Trabalho Conclusão Curso

José Victor Feijó de Araujo

ANÁLISE E CLASSIFICAÇÃO DE IMAGENS PARA APLICAÇÃO DE *OCR* EM CUPONS FISCAIS

Florianópolis
2017

Universidade Federal de Santa Catarina
Centro Tecnológico
Ciências da Computação

José Victor Feijó de Araujo

ANÁLISE E CLASSIFICAÇÃO DE IMAGENS PARA APLICAÇÃO DE *OCR* EM CUPONS FISCAIS

Trabalho Conclusão do Curso de Graduação em Ciências da Computação do Centro Tecnológico da Universidade Federal de Santa Catarina como requisito para a obtenção do Título de Bacharel em Ciências da Computação. Orientador: Prof. Dr. Elder Rizzon Santos. Coorientador: Prof. Dr. Alexandre Gonçalves Silva

Florianópolis
2017

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Araujo, José Victor Feijó de Araujo
Análise e classificação de imagens para aplicação de OCR em cupons fiscais. / José Victor Feijó de Araujo Araujo ; orientador, Elder Rizzon Santos, coorientador, Alexandre Gonçalves Silva, 2017.
90 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Ciências da Computação, Florianópolis, 2017.

Inclui referências.

1. Ciências da Computação. 2. Classificação de Imagens. 3. Técnicas de PDI. 4. OCR. 5. Cupons Fiscais. I. Rizzon Santos, Elder. II. Gonçalves Silva, Alexandre. III. Universidade Federal de Santa Catarina. Graduação em Ciências da Computação. IV. Título.

José Victor Feijó de Araujo

ANÁLISE E CLASSIFICAÇÃO DE IMAGENS PARA APLICAÇÃO DE *OCR* EM CUPONS FISCAIS

Este Trabalho Conclusão de Curso foi julgado adequado para obtenção do Título de "Bacharel em Ciências da Computação" e aprovado em sua forma final pelo Departamento de Informática e Estatística.

Florianópolis, 16 de novembro de 2017.

_____

Prof. Rafael Luiz Cancian, Dr.

Coordenador do Curso

Banca Examinadora:

_____

Prof. Elder Rizzon Santos, Dr.

Orientador

Universidade Federal de Santa Catarina - UFSC

_____

Prof. Alexandre Gonçalves Silva, Dr.

Coorientador

Universidade Federal de Santa Catarina – UFSC

_____

Prof.ª Jerusa Marchi, Dr.ª

Universidade Federal de Santa Catarina - UFSC

RESUMO

A proposta sugerida por este trabalho foi de analisar o impacto de um modelo de classificação, seguido de técnicas de PDI e *OCR* para extração de texto em cupons fiscais, classificando-os em subgrupos. Técnicas selecionadas de PDI foram aplicadas para cada grupo com suas devidas características, por fim extraindo texto dessas imagens através de um algoritmo de *OCR*. Foi realizado um estudo sobre os algoritmos clássicos de classificação na área de aprendizado de máquinas, com foco nos algoritmos de "clusterização" e sua correlação com a classificação de imagens em um modelo de aprendizado não supervisionado. Também foi feita uma análise sobre as características das imagens de cupons fiscais e das possíveis técnicas de PDI que podem ser aplicadas. Em relação ao *OCR,* também foi realizado um estudo para verificar possíveis soluções na extração de texto e entender seu comportamento, possibilitando desta maneira implementar a arquitetura proposta. Sendo assim, foram desenvolvidos métodos para classificar as imagens em *clusters* utilizando algoritmos de "clusterização". Também foram propostas três técnicas de PDI, a primeira aplicando uma série de realces, a segunda uma binarização adaptativa e a terceira técnica utilizando a compressão de dados JPEG. Essas imagens foram enviadas para o serviço de *OCR* do *Google Vision*, onde foi possível extrair o texto das imagens em formato de blocos. Os resultados do modelo desenvolvido foram avaliados comparando a taxa de acerto do *OCR* com os valores de texto reais presentes nos cupons fiscais, onde foi possível analisar a precisão de cada técnica proposta e da arquitetura como um todo. Foram obtidos resultados positivos utilizando o modelo desenvolvido, melhorando a extração do valor total da compra em aproximadamente 6%. Além disso, os resultados da compressão JPEG melhoraram também a extração de outros dados do cupom fiscal, como por exemplo o CNPJ e a data da compra.

Palavras-chave: Classificação de Imagens, Técnicas de PDI, *OCR*, Cupons Fiscais.

ABSTRACT

The proposal suggested by this work was to analyze the impact of a classification model, followed by image processing techniques and *OCR* for text extraction of tax coupons. This model had as proposition to classify the images of coupons in subgroups and then apply image processing techniques selected for each group with its proper characteristics, finally extracting text of these images through an *OCR* algorithm. A study was conducted on classification algorithms in learning machine models, with a focus on clustering algorithms and its correlation with image classification in unsupervised learning. It was also made a study on the tax coupons images characteristics and it's possible image processing techniques that can be applied. A study on *OCR* was also carried out to check for possible solutions in text extraction and understand their behavior, allowing to implement the proposed architecture. Thus, methods have been developed to classify images into clusters using clustering algorithms. Were also proposed three techniques to be applied, each one applying different algorithms to change the images. These images were sent to the Google *OCR* Vision service, where it was possible to extract the text of images in blocks. The results of the developed model were evaluated by comparing the accuracy rate of OCR with the actual text values present in the tax coupons, where it was possible to analyze the accuracy of each technique proposed and the architecture as a whole. Positive results were obtained using the developed model, improving the extraction of the total purchase value by approximately 6%. In addition, JPEG compression results have also improved other coupon data extraction, such as the CNPJ and the purchase date and time.

Keywords: Image Classification, Image Processing Techniques, *OCR*, Tax Coupon.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

ABNT – Associação Brasileira de Normas Técnicas
CCD – Charge Coupled Device
CMY – Cyan, Magenta, Yellow (modelo para representação de cores)
CMYK – Variante do modelo CMY, onde K=black
CNPJ – Cadastro Nacional da Pessoa Jurídica
DCT – Transformada Discreta de Cossenos
HSI – Hue, Saturation, Intensity
I – Dimensão de atributos selecionados
K – Número de Clusters
N – Conjunto de dados de imagens
OCR – Optical Character Recognition
PDI – Processamento de Imagens
RGB – Rred, Green,Blue (modelo para representação de cores)
T – Função de limiarização
UFSC – Universidade Federal de Santa Catarina
YIQ – Padrao NTSC de TV em cores

SUMÁRIO

1 INTRODUÇÃO

As aplicações que utilizam visão computacional vêm se popularizando cada vez mais com os avanços tecnológicos em processamento de imagens. Além desses avanços, os dispositivos com câmeras de boa qualidade tiveram uma significativa redução de custos nos últimos anos, aumentando o acesso a essas tecnologias. De acordo com Charles Arthur, o preço médio de venda de um *smartphone* vem caindo 12,5% a cada ano, indicando que um número maior de *smartphones* estão sendo vendidos com baixo custo. Isso permitiu a popularização de aplicações que extraem dados de imagens, como por exemplo a leitura de informações de uma foto de cupom fiscal.

Este trabalho tem como proposta analisar e classificar imagens de cupons fiscais para extração de conteúdo em texto, utilizando técnicas de análise em *cluster*, processamento de imagens (PDI) e reconhecimento óptico de caracteres (*OCR*).

A análise em *cluster* tem sido identificada como uma tarefa essencial em mineração de dados (HAN; KAMBER, 2000). Por um ponto de vista de alto nível, a "clusterização" pode ser definida como a segmentação de uma população heterogênea em um número maior de subgrupos homogêneos (ALDENDERFER; BLASHFIELD, 1984). Já partindo de um ponto de vista de baixo nível, "clusterização" pode ser definida como a busca de grupos em um conjunto de dados por algum critério natural de similaridade (DUDA; HART, 1973). Portanto, nesta proposta foi utilizado um modelo de "clusterização" de imagem no qual é factível verificar e separar claramente as propriedades de cada grupo de imagens. A partir desses grupos há a possibilidade de analisar as características semelhantes e usar algoritmos de PDI específicos para cada grupo classificado.

De acordo com Pedrini e Schwartz (2007), visão computacional é a área de pesquisa que procura auxiliar a resolução de problemas altamente complexos, buscando simular a interpretação humana e a habilidade do ser humano em tomar decisões de acordo com as informações contidas na imagem. A visão computacional pode ser estudada em dois níveis de abstração: processamento de imagens (baixo nível) e análise de imagens (alto nível). A área de processamento de imagens viabiliza um grande número de aplicações no escopo de aprimoramento de informações pictóricas para interpretação humana (FILHO; NETO, 1999). Alguns exemplos de técnicas que podem ser aplicadas no processamento e imagem são filtragem, realce, operações de borda, transformações, orientação de superfície, fluxo óptico e pirâmide. Utilizando um algoritmo de "clusterização" para agrupar as imagens possibilita que

o algoritmo de PDI seja mais eficiente em seu direcionado *cluster*, pois as imagens podem ter diferentes resoluções, tamanho, luminosidade entre outras propriedades.

A extração de textos através de imagens de cupons fiscais foi realizada neste trabalho utilizando um procedimento de reconhecimento de padrões chamado de Reconhecimento Óptico de Caracteres (*Optical Character Recognition, OCR*). O *OCR* pode ser definido como uma técnica que lida com o problema de reconhecimento de caracteres oticamente processados, ou seja, obtidos por um escaneamento, escrita ou até uma fotografia (EIKVIL, 1993). A análise qualitativa, que consiste em tratar com precisão e legibilidade dos resultados da extração de dados foi feita com base na taxa de acerto dos campos definidos que o *OCR* conseguiu identificar comparando com os dados reais no cupom fiscal.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Analisar o impacto de técnicas de "clusterização", processamento de imagens e *OCR* na extração de textos de imagens de cupons fiscais.

1.1.2 Objetivos Específicos

- Analisar o estado da arte em algoritmos de "clusterização", processamento de imagens e *OCR*.
- Propor uma arquitetura para transformar imagens de cupons fiscais em textos através de "clusterização", processamento de imagens e *OCR*.
- Especificar os requisitos e as técnicas para análise em *cluster* de imagens de cupons fiscais.
- Especificar o conjunto de técnicas de processamento de imagens que sejam adequadas a imagens de cupons fiscais.
- Analisar qual algoritmo de *OCR* pode ser utilizado com eficiência em imagens de cupons fiscais.
- Analisar os resultados obtidos das possíveis técnicas utilizadas na arquitetura proposta.

1.1.3 Métodologia

- Estudar a bibliografia clássica e artigos relacionados, escrevendo os principais tópicos relacionados a “clusterização”, processamento de imagens e *OCR*.
- Implementar um modelo que possibilite o uso de técnicas de “clusterização”, processamento de imagens e *OCR* em um conjunto de imagens de cupons fiscais.
- Analisar os possíveis atributos semelhantes no conjunto de imagens, utilizando esses atributos na implementação dos algoritmos de “clusterização”.
- Analisar a partir de artigos e referências, quais técnicas de processamento de imagens já foram utilizadas em imagens de cupons fiscais e obtiveram resultados positivos.
- Realizar uma análise do serviço de *OCR Google Vision*, implementando heurísticas para extrair os valores do texto retornado pelo serviço.
- Avaliar os resultados das técnicas implementadas na arquitetura proposta, analisando os resultados por *clusters* em quadros comparativos e gráficos.

1.2 ORGANIZAÇÃO

O escopo do trabalho foi definido de acordo com especificação e regras definidas pela Biblioteca Universitária. O trabalho é composto por uma introdução, que explica detalhadamente a motivação por trás do tema escolhido e os objetivos gerais e específicos da pesquisa realizada. Junto aos objetivos específicos é apresentado o método utilizado para alcançar o mesmo. Além disso, são apresentados capítulos que abordam conceitos básicos relacionados à área da aplicação e que, portanto, devem ser explicados a qualquer usuário (leitor) do trabalho que será desenvolvido. O capítulo de desenvolvimento exemplifica isso por meio de uma proposta de arquitetura que pode ser utilizada na extração de texto de cupom fiscal. Esta proposta será um programa capaz de extrair dados de uma imagem de cupom fiscal utilizando algoritmos de classificação ,técnicas de processamento de imagens e *OCR*.

Após o desenvolvimento dos capítulos mencionados no parágrafo anterior, são apresentados todos os resultados obtidos com as técnicas desenvolvidas. E por fim, as considerações finais do que foi aprendido durante a pesquisa e durante os meus quatro anos de graduação, assim como uma proposta para trabalhos futuros.

1.3 JUSTIFICATIVA

O desenvolvimento acelerado nas áreas de inteligência artificial e processamento de imagens possibilitou a criação de novas aplicações no mercado, como por exemplo

identificação de objetos, reconhecimento facial, identificação de texto em documentos entre outras aplicações. Entretanto, mesmo com os avanços dessas áreas ainda é uma tarefa difícil o processamento automático de certos tipos de documentos. Dependendo da qualidade da imagem, os textos identificados podem não ser corretos, e essa margem de erro pode ser um fator decisivo para utilização dessas tecnologias na identificação de texto automática em documentos.

Um documento que contém informações em texto que podem ser utilizadas em diversas aplicações é o cupom fiscal. O cupom fiscal representa uma interação de compra entre um cliente e uma loja em um determinado tempo. Diversas aplicações podem ser desenvolvidas utilizando os dados de um cupom fiscal, como por exemplo controle financeiro, programas de fidelidade, detecção de fraudes, sistemas de promoções, entre outras aplicações.

Tendo em vista que alguns algoritmos como *Tesseract* e *Google Vision* foram desenvolvidos para um uso de propósito geral na extração do texto da imagem, os resultados podem não ser tão satisfatórios para uma análise completa dos dados. De uma maneira geral, esses algoritmos de *OCR* podem ser descritos como uma caixa preta, ou seja, podem receber qualquer tipo de imagem que contenha um texto, por exemplo, placa de carro, página de livro, entre outras. Porém, aplicando técnicas de "clusterização" e processamento de imagens é possível melhorar sua disposição antes de ser enviada ao *OCR*, assim possivelmente melhorando a extração dos dados para ter resultados mais precisos.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 ANÁLISE EM *CLUSTER*

De acordo com Han e Kamber (2000), a análise em *cluster* é o processo que consiste em agrupar um conjunto de objetos físicos ou abstratos em classes similares de objetos. Já o *cluster* é uma coleção de objetos que são similares nessa mesma coleção e não similares a objetos em diferentes *clusters*. Podemos definir que objetos similares contêm características semelhantes entre eles, ou seja, idênticas ou parecidas. Em aprendizado de máquina, "clusterização" também pode ser referenciada como aprendizado não supervisionado. Diferente de classificação, a análise em *cluster* não utiliza classes pré-definidas e exemplos de dados para treino. Portanto, é uma forma de aprendizado por observação das características semelhantes entre os objetos (HAN; KAMBER, 2000).

O bom funcionamento da análise em *cluster* está relacionado com os requisitos da aplicação a ser desenvolvida. Entretanto, Han e Kamber (2000) definem alguns requerimentos típicos para o bom funcionamento da análise em cluster:

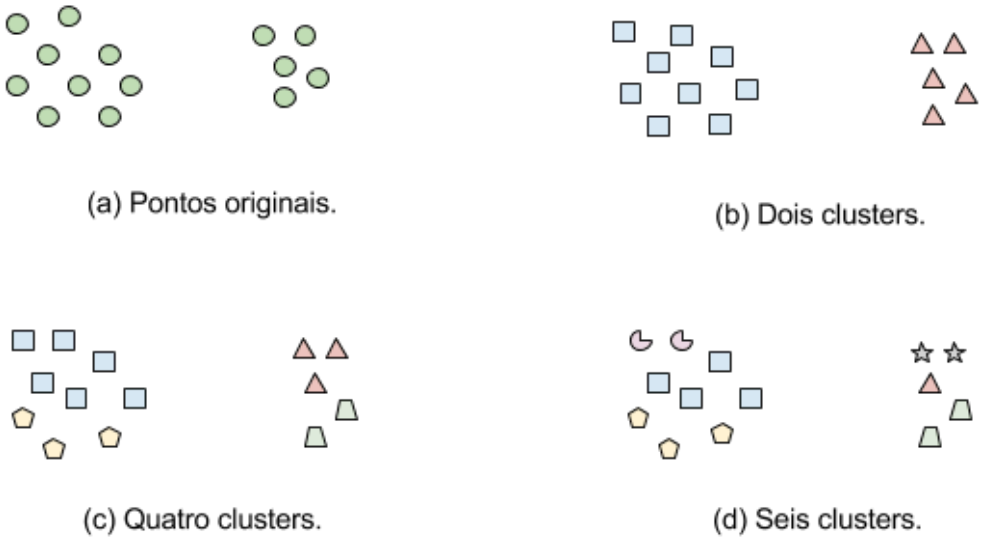
1. **Escalabilidade:** o algoritmo de "clusterização" deve funcionar não apenas para pequenas quantidades de dados, e sim ser executado sobre um volume maior de dados para ter um resultado mais preciso.
2. **Lidar com diferentes atributos:** algumas características podem variar em um conjunto de dados, podendo ser um valor numérico, binário ou até uma mistura de ambos. O algoritmo de "clusterização" deve operar sobre esses diferentes atributos na definição dos *clusters*.
3. **Descobrir *clusters* de tamanho arbitrário:** os *clusters* podem ser formados por diferentes formas geométricas, resultando em um algoritmo que saiba lidar com diferentes formas na classificação.
4. **Habilidade de lidar com ruídos:** grande parte dos bancos de dados contêm alguma característica incorreta, incompleta ou não conhecida. Não saber tratar esses casos pode resultar em *clusters* de baixa qualidade.
5. **Alta dimensionalidade:** um banco de dados pode conter algumas dimensões ou atributos. Na observação dos *clusters* gerados, é fácil para um humano julgar a qualidade do resultado em até 3 dimensões, sendo um desafio para os algoritmos de "clusterização" operar sobre mais atributos ou dimensões.

6. **Interpretabilidade e usabilidade:** é importante que seja possível interpretar os resultados de cada *cluster* para seu devido uso.

2.1.2 Tipos de "Clusterização"

Os métodos de "clusterização" podem ser classificados em diferentes tipos: hierárquico (aninhado) *versus* particionado (não aninhado), exclusivo *versus* sobreposto *versus Fuzzy*, e completo *versus* parcial.

Figura 1 - Diferentes maneiras de "clusterizar" o mesmo conjunto de pontos.



Fonte: TAN; STEINBACH; KUMAR (2005).

