6 & 84 & 8 & 2 & 0 \\ 2 & 0 & 8 & 78 & 10 & 2 \\ 6 & 12 & 8 & 2 & 72 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \end{bmatrix}$
10	83,0%	83,25%	77,87%	$\begin{bmatrix} 96 & 0 & 0 & 0 & 4 & 0 \\ 2 & 72 & 12 & 6 & 8 & 0 \\ 0 & 6 & 66 & 8 & 20 & 0 \\ 2 & 4 & 6 & 84 & 4 & 0 \\ 4 & 0 & 14 & 0 & 80 & 2 \\ 0 & 0 & 0 & 0 & 0 & 100 \end{bmatrix}$
11	58,67%	68,01%	34,16%	$\begin{bmatrix} 84 & 0 & 0 & 0 & 12 & 4 \\ 26 & 8 & 0 & 4 & 18 & 44 \\ 8 & 2 & 48 & 4 & 16 & 22 \\ 8 & 0 & 0 & 62 & 12 & 18 \\ 0 & 0 & 0 & 0 & 52 & 48 \\ 2 & 0 & 0 & 0 & 0 & 98 \end{bmatrix}$
12	73,67%	74,83%	64,88%	$\begin{bmatrix} 86 & 0 & 4 & 6 & 2 & 2 \\ 20 & 58 & 8 & 8 & 0 & 6 \\ 12 & 14 & 60 & 2 & 6 & 6 \\ 8 & 2 & 2 & 82 & 4 & 2 \\ 12 & 8 & 0 & 14 & 64 & 2 \\ 2 & 0 & 0 & 2 & 4 & 92 \end{bmatrix}$

Fonte: o autor.

O classificador KNN apresentou acurácias variando de 49,67%, sendo o teste 7, até 88% para o teste 6, que utiliza-se da divisão da imagem em sub-imagens para realizar uma votação com base na distribuição das cores em cada sub-imagem. Para o teste 6

no KNN, as notas de R$ 2,00, R$ 20,00 e R$ 100,00 tiveram os maiores acertos e houve confusão com as notas de R$ 10,00.

Tabela 5.3 – Resultados dos testes para cada configuração usando MLP.

(continua)

Id	**Acurácia**	**Precisão**	**Sensibilidade**	**Matriz de confusão**
1	86,67%	85,69%	86,60%	$\begin{bmatrix} 95 & 0 & 0 & 0 & 5 & 0 \\ 0 & 92 & 3 & 2 & 3 & 0 \\ 2 & 8 & 81 & 6 & 3 & 0 \\ 1 & 2 & 9 & 80 & 6 & 2 \\ 9 & 5 & 8 & 5 & 72 & 1 \\ 0 & 0 & 0 & 0 & 0 & 100 \end{bmatrix}$
2	84,67%	85,98%	81,23%	$\begin{bmatrix} 95 & 0 & 0 & 0 & 5 & 0 \\ 2 & 83 & 9 & 3 & 3 & 0 \\ 3 & 10 & 80 & 6 & 1 & 0 \\ 5 & 0 & 9 & 80 & 4 & 2 \\ 10 & 4 & 7 & 7 & 70 & 2 \\ 0 & 0 & 0 & 0 & 0 & 100 \end{bmatrix}$
3	84,67%	87,32%	82,37%	$\begin{bmatrix} 94 & 0 & 0 & 0 & 6 & 0 \\ 1 & 85 & 7 & 2 & 4 & 1 \\ 3 & 7 & 83 & 7 & 0 & 0 \\ 3 & 1 & 14 & 75 & 5 & 2 \\ 8 & 2 & 8 & 9 & 71 & 2 \\ 0 & 0 & 0 & 0 & 0 & 100 \end{bmatrix}$
4	84,67%	88,42%	81,08%	$\begin{bmatrix} 96 & 0 & 0 & 0 & 4 & 0 \\ 2 & 82 & 12 & 2 & 2 & 0 \\ 3 & 8 & 82 & 6 & 1 & 0 \\ 4 & 1 & 14 & 76 & 4 & 1 \\ 9 & 1 & 7 & 9 & 72 & 2 \\ 0 & 0 & 0 & 0 & 0 & 100 \end{bmatrix}$

Tabela 5.3 – Resultados dos testes para cada configuração usando MLP.

(continuação)

Id	Acurácia	Precisão	Sensibilidade	Matriz de confusão
5	62,0%	56,89%	51,38%	$\begin{bmatrix} 82 & 4 & 0 & 3 & 7 & 4 \\ 5 & 47 & 22 & 13 & 12 & 1 \\ 0 & 14 & 71 & 11 & 3 & 1 \\ 8 & 16 & 17 & 43 & 6 & 10 \\ 11 & 6 & 8 & 14 & 46 & 15 \\ 7 & 0 & 0 & 3 & 7 & 83 \end{bmatrix}$
6	83,83%	82,81%	78,29%	$\begin{bmatrix} 95 & 2 & 1 & 0 & 2 & 0 \\ 0 & 77 & 17 & 2 & 4 & 0 \\ 0 & 9 & 80 & 6 & 5 & 0 \\ 1 & 0 & 9 & 80 & 9 & 1 \\ 3 & 5 & 9 & 11 & 71 & 1 \\ 0 & 0 & 0 & 0 & 0 & 100 \end{bmatrix}$
7	48,17%	58,83%	48,32%	$\begin{bmatrix} 60 & 4 & 9 & 18 & 5 & 4 \\ 14 & 52 & 8 & 19 & 5 & 2 \\ 14 & 10 & 39 & 24 & 9 & 4 \\ 23 & 5 & 13 & 44 & 7 & 8 \\ 12 & 12 & 17 & 14 & 40 & 5 \\ 12 & 3 & 8 & 21 & 2 & 54 \end{bmatrix}$
8	80,17%	75,84%	75,52%	$\begin{bmatrix} 94 & 0 & 2 & 0 & 4 & 0 \\ 1 & 74 & 7 & 0 & 17 & 1 \\ 0 & 15 & 73 & 11 & 1 & 0 \\ 0 & 2 & 20 & 73 & 4 & 1 \\ 8 & 9 & 8 & 4 & 70 & 1 \\ 0 & 2 & 1 & 0 & 0 & 97 \end{bmatrix}$

Tabela 5.3 – Resultados dos testes para cada configuração usando MLP.

(conclusão)

Id	**Acurácia**	**Precisão**	**Sensibilidade**	**Matriz de confusão**
9	80,5%	80,31%	74,78%	$\begin{bmatrix} 97 & 0 & 0 & 0 & 3 & 0 \\ 2 & 74 & 13 & 6 & 5 & 0 \\ 1 & 8 & 80 & 8 & 3 & 0 \\ 4 & 0 & 13 & 66 & 13 & 4 \\ 6 & 8 & 8 & 9 & 67 & 2 \\ 0 & 0 & 0 & 0 & 1 & 99 \end{bmatrix}$
10	81,33%	77,25%	77,35%	$\begin{bmatrix} 97 & 0 & 0 & 0 & 3 & 0 \\ 0 & 79 & 0 & 4 & 17 & 0 \\ 0 & 8 & 70 & 13 & 9 & 0 \\ 0 & 4 & 13 & 77 & 6 & 0 \\ 6 & 9 & 14 & 4 & 65 & 2 \\ 0 & 0 & 0 & 0 & 0 & 100 \end{bmatrix}$
11	58,33%	53,62%	49,97%	$\begin{bmatrix} 72 & 7 & 5 & 6 & 5 & 5 \\ 14 & 47 & 5 & 9 & 12 & 13 \\ 4 & 7 & 46 & 14 & 24 & 5 \\ 3 & 6 & 10 & 66 & 8 & 7 \\ 2 & 12 & 13 & 10 & 41 & 22 \\ 5 & 4 & 2 & 5 & 6 & 78 \end{bmatrix}$
12	87,83%	88,11%	83,12%	$\begin{bmatrix} 92 & 7 & 0 & 1 & 0 & 0 \\ 5 & 77 & 8 & 4 & 5 & 1 \\ 3 & 3 & 84 & 2 & 2 & 6 \\ 0 & 0 & 4 & 88 & 4 & 4 \\ 2 & 0 & 4 & 6 & 88 & 0 \\ 0 & 0 & 0 & 0 & 2 & 98 \end{bmatrix}$

Fonte: o autor.

Por fim, os resultados do MLP mostram que o melhor teste foi o 12, que utilizava o SIFT, e o pior o teste 7. Para o teste 12, as classes menos confusas foram a de R$ 2,00 e R$ 100,00 e a mais confusa foi a de R$ 5,00.

Além da análise da melhor configuração para cada classificador também é possível realizar comparações entre as opções de processamentos e características usadas. Na comparação entre os filtros de borramento, presentes nos testes 1, 2, 3 e 4, é possível ver que os filtros não apresentaram grande diferença para o SVM, sendo o melhor resultado sem o uso de filtros. Para o KNN, o Bilateral foi o melhor e o pior resultado foi quando não foram utilizados filtros. Para o MLP, nenhum filtro é o melhor resultado.

Comparando a forma de representar as informações de cor da imagem, foram testadas as características do Histograma Completo, Filtrado e Reduzido. O melhor foi o Histograma Completo e o pior, o Reduzido, para os três classificadores.

Também existe a comparação que pode ser realizada entre utilizar a cor da imagem, a forma dos objetos ou a imagem completa. Para isso, foram comparados o Histograma Completo (cor), o SIFT (forma) e a matriz do canal H (imagem completa). O Histograma Completo foi o melhor resultado obtido com KNN e SVM. Já o MLP indicou o SIFT como melhor característica.

Por fim, vendo os resultados dos classificadores é possível notar que todos concordaram em qual seria a pior característica, sendo a do teste 7, que utiliza a segmentação dos números. Porém, houve divergência na melhor característica. Já na avaliação em relação às classes, os três classificadores não tiveram dificuldades para classificar notas de R$ 100,0 e R$ 2,00, mas cada um teve uma classe mais confusa diferente.