Hierárquica *versus* Particionada: a distinção mais comum entre os diferentes tipos de "clusterização" é quando o conjunto de *clusters* é aninhado ou não aninhado, em outras palavras, hierárquico ou particionado. Tan, Steinbach e Kumar (2005) definem que o modelo particionado é a simples divisão do conjunto de dados em subconjuntos (*clusters*) em que cada objeto está em exatamente um subconjunto. Um exemplo seria cada coleção de *clusters* na Figura 1 (b-d) na qual são "clusterizações" particionadas. Caso seja permitido que os *clusters* tenham sub *clusters*, então obtemos o modelo de "clusterização" hierárquico, o qual consiste em um conjunto de *clusters* alinhados que são organizados como uma árvore. Cada nó (*cluster*) na árvore (exceto os nós folha) é a união de seus nós filho (sub *clusters*), e a raiz da árvore é o *cluster* contendo todos os objetos. Portanto, uma "clusterização" hierárquica pode ser vista como uma sequência de "clusterização" particionadas e a "clusterização"

particionada pode ser obtida pegando qualquer membro dessa sequência (TAN; STEINBACH; KUMAR, 2005).

Exclusivo *versus* Sobreposto versus *Fuzzy*: as análises em *clusters* mostradas na Figura 1 são todas exclusivas, pois assimilam cada objeto a um único *cluster*. Em determinadas situações, um objeto pode pertencer simultaneamente a mais de um grupo (*cluster*), no qual é utilizado a "clusterização" sobreposta ou não exclusiva para representar esse modelo. Segundo Tan, Steinbach e Kumar (2005), na "clusterização" *Fuzzy* cada objeto pertence a todos os *clusters* com um peso que tem seu valor entre 0 (absolutamente não pertence) e 1 (absolutamente pertence). Em outras palavras, os *clusters* são tratados como conjuntos *Fuzzy*. Matematicamente, um conjunto *Fuzzy* é o conjunto em que existe um grau de pertinência de cada elemento a um determinado conjunto (CELINA ABAR, 2004). Similarmente, técnicas de "clusterização" probabilísticas computam a probabilidade de cada objeto pertencer a um *cluster*, e essas probabilidades também devem ter a soma igual a 1. Na prática, a "clusterização" probabilística ou *Fuzzy* acaba sendo convertida a um modelo exclusivo por atribuir a cada objeto o *cluster* com o maior peso (TAN; STEINBACH; KUMAR, 2005).

Completa *versus* Parcial: de uma maneira simplificada, a "clusterização" completa atribui a todos os objetos um *cluster*, ao contrário da parcial que não faz o mesmo. A motivação para uma "clusterização" parcial é que alguns objetos em um conjunto de dados podem não pertencer a grupos bem definidos (TAN; STEINBACH; KUMAR, 2005). Isso pode acontecer pois muitas vezes os objetos podem conter características incompletas, ruídos ou uma característica não reconhecida pelo algoritmo. Por exemplo, uma aplicação que utiliza análise em *cluster* para organizar e classificar artigos precisa garantir que todos os artigos podem ser buscados, portanto é necessário que a "clusterização" seja completa.

2.1.3 Algoritmos de "Clusterização"

Existem diversos algoritmos de "clusterização", sendo que cada um pode performar de diferentes maneiras dependendo do conjunto de dados a ser "clusterizado". Os três principais algoritmos da literatura são conhecidos como:

- ***K-means*:** é um algoritmo de "clusterização" particionado que tem como objetivo buscar o número de *clusters (K)* especificado pelo usuário, no qual os *clusters* são representados por centróides.

- **Agrupamento Hierárquico Aglomerativo:** essa abordagem se refere a uma coleção de técnicas relacionadas de "clusterização" que produzem uma "clusterização" hierárquica. Cada ponto é interpretado como um *cluster* único e, então, são unidos repetidamente os dois *clusters* mais próximos até restar apenas um único *cluster*.
- **DBSCAN:** é um algoritmo de "clusterização" baseado em densidade que produz uma "clusterização" particionada, em que o número *clusters* é automaticamente determinado pelo próprio algoritmo.

2.1.4 *K-means*

MacKay e David (2003) definem que o algoritmo *K-means* tem a funcionalidade de colocar N pontos de dados de um espaço dimensional I em K *clusters*. Cada *cluster* é parametrizado por um vetor $m^{(k)}$. Os pontos de dados são denotados por $\left\{x^{(n)}\right\}$ no qual n irá rodar no algoritmo de 1 até o número de pontos de dados N. Cada vetor x tem I componentes x_i. Assumindo que x está em um espaço real, então a métrica para definir a distância entre dois pontos é dada por,

$$d(x,y) = \frac{1}{2}\sum_i (x_i - y_i)^2 \qquad (1)$$

Para iniciar o algoritmo (Algoritmo 1), os valores $m^{(k)}$ precisam ser inicializados de alguma maneira, como por exemplo com valores aleatórios. Ele funciona basicamente em duas etapas, atribuição e atualização. Na etapa de atribuição cada ponto de dado *n* é atribuído ao valor de $m^{(k)}$ mais próximo. Já na etapa de atualização, os valores de $m^{(k)}$ são ajustados para condizer com os valores de cada ponto de dados que eles são responsáveis. Na Figura 2 é possível ver um exemplo das iterações do algoritmo com um conjunto de dados em duas dimensões de 40 pontos e K=2. A atribuição dos valores aos *clusters* é mostrada pelos dois tipos de pontos e os valores de $m^{(k)}$ (centróides) pelos círculos.

Algoritmo 1 – "clusterização" *K-means*.

Inicialização. Define os *K-means* $m^{(k)}$ para valores aleatórios.

Atribuição. Cada ponto é atribuído ao valor mais próximo de $m^{(k)}$. Denotamos que o chute para o *cluster* k^n na qual o ponto x^n pertence a $k'^{(n)}$.

$$k'^{(n)} = \arg\min_k \left\{ d\left(m^{(k)},\ x^{(n)}\right) \right\}.$$

Nessa mesma etapa, é atribuído um valor $r_n^{(k)}$ que representa a responsabilidade sobre o ponto. Caso k seja o parâmetro mais próximo do ponto $x^{(n)}$, é atribuído 1 ao valor de responsabilidade, caso contrário $r_n^{(k)}$ será 0.

$$r_k^{(n)} = \begin{cases} 1 & k'^{(n)} = k \\ 0 & k'^{(n)} \neq k \end{cases}$$

Não é esperado que aconteça um empate, mas se houver o valor $k'^{(n)}$ é atribuído para o menor valor do veqncedor $\{k\}$.

Atualização. Os parâmetros dos *clusters* são ajustados para satisfazer o valor dos pontos de dados que eles são responsáveis.

$$m^{(k)} = \frac{\sum_n r_k^{(n)} x^{(n)}}{R^{(k)}}$$

onde $R^{(k)}$ é a total responsabilidade do parâmetro k,

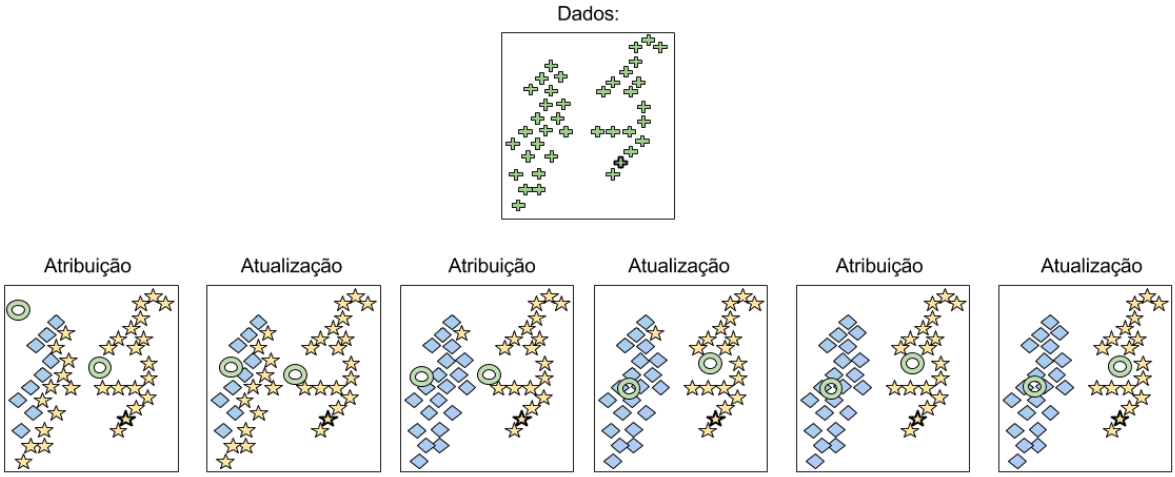
$$R^{(k)} = \sum_n r_k^{(n)}$$

Repete a etapa de Atribuição e Atualização até que as atribuições não alterem.

Fonte: MACKAY; DAVID, 2003.

Na primeira etapa de atribuição é possível visualizar que como o algoritmo foi inicializado com valores de $m^{(k)}$ aleatórios, os *clusters* não estão corretamente definidos. Em seguida, com a atualização já é possível ver o ajuste das centróides em conformidade aos pontos de dados, os quais começam a convergir ao ponto central de cada *cluster*. Quando o algoritmo chega ao seu fim, pois a etapa de atualização não altera os valores de $m^{(k)}$, as centróides identificaram os pontos centrais de cada *cluster*. O algoritmo chega a um estado no qual os pontos não estão passando de um *cluster* para o outro, ou seja, as centróides não alteram.

Figura 2 - Algoritmo *K-means* aplicado a um conjunto de 40 dados e K=2.



Fonte: MACKAY; DAVID (2003).

Segundo Tan, Steinbach e Kumar (2005), na maior parte das execuções o algoritmo irá convergir nas etapas iniciais, contudo, a última etapa do Algoritmo 1 também pode ter uma verificação fraca na qual irá repetir o algoritmo até que apenas 1% dos pontos alterarem os *clusters*.

K-means é simples e pode ser usado para uma grande variedade de tipos de dados. Ele também é bastante eficiente, mesmo que múltiplas execuções sejam executadas frequentemente (TAN; STEINBACH; KUMAR, 2005). Porém, o algoritmo *K-means* não é adequado para todos os tipos de dados, tendo dificuldade de ligar com *clusters* globulares e de tamanhos e densidades diferentes. De acordo com MacKay e David (2003), uma crítica ao algoritmo seria que ele é *'hard'* ao invés de *'soft'*, pois os pontos são atribuídos a exatamente um *cluster* e todos os outros pontos atribuídos a um *cluster* são iguais neste mesmo *cluster*. Os pontos localizados nas bordas entre dois *clusters*, deveriam ter uma classificação parcial entre todos os possíveis *clusters* que ele poderia ser atribuído, porém isso não acontece no algoritmo *K-means*. Esses problemas motivaram a criação de diversas variações do algoritmo. Uma delas é chamada de *Soft K-means*, na qual utiliza um parâmetro β denominado rigidez. O *Soft K-means* é parecido com o algoritmo original, sendo a maior diferença a atribuição de um valor no intervalo de 0 a 1 no cálculo das responsabilidades (MACKAY; DAVID, 2003).

2.1.5 Agrupamento Hierárquico Aglomerativo

Técnicas de "clusterização" hierárquica são uma segunda categoria importante de métodos de "clusterização". Segundo Hastie, Tibshirani e Friedman (2009) a abordagem aglomerativa, também chamada de *"Bottom-Up"*, inicia os pontos como *clusters* individuais e recursivamente reúne o par de *clusters* selecionado em um único *cluster*. O par selecionado

para união consiste nos dois *clusters* com a melhor proximidade entre *clusters*. Esse método requer a definição de proximidade em *cluster*. Segundo Tan, Steinbach e Kumar (2005), o algoritmo pode ser expresso mais detalhadamente em Algoritmo 2.

Algoritmo 2 – Agrupamento Hierárquico Aglomerativo.

Inicialização. Calcula a matriz de proximidade se necessário.

repita.

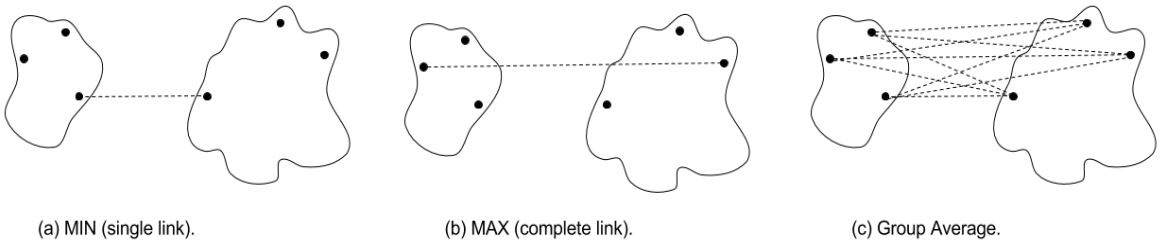
Une os dois *clusters* mais próximos.
Atualiza a matriz de proximidade para refletir na proximidade entre o novo cluster e os originais.

até que. Apenas um sobrar um *cluster*.

Fonte: TAN; STEINBACH; KUMAR, 2005.

A operação chave do Algoritmo 2 é a definição da proximidade entre *clusters*, na qual essa definição é o que diferencia as técnicas de agrupamento hierárquico aglomerativo. De acordo com Tan, Steinbach e Kumar (2005), diversas dessas técnicas, como por exemplo, MIN, MAX e *Group Average*, vem de um ponto de vista baseado em grafo dos *clusters*. A abordagem MIN define que a proximidade entre os *clusters* vem da menor distância entre dois pontos de *clusters* distintos, em um ponto de vista de grafos, a menor aresta entre dois nodos de diferentes subconjuntos de nodos. Alternativamente, MAX leva em consideração a proximidade como sendo a maior distância entre dois pontos de *clusters* distintos, que em termos de grafos, a maior aresta entre dois nodos de diferentes subconjuntos de nodos. Também utilizando uma abordagem baseada em grafos, a técnica *Group Average* utiliza a média de todas as distâncias entre todos os pontos entre *clusters* distintos como definição de proximidade. Outras técnicas também podem ser utilizadas na definição de proximidade do algoritmo, como por exemplo a análise por centróides na qual define a proximidade como sendo a distância entre as centróides e o método de *Ward's* que também utiliza centróides, mas usa um critério de variação mínima no cálculo das proximidades (TAN; STEINBACH; KUMAR, 2005). Na Figura 3 é possível ver uma ilustração das três técnicas descritas.

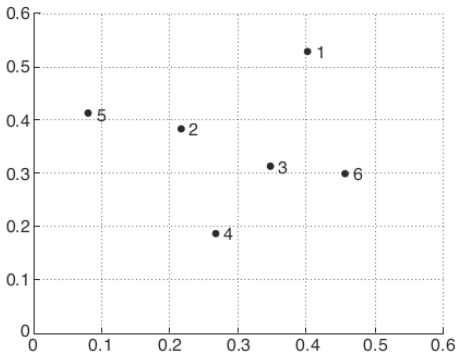
Figura 3 - Definições de proximidade em clusters baseado em grafos.



Fonte: TAN; STEINBACH; KUMAR (2005).

Para ilustrar o comportamento das variações do algoritmo, Tan, Steinbach e Kumar (2005) utilizam um conjunto de dados de exemplo que consiste em 6 pontos de duas dimensões, apresentados na Figura 4. As coordenadas x e y dos pontos e a distância euclidiana entre eles podem ser visualizados nos Quadros 1 e 2, respectivamente.

Figura 4 - Conjunto de 6 pontos



Fonte: TAN; STEINBACH; KUMAR (2005)

Quadro 1 – Coordenadas x e y

Point	x Coordinate	y Coordinate
p1	0.40	0.53
p2	0.22	0.38
p3	0.35	0.32
p4	0.26	0.19
p5	0.08	0.41
p6	0.45	0.30

Fonte: TAN; STEINBACH; KUMAR (2005)

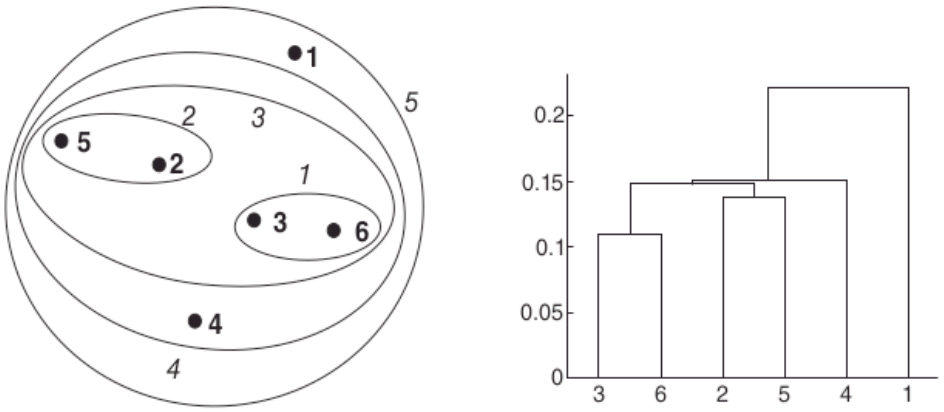
Quadro 2 - Matriz de distância euclidiana dos 6 pontos.

	p1	p2	p3	p4	p5	p6
p1	0.00	0.24	0.22	0.37	0.34	0.23
p2	0.24	0.00	0.15	0.20	0.14	0.25
p3	0.22	0.15	0.00	0.15	0.28	0.11
p4	0.37	0.20	0.15	0.00	0.29	0.22
p5	0.34	0.14	0.28	0.29	0.00	0.39
p6	0.23	0.25	0.11	0.22	0.39	0.00

Fonte: TAN; STEINBACH; KUMAR (2005).

MIN ou *Single Link*: essa técnica costuma ter bons resultados ao lidar com formatos de *clusters* não elípticos, porém é sensível a ruídos no conjunto de dados. A Figura 5 mostra o resultado da aplicação da técnica MIN ao conjunto de dados de 6 pontos. Na Figura 5(a) é possível observar os *clusters* aninhados com uma sequência de elipses, onde cada número associado a elipse indica a ordem da "clusterização". A Figura 5(b) mostra a mesma informação, porém em formato de dendograma. A altura na qual os dois *clusters* são unidos no dendograma reflete a distância calculada pela técnica MIN (TAN; STEINBACH; KUMAR, 2005).