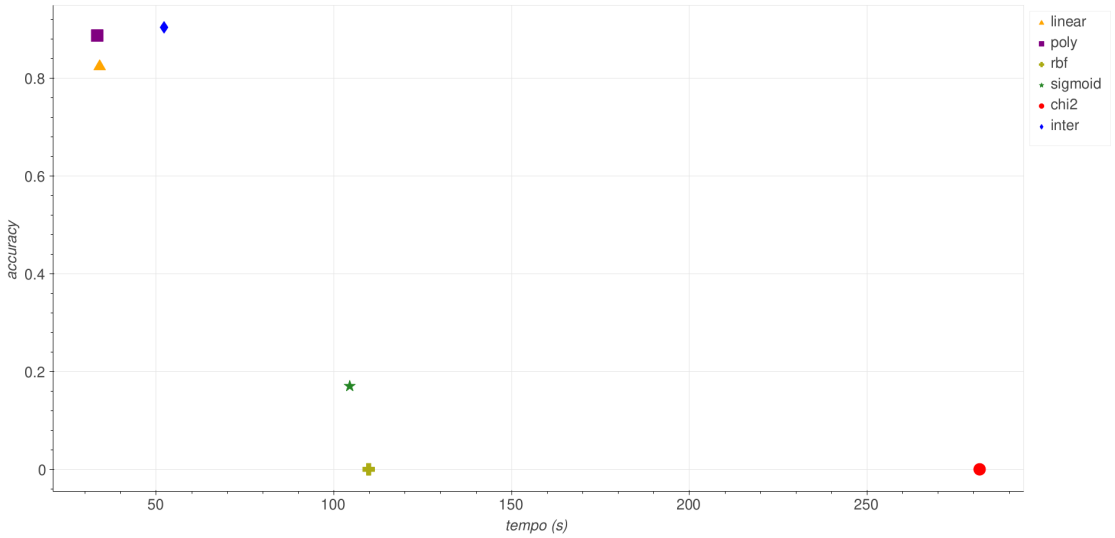
5.2 Resultados específicos do melhor teste

Dos 39 classificadores testados, dois obtiveram a maior acurácia de 90,33%, sendo ambos utilizando o método SVM. O critério de escolha utilizado foi uma média entre as três métricas que resultou em 89,28% para o teste 1 e para o teste 4 89,04%. Desta forma, a melhor configuração para diferenciar as notas foi a 1. Para esta configuração foi realizado um *grid seach* do classificador SVM para obter os melhores parâmetros. Para cada parâmetro foram testados 5 valores diferentes.

A Figura 5.1 apresenta os resultados obtidos na comparação entre os *kernels* do SVM. Para os demais valores foram utilizados C=0,1. Analisando o gráfico é possível ver que o melhor *kernel* foi o Interseção de Histograma seguido pelo Polinomial. O tempo de execução do *cross validation* para o Polinomial foi menor, mas não muito menor. Por isso o *kernel* escolhido foi o Interseção de Histograma.

A Figura 5.2 apresenta os resultados obtidos na comparação entre os valores 0,1,

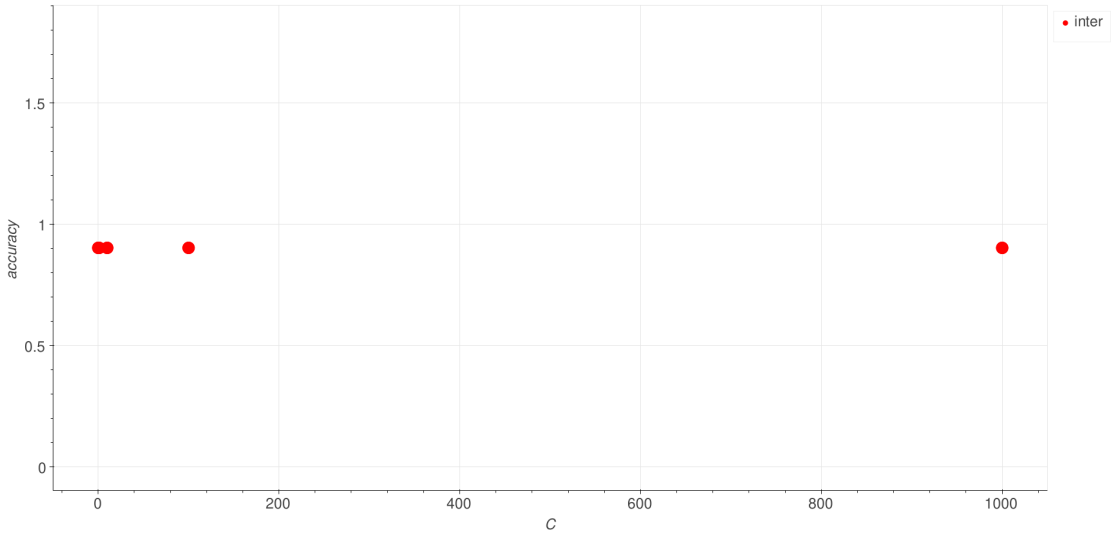
Figura 5.1 – Acurácia pelo tempo de execução dos *kernels* SVM.



Fonte: o autor.

1, 10, 100 e 1000 de C do SVM. Analisando o gráfico é possível ver que a variação do C não provocou interferência no resultado. Assim, foi escolhido o valor 0,1 por apresentar menor tempo de execução.

Figura 5.2 – Parâmetros de C por Acurácia.



Fonte: o autor.

5.3 Resultados do celular

Após a construção do aplicativo com a melhor técnica de classificação encontrada, ou seja, a SVM recebendo o histograma do canal H de uma imagem que não passou por

nenhum filtro, foi obtido a acurácia, sensibilidade, precisão e a matriz de confusão dos resultados do celular.

Foram criadas duas condições para se obter os testes. Na primeira condição, chamada de ideal, o aplicativo foi usado em um ambiente iluminado pela luz do sol com as notas sobre a mesa. Neste, foram usadas 5 notas para cada classe e obtidas 35 imagens diferentes entre si.