Figura 5 - "Clusterização" dos pontos da Figura 4 utilizando a técnica MIN (*Single Link*)

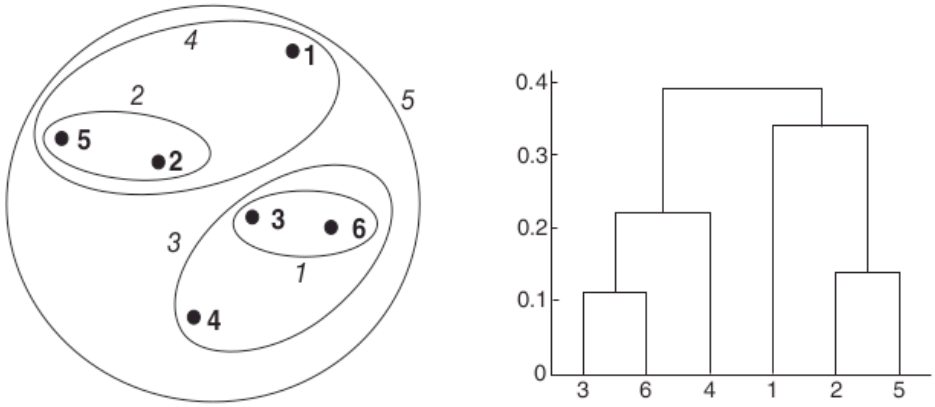


(a) Single link clustering. (b) Single link dendrogram.

Fonte: TAN; STEINBACH; KUMAR (2005).

MAX ou *Complete Link*: apesar de ser menos suscetível a ruídos no conjunto de dados, essa técnica pode não funcionar muito bem em *clusters* largos favorecendo formatos globulares. A Figura 6 mostra o resultado da aplicação do MAX para o conjunto de dados de 6 pontos. Na técnica MIN o *cluster* $\{3,\ 6\}$ é unido com *cluster* $\{2,\ 5\}$, porém com a técnica MAX, essa união é feita entre o *cluster* $\{3,\ 6\}$ e o *cluster* $\{4\}$ por utilizar a máxima distância entre os pontos (TAN; STEINBACH; KUMAR, 2005).

Figura 6 - “Clusterização” dos pontos da Figura 4 utilizando a técnica MAX (*Complete link*)

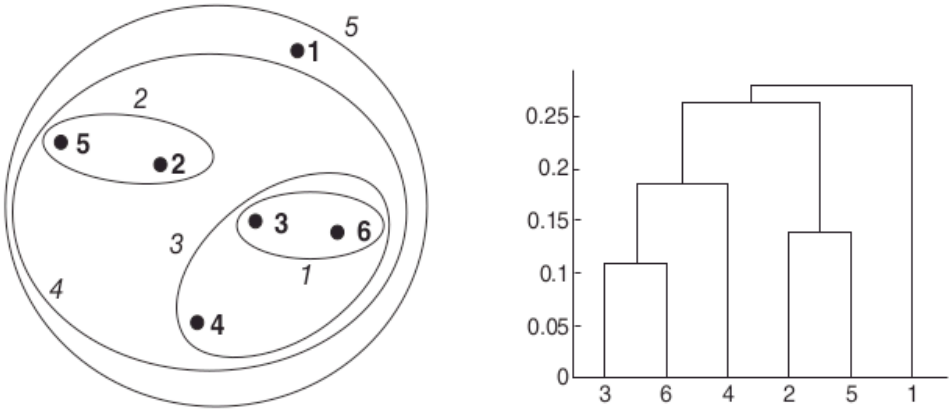


(a) Complete link clustering. (b) Complete link dendrogram.

Fonte: TAN; STEINBACH; KUMAR (2005).

***Group Average*:** Para a essa versão do algoritmo, a proximidade entre dois *clusters* é definida como a média da proximidade de cada par de pontos entre todos os pontos de diferentes *clusters*. Essa é uma abordagem intermediária entre a MIN (*Single Link*) e a MAX (*Complete Link*). A Figura 7 mostra os resultados da aplicação da técnica *Group Average* no conjunto de dados de 6 pontos (TAN; STEINBACH; KUMAR, 2005).

Figura 7 – “Clusterização” dos pontos da Figura 4 utilizando a técnica *Groupo Average*.



(a) Group average clustering. (b) Group average dendrogram.

Fonte: TAN; STEINBACH; KUMAR (2005).

De acordo com Tan, Steinbach, Kumar (2005) de um modo geral, esse tipo de algoritmo e suas determinadas técnicas são utilizados em casos específicos onde requer uma hierarquia entre os *clusters* criados, como por exemplo uma taxonomia. Contudo, as aplicações do algoritmo hierárquico aglomerativo são custosas em termos de seus requisitos computacionais e de armazenamento.

2.1.6 DBSCAN

Segundo Ester, Kriegel, Sander e Xu (1996), o DBSCAN é um algoritmo que depende da noção de densidade em *clusters*, no qual foi criado para lidar com *clusters* de tamanhos arbitrários. O algoritmo transforma regiões de densidade suficientemente alta em *clusters*, também descobrindo *clusters* de tamanhos arbitrários mesmo com ruído (HAN; KAMBER, 2000). Na abordagem tradicional baseada em centro, a densidade é avaliada para um determinado ponto no conjunto de dados contando-se o número de pontos dentro de um raio específico, como por exemplo, o do próprio ponto (TAN; STEINBACH; KUMAR, 2005). Essa abordagem baseada em centro permite classificar um ponto como estando no interior de uma região densa (Ponto Central), no limite de uma região densa (Ponto Limite) ou em uma região ocupada esparsamente (Ruído ou Ponto de Segundo Plano).

Tan, Steinbach, Kumar (2005) descrevem o algoritmo informalmente como: quaisquer pontos centrais que são próximos o suficiente (com uma distância *Eps* entre eles) são colocados no mesmo *cluster*. Da mesma forma, qualquer ponto limite que é próximo o suficiente de um ponto central é colocado no mesmo *cluster* do ponto central. Os detalhes e passos executados podem ser vistos em Algoritmo 3. Na execução do algoritmo o usuário precisa selecionar dois parâmetros, *Eps* e *MinPts*. A abordagem básica é olhar para o comportamento da distância de um ponto ao seu k^{th} vizinho mais próximo, sendo chamado de k-dist (TAN; STEINBACH; KUMAR, 2005). Se selecionarmos a distância como o parâmetro *Eps* e tomar o valor de k como parâmetro *MinPts*, então os pontos na qual a k-dist é menor que *Eps* serão nomeados pontos centrais, enquanto outros pontos serão classificados como ruídos ou pontos de segundo plano. De acordo com Ester, Kriegel, Sander e Xu (1996) o parâmetro *MinPts* pode ser utilizado com o valor fixado em 4 para um conjunto de dados de duas dimensões.

Algoritmo 3 – “Clusterização” DBSCAN.

Inicialização. Rotula todos os pontos como sendo pontos centrais.

Eliminação. Remove todos os pontos considerados como ruídos.

Definição das distâncias. Coloca uma aresta entre todos os pontos centrais que estão com uma distância *Eps* entre si.

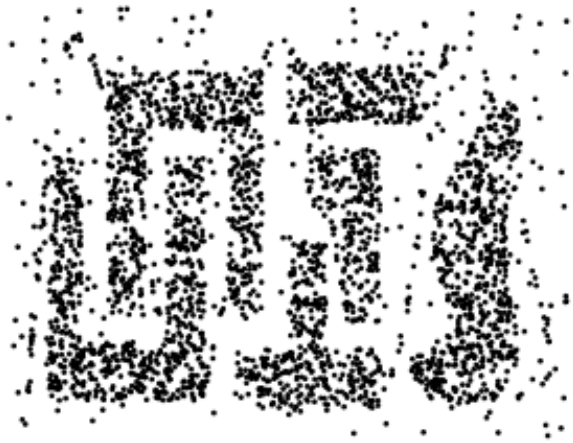
Separação. Transforma cada grupo de pontos centrais conectados em *clusters* separados.

Atribuição. Cada ponto limite é atribuído para um cluster de seus pontos centrais assimilados.

Fonte: TAN; STEINBACH; KUMAR, 2005.

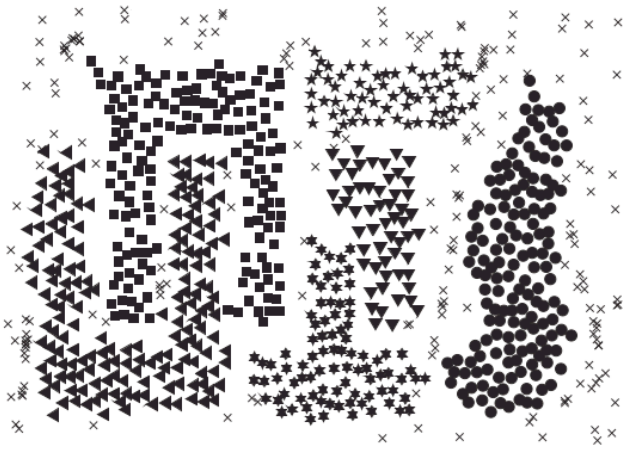
Para ilustrar o funcionamento do DBSCAN, utilizaremos o conjunto de dados de exemplo usado por Tan, Steinbach e Kumar (2005). A Figura 8 mostra um conjunto de dados relativamente complicado, o qual consiste em 3000 pontos de duas dimensões. Na escolha do parâmetro *Eps* é utilizado uma técnica a qual consiste em ordenar as distâncias dos quatro vizinhos mais próximos de cada ponto, identificando qual valor provoca um aumento acentuado. Portanto, é selecionado *Eps*=10 que corresponde a esse aumento acentuado (TAN; STEINBACH; KUMAR, 2005). Os *clusters* encontrados pelo algoritmo DBSCAN usando os parâmetros *Eps*=10, *MinPts*=4, podem ser visualizados na Figura 9. Os pontos centrais, pontos limite e pontos de segundo plano (ruídos) estão exibidos na Figura 10.

Figura 8 - Conjunto de dados de 3000 pontos de duas dimensões



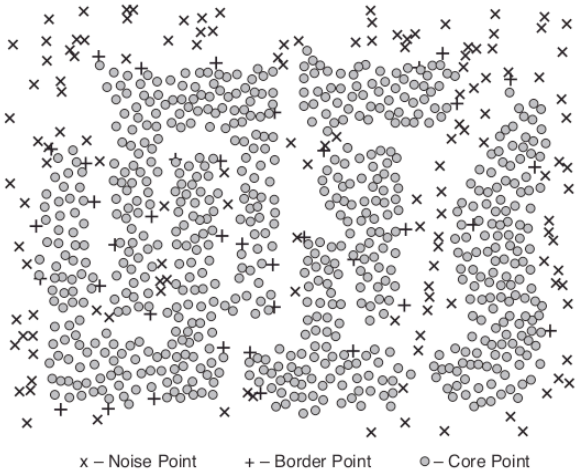
Fonte: TAN; STEINBACH; KUMAR (2005).

Figura 9 - *Clusters* encontrados pelo algoritmo DBSCAN.



Fonte: TAN; STEINBACH; KUMAR (2005).

Figura 10 - Pontos de segundo plano (ruído), limite e centrais



Fonte: TAN; STEINBACH; KUMAR (2005).

Pelo fato de o DBSCAN utilizar a definição de densidade, o torna relativamente resistente a ruídos, podendo lidar com *clusters* de tamanhos e formatos arbitrários. Por exemplo, ele pode encontrar *clusters* que um algoritmo como o *K-means* não conseguiria encontrar. Entretanto, o DBSCAN tem dificuldade de lidar com *clusters* com uma grande variedade de densidades. Em conclusão, o algoritmo pode ser custoso em requisitos computacionais ao calcular todas as distâncias entre os pontos vizinhos (dados com muitas dimensões), porém para conjuntos de dados de duas dimensões ele provê bons resultados.

2.1.7 *Spectral Clustering*

Segundo Hastie, Tibshirani e Friedman (2009), o algoritmo *Spectral Clustering* é uma generalização dos métodos de agrupamento padrões, projetado para funcionar em situações onde os *clusters* não têm apenas um formato circular ou elíptico. A ideia principal é construir uma matriz que representa as relações locais de vizinhança sobre as observações. Existem diversas maneiras de definir a matriz de similaridade e seu grafo que representa o comportamento local, sendo o *K-nearest-neighbor* o mais popular. Também é necessário calcular a matriz *Laplaciana*, podendo ser não-normalizada, normalizada, entre outras (VON LUXBURG, 2007). No Algoritmo 6 é possível ver cada etapa do *Spectral Clustering* proposto por Ng, Jordan e Weiss (2001).

Algoritmo 6 – *Spectral Clustering* com a matriz *Laplaciana* normalizada.

Entrada: Matriz de similaridade S, número K de *clusters* a serem criados.

• Construir a matriz de similaridade, utilizando por exemplo o *K-nearest-neighbor*.

• Calcular a matriz *Laplaciana* L normalizada.

• Calcular os primeiros k *eigenvectors* $u1, \ \dots \ , \ uk$ de L.

• Seja U a matriz contendo os vetores $u1, \ \dots \ , \ uk$ como colunas.

• Criar a matriz T a partir de U, normalizando as linhas tal que $t_{ij} = \frac{u_{ij}}{\left(\sum_k u_{ik}^2\right)^{\frac{1}{2}}}$.

• Seja $i = 1, \ \dots \ , \ n$ o vetor correspondente a i-ésima linha de T.

• Clusterizar os pontos $(y_i)_{i=1,\dots,n}$ com o K-means em clusters $C_1, \ \dots \ , C_k$

Saída: Clusters $A_1, \ \dots \ , A_k$.

Fonte: Ng, Jordan e Weiss (2001).

2.2 PROCESSAMENTO DIGITAL DE IMAGENS

Segundo Huang (1996) visão computacional tem um objetivo duplo. Partindo de um ponto de vista biológico, visão computacional é o estudo dos modelos computacionais do sistema visual humano. Já partindo de um ponto de vista de engenharia, visão computacional tem o objetivo de criar sistemas autônomos capazes de executar tarefas que o sistema visual humano executa, e até superar em alguns casos.

Conforme Pedrini e Schwartz (2007), a visão computacional pode ser dividida em dois níveis de abstração: processamento de imagens (baixo nível) e análise de imagens (alto nível). O processamento digital de imagens pode ser definido como um conjunto de técnicas para capturar, representar e transformar imagens utilizando um computador. A aplicação dessas técnicas permite extrair e identificar informações das imagens e melhorar a qualidade visual de aspectos estruturais, simplificando a percepção humana e a interpretação automática por meio de máquinas (PEDRINI; SCHWARTZ, 2007). Essa abordagem de baixo nível utiliza técnicas como o aumento de contraste, a redução de ruído, a extração de bordas e a compressão de imagens.

De acordo com Filho e Neto (1999), a área de PDI vem apresentando crescimento expressivo e suas aplicações permeiam quase todos os ramos que envolvem atividades humanas. O uso de imagens no diagnóstico médico tornou-se rotineiro na medicina atual. Em biologia, a capacidade de processar imagens automaticamente obtidas de microscópios facilita a execução de tarefas laboratoriais com alto grau de precisão (FILHO; NETO, 1999). Inúmeras outras áreas como Astronomia, Meteorologia, Segurança entre diversas outras estão sendo beneficiadas com os avanços nas áreas de PDI e visão computacional.

2.2.2 Fundamentos de Imagens Digitais

Para manipular uma imagem em computador é necessário a definição de um modelo matemático adequado para sua representação. Pedrini e Schwartz (2007) definem a imagem em níveis de cinza como uma função $f(x, y)$ de intensidade luminosa, cujo valor ou amplitude nas coordenadas espaciais (x, y) fornece a intensidade ou o brilho da imagem naquele ponto. A função $f(x, y)$ representa o produto da interação entre a função de iluminância $i(x, y)$ que relata a quantidade de luz que incide sobre o objeto e as

propriedades de reflectância que pode ser representada pela função $r\,(x,\ y)$, cujo valor representa a quantidade de luz refletida pelo objeto no ponto $(x,\ y)$ (FILHO; NETO, 1999). Pode ser expressa matematicamente por:

$$f\,(x,\ y)\ =\ i\,(x,\ y)\ .\ r\,(x,\ y) \tag{2}$$

para

$$0 < i(x,y) < \infty \ \text{ e } \ 0 < r(x,y) < 1 \tag{3}$$

A natureza de $r\,(x,\ y)$ é determinada pelas características dos objetos na cena, enquanto $i\,(x,\ y)$ é determinada pela fonte de luz. A iluminância é medida em lúmen/m² ou lux, enquanto a reflectância é medida em porcentagem ou no intervalo entre 0 e 1 (PEDRINI; SCHWARTZ, 2007).

A conversão de uma cena real para uma imagem digital é possível pela execução de duas etapas: aquisição e digitalização. Conforme Filho e Neto (1999) a etapa de aquisição da imagem é um processo de conversão de uma cena real tridimensional em uma imagem analógica. Atualmente existem diversos dispositivos de aquisição de imagens, porém o mais utilizado é a câmera *CCD* (*Charge Coupled Device*), em que consiste de uma matriz de células semicondutoras e fotossensíveis atuando como capacitores, armazenando carga elétrica proporciona à energia luminosa incidente (FILHO; NETO, 1999). Para transformar o sinal analógico obtido pela aquisição da imagem em uma imagem digital, duas etapas são necessárias: amostragem e quantização. Na etapa de amostragem é convertido a imagem analógica em uma matriz de *M* por *N* pontos, cada qual é denominado *pixel* (FILHO; NETO, 1999). Já a etapa de quantização faz com que cada um destes pixels assuma um valor inteiro, representado no intervalo de 0 a $2^n - 1$, sendo que quanto maior o valor de n maior o número de níveis de cinza presentes na imagem digitalizada (FILHO; NETO, 1999).

Segundo Pedrini e Schwartz (2007) uma imagem digital pode ser representada através de uma matriz bidimensional, na qual cada pixel da imagem corresponde a um elemento da matriz. Na Figura 11 é possível ver um exemplo de representação matricial de uma imagem, sendo que uma pequena região destacada é formada por números inteiros que representam o nível de cinza dos *pixels* da imagem. Existem várias vantagens ao utilizar matrizes para representação de imagens, pois são estruturas simples para armazenar, manipular e visualizar dados (PEDRINI; SCHWARTZ, 2007).

Figura 11 - Representação matricial; (a) imagem; (b) níveis de cinza correspondentes a região destacada da imagem.

(a)

120	138	120	151	139
110	129	129	139	146
150	138	137	138	129
137	129	129	128	137
146	125	131	132	145

(b)

Fonte: PEDRINI; SCHWARTZ (2007).