Na segunda condição, chamada de péssima, o aplicativo foi usado em uma sala fechada iluminada por uma lampada, foi usada a parte mais escura do local e as notas foram seguradas com a mão. Para este foi usada somente uma nota de cada valor para obter 6 imagens diferentes para o aplicativo classificar.

A Tabela 5.4 apresenta os resultados para a condição ideal de manipulação do aplicativo. A Figura 5.3 apresenta imagens das notas classificadas erradas, onde as notas de R$ 2,00, R$ 10,00, R$ 20,00 e R$ 50,00 foram classificadas como sendo $ 100,00, R$ 20,00, R$ 50,00 e R$ 2,00, respectivamente.

Tabela 5.4 – Resultados do aplicativo nas condições ideais de operação.

Acurácia	**Precisão**	**Sensibilidade**	**Matriz de confusão**
87,62%	89,11%	94,03%	$\begin{bmatrix} 32 & 0 & 0 & 0 & 1 & 2 \\ 0 & 35 & 0 & 0 & 0 & 0 \\ 0 & 0 & 34 & 1 & 0 & 0 \\ 0 & 0 & 2 & 17 & 16 & 0 \\ 2 & 0 & 1 & 1 & 31 & 0 \\ 0 & 0 & 0 & 0 & 0 & 35 \end{bmatrix}$

Fonte – Produzido pelo autor.

A Tabela 5.5 apresenta os resultados para a condição péssima para a utilização do aplicativo. A Figura 5.4 apresenta imagens das notas classificadas erradas, onde as notas de R$ 2,00, R$ 5,00, R$ 10,00, R$ 20,00 e R$ 50,00 foram classificadas como sendo R$ 20,00, R$ 50,00, R$ 5,00, R$ 10,00 e R$ 100,00, respectivamente.

Figura 5.3 – Imagens classificadas erroneamente pelo APP nas melhores condições.

Fonte: o autor.

Tabela 5.5 – Resultados do aplicativo nas condições péssimas de operação.

Acurácia	**Precisão**	**Sensibilidade**	**Matriz de confusão**
41,67%	23,69%	33,87%	$\begin{bmatrix} 5 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 4 & 1 \\ 0 & 5 & 0 & 0 & 1 & 0 \\ 0 & 0 & 3 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 3 & 3 \\ 0 & 0 & 0 & 0 & 0 & 6 \end{bmatrix}$

Fonte – Produzido pelo autor.

Figura 5.4 – Imagens classificadas erroneamente pelo APP nas piores condições.

Fonte: o autor.

5.4 Considerações finais

Foi realizada uma grande bateria de testes para se escolher o melhor algoritmo classificador de notas de Real. O critério utilizado para escolher e comparar estes algoritmos foi a acurácia dos resultados obtidos usando *cross-validation leave-one-out*, tendo como suporte os valores da precisão e sensibilidade.

O melhor resultado foi conseguido utilizando o SVM com a imagem processada unicamente pela conversão do modelo RGB para HSV. Também foi visto que as notas de R$ 2,00 e R$ 100,00 foram as que apresentaram os melhores resultados usando a análise da matriz de confusão. Para este melhor resultado foi realizado o processo de *grid search* para encontrar o melhores parâmetros que ajudem a aumentar o valor da acurácia. Foi visto que a mudança do *kernel* do SVM causava grande impacto na classificação, mas o que apresentou melhor resultado foi o Interseção de Histograma. O parâmetro C, ao contrário do *kernel*, não apresentou impacto no valor da acurácia, assim foi escolhido o que apresentou menor tempo de execução, sendo C=0,1.

Este algoritmo foi implementado e testado no celular. Para validar o aplicativo foram criados dois testes, o primeiro simulava condições ideais de uso em um local bem iluminado e com a nota bem enquadrada na imagem, o outro foi realizado em um ambiente escuro com objetos presentes na imagem que não fazem parte da cédula.

Em condições ideais de uso, a nota apresentou 87,62% de acurácia, um valor abaixo do encontrado nos testes realizados usando o banco de dados, mas só 2,71% de diferença. A nota de R$100,00 continua tendo uma maior quantidade de acertos, mas foi inesperado o resultado para a nota de R$ 5,00. A nota mais confusa continua sendo a de R$ 20,00.

Já nas piores condições é visto uma acurácia de 41,67%, também foi visto que as notas mais impactadas pela confusão do aplicativo foram as de R$ 5,00, R$ 10,00 e R$ 20,00, mas a de R$ 100,00 manteve-se com 100% de acertos mesmo nestas condições.

Logo, a obtenção dos resultados usando quatro métricas de análise permitiu escolher o melhor algoritmo de classificação com os melhores parâmetros de configuração. E este algoritmo escolhido possibilitou a criação de um aplicativo capaz de reconhecer e diferenciar os valores das cédulas de Real com uma acurácia de 87,62%.

6 Conclusão

Os deficientes visuais apresentam certas dificuldades para identificar as cédulas de Real, mesmo estas apresentando características de acessibilidade, como tamanhos diferentes. Assim, é proposta a criação de um aplicativo que capture uma imagem da cédula para a qual o usuário tenha dúvida do valor e a inteligência artificial do aplicativo retorne o valor correto desta.

O objetivo deste trabalho foi criar um aplicativo para o reconhecimento de cédulas monetárias que utiliza um conjunto de técnicas tanto de processamento de imagem, extração de características e de classificadores para o reconhecimento de objetos. Estas técnicas foram comparadas entre si para escolher a que geraria maior eficiência, comprovada com base nos experimentos realizados.

Para tanto, é necessário entender conceitos e fundamentos por trás de cada uma das ferramentas utilizadas no projeto. Dentre os recursos utilizados, destacam-se a linguagem de programação utilizada, os métodos de manipulação de imagens digitais, procedimentos de obtenção de atributos responsáveis por diferenciar os grupos de imagens entre si, métodos pertencentes ao estudo referente ao reconhecimento de objetos e finalmente a possibilidade de incorporar todos os passos anteriores em um sistema Android.

Este projeto é orientado a implementar e estudar a utilização de algoritmos de reconhecimento de objetos já sancionados pela comunidade científica: SVM, KNN e ANN. Esses classificadores são usados para a identificação de cédulas monetárias de modo satisfatório para o uso geral do público, direcionado para os deficientes visuais. A linguagem de programação Python foi usada para o desenvolvimento do algoritmo contendo um desses classificadores, que será escolhido em uma etapa de comparação. O Python também foi utilizado para a criação da *interface* do aplicativo compatível com sistemas Android.

Dos processamentos de imagens propostos e testados são eles: filtros para remover ruído como Gaussiano, a Mediana e o Bilateral; conversões de imagem para HSV e tons de cinza; e outros processamentos como a utilização do K-Means, equalização de histograma, limiarização, filtro morfológico de fechamento e Canny para detecção de contornos. Cada um destes foi usado em conjunto ou separadamente para se extrair diferentes características de cor e contorno da imagem com as cédulas. Ao todo foram gerados 12 processamentos de imagens que resultaram em 12 vetores de características distintos entre si. Estes vetores foram aplicados aos classificadores para a verificação do desempenho

destes ao usá-los.

Os resultados obtidos mostraram desempenhos variando de 49,76% até 90% de acurácia dos classificadores. Apresar de alguns apresentarem desempenho baixo, o importante foi obter acurácias altas, acima de 90%, pois somente o melhor algoritmo é que será introduzido no aplicativo. Também é possível ver que a estratégia de testar vários tipos de processamento e classificadores ajudou a encontrar um que gerasse uma acurácia alta o suficiente para permitir a construção do aplicativo.

O melhor teste foi obtido usado o SVM com processamento de imagem que realiza a conversão para o formato HSV. Foi realizado um *grid search* para verificar como a variação dos parâmetros impacta nos valores de acurácia e foi visto que somente a variação do *kernel* provocou grande mudanças, fazendo este classificador cair para valores de acurácia de 0%. Com isso é possível ver a importância da variação dos parâmetros dos classificadores, pois eles podem mudar drasticamente o desempenho final. Foi notado também o impacto no tempo de processamento de cada *kernel*. Logo, observando o tempo e a acurácia, o melhor é o *kernel* Interseção de Histograma. É importante apresentar que a execução do *grid search* somente para o melhor resultado pode ter causado um erro na escolha do melhor combinação de processamentos e classificador. O procedimento ideal seria realizar o *grid search* para cada um dos testes, mas devido ao longo tempo de execução que esta tarefa levaria este procedimento não seria viável.

O aplicativo tem a capacidade processar a imagem obtida por sua câmera em tempo real, onde são aplicadas a melhor técnica de programação testada para o reconhecimento de objetos, permitindo o reconhecimento das notas do Real de R$ 2,00, de R$ 5,00, de R$ 10,00, de R$ 20,00,de R$ 50,00 e de R$ 100,00 com uma acurácia de quase 88%(64).

Pensando na utilização diária do aplicativo, foi visto que dependendo das condições de uso sua acurácia pode variar. Assim foram realizados dois teste para medir a acurácia miníma e máxima. No teste com as melhores condições foi obtido um valor de acurácia de 87,62% e nas piores se obteve 41,67%.

Analisando estes dois resultados do aplicativo foi visto a sua sensibilidade a luminosidade e a elementos estranhos na imagem como dedos e plano de fundo. As notas de R$ 100,00 de R$ 2,00 tiveram poucos erros porque ela não apresentam uma cor mais avermelhada, ao contrário das outras, ajudando na classificação e a nota de R$ 100,00 tem a vantagem de sofrer menos desgaste, pois existe um cuidado maior das pessoas ao utilizá-la e normalmente ela é obtida diretamente de caixas eletrônicos.

Com isso, pode-se dizer que o objetivo geral do trabalho, sendo a construção de um aplicativo que reconhecia notas do Real brasileiro, foi atingido. Comprovando também as hipóteses do trabalho, onde se afirmava a possibilidade da construção de um aplicativo de inteligência artificial com Python que reconhece os objetos presentes na imagem capturada pela câmera do aparelho e que este aplicativo, após realizados diversos teste, teria uma alta taxa de acertos.