Filho e Neto (1999) definem em seu texto um conjunto de operações lógicas e aritméticas que podem ser aplicadas em imagens. Sejam duas imagens X e Y de igual tamanho, uma terceira imagem Z pode ser produzida através do processamento *pixel* a *pixel* de $X \; opn \; Y$, sendo que opn é um operador aritmético (+, -, x, e /) ou lógico (AND, OR, XOR). As principais aplicações das operações aritméticas sobre imagens estão descritas no Quadro 3, a segunda coluna do quadro avalia os efeitos qualitativos das operações.

Quadro 3 - Efeitos e aplicações das operações aritméticas em imagens.

Operação	Efeito sobre a imagem	Aplicações
Adição	Z é o resultado da soma dos valores de intensidade de X e Y. Se for Y for um escalar positivo, Z será uma versão mais clara de X.	Normalização de brilho de imagens; Remoção de ruídos.
Subtração	Z é o resultado da diferença dos valores de intensidade de X e Y. Se for Y for um	Detecção de diferenças entre duas imagens da mesma cena.

	escalar positivo, Z será uma versão mais escura de X.	
Multiplicação	Z é o produto dos valores de intensidade de X e Y. Se for Y for um escalar positivo, os valores de intensidade de Z serão diretamente proporcionais a X por um fator de Y.	Calibração de brilho.
Divisão	Z é a razão dos valores de intensidade de X pelos valores correspondentes em Y. Se for Y for um escalar positivo, os valores de intensidade de Z serão inversamente proporcionais a X por um fator de Y.	Normalização de brilho.

Fonte: FILHO; NETO (1999).

2.2.3 Realce de Imagens

De acordo com Pedrini e Schwartz (2007), as técnicas de realce de imagens buscam aprimorar a aparência de determinadas características da imagem, tornando-a mais adequada à destinada aplicação. Essas técnicas são normalmente utilizadas em problemas específicos, ou seja, um método que é útil para melhorar imagens de raios-x não necessariamente é o melhor método para melhorar imagens transmitidas de Marte (GONZALEZ; WOODS, 2001). Conforme Filho e Neto (1999) os métodos de filtragem de imagem podem ser classificados em duas categorias: as técnicas de filtragem espacial e as técnicas de filtragem no domínio da frequência. Os métodos de domínio espacial operam diretamente sobre a matriz de *pixels* que é a imagem digitalizada. Por outro lado, os métodos de domínio de frequência se baseiam na modificação da transformada de Fourier da imagem. Existem técnicas que utilizam métodos de ambos os domínios.

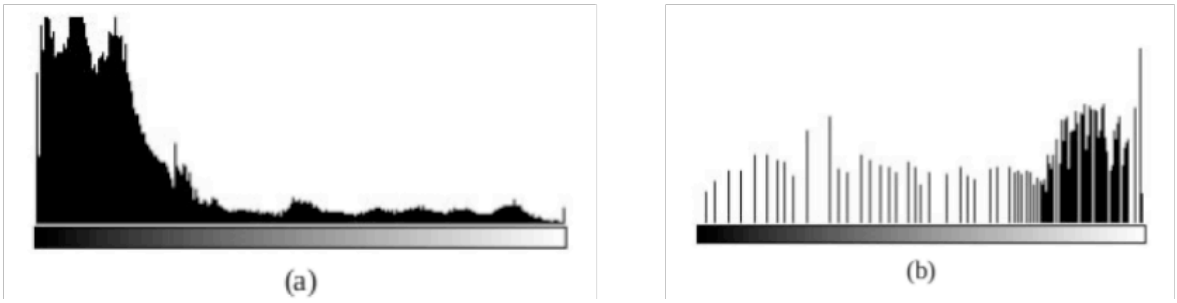
2.2.4 Brilho e Contraste

O brilho está relacionado a sensação visual da intensidade luminosa de uma fonte. A habilidade do sistema visual humano para diferenciar os níveis de brilho é um fator importante na avaliação de resultados que envolvem imagens digitais (PEDRINI; SCHWARTZ, 2007). A definição de contraste é dada como uma medida de variação relativa da luminância, isto é, da intensidade luminosa por unidade de área. De acordo com a lei de Weber, a resposta do sistema visual humano depende significativamente de variações locais de luminância (PEDRINI; SCHWARTZ, 2007). A lei de Weber estabelece uma relação chamada de contraste de Weber, na qual define que a intensidade adicional de luminância necessária para que o sistema visual humano possa reparar uma alteração é proporcional à intensidade inicial.

2.2.5 Histograma

Filho e Neto (1999) definem o histograma de uma imagem como sendo um conjunto de números indicando o percentual de *pixels* naquela imagem que apresentam um determinado nível de cinza. O histograma pode ser representado por um gráfico indicando o número de *pixels* na imagem para cada nível de cinza (PEDRINI; SCHWARTZ, 2007). Analisando o histograma de uma imagem é possível obter uma indicação de sua qualidade quanto ao nível de contraste e quanto ao seu brilho médio, ou seja, se a imagem é predominante clara ou escura (FILHO; NETO, 1999). A Figura 12 apresenta dois tipos diferentes de histogramas que são frequentemente encontrados em imagens comuns. O histograma da Figura 12(a) apresenta uma concentração de *pixels* nos valores baixos de cinza, o que corresponde a uma imagem predominantemente escura. Já o histograma da Figura 12(b) os *pixels* estão concentrados próximo ao limite superior da escala de cinza, caracterizando uma imagem clara (FILHO; NETO, 1999).

Figura 12 - Exemplo de histogramas.



Fonte: FILHO; NETO (1999).

De acordo com Pedrini e Schwartz (2007), dado uma imagem $f(x,\ y)$ representada por uma matriz bidimensional, com dimensões de $M\ x\ N$ pixels e contendo L níveis de cinza, o cálculo do histograma é definido pelo Algoritmo 4. Embora o histograma de uma imagem proporciona diversas características sobre ela (nível de cinza mínimo, médio e máximo, predominância de pixels claros ou escuros etc.), outras características qualitativas somente podem ser extraídas dispondo-se da imagem propriamente dita (FILHO; NETO, 1999).

Algoritmo 4 – Cálculo do histograma de uma imagem.

Inicialização. Atribui o valor 0 a todos os elementos do vetor,

para $i\ =\ 0$ **até** $L\max$ **faça**
$H[i]\ =\ 0$
fim

Calcular Distribuição. Níveis de cinza para cada *pixel* da imagem, tal que

para $x\ =\ 0$ **até** $M-1$ **faça**
para $y\ =\ 0$ **até** $N-1$ **faça**
$H[f(x,y)]\ =\ H[f(x,y)]+\ 1$
fim
fim

Fonte: PEDRINI; SCHWARTZ, 2007.

As técnicas de modificação de histograma são conhecidas como técnicas ponta-a-ponto, pois o valor de tom de cinza de um determinado *pixel* após o processamento depende apenas de seu valor original. Em contrapartida, as técnicas de processamento orientadas a vizinhança, o valor resultante depende dos pixels que circundam o elemento da imagem

original (FILHO; NETO, 1999). Segundo Pedrini e Schwartz (2007), o intervalo de contraste é a diferença entre os valores de intensidade máximo e mínimo que f (x, y) pode assumir. Seja um valor L, representando o nível de cinza dos *pixels* na imagem em um intervalo de [Lmin, Lmax], a transformação de intensidade é uma função na qual pode ser descrita como:

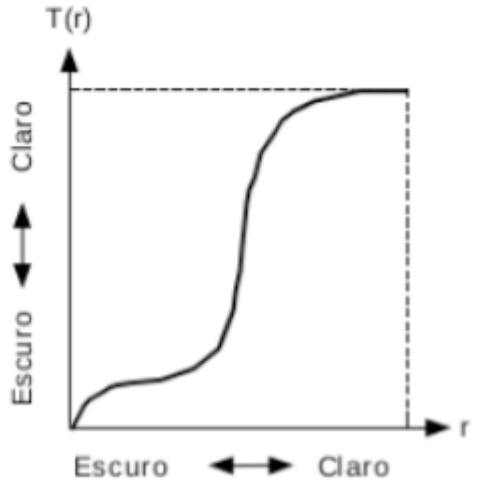
$$g = T(L) \qquad (4)$$

que mapeia cada tom de cinza L em um novo tom de cinza g na imagem destino. Na Figura 13 é possível ver o efeito de uma função de transformação de intensidade não-linear sobre uma imagem com a finalidade de aumentar seu contraste. As transformações de intensidade podem ser classificadas em lineares ou não-lineares. As transformações lineares podem ser genericamente expressas pela equação:

$$g = c.\ L + b \qquad (5)$$

onde o parâmetro c controla o contraste da imagem resultante e o parâmetro b o seu brilho (FILHO; NETO, 1999). As funções de mapeamento não-lineares também podem ser utilizadas para ressaltar características específicas em imagens, sendo que as principais são baseadas na função logaritmo, raiz quadrada, exponencial e quadrática (PEDRINI; SCHWARTZ, 2007).

Figura 13 - Transformação de intensidade.



Fonte: FILHO; NETO (1999).

2.2.6 Equalização de Histograma

A escolha de uma transformação de escala de cinza é em geral empírica, dependendo da aplicação e do conjunto de imagens para fornecer bons resultados. Entretanto, a equalização de histograma é um método de transformação que tem por finalidade produzir uma imagem cujo o histograma tenha um formato mais uniforme dos seus níveis de cinza, ou seja, os níveis devem aparecer na imagem aproximadamente com a mesma frequência (PEDRINI; SCHWARTZ, 2007). Conforme Filho e Neto (1999) a imagem obtida pela equalização de histograma contém praticamente o mesmo número (percentual) de *pixels* de qualquer nível de cinza. A forma mais comum de equalizar o histograma de uma imagem é utilizando uma função de distribuição acumulada da distribuição de probabilidades original, que pode ser expressa como:

$$g_k = T(f_k) = \sum_{i=0}^{k} p_f(f_i) = \sum_{i=0}^{k} \frac{n_i}{n} \qquad k = 0, 1, ..., L-1 \tag{6}$$

em que n é o número de pixels, n_i é o número de ocorrências do nível de cinza i, e $p_f\ (f_i)$ é a probabilidade do i-ésimo nível de cinza (PEDRINI; SCHWARTZ, 2007). Para que a transformação possa ser utilizada, é necessário normalizar os níveis de cinza da imagem no intervalo de $0 \ \leq \ f_k \ \leq \ 1$. O algoritmo 5 apresenta a técnica de equalização de histograma através da função de distribuição acumulada de probabilidade. Os níveis de cinza para imagem original f e para imagem equalizada g são respectivamente representados por f_k e g_k, com $0 \leq k \leq L-1$. A Figura 14 apresenta um exemplo da aplicação de equalização de histograma para aumentar o contraste de uma imagem com dimensões 446 x 297 e 256 tons de cinza. Na Figura 14(a) é apresentado a imagem original cujo o histograma é a Figura 14(b), e a Figura 14(c) mostra o resultado da equalização de seu histograma com seu correspondente histograma em Figura 14(d). As técnicas de equalização de histogramas também podem ser aplicadas a trechos de imagens, como por exemplo janelas $m \times n$ que servem principalmente para realçar detalhes sutis de pequenas porções de imagens (FILHO; NETO, 1999).

Algoritmo 5 – Equalização de histograma de uma imagem.

Inicialização. Calcular o histograma da imagem a ser transformada (Algoritmo 4).

Normalização. Ajustar os valores do histograma, tal que $0 \leq f_k \leq 1$.

Calcular Distribuição. Função distribuição acumulada de probabilidade tal que,

para $k = 0$ **até** $L-1$ **faça**

$$g_k = \sum_{i=0}^{k} p_f(f_i)$$

arredondando o valor para nível de cinza mais próximo,

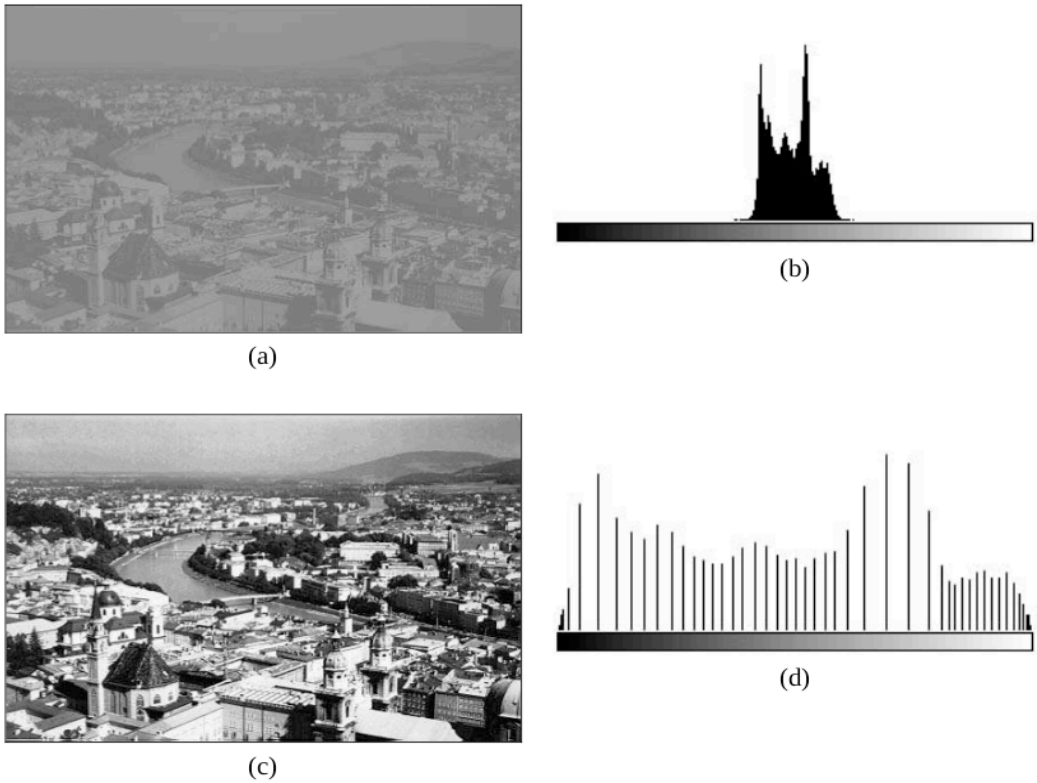
$$g_k = round(g_k \times L\max)$$

fim

Agrupamento. Agrupar valores de níveis de cinza f_k para g_k.

Fonte: PEDRINI; SCHWARTZ, 2007.

Figura 14 - Aplicação da equalização de histograma a uma imagem com baixo contraste.



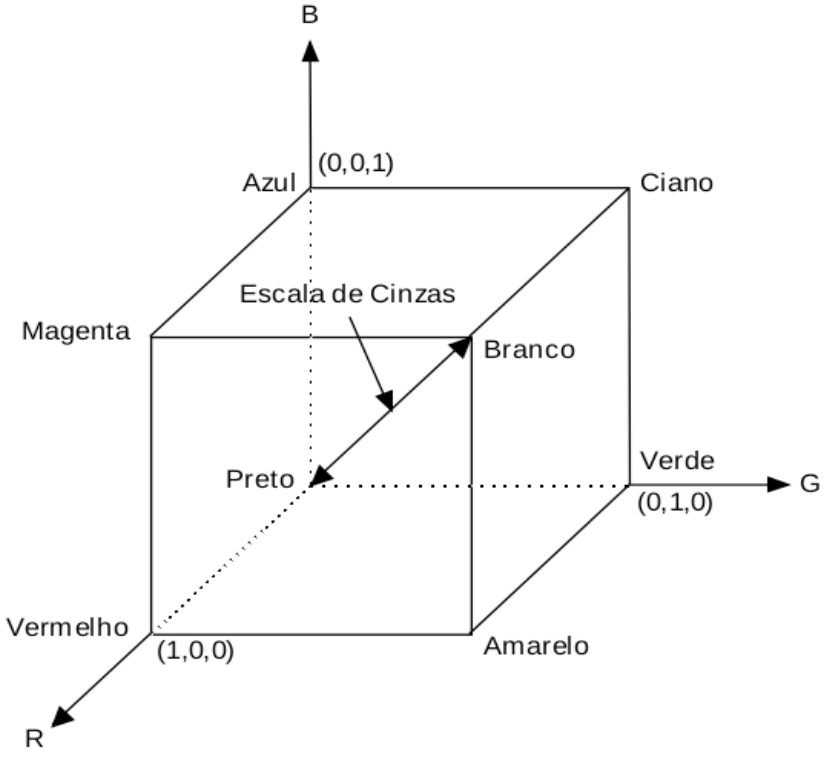
Fonte: FILHO; NETO (1999).

2.2.7 Processamento de Imagens Coloridas

As cores presentes em uma imagem são de grande importância no processo de identificação de objetivos realizado tanto pelos seres humanos quanto pelos computadores (PEDRINI; SCHWARTZ, 2007). De modo geral, um modelo de cores é uma representação tridimensional (sistemas de coordenadas 3-D), em que cada cor é representada por um ponto neste modelo. Segundo Filho e Neto (1999) os modelos mais utilizados para representação de cores são: *RGB* (*red, green, blue*), *CMY* (*cyan, magenta, yellow*), *CMYK* (variante do modelo CMY, onde K denota *black*), *YCbCr* (padrão normalizado pela recomendação ITU-R BT.601 e utilizado em técnicas de compressão de vídeo), *YIQ* (padrão NTSC de TV em cores) e *HSI* (*hue, saturation, intensity*).

O modelo *RGB* é baseado em um sistema de coordenadas cartesianas, que pode ser interpretado como um cubo onde três de seus vértices são as cores primárias, outros três as cores secundárias, o vértice junto à origem é o preto e o mais afastado da origem corresponde à cor branca, conforme é ilustrado na Figura 15 (FILHO; NETO, 1999). O modelo *RGB* é o mais comum pois é utilizado por câmeras e monitores de vídeo.

Figura 15 - Modelo *RGB* representado pelo sistema de coordenadas cartesianas.



Fonte: FILHO; NETO (1999).