Visando melhorar o aplicativo nas próximas versões, foi-se pensado em outras funcionalidades que são relevantes de serem implementadas no aplicativo como o reconhecimento de notas estrangeiras e reconhecimento de notas falsas. Esta última funcionalidade pode mudar o público alvo, se tornando uma ferramenta de interesse para se ter no bolso, para qualquer pessoa. Usar mais de um classificador para detectar notas. Controlar a intensidade do brilho da lanterna celular para reduzir a influência da iluminação do ambiente. Outra característica que pode ser adicionada ao aplicativa é o reconhecimento se qualquer nota está ou não presente ou não na imagem, isso elevaria a qualidade do aplicativo, evitando que o mesmo classifique o chão ou a mão da pessoa em uma nota de R$ 50,00. Outras funções funções que podem ser adicionadas são sensor de luminosidade e detecção cartão de crédito. No futuro podem ser testadas também outros algoritmos visando aumentar a eficiência do aplicativo como os de processamento de imagem como HOG (*Histogram of Oriented Gradients* - Histograma de Gradientes Orientado) e outros classificadores como a rede neural profunda e arvore de decisão. Também é possível usar técnicas de redução de dimensões do vetor de característica como o PCA (*Principal Component Analysis* - Análise de componentes principais). Por fim, uma opção para aumentar a qualidade do aplicativo sem usar novos recursos é usar um número ímpar de classificadores no aplicativo para realizar uma votação do valor da nota entre os classificadores, mas isso poderia aumentar o tempo de classificação no aplicativo, tornando-o mais lento.

Referências

1 GONZALEZ, R. **Processamento Digital De Imagens**. [S.l.]: ADDISON WESLEY BRA, 2010. ISBN 9788576054016.

2 PEDRINI, H.; SCHWARTZ, W. **Análise de imagens digitais: princípios, algoritmos e aplicações**. São Paulo: CENGAGE - UM LIVRO, 2008. ISBN 9788522105953.

3 SHIH, F. Y. **Image processing and pattern recognition: fundamentals and techniques**. [S.l.]: Wiley, 2010. ISBN 9780470590409.

4 PORTAL DE PESQUISAS TEMÁTICAS E EDUCACIONAIS. **Cinco Sentidos**. 2018. Disponível em: https://www.suapesquisa.com/pesquisa/cinco_sentidos.htm. Acesso em: 23 ago. 2018.

5 ANDROID ACCESSIBILITY HELP. **Get started on Android with TalkBack**. 2018. Disponível em: https://support.google.com/accessibility/android/answer/6283677?hl=en. Acesso em: 23 ago. 2018.

6 GOOGLE PLAY. **Android Accessibility Suite**. 2018. Disponível em: https://play.google.com/store/apps/details?id=com.google.android.marvin.talkback&hl=pt_BR. Acesso em: 23 ago. de 2018.

7 Hayaku s. r. o. **Cash Reader**. 2019. Disponível em: https://cashreader.app/en/. Acesso em: 09 nov. 2022.

8 TARSASAG, J. I. K. F. **LetSeeApp**. 2020. Disponível em: https://apps.apple.com/br/app/letseeapp/id1170643143. Acesso em: 20 dez. 2022.

9 Hayaku s. r. o. **Qatari Money Reader**. 2019. Disponível em: https://apps.apple.com/br/app/qatari-money-reader/id1314377588. Acesso em: 20 dez. 2022.

10 MCT DATA. **LetSeeApp**. 2020. Disponível em: https://play.google.com/store/apps/details?id=com.mctdata.ParaTanima. Acesso em: 20 dez. 2022.

11 COLNECT COLLECTORS COMMUNITY. **Banknote Identifier**. 2022. Disponível em: https://play.google.com/store/apps/details?id=com.colnect.identify.banknote&hl=en_US&gl=US. Acesso em: 21 dez. 2022.

12 SERVIÇOS E INFORMAÇÕES DO BRASIL. **Dinheiro Brasileiro**. 2022. Disponível em: https://play.google.com/store/apps/details?id=br.gov.bcb.mobile.android.appnotas. Acesso em: 21 dez. 2022.

13 PURGATO, V. **Aluno de Engenharia de Computação desenvolve aplicativo para cegos**. 2018. Disponível em: https://www.puc-campinas.edu.br/aluno-de-engenharia-de-computacao-desenvolve-aplicativo-para-cegos/. Acesso em: 09 nov. 2022.

14 CARVALHO, L. **Android cresce no Brasil e aumenta distância para iOS e Windows Phone**. 2017. Disponível em: https://olhardigital.com.br/noticia/android-cresce-no-brasil-e-aumenta-distancia-para-ios-e-windows-phone/68023. Acesso em: 24 ago. 2018.

15 LOPES, L. **Como os cegos diferenciam as notas de dinheiro?** 2018. Disponível em: http://revistaepoca.globo.com/Revista/Epoca/0,,EMI103120-15223,00-COMO+OS+CEGOS+DIFERENCIAM+AS+NOTAS+DE+DINHEIRO.html. Acesso em: 31 ago. 2018.

16 CHORAS, R. S. Image feature extraction techniques and their applications for cbir and biometrics systems. **International Journal of Biology and Biomedical Engineering**, v. 1, 01 2007.

17 INTEL SOFTWARE. **Color Models.** 2019. Disponível em: https://software.intel.com/en-us/ipp-dev-reference-color-models. Acesso em: 05 dez. de 2019.

18 KIM, Y. et al. Robust color-to-gray via nonlinear global mapping. **ACM SIGGRAPH Asia 2009 papers**, 2009.

19 ITU-T. **Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios**. Geneva, 2011.

20 FITRIYAH, H.; WIHANDIKA, R. C. An analysis of rgb, hue and grayscale under various illuminations. In: **2018 International Conference on Sustainable Information Engineering and Technology (SIET)**. [S.l.: s.n.], 2018. p. 38–41.

21 ROBERT FISHER. **Bilateral Filtering for Gray and Color Images**. 2022. Disponível em: https://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/MANDUCHI1/Bilateral_Filtering.html. Acesso em: 06 dez. 2022.

22 Wikipedia contributors. **Gaussian filter — Wikipedia, The Free Encyclopedia**. 2019. https://en.wikipedia.org/w/index.php?title=Gaussian_filter&oldid=921876876. [Online; accessed 5-December-2019].

23 LEACH, R. (Ed.). **Fundamental Principles of Engineering Nanometrology (Second Edition)**. Second edition. Oxford: William Andrew Publishing, 2014. (Micro and Nano Technologies). ISBN 978-1-4557-7753-2. Disponível em: http://www.sciencedirect.com/science/article/pii/B9781455777532000153.

24 OPENCV TEAM. **Miscellaneous Image Transformations**. 2020. Disponível em: https://play.google.com/store/apps/details?id=org.kivy.pygame&hl=en. Acesso em: 06 mar. 2020.

25 JUSTUSSON, B. I. Median filtering: Statistical properties. In: _____. **Two-Dimensional Digital Signal Prcessing II: Transforms and Median Filters**. Berlin, Heidelberg: Springer Berlin Heidelberg, 1981. p. 161–196. ISBN 978-3-540-38446-5. Disponível em: https://doi.org/10.1007/BFb0057597.

26 OPENCV TEAM. **Kivy Launcher.** 2020. Disponível em: https://docs.opencv.org/4.x/d7/d1b/group___imgproc___misc.html#gaa9e58d2860d4afa658ef70a9b1115576. Acesso em: 06 mar. 2020.

27 OPENCV TEAM. **Canny Edge Detection**. 2020. Disponível em: https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html. Acesso em: 06 mar. 2020.

28 OPENCV TEAM. **Histograms - 1 : Find, Plot, Analyze !!!** 2020. Disponível em: https://docs.opencv.org/4.x/d1/db7/tutorial_py_histogram_begins.html. Acesso em: 06 mar. 2020.

29 OPENCV TEAM. **Histogram Equalization**. 2022. Disponível em: https://docs.opencv.org/3.4/d4/d1b/tutorial_histogram_equalization.html. Acesso em: 20 dez. 2022.

30 DUBEY, A.; CHOUBEY, A. A systematic review on k-means clustering techniques. **Int J Sci Res Eng Technol (IJSRET, ISSN 2278–0882)**, v. 6, n. 6, 2017.

31 OPENCV TEAM. **K-Means Clustering in OpenCV**. 2022. Disponível em: https://docs.opencv.org/3.4/d1/d5c/tutorial_py_kmeans_opencv.html. Acesso em: 20 dez. 2022.

32 OPENCV TEAM. **Morphological Transformations**. 2020. Disponível em: https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html. Acesso em: 06 mar. 2020.

33 GONZÁLES, G. L. G. **Aplicação da Técnica SIFT para Determinação de Campos de Deformações de Materiais usando Visão Computacional**. Dissertação (Mestrado) — Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, RJ, 2010. Disponível em: https://www.maxwell.vrac.puc-rio.br/17050/17050_5.PDF. Acesso em: 13 jan. 2023.

34 OPENCV TEAM. **Introduction to SIFT (Scale-Invariant Feature Transform)**. 2022. Disponível em: https://docs.opencv.org/4.x/da/df5/tutorial_py_sift_intro.html. Acesso em: 13 jan. 2023.

35 SHALEV-SHWARTZ S. E BEN-DAVID, S. **Understanding Machine Learning: From Theory to Algorithms**. Cambridge University Press, 2014. ISBN 9781107057135. Disponível em: https://books.google.com.br/books?id=ttJkAwAAQBAJ.

36 SHARMA, S. **SVM: What makes it superior to the Maximal-Margin and Support Vector Classifiers?** 2022. Disponível em: https://www.analyticsvidhya.com/blog/2021/05/support-vector-machines/. Acesso em: 12 jul. 2022.

37 OPENCV TEAM. **Smoothing Images**. 2022. Disponível em: https://docs.opencv.org/3.4/dc/dd3/tutorial_gausian_median_blur_bilateral_filter.html. Acesso em: 6 dez. 2022.

38 PARK, C.-S. et al. Automatic modulation recognition of digital signals using wavelet features and svm. **International Conference on Advanced Communication Technology, ICACT**, v. 1, p. 387 – 390, fev. 2008.

39 S., A.; ZHANG, Y. A review on back-propagation neural networks in the application of remote sensing image classification. **Journal of Earth Science and Engineering**, v. 5, 01 2015.

40 OPENCV TEAM. **cv::ml::ANN_MLP Class Reference**. 2022. Disponível em: https://docs.opencv.org/4.x/d0/dce/classcv_1_1ml_1_1ANN_MLP.html. Acesso em: 20 dez. 2022.

41 SHARMA, A.; SHRIMALI, V. R.; BEYELER, M. **MACHINE LEARNING FOR OPENCV 4 : intelligent algorithms for building image processing apps... using opencv 4, python, and scikit-learn, 2nd edit.** 2. ed. [S.l.]: PACKT PUBLISHING LIMITED, 2019. ISBN 9781789536300,1789536308.