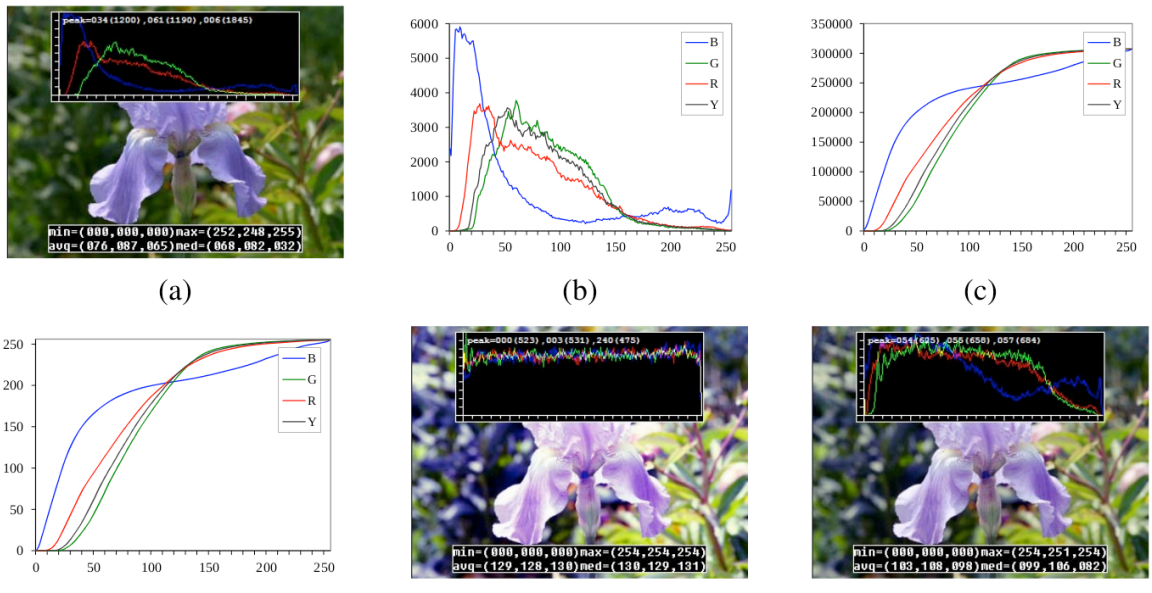
O modelo *CMY* é baseado nos pigmentos primários ciano, magenta e amarelo, em que a maioria dos dispositivos que utilizam esse modelo são as impressoras e as fotocopiadoras coloridas.

O modelo *YIQ* utilizado no padrão NTSC de TV em cores foi desenvolvido sob o princípio da dupla compatibilidade, que guiou os projetos de TV colorida para garantir a convivência entre o sistema colorido e o sistema preto e branco já existente. A sua principal vantagem é sua capacidade de permitir a separação entre o componente de brilho (Y) e os componentes de cromaticidade (I e Q) (FILHO; NETO, 1999).

Filho e Neto (1999) definem o modelo *HSI* como sendo de grande interesse, uma vez que permite separar as componentes de matiz, saturação e intensidade da informação de cor em uma imagem, da forma como o ser humano as percebe. Esse modelo tem sua utilização mais intensa em sistemas de visão artificial baseados no modelo de percepção de cor pelo ser humano.

A partir da decomposição da imagem colorida nas componentes adequadas, diversas técnicas existentes para imagens monocromáticas podem ser aplicadas com sucesso a imagens coloridas, como por exemplo a equalização de histograma. Na Figura 16 é ilustrado um exemplo de equalização de histograma em imagens coloridas.

Figura 16 - Análise de histograma e equalização: (a) imagem original; (b) histogramas em canais de cores; (c) funções de distribuição cumulativa; (d) funções de equalização; (e) equalização de histograma completa, e (f) equalização de histograma parcial.



(a) (b) (c)

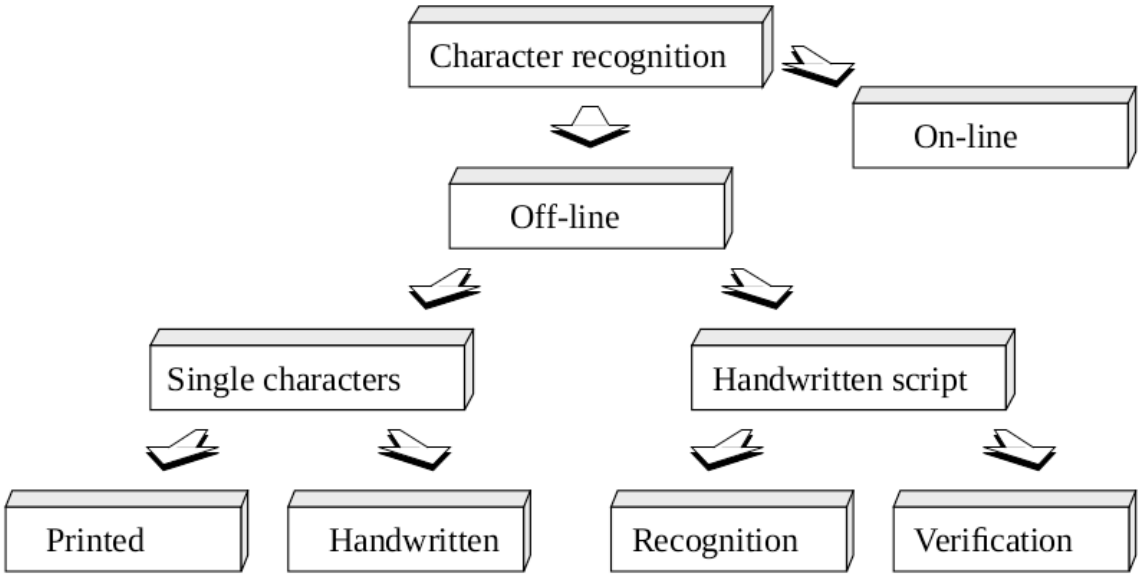
(d) (e) (f)

Fonte: SZELISKI (2010).

2.3 RECONHECIMENTO ÓPTICO DE CARACTERES (OCR)

O *OCR* é um método de reconhecimento de padrões que pertence a família de técnicas de identificação automática (EIKVIL, 1993). Existem diversas técnicas de identificação automática, na qual cada uma satisfaz às necessidades de sua área de aplicação. Reconhecimento de voz, tarja magnética, código de barra, impressão digital e *OCR* são exemplos de aplicações que utilizam tecnologias de identificação automática. Segundo Bhatia (2014), os sistemas de *OCR* são aqueles capazes de transformar uma grande quantidade de documentos, tanto impresso como manuscrito, em texto de máquina codificado. O reconhecimento de caracteres pode ser desempenhado *off-line*, onde ocorre após a escrita ou impressão do documento estiver finalizada, porém o reconhecimento também pode ser *on-line* onde o computador reconhece os caracteres na maneira que eles vão sendo desenhados (EIKVIL, 1993). A Figura 17 mostra as possíveis etapas para o reconhecimento de caracteres.

Figura 17 - Diferentes áreas de reconhecimento de caracteres.



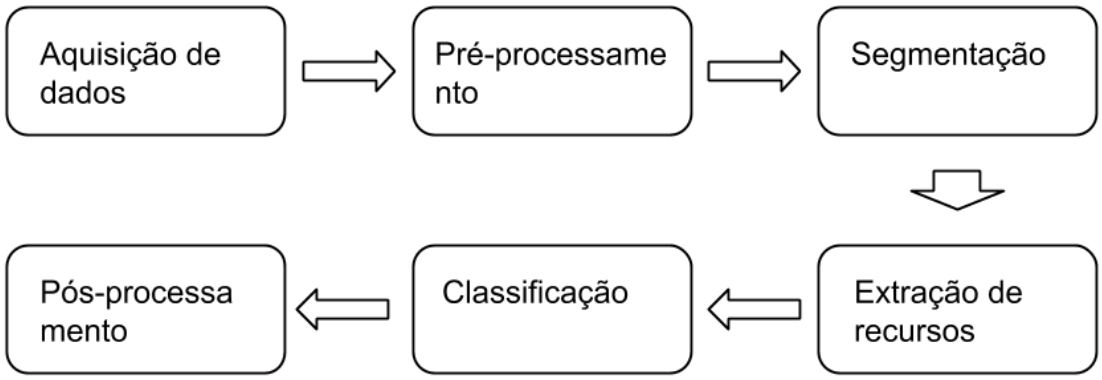
Fonte: EIKVIL (1993).

Em reconhecimento automático de padrões, o princípio fundamental é primeiro ensinar a máquina qual classes de padrões vão ocorrer e como eles se parecem. No *OCR*, esses padrões são letras, números e alguns símbolos especiais, enquanto as diferentes classes correspondem aos diferentes caracteres (EIKVIL, 1993). Com a evolução das técnicas de *OCR* novas áreas de aplicações estão surgindo, como por exemplo, melhora de sistemas e bancos de dados multimídia, bibliotecas eletrônicas, sistemas de monitoramento de trânsito entre diversas outras aplicações (GOSWAMI; SHARMA, 2013).

2.3.2 Componentes de um Sistema *OCR*

Um sistema típico de *OCR* consiste em diversos componentes, podendo variar de acordo com a aplicação e a metodologia utilizada. Na Figura 18 podem ser vistos uma série de componentes envolvendo as etapas do reconhecimento de caracteres.

Figura 18 - Sequência de etapas em um sistema *OCR*.



Fonte**:** GOSWAMI; SHARMA (2013).

A primeira etapa é a aquisição de dados, que pode ser feita através de um escaneamento óptico ou por uma câmera. Em seguida, a etapa de pré-processamento com a finalidade melhorar o documento para aplicação da segmentação. Na etapa de segmentação os caracteres são separados e extraídos para serem identificados. E por fim, os caracteres são reconstruídos em palavras pela etapa de pós-processamento.

Aquisição de dados: uma imagem digital do documento original pode ser obtida através de um processo de digitalização ou por uma captura fotográfica. Na criação de um sistema de reconhecimento de caracteres *on-line*, é utilizado digitalizadores que capturam diretamente a escrita pela ordem dos traços e pela velocidade (GOSWAMI; SHARMA, 2013). A qualidade da imagem obtida, seja através de um digitalizador ou por uma câmera fotográfica, é de grande relevância para o bom resultado na extração dos caracteres.

Pré-processamento: a imagem digital obtida na etapa de aquisição de dados pode conter uma certa quantidade de ruídos e imperfeições (EIKVIL, 1993). Portanto, o objetivo do pré-processamento é utilizar uma série de técnicas com a finalidade de melhorar a imagem para as etapas subsequentes (GOSWAMI; SHARMA, 2013). Das diversas técnicas que podem ser utilizadas, as principais são: redução de ruído e normalização. A redução de ruído tem o objetivo de resolver problemas como segmentos de linhas desconectados, lacunas nas linhas, distorção, entre outras imperfeições através de métodos de filtragem, operações morfológicas, suavização e modelagem de ruído (GOSWAMI; SHARMA, 2013). A operação

de normalização é considerada a etapa mais importante do pré-processamento, pois consegue obter caracteres de tamanho, inclinação e rotação uniforme (EIKVIL, 1993). Na Figura 19 é possível ver a aplicação das técnicas de normalização e suavização.

Figura 19 - Normalização e suavização de um caractere.

Fonte: EIKVIL (1993).

Segmentação: é o processo que determina os constituintes de uma imagem (EIKVIL, 1993). A segmentação é basicamente realizada em dois tipos: segmentação externa, na qual faz a separação de unidades escritas como palavras, frases ou parágrafos e a segmentação interna, que lida com a separação de letras geralmente escritas de forma cursiva (GOSWAMI; SHARMA, 2013). Em alguns casos os componentes de dois caracteres adjacentes podem estar sobrepostos criando dificuldades na etapa de segmentação, por isso é importante que esses caracteres sejam separados corretamente para que seja possível a identificação (BATHIA, 2014).

Extração de características: o objetivo desta etapa é capturar as características essenciais dos símbolos, no qual é um problema muito difícil de reconhecimento de padrões (EIKVIL, 1993). As diversas técnicas para extração de características podem ser classificadas nos três seguintes grupos.

1) *Transformação Global e Expansão em Série*: Um sinal contínuo geralmente contém as informações necessárias para o propósito de classificação. Uma abordagem para representar um sinal é a combinação linear de uma sequência de funções bem definidas (GOSWAMI; SHARMA, 2013). Os métodos mais comuns para transformação e expansão em série são as transformadas de Fourier, Gabor, Walsh, Haar, Hadamard, Hough e Karhunen-Loeve (EIKVIL, 1993).
2) *Representação Estatística:* As técnicas de representação estatística são utilizadas na variação de estilo do caractere. As principais técnicas utilizadas são Zoneamento, Cruzamentos e Projeções (GOSWAMI; SHARMA, 2013).
3) *Representação Topológica e Geométrica:* Diversas propriedades globais e locais dos caracteres podem ser representadas por características geométricas e topológicas. Essa

representação também pode codificar algum conhecimento sobre a estrutura do objeto ou fornecer alguma informação sobre quais tipos de componentes o objeto é composto.

Classificação: a classificação é o processo de identificar cada caractere e atribuir a ele a sua correspondente classe. De acordo com Eikvil (1993), existem duas principais abordagens para as metodologias de classificação: análise teórica de decisão e estrutura física do caractere. Em análise teórica os métodos são usados quando a descrição do caractere pode ser representada numericamente por um vetor de características. As principais técnicas utilizadas em análise teórica são classificadores de distância mínima, classificadores estatísticos e redes neurais. Já na abordagem derivada das propriedades físicas, é de grande importante uma relação entre as características extraídas do caractere ao decidir à qual classe ele pertence. Os principais métodos utilizados nessa abordagem são os métodos sintáticos, no qual medem a similaridade entre duas estruturas de componentes utilizando conceitos de gramática (EIKVIL, 1993).

Pós-processamento: nesta última etapa é feito o processo de agrupamento, que basicamente concatena os símbolos classificados para formar palavras de acordo com a distância entre eles no documento. Os símbolos que são encontrados próximos o suficiente são agrupados, e os outros separadas em palavras. Outra técnica também utilizada no pós-processamento é a detecção e correção de erros, que utiliza dicionários e regras de sintaxe para analisar as palavras agrupadas (EIKVIL, 1993).

2.3.3 Sistemas *OCR*

As primeiras máquinas de reconhecimento eram todos dispositivos físicos de *hardware*. Atualmente essas máquinas ainda são utilizadas para aplicações específicas onde a velocidade é de grande importância, como por exemplo ordenação de cartas, leitura de cheques e verificação de passaportes (EIKVIL, 1993). Pelo fato de essas máquinas terem um custo de fabricação alto e com os avanços da computação, foi tornando possível a implementação de *softwares* de *OCR* que operam em computadores pessoais. Entretanto, existem algumas limitações para o sistema de *OCR*, especialmente em relação a velocidade e o tipo de caractere a ser lido (EIKVIL, 1993).

Entre os anos de 1985 e 1994 foi desenvolvido um sistema de *OCR* compatível com diversos sistemas operacionais chamado de *Tesseract*. No ano de 1996 ele teve a portabilidade para o sistema operacional *Windows* e logo em seguida em 1998 seu código foi

reescrito da linguagem C para C++. Após seu funcionamento apresentar bons resultados, em 2005 seu código foi liberado com a ideologia de *software livre*. A partir de 2006 seu desenvolvimento foi apoiado pela *Google* e por desenvolvedores de todo o mundo. Embora o *Tesseract* seja uma aplicação de *OCR* gratuita, o resultado de sua aplicação em imagens digitais de documentos pode não conter todas as informações desejadas. Entretanto, existem outras aplicações modernas que conseguem desempenhar bons resultados na extração de texto de imagens, como por exemplo o *Google Cloud Vision API* e o *Microsoft Vision API*. Essas aplicações utilizam modelos de aprendizado de máquina capazes de extrair diversas características da imagem e possivelmente bons resultados na extração de texto.

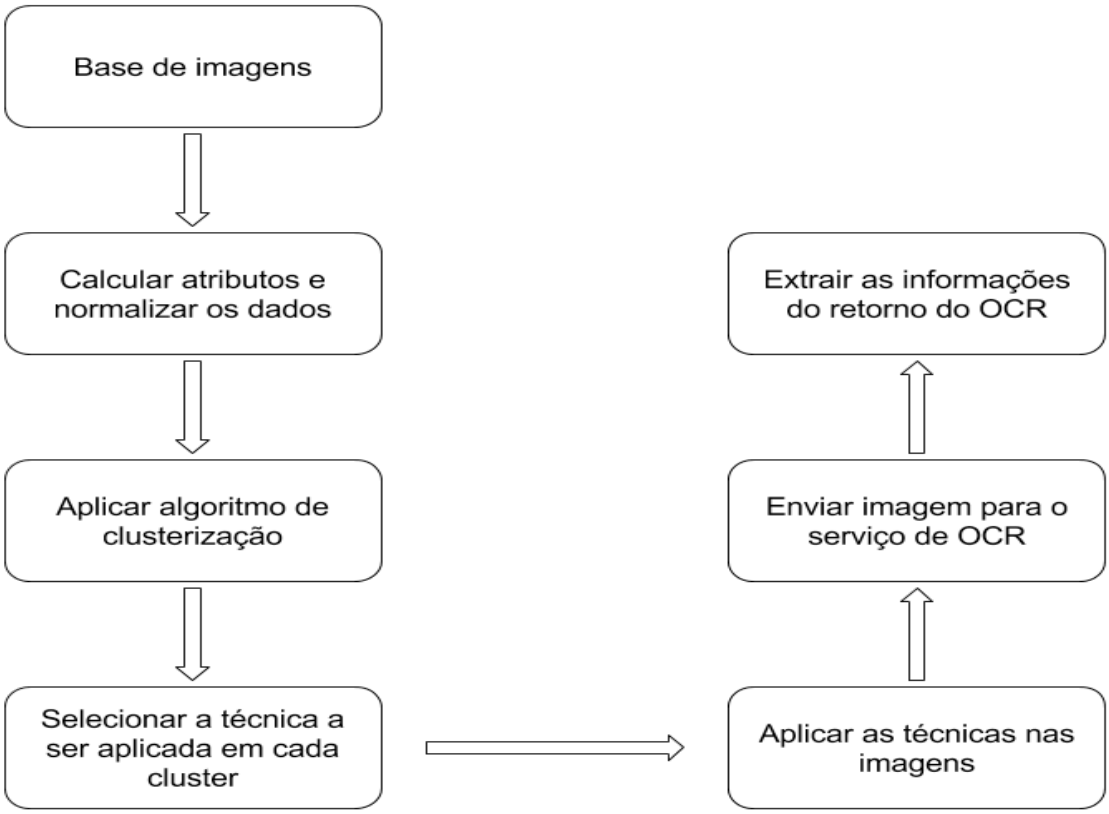
3 PROPOSTA DE ANÁLISE E CLASSIFICAÇÃO

A proposta de análise e classificação foi desenvolvida com base na concepção de que o *OCR* aplicado em uma imagem é um algoritmo de uso geral, que pode ser utilizado tanto para extrair os dados de um cupom fiscal quanto para extrair o número da placa de um carro, por exemplo. Contudo, considerando que as imagens a serem analisadas são de cupons fiscais, é possível modificar as suas características, visando obter melhora nos resultados do *OCR*. Diante disso, foi desenvolvida uma arquitetura em que as imagens percorrem pelas seguintes etapas: algoritmo de "clusterização", técnica de processamento de imagem e aplicação do *OCR*. Essa metodologia tem como objetivo analisar as imagens em subgrupos (*clusters*), e para cada um desses subgrupos aplicar uma técnica de PDI antes da aplicação do *OCR*. De acordo com o fluxograma da Figura 20, é possível observar cada etapa em que a imagem percorre na metodologia desenvolvida.

A primeira etapa consiste em selecionar a base de imagens distintas de cupons fiscais. Após obter os dados, a próxima etapa calcula e normaliza os atributos desejados das imagens para aplicar o algoritmo de "clusterização". A partir da normalização dos atributos, é aplicado um algoritmo para "clusterizar" as imagens. Os *clusters* gerados vão conter imagens com os atributos semelhantes entre elas, baseando-se nessa semelhança, na próxima etapa é selecionada qual a melhor técnica a ser aplicada em cada *cluster*. A técnica selecionada aproveita das características das imagens para aplicar modificações nestas de forma adaptativa. Tomando como exemplo a seleção de dois *clusters*, um contendo imagens com baixa iluminação e outro com alta, é possível aplicar uma técnica que melhore a iluminação para o primeiro *cluster*, e também outra técnica distinta que altere o contraste no segundo *cluster*. Com as técnicas selecionadas e aplicadas em cada *cluster*, as imagens resultantes são finalmente enviadas para um serviço de *OCR*. Por fim, a última etapa consiste em extrair as informações do resultado da aplicação do *OCR*.

A arquitetura proposta funciona de maneira genérica, ou seja, pode ser utilizado qualquer conjunto de atributos e algoritmos para aplicar a "clusterização" na base de imagens. Também pode ser desenvolvido um número de técnicas de até o número de *clusters* gerados, ou então aplicar a mesma técnica para dois *clusters* distintos. Em relação ao serviço de *OCR*, também pode ser utilizado qualquer um que funcione com imagens de cupons fiscais.

Figura 20 - Fluxograma da arquitetura desenvolvida.



Fonte: desenvolvido pelo autor.

No desenvolvimento da metodologia proposta, foi utilizado um conjunto de dados com 3200 imagens de cupons fiscais. O conjunto de dados foi fornecido pela empresa *SumOne*, que utilizava as imagens em suas aplicações de mercado. Na Figura 21, é possível ver alguns exemplos de imagens extraídas desse conjunto.

Essas imagens foram obtidas através de distintos *smartphones*, por isso elas variam a resolução, qualidade da câmera, iluminação e diversos aspectos. Cada elemento do conjunto de dados contém uma *url* que indica o endereço de onde a imagem está armazenada, e outros três campos que foram extraídos manualmente: CNPJ, valor total e a data e hora que foi efetuada a compra. Apesar de o cupom fiscal conter diversas informações da compra, esses três campos são de grande importância para as aplicações modernas, tendo em vista que é possível derivar outras informações como nome e local do estabelecimento, valor médio de consumo por data, entre outros.

Figura 21 - Exemplos de imagens do conjunto de dados.

Fonte: desenvolvido pelo autor.

A construção da arquitetura foi feita utilizando a linguagem de programação *Python*. Sua escolha foi devido a praticidade na instalação de dependências, facilidade para desenvolver o código e principalmente pela disponibilidade de bibliotecas gratuitas de desenvolvimento científico nas áreas de análise em cluster e processamento de imagens. A biblioteca para análise em *cluster* utilizada no projeto foi o *scikit-learn*, que contém a implementação de diversos algoritmos clássicos de "clusterização". No desenvolvimento das técnicas de PDI, foram utilizadas as bibliotecas *Pillow, scikit-image* e *OpenCV*. As três bibliotecas contêm diversos algoritmos clássicos de processamento de imagens, porém são distintas em quesitos de implementação e quantia de funcionalidades.

3.1 ANÁLISE EM *CLUSTER*

O propósito da etapa de análise em *cluster* é separar as imagens com características semelhantes em subgrupos, analisando-os para aplicação de técnicas de PDI direcionadas. No desenvolvimento da metodologia para alcançar esse objetivo, foram estabelecidos quatro requisitos: seleção dos atributos, normalização, aplicação do algoritmo e visualização dos resultados.

3.1.1 Seleção dos Atributos

A etapa inicial da análise em *cluster* consiste na seleção dos atributos que serão utilizados para criação dos *clusters*. As imagens do conjunto de dados possuem diversos atributos que podem ser utilizados para realizar a "clusterização", porém é necessário selecionar apenas os que possam gerar bons resultados para uma futura análise e aplicação das técnicas de PDI. Foram selecionados dois atributos na proposta desenvolvida: o número de *pixels* e o nível de contraste de cada imagem.

O atributo número de *pixels* de uma imagem é obtido através da quantidade de *pixels* que a imagem possui, ou seja, levando em consideração que a imagem é uma matriz de *pixels*, esse valor é a multiplicação do número de linhas pelo número de colunas. O atributo pode ser considerado interessante para criação dos *clusters* pelo fato de que o conjunto de dados contêm imagens de diferentes câmeras, contendo uma boa distribuição desse valor. A Figura 22 mostra duas imagens do conjunto de dados, sendo a primeira com um alto número de *pixels* e a segunda com um baixo número de *pixels*.

O segundo atributo utilizado, o nível de contraste, tem como finalidade medir o grau de contraste de uma imagem. Essa medida é obtida através do algoritmo do Código 1, que foi inspirado pela função *is_low_contrast* da biblioteca *scikit-image*. Descrevendo informalmente o algoritmo, ele recebe como parâmetro a imagem e outros dois valores fixos que são utilizados para calcular os limites dos níveis de cinza. Primeiro, a imagem é convertida para um *array* do *numpy* e em seguida é verificado se a imagem contém três canais de cores, se sim é convertida para apenas níveis de cinza, caso contrário ela já está em escala de cinzas. Após isso, o algoritmo calcula os limites de intensidade da imagem e os limites da escala de cinza. Por fim, a função retorna a razão da diferença dos limites, um valor numérico que representa o nível de contraste da imagem. A Figura 23 contém duas imagens com diferentes valores de nível de contraste calculado pela função do Código 1. Na base de dados, o nível de contraste varia em relação a quantidade de iluminação e ao ambiente na qual foi batida a foto do cupom fiscal.

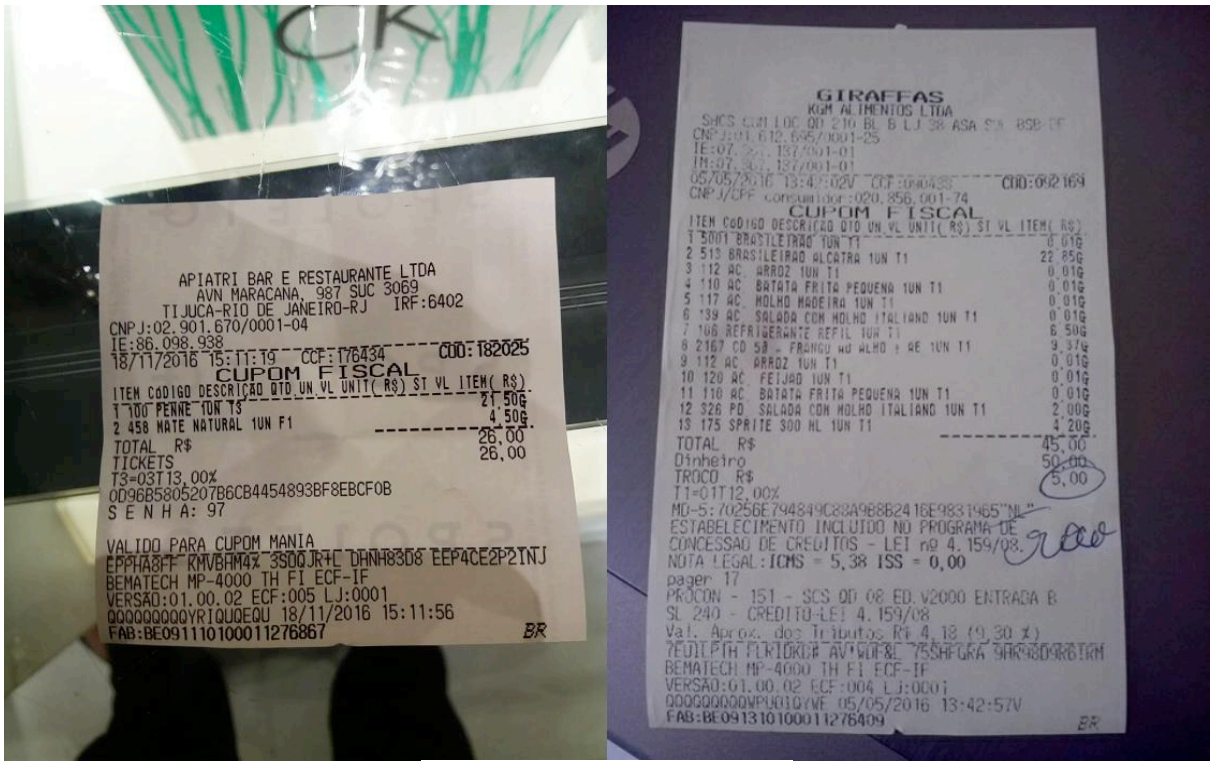
Código 1 – Função que calcula o nível de contraste de uma imagem.

```
def contrast_ratio(image, lower_percentile=1, upper_percentile=99):
    image = np.asanyarray(image)
    if image.ndim == 3 and image.shape[2] in [3, 4]:
        image = rgb2gray(image)
```

```
dlimits = dtype_limits(image, clip_negative=False)
limits = np.percentile(image, [lower_percentile, upper_percentile])

return (limits[1] - limits[0]) / (dlimits[1] - dlimits[0])
```

Fonte: desenvolvido pelo autor.

Figura 22 - Comparação de imagens com diferentes resoluções de *pixels*. Imagem da esquerda com *2560x1920* e da direita com 640x480.



Fonte: desenvolvido pelo autor.

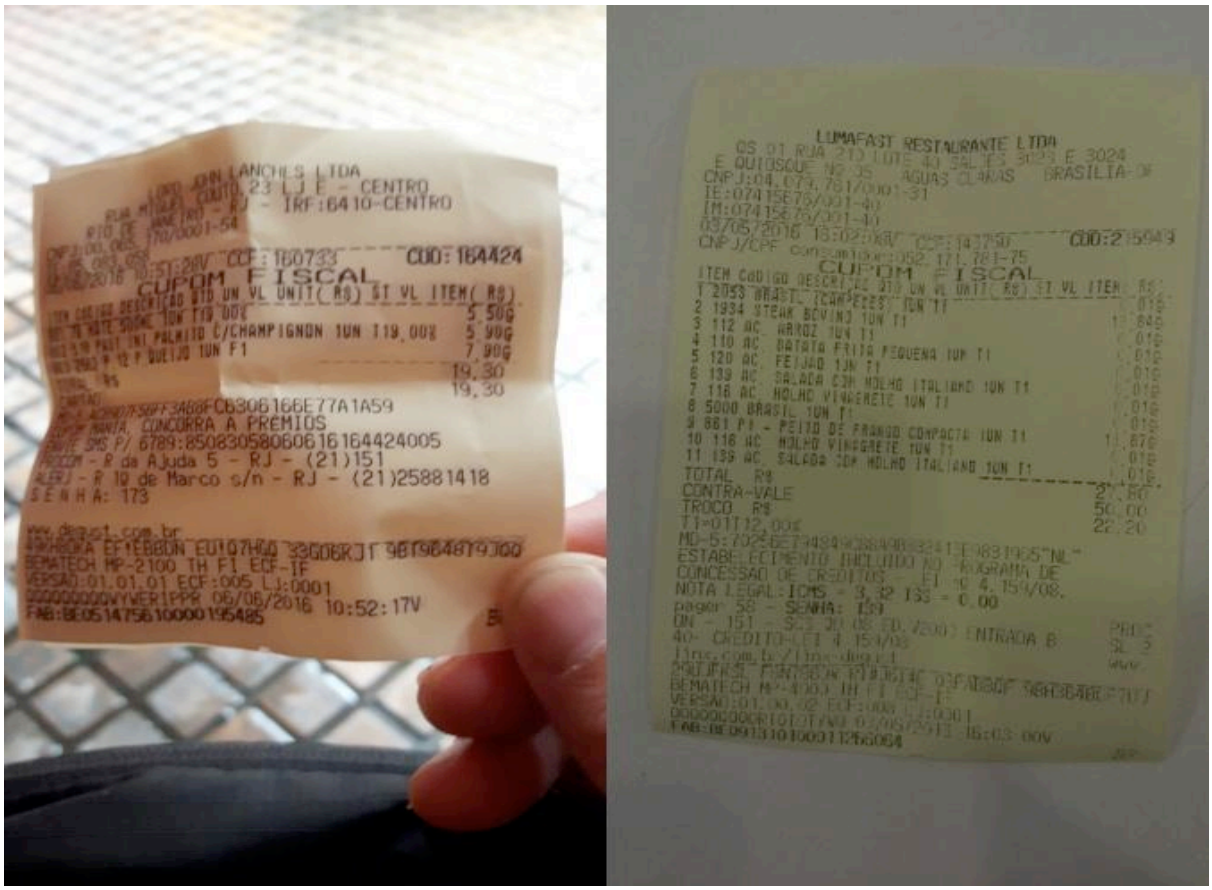
3.1.2 Normalização

De acordo com Han e Kamber (2000), normalização é o ato de dimensionar os dados do atributo de modo a cair dentro de um pequeno intervalo especificado, como -1 a 1 ou 0 a 1. Os atributos selecionados contêm distintas ordens de magnitude, e para que isso não afete os resultados dos algoritmos de "clusterização", é necessário normalizá-los para obter valores em ordens semelhantes. Entre os diversos métodos de normalizar dados, o utilizado na proposta foi a normalização min-max, também conhecida como normalização linear. Esse método transforma o menor valor em 0 e o maior valor em 1, fornecendo uma maneira simples de

comparar os valores medidos em diferentes escalas. Sua definição é dada pelo seguinte cálculo (MOHAMED, ISMAIL; USMAN, 2013):

$$MM\left(X_{ij}\right) = \frac{X_{ij} - X_{\min}}{X_{\max} - X_{\min}} \qquad (7)$$

Figura 23 - Comparação de imagens com diferentes níveis de contraste. Imagem da esquerda com *nível > 0,45* e imagem da direita com *nível < 0,15*.



Fonte: desenvolvido pelo autor.

3.1.3 Aplicação dos Algoritmos de *Cluster*

Considerando todos os dados devidamente pré-processados, a próxima etapa é aplicar de fato um algoritmo que realize a distribuição do conjunto de dados em *clusters*. Na proposta realizada, o escopo foi limitado a análise de três algoritmos de "clusterização", sendo eles: *K-means*, DBSCAN e *Spectral Clustering*. Cada um desses algoritmos utiliza um

método distinto para verificar a semelhança entre os dados, diante disso foi realizada uma análise comparativa entre eles para determinar a qualidade dos *clusters* gerados.

3.1.4 *K-Means*

De acordo com a descrição apresentada na seção 2.1.4, o algoritmo *K-means* tem a funcionalidade de posicionar N pontos de dados de um espaço dimensional I em K *clusters*. Assim sendo, N é o conjunto de dados de imagens, I a dimensão dos atributos selecionados e K o número de *clusters* que serão gerados. Analisando o Código 2, é possível visualizar cada etapa da aplicação do algoritmo, primeiro a seleção dos atributos, seguido da normalização e aplicação do *K-means*.

A variável *cluster_data* contém o conjunto das 3200 imagens, cada elemento contendo seus atributos, incluindo os selecionados para análise em *cluster*. Na primeira linha é feito um mapeamento para extrair apenas os valores selecionados para a "clusterização", o número de *pixels* e o nível de contraste. Em seguida, é aplicado a normalização min-max nos atributos mapeados. Após isso, o objeto que representa a "clusterização" é criado utilizando a função *K-Means*. O K número de *clusters* que o algoritmo irá gerar é decidido pelo parâmetro *n_clusters*, que foi fixado em 5 *clusters* na análise desenvolvida. Além destes, outros parâmetros podem ser utilizados, como por exemplo o *init*, no qual o algoritmo utilizará inicializar os valores das centróides. A lista com todos os parâmetros para o algoritmo *K-Means* pode ser encontrada em SCIKIT-LEARN (2017). Por fim, o método *fit_predict* retorna uma lista com o número do cluster criado para cada elemento do conjunto de dados.

Código 2 – Aplicação do algoritmo *K-means* no conjunto de dados.

```
x_values = np.array([[data['n_pixel'], data['contrast_ratio']] for data in cluster_data])
normalized_x_values = normalize(x_values)
cluster = KMeans(n_clusters=5, init='k-means++')
y_values = cluster.fit_predict(normalized_x_values)
```

Fonte: desenvolvido pelo autor.

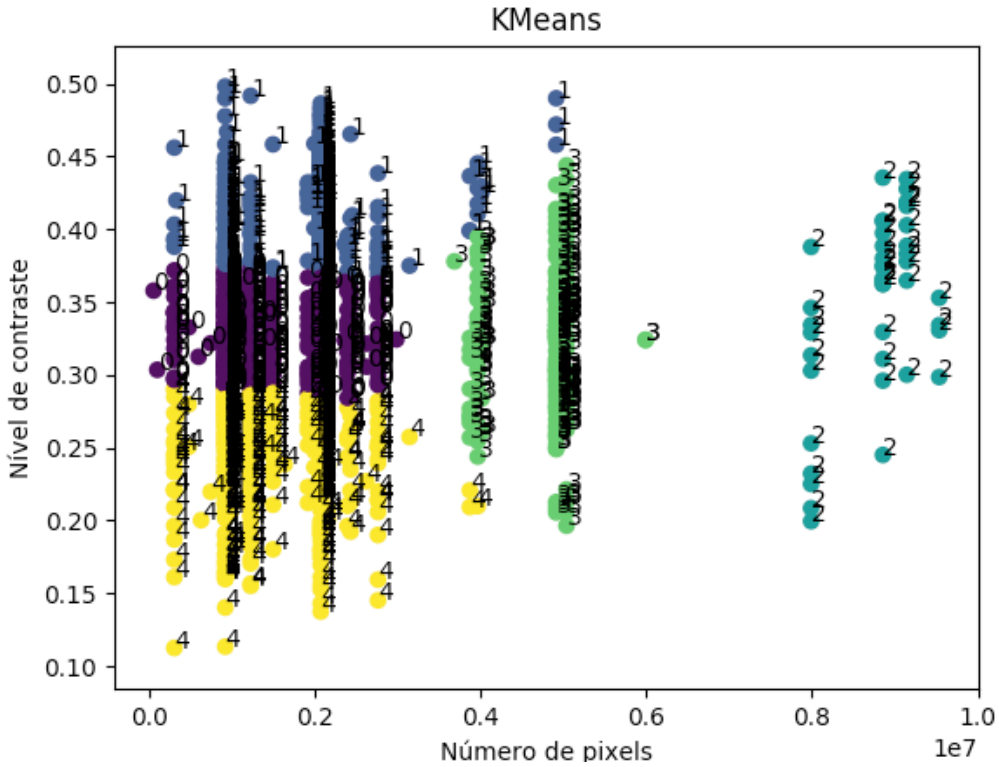
Para visualizar os resultados, o Quadro 4 mostra o *cluster* gerado com o correspondente número de imagens atribuídas a ele, e a Figura 24 demonstra o gráfico de número de *pixels* por nível de contraste com os respectivos *clusters* gerados.