42 GUAMÁ, J. **Métricas de avaliação de classificadores**. 2019. Disponível em: https://medium.com/pyladiesbh/m%C3%A9tricas-de-avalia%C3%A7%C3%A3o-de-classificadores-6aadc3dacd51. Acesso em: 09 nov. 2022.

43 GRANDINI, M.; BAGLI, E.; VISANI, G. Metrics for multi-class classification: an overview. **ArXiv**, abs/2008.05756, 2020.

44 PEDREGOSA, F. et al. Scikit-learn: Machine learning in Python. **Journal of Machine Learning Research**, v. 12, p. 2825–2830, 2011.

45 LYASHENKO, V. **Cross-Validation in Machine Learning: How to Do It Right**. 2022. Disponível em: https://neptune.ai/blog/cross-validation-in-machine-learning-how-to-do-it-right. Acesso em: 05 jan. 2023.

46 DIKE, H. U. et al. Unsupervised learning based on artificial neural network: a review. **2018 IEEE INTERNATIONAL CONFERENCE ON CYBORG AND BIONIC SYSTEMS (CBS)**, v. 1, n. 2, p. 322–327, 2018.

47 SCIKIT-LEARN DEVELOPERS. **Scikit-learn**. [S.l.], 2019. Disponível em: https://scikit-learn.org/0.21/_downloads/scikit-learn-docs.pdf. Acesso em: 29 ago. 2021.

48 PUTANO, B. **Java vs. Python: Coding Battle Royale**. 2019. Disponível em: https://stackify.com/java-vs-python/?utm_referrer=https://www.google.com.br. Acesso em: 30 set. 2019.

49 IBM COMMUNITY. **iMasters**. 2018. Disponível em: https://www.ibm.com/developerworks/community/blogs/fd26864d-cb41-49cf-b719-d89c6b072893/entry/primeiros_passos_com_pil_a_biblioteca_de_imagens_do_python2?lang=en. Acesso em: 24 ago. de 2018.

50 KIVY. **Kivy**. 2020. Disponível em: https://kivy.org/#home. Acesso em: 05 jan. 2020.

51 OPENCV. **OpenCV-Python Tutorials**. 2020. Disponível em: https://docs.opencv.org/4.2.0/d6/d00/tutorial_py_root.html. Acesso em: 12 mar. 2020.

52 KIVY. **Kivy Launcher.** 2020. Disponível em: https://play.google.com/store/apps/details?id=org.kivy.pygame&hl=en. Acesso em: 06 mar. 2020.

53 PUTANO, B. **A Look At 5 of the Most Popular Programming Languages of 2019**. 2019. Disponível em: https://stackify.com/popular-programming-languages-2018/. Acesso em: 07 out. 2019.

54 WIKIPEDIA CONTRIBUTORS. **Tipo de dado — Wikipédia, a enciclopédia livre**. 2018. Disponível em: https://pt.wikipedia.org/w/index.php?title=Tipo_de_dado&oldid=51450813. Acesso em: 07 out. 2019.

55 BORGES, L. E. **Python para Desenvolvedores**. 2018. Disponível em: https://ark4n.wordpress.com/python. Acesso em: 07 out. 2019.

56 MATTHES, E. **Curso Intensivo de Python: Uma Intridução Prática e Baseada em Projetos à Programação**. 1. ed. São Paulo: No Starch Press, 2015. Disponível em: https://books.google.com.br/books?id=igYvDwAAQBAJ&printsec=frontcover&dq=curso+intensivo+de+python&hl=en&sa=X&ved=0ahUKEwi--9OC5YrlAhXLI7kGHYyxDWEQ6AEIMjAB#v=onepage&q&f=false. Acesso em: 07 out. 2019.

57 GRIES, P.; CAMPBELL, J.; MONTOJO, J. **Practical Programming, Third Edition: An Introduction to Computer Science Using Python 3.6**. 3. ed. Raleigh, North Carolina: Pragmatic Bookshelf, 2017. ISBN 139781680502688.

58 NUMPY COMMUNITY. **NumPy Reference**. 2016. Disponível em: https://docs.scipy.org/doc/numpy-1.11.0/numpy-ref-1.11.0.pdf. Acesso em: 07 out. 2019.

59 RASCHKA, S.; MIRJALILI, V. **Machine Learning con Python - Nuova edizione: Costruire algoritmi per generare conoscenza**. [S.l.]: Feltrinelli Editore, 2021. ISBN 9788850318933.

60 GALIMBERTI, M. **python-for-android.** 2020. Disponível em: https://github.com/kivy/python-for-android. Acesso em: 06 mar. 2020.

61 GALIMBERTI, M. **Buildozer.** 2020. Disponível em: https://github.com/kivy/buildozer. Acesso em: 06 mar. 2020.

62 TAYLOR, A. **Recipes**. 2015. Disponível em: https://python-for-android.readthedocs.io/en/latest/recipes/. Acesso em: 16 nov. 2022.

63 PEDREGOSA, F. et al. Scikit-learn: Machine learning in Python. **Journal of Machine Learning Research**, v. 12, p. 2825–2830, 2011.

64 TUTORIAIS DA OPENCV UTILIZANDO A LINGUAGEM DE PROGRAMAÇÃO PYTHON. **Reconhecimento de objetos com Python e OpenCV**. 2018. Disponível em: http://www.galirows.com.br/meublog/opencv-python/opencv2-python27/capitulo2-deteccao/reconhecimento-objetos. Acesso em: 24 ago. 2018.