Quadro 4 - Resultado da aplicação do algoritmo *K-Means*, com o cluster gerado e respectivo número de imagens.

Cluster	Quantidade de imagens
0	1323
1	1062
2	42
3	208
4	565

Fonte: desenvolvido pelo autor.

Figura 24 - Gráfico representativo dos *clusters* gerados utilizando o algoritmo *K-Means* no conjunto de imagens. *OBS.:* Os dados do gráfico não estão normalizados para facilitar a visualização.



Fonte: desenvolvido pelo autor.

Cada ponto do gráfico é denotado por uma cor e pelo número do correspondente *cluster* atribuído. O primeiro *cluster*, número zero e cor roxa, é um conjunto de imagens de baixa resolução de *pixels* e nível de contraste médio. Já o próximo *cluster*, número 1 e cor

azul marinho, representa em sua maioria imagens com baixa resolução de *pixels*, porém todas com um alto nível de contraste. O terceiro *cluster*, número 2 e cor azul claro, é o conjunto de imagens com alta resolução de *pixels*, e como o número de elementos atribuídos a esse *cluster* foi relativamente inferior, ele contém imagens tanto de alto como de baixo contraste. O *cluster* de número 3 e cor verde, contém um conjunto de imagens com número médio de *pixels*, porém variando em quase todos os níveis de contraste. Enfim, o último *cluster* gerado com número 4 e cor amarela, denota as imagens de baixa resolução de *pixels* e baixo nível de contraste.

3.1.5 DBSCAN

O DBSCAN é um algoritmo de "clusterização" baseado em densidade, que necessita da definição de dois parâmetros para calcular os *clusters*, sendo eles: *min_samples* que representa o número mínimo de pontos para definir se um ponto é central, e o *eps* que denota a máxima distância entre dois pontos para que sejam considerados na mesma vizinhança. A aplicação do algoritmo pode ser visualizada no Código 3, considerando que a seleção dos atributos e a normalização dos dados foram feitas igualmente a aplicada no algoritmo *K-Means*, no Código 2.

Código 3 – Aplicação do algoritmo DBSCAN no conjunto de dados.

```
cluster = DBSCAN(eps=0.13, min_samples=300)
y_values = cluster.fit_predict(normalized_x_values)
```

Fonte: desenvolvido pelo autor.

A chamada de função *DBSCAN*, pode receber diversos outros parâmetros além dos necessários *eps* e *min_samples*, como por exemplo distintas métricas de distância e variações da implementação do algoritmo (SCIKIT-LEARN, 2017). Os parâmetros *eps* e *min_samples* foram fixados respectivamente em 0.13 e 300, com a finalidade de gerar pequenos e densos clusters. Porém, como é possível observar no Quadro 5, apenas dois *clusters* foram gerados, o primeiro contendo 92% dos dados e o segundo apenas 8%.

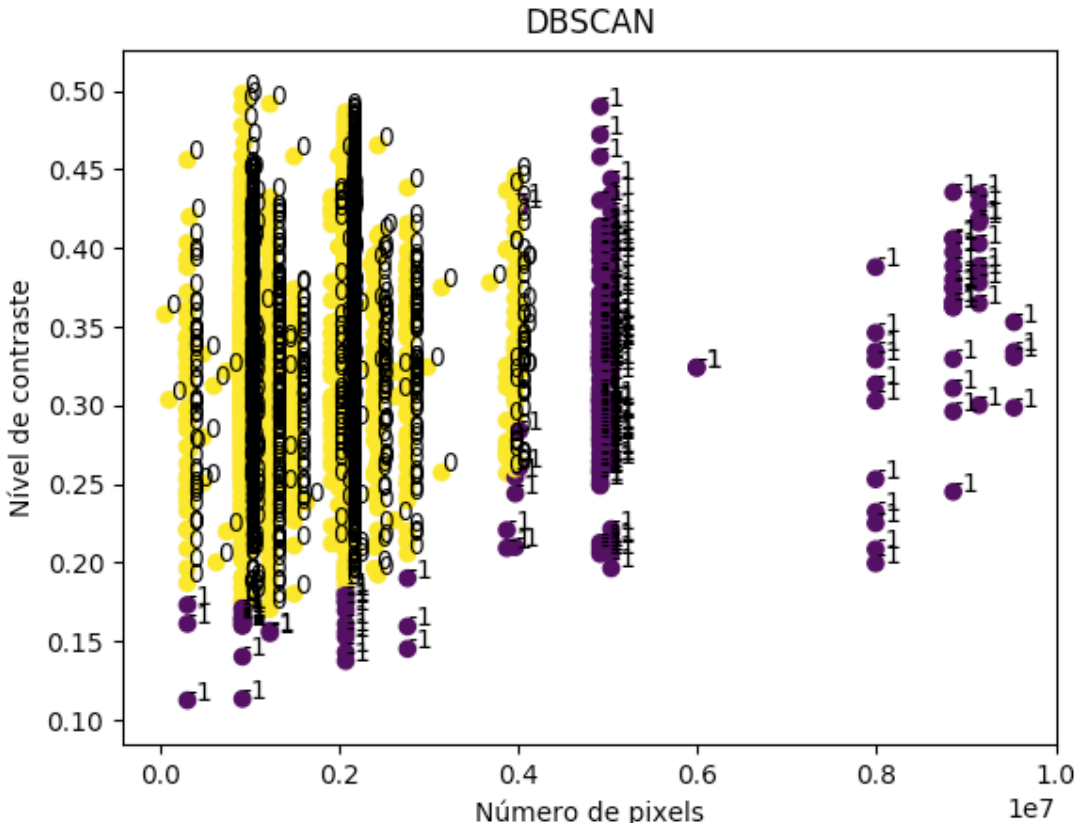
Quadro 5 - Resultado do algoritmo DBSCAN, com o cluster gerado e o respectivo número de imagens.

Cluster	Quantidade de imagens
0	2957
1	243

Fonte: desenvolvido pelo autor.

A Figura 25 apresenta o gráfico com os respectivos *clusters* gerados, contendo os pontos com a cor e o número do *cluster* atribuído. O primeiro *cluster*, número 0 e cor amarela, contém em geral imagens entre baixa e média resolução de *pixel*, variando em todos as faixas de níveis de contraste. Já o outro *cluster*, número 1 e cor roxa, denota imagens com alta resolução de *pixel* e também com o nível de contraste em todos os valores.

Figura 25 - Gráfico representativo dos clusters gerados utilizando o algoritmo DBSCAN no conjunto de imagens. *OBS: Os dados do gráfico não estão normalizados para facilitar a visualização.*



Fonte: desenvolvido pelo autor.

Fazendo uma análise comparativa entre os clusters gerados pelo *K-Means*, e pelo DBSCAN, fica evidente que o primeiro algoritmo separou os *clusters* de maneira mais eficiente. Isso é justificado pelo número de *clusters* e pela quantidade de imagens distribuídas

entre eles. Além disso, o *K-means* utilizou melhor o atributo nível de contraste para distinguir os *clusters*, possibilitando analisar clusters com baixo contraste e baixa resolução de *pixel* e também *clusters* de alto contraste e baixa resolução de *pixels*.

3.1.6 *Spectral Clustering*

Para a aplicação do *Spectral Clustering* na análise proposta, foi necessário definir o número de clusters que seria gerado pelo algoritmo. O número de clusters foi estabelecido em 5, da mesma maneira que a aplicação do *K-means* para facilitar a comparação dos resultados entre os algoritmos. Tanto os atributos selecionados como a normalização foram calculados do mesmo jeito que no *K-means*, no Código 2. Outros parâmetros podem ser utilizados para alterar o funcionamento do algoritmo, sua implementação pode ser visualizada em SCIKIT-LEARN (2017).

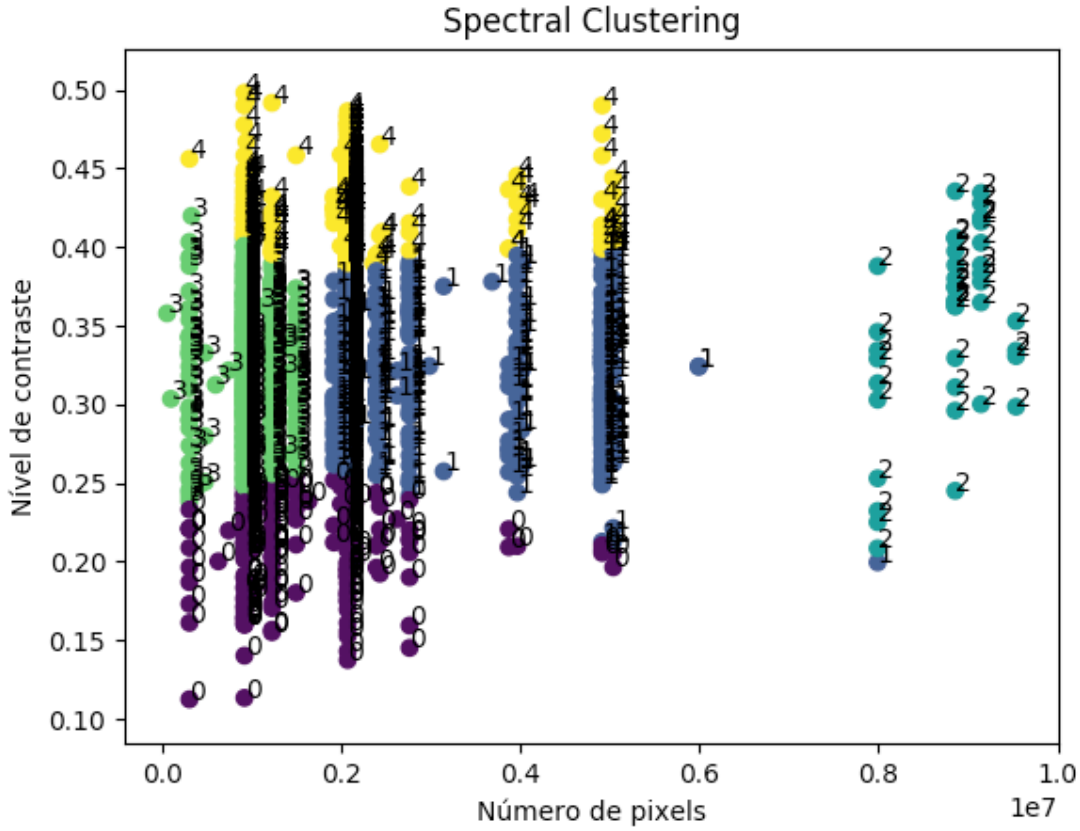
Em relação aos resultados do *Spectral Clustering*, o Quadro 6 demonstra os *clusters* gerados e seus respectivos números de imagens. Já na Figura 26 é possível verificar cada *cluster* no conjunto de dados com seu correspondente número e cor. O primeiro cluster gerado, número 0 e cor roxa, contém imagens de baixa e média resolução de *pixels*, porém em geral com baixo nível de contraste. O segundo *cluster*, número 1 e cor azul marinho, representa as imagens de médio número de *pixels* e médio contraste. Em relação ao terceiro *cluster*, número 2 e cor azul claro, representa as imagens com alta resolução de *pixels*. O próximo *cluster*, número 3 e cor verde, denota as imagens com baixa resolução de *pixels* e médio nível de contraste. Por fim, o último *cluster* gerado, número 4 e cor amarela, é o conjunto de imagens de baixo a médio número de *pixels*, porém com alto nível de contraste.

Quadro 6 - Resultado da aplicação do algoritmo *Spectral Clustering*, com o *cluster* gerado e respectivo número de imagens.

Cluster	Quantidade de imagens
0	262
1	1313
2	41
3	808
4	776

Fonte: desenvolvido pelo autor.

Figura 26 - Gráfico representativo dos *clusters* gerados utilizando o algoritmo *Spectral Clustering* no conjunto de imagens. *OBS: Os dados do gráfico não estão normalizados para facilitar a visualização.*



Fonte: desenvolvido pelo autor.

Comparando esses resultados com o dos algoritmos previamente analisados, é possível concluir que o *Spectral Cluster* obteve um bom resultado na geração dos *clusters*. As imagens foram distribuídas de maneira semelhante ao *K-means*, tendo como maior diferença os *clusters* centrais. No *K-Means* o *cluste*r no centro do gráfico, número 3 e cor verde, obteve imagens de média resolução de *pixels* e com todos os níveis de contraste. Em contrapartida, o *Spectral* separou o cluster do centro, número 1 e azul marinho, em uma maior faixa de número de *pixels* e com níveis de contraste entre aproximadamente 0,22 e 0,4, ou seja, contraste médio. Em conclusão, os resultados de ambos são relevantes para analisar a aplicação das técnicas de PDI que serão descritas na próxima seção.

3.2 TÉCNICAS DE PROCESSAMENTO DE IMAGENS

Retomando a arquitetura proposta na Figura 20, a aplicação das técnicas de PDI no conjunto de dados é a última etapa antes do envio das imagens para o serviço de *OCR*. Neste

passo, o conteúdo da imagem é alterado através da aplicação de algum algoritmo, podendo ter o objetivo de ajustes de brilho e contraste, filtros adaptativos ou até compressões na imagem. Isto posto, é possível desenvolver diversas técnicas para serem aplicadas no conjunto de imagens, mais especificamente, aplicando uma técnica distinta para cada cluster gerado ou a mesma técnica em demais clusters.

Na proposta desenvolvida, foram elaboradas três técnicas: Técnica de Realces, Técnica de Binarização e Técnica de Compressão. Cada uma dessas técnicas utiliza um algoritmo distinto para alterar a imagem, permitindo analisar os resultados da aplicação de uma determinada técnica em um ou mais clusters. Nesta seção, é apresentado como foram desenvolvidas cada uma dessas três técnicas, e posteriormente no capítulo 4, a repercussão da aplicação delas nos *clusters* gerados.

3.2.1 Técnica de Realces

A primeira técnica desenvolvida, nomeada Técnica de Realces, aplica um conjunto de quatro algoritmos de realce em imagens, sendo eles: *Sharpness*, ajuste de brilho, ajuste de contraste e balanceamento de cor. O objetivo desses algoritmos é alterar as imagens de modo que a aplicação do *OCR* consiga extrair de forma mais eficiente os dados do cupom fiscal.

Aplicar um ajuste de *Sharpness*, geralmente traduzido como nitidez, altera visualmente a imagem em relação a textura e clareza dos detalhes. Para utilizar esse ajuste, é necessário aplicar o algoritmo de *Unsharp Mask* na imagem, sendo obtido pela subtração da imagem original com sua versão *borrada*. Após isso, é necessário somar o resultado da subtração com uma versão em alto contraste da imagem e com a imagem original, obtendo uma imagem mais nítida [CAMBRIDGE, 2017).

O brilho de uma imagem é um atributo da percepção visual em que uma fonte parece irradiar ou refletir luz. Em outras palavras, o brilho é a percepção ocasionada pela *luminância* de um alvo visual. Na escala de cores *RGB*, a taxa de brilho pode ser calculada através da média aritmética dos canais de cores (WIKIPEDIA, 2017). Para aumentar o brilho de uma imagem é necessário somar um *offset* nos canais de cores de cada pixel, e para diminuir é necessário subtrair esse *offset* (SZELISKI, 2010).

Em relação ao contraste da imagem, este representa a diferença de luz entre regiões claras e escuras da imagem. O contraste pode ter um impacto significante na percepção visual de uma imagem por enfatizar suas texturas (CAMBRIDGE, 2017). Para interpretar o nível de contraste da imagem, é possível analisar a distribuição dos pixels na escala de cinza em seu

histograma. Diversas técnicas de modificação de histograma podem ser implementadas a partir do conceito de transformações de intensidade, como por exemplo uma transformação linear (FILHO; NETO, 1999).

No processamento de imagens, o equilíbrio de cores se refere ao ajuste global das intensidades das cores, tipicamente nos canais primários vermelho, verde e azul. O objetivo deste ajuste é renderizar cores específicas, que pode ser chamado de equilíbrio de cinza, equilíbrio de neutro ou balanço de branco. O equilíbrio de cores altera o envolvimento geral das cores em uma imagem e é utilizado para correção de cores (WIKIPEDIA, 2017).

No Código 4 é possível ver a aplicação dos quatro realces descritos em uma imagem qualquer. A biblioteca utilizada *Pillow*, contém o módulo de *ImageEnhance* que implementa todos os quatro realces. O uso do realce é feito pela chamada da função *enhance*, que recebe por parâmetro um fator de intensidade para ser utilizado no método. O fator de intensidade em 1 indica que a imagem não será alterada, já entre 0 e 1 o fator de intensidade é baixo e de 1 a 2 o fator de intensidade é alto (PILLOW, 2017).

Código 4 – Aplicação da Técnica de Realces em uma imagem.

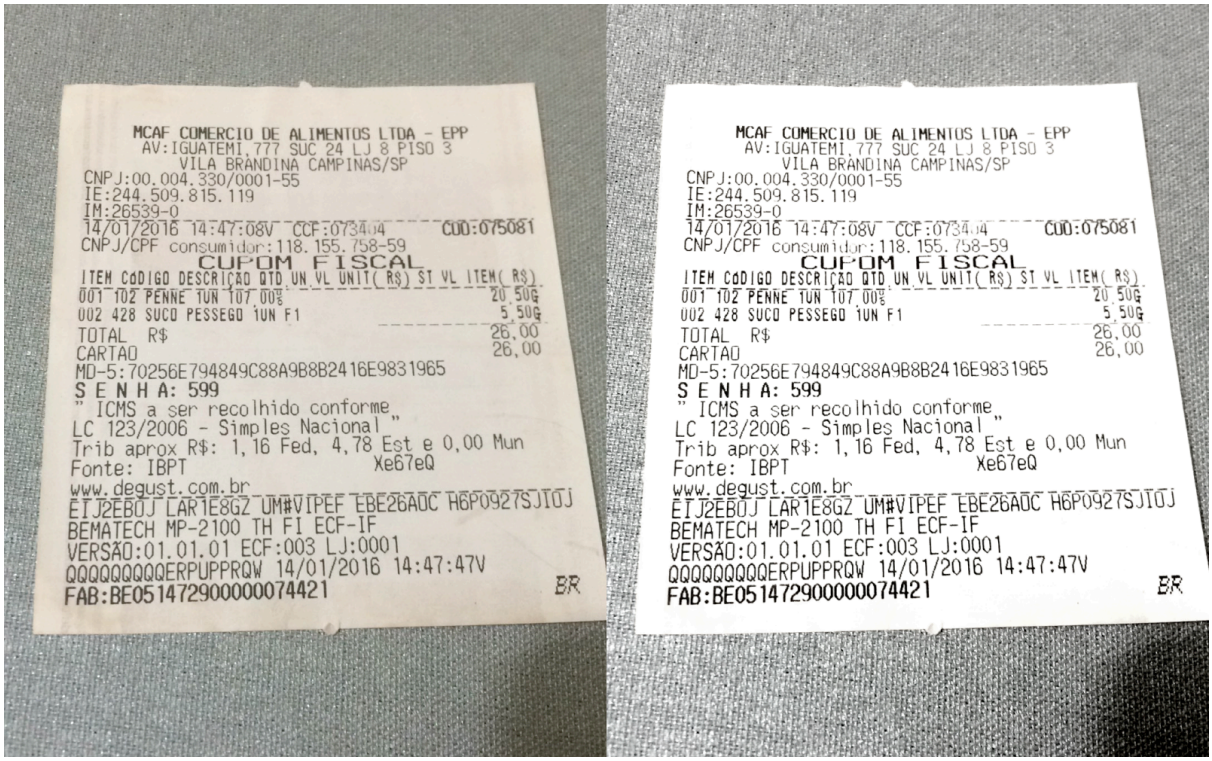
```
def techniqueEnchance(image):
    filtered = ImageEnhance.Sharpness(image).enhance(1.3)
    filtered = ImageEnhance.Brightness(filtered).enhance(1.3)
    filtered = ImageEnhance.Contrast(filtered).enhance(1.6)
    filtered = ImageEnhance.Color(filtered).enhance(0.2)

    return filtered
```

Fonte: desenvolvido pelo autor.

O primeiro realce a ser aplicado é o *Sharpness*, utilizando um fator de intensidade em 1.3, pois seu objetivo é deixar os detalhes dos textos na imagem mais nítidos, não exagerando para não criar ruídos. Após a aplicação do *Sharpness,* o próximo realce é o ajuste de brilho que também utiliza um fator em 1.3, aumentando sucintamente o brilho da imagem. Com o ajuste de brilho, é aplicado o realce de contraste com fator de intensidade mais elevado em 1.6, com objetivo de melhorar visualmente os destaques da imagem. Por fim, o último realce aplicado é o equilíbrio de cores com um fator de intensidade baixo em 0.2, o que representa uma imagem com cores mais próxima dos tons de cinza.

Na Figura 27 é possível visualizar um exemplo da aplicação da Técnica de Realces em uma imagem de cupom fiscal. Na esquerda está a imagem original, e na direita o resultado da aplicação da técnica.

Figura 27 - Demonstração da aplicação da Técnica de Realces em uma imagem. Na esquerda Imagem original, e na direita o resultado da técnica.



Fonte: – desenvolvido pelo autor.

Fazendo uma análise comparativa entre as duas imagens, é possível observar que na imagem original as cores do papel do cupom estão meio amarelas e manchadas, não dando contraste aos tons de pretos do texto. Porém, na imagem com a aplicação da técnica, a cor do papel é normalizada em branco e os tons de preto do texto contêm um melhor destaque, ficando mais nítido e facilitando a leitura dos valores.

3.2.2 Técnica de Binarização

O objetivo da Técnica de Binarização é aplicar um algoritmo de limiarização *(thresholding)* na imagem, separando suas regiões em fundo e objeto, ou seja, no escopo de cupons fiscais em papel e texto. Conforme Filho e Neto (1999), a forma mais simples de limiarização consiste na bipartição do histograma, convertendo os pixels cujo tom de cinza é maior ou igual a um certo valor de limiar T em brancos e os demais em pretos. A operação de limiarização pode ser descrita como uma técnica de processamento de imagens na qual uma imagem de entrada $f(x,\ y)$ de N níveis de cinza produz na saída uma imagem $g(x,\ y)$,

chamada de imagem limiarizada, cujo o número de níveis de cinza é geralmente dois, ou seja, também chamada de imagem binarizada (FILHO; NETO, 1999).

A função de limiarização T pode ser obtida de duas maneiras, forma global ou adaptativa. Quanto T depende apenas de $f(x, y)$, ou seja, o tom de cinza original no ponto (x, y), o limiar é classificado como global. Porém, quando T depende também de $p(x, y)$, isto é, alguma propriedade local de $f(x, y)$, o limiar é classificado como local ou adaptativo. Se a imagem tiver um fundo relativamente uniforme, pode ser aplicado a limiarização global. No entanto, se houver uma grande variação na intensidade do fundo, aplicar a limiarização local é mais adequado para obter melhores resultados (SCKIT-IMAGE, 2017a).

No desenvolvimento da Técnica de Binarização, foi utilizado a biblioteca *Scikit-Image* para aplicar a limiarização local nas imagens. Através do Código 5, é possível analisar cada etapa envolvida na aplicação da técnica em uma imagem. O primeiro passo do algoritmo é transformar a imagem em tons de cinza, pois é necessário à aplicação da limiarização. A chamada de função *threshold_local* é o que calculada de fato o limiar, sendo que a função recebe por parâmetro a imagem em tons de cinza, o tamanho do bloco de vizinhos e um *offset* para ser subtraído do peso do bloco.

Como o número de *pixels* do conjunto de imagens é bastante variável, foi criado uma função no Código 6 com a finalidade de selecionar o tamanho do bloco para calcular o limiar de acordo com o tamanho da imagem. Quanto maior o número de *pixels* da imagem, maior o tamanho do bloco e quanto menor o número de pixels, menor o número do bloco. Outros parâmetros, como por exemplo a função que será utilizada para calcular o limiar a partir do bloco de cada *pixel* também pode ser utilizada, assim como outros em SCKIT-IMAGE (2017b). Em seguida, a imagem é binarizada aplicando o limiar calculado em cada *pixel* na imagem em tons de cinza, e por fim o array de pixels é convertido para *uint8*, o que retorna um array de pixels com valores entre 0 e 1, sendo necessário multiplicar o resultado por 255.

Código 5 – Aplicação da Técnica de Binarização em uma imagem.

```
def techniqueB(image):
    # Transform PIL image to grayscale array
    gray = ImageOps.grayscale(image)
    gray_arr = np.asarray(gray)

    # Dynamically calculate neighbours block size
    block = block_size(gray_arr)
```

```
    # Apply local thresh on image
    local_thresh = threshold_local(gray_arr, block, offset=10)
    binary = gray > local_thresh
    binary = binary.astype("uint8") * 255

    return binary
```

Fonte: desenvolvido pelo autor.

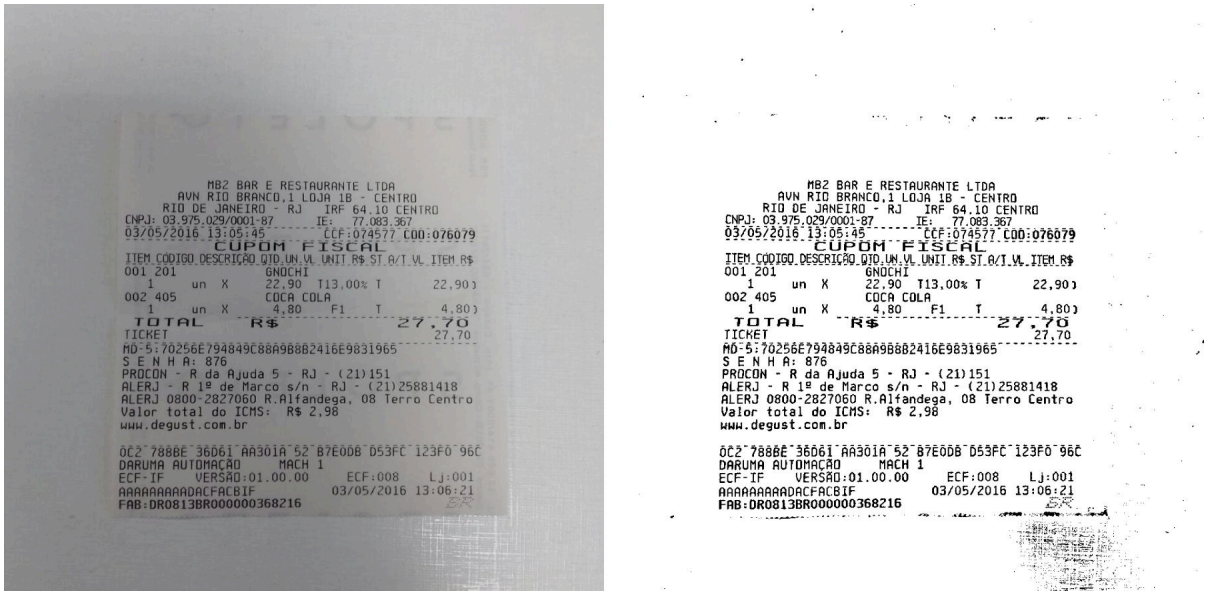
Código 6 – Cálculo do número de blocos para limiarização local.

```
def block_size(image):
    if image.size <= 2073600.0:
        return 75
    elif image.size <= 4915200.0:
        return 95
    elif image.size <= 8856576.0:
        return 131
    else:
        return 151
```

Fonte: desenvolvido pelo autor.

É possível ver um exemplo da aplicação da Técnica de Binarização na Figura 28, onde na esquerda é a imagem original e no lado direito o resultado da técnica. Mesmo com a presença de sombra na imagem original, a aplicação da limiarização local obteve um bom resultado. De um modo geral, o resultado da técnica foi uma imagem com apenas o texto do cupom fiscal, seu fundo perdeu destaque pois foi normalizado em branco e é possível visualizar facilmente o texto em preto. Os resultados da aplicação dessa técnica nos clusters será demonstrado na seção de resultados obtidos, sendo feita uma análise comparativa com as outras técnicas.

Figura 28 - Demonstração da aplicação da Técnica de Binarização em uma imagem. Na esquerda Imagem original, e na direita o resultado da técnica.



Fonte: desenvolvido pelo autor.

3.2.3 Técnica de Compressão

A terceira e última técnica desenvolvida apresenta um funcionamento um pouco diferente da técnicas anteriores, ao invés de aplicar métodos de filtro ou realce na imagem, é explorado a compressão de dados. De acordo com Filho e Neto (1999), o termo compressão de dados se refere ao processo de redução da quantidade de dados exigidos para representar uma certa quantidade de informação. Na compressão de imagens digitais, a redundância de dados é um aspecto essencial. Existem três redundâncias básicas de dados podem ser identificadas e exploradas: redundância de codificação, redundância interpixel, e redundância psicovisual. Para efetivamente obter a compressão de dados, uma ou mais dessas redundâncias devem ser reduzidas ou eliminadas (FILHO; NETO, 1999).

Nos modelos de compressão de imagens, um codificador parte de uma imagem de entrada $f(x,\ y)$ e cria um conjunto de símbolos que serão interpretados no futuro. O decodificador recebe esses símbolos e produz uma imagem de saída reconstruída $f'(x,\ y)$. Se a imagem de saída, $f'(x,\ y)$, for uma réplica exata de $f(x,\ y)$ então a compressão é dita livre de perdas ou imune a erros. Caso contrário, se a imagem de saída for diferente, significa que houve perda de informação, ou seja, é uma compressão com perdas (FILHO; NETO, 1999). Nesta técnica, é utilizado a compressão *JPEG* que é classificada como uma compressão com perdas de informação.

Apesar de existirem implementações para compressão sem perdas, o *JPEG* é fundamentalmente um método de compressão com perdas baseado na DCT (Transformada Discreta de Cossenos). O algoritmo explora as características do olho humano, pelo fato de que variações de cor são menos perceptíveis que variações de brilho. O grau de perda pode ser variado ajustando-se parâmetros de compressão, geralmente é selecionado um nível de qualidade que costuma ir de 0 a 100 dependendo da implementação do compressor. Segundo Filho e Neto (1999), o *JPEG* tem quatro modos de operação:

- **sequencial:** a imagem é codificada em uma única varredura (da esquerda para a direita, de alto a baixo);
- **progressiva:** a imagem é codificada em múltiplas varreduras, aumentando a qualidade e resolução a cada nova varredura;
- **hierárquica:** a imagem é codificada em múltiplas resoluções;
- **sem perda.**

A biblioteca utilizada para aplicar a compressão *JPEG* foi a versão 3.3.0 do *OpenCV* para python. No Código 7 é possível ver a implementação da Técnica de Compressão, na qual é uma função que recebe uma *url* de onde está armazenada a imagem, e retorna a imagem resultado codificada em bytes.

A primeira etapa do algoritmo é obter o conteúdo da *url* da imagem utilizando a função *url_to_image* da biblioteca *imutils*, que contém diversas funções auxiliares do *OpenCV*. Após obter a imagem no formato do *OpenCV*, um array de pixels, a próxima etapa é aplicar a função *imencode* que aplica algum formato de compressão. A função *imencode* recebe como primeiro parâmetro qual o formato que será utilizado na compressão, como segundo a imagem que será comprimida e como terceiro parâmetro uma série de opções que serão utilizadas na aplicação do algoritmo compressão. Todos os formatos disponíveis para compressão com seus respectivos parâmetros podem ser visualizados em OPENCV (2017).

Como é possível ver no Código 7, o formato escolhido foi o *JPEG*, utilizando a *flag IMWRITE_JPEG_QUALITY* como parâmetro de nível de qualidade na compressão. Selecionar o nível em 1 significa ter maior perda de informação com a compressão, e escolher o valor máximo de 100 representa comprimir na maior qualidade possível, ou seja, com poucas perdas perceptíveis. Por fim, o resultado em *bytes* é convertido em *string* para que possa ser salvo em um arquivo ou mandado direto para o serviço de *OCR*.

Código 7 – Extração do CNPJ do texto de uma imagem de cupom fiscal.

```
def techniqueC(url):
```

```
image = imutils.url_to_image(url)

encoded = cv2.imencode('.jpeg', image, [cv2.IMWRITE_JPEG_QUALITY, 100])

return encoded[1].tostring()
```

Fonte: desenvolvido pelo autor.

O objetivo dessa técnica é utilizar a compressão não para diminuir o tamanho do arquivo final da imagem, mas na verdade o contrário, aumentando o seu tamanho. Como todas as imagens do conjunto de dados são de câmeras de *smartphones*, em algum momento já foi aplicada alguma compressão *JPEG*, ou seja, seu tamanho em *bytes* já está reduzido. Portanto, o algoritmo recebe uma imagem com um tamanho de N *bytes*, e aplica a compressão *JPEG* com parâmetro de qualidade máximo, retornando uma imagem comprimida com um tamanho $N' > N$.

Para exemplificar, vamos utilizar a seguinte url como entrada do algoritmo da Técnica de Compressão. Realizando o download da imagem, temos um arquivo *JPEG* com o tamanho de 413 kB. Utilizando a primeira linha do Código 6, ou seja, convertendo a url da imagem para um um array de pixels, obtemos um total de 1080x1920 *pixels*, em que cada pixel é um array de 3 valores *uint8* representando os canais de cores. Portanto, são necessários 3 bytes para armazenar cada *pixel*, dando um total de 6.220.800 *bytes*. Caso essa imagem for salva em um formato sem perdas *bmp* ou *raw*, seria necessário aproximadamente 6.2 MB para armazenar a imagem em um arquivo. Porém, aplicando a compressão *JPEG* com nível de qualidade em 100, é obtido uma imagem com 1.104.386 *bytes*, isto é, aproximadamente 1,1 MB armazenada em um arquivo.

Mesmo parecendo contra intuitivo utilizar a compressão de imagens para aumentar o tamanho da imagem final, a aplicação dessa técnica no conjunto de imagens mostrou resultados positivos em relação à extração de dados via *OCR*. No capítulo 4, são apresentados todos os resultados da aplicação dessa técnica nos *clusters*, comparando com a aplicação das Técnicas de Realces e Binarização.

3.3 APLICAÇÃO DO *OCR*

Dispondo das imagens classificadas em *clusters* e as técnicas de PDI aplicadas, as duas últimas etapas da arquitetura desenvolvida constituem em enviar as imagens para o serviço de *OCR* e extrair as informações dos resultados. O serviço de *OCR* utilizado foi o

Google Cloud Vision API, uma aplicação que permite aos desenvolvedores entender o conteúdo de uma imagem por meio do encapsulamento de poderosos modelos de aprendizado de máquina em uma *API REST* (GOOGLE, 2017a). Também é necessário interpretar o resultado obtido pela Google Vision para extrair os valores desejados do cupom fiscal, sendo eles o CNPJ, data e hora da venda e o valor total da compra. Após isso, é possível verificar se o valor extraído é igual ao valor original que está no cupom fiscal, permitindo uma análise qualitativa dos resultados do *OCR*.

3.3.1 *Google Cloud Vision API*

O *Google Vision* é uma aplicação em formato de API capaz de analisar e identificar diversas propriedades de uma imagem através de modelos de aprendizado de máquina. Entre os diversos recursos do *Google Vision*, os principais são: Detecção Facial, Atributos de imagens, Detecção na *Web*, Detecção de pontos de referência e Reconhecimento óptico de caracteres. O recurso utilizado no desenvolvimento foi o reconhecimento óptico de caracteres (*OCR*), que extrai textos de uma imagem com compatibilidade para um extenso grupo de idiomas, além do suporte a identificação automática de idioma. Cada um desses recursos tem um preço pelo seu uso, sendo que a detecção de texto tem um custo de US$ 1,50 para cada 1000 imagens processadas.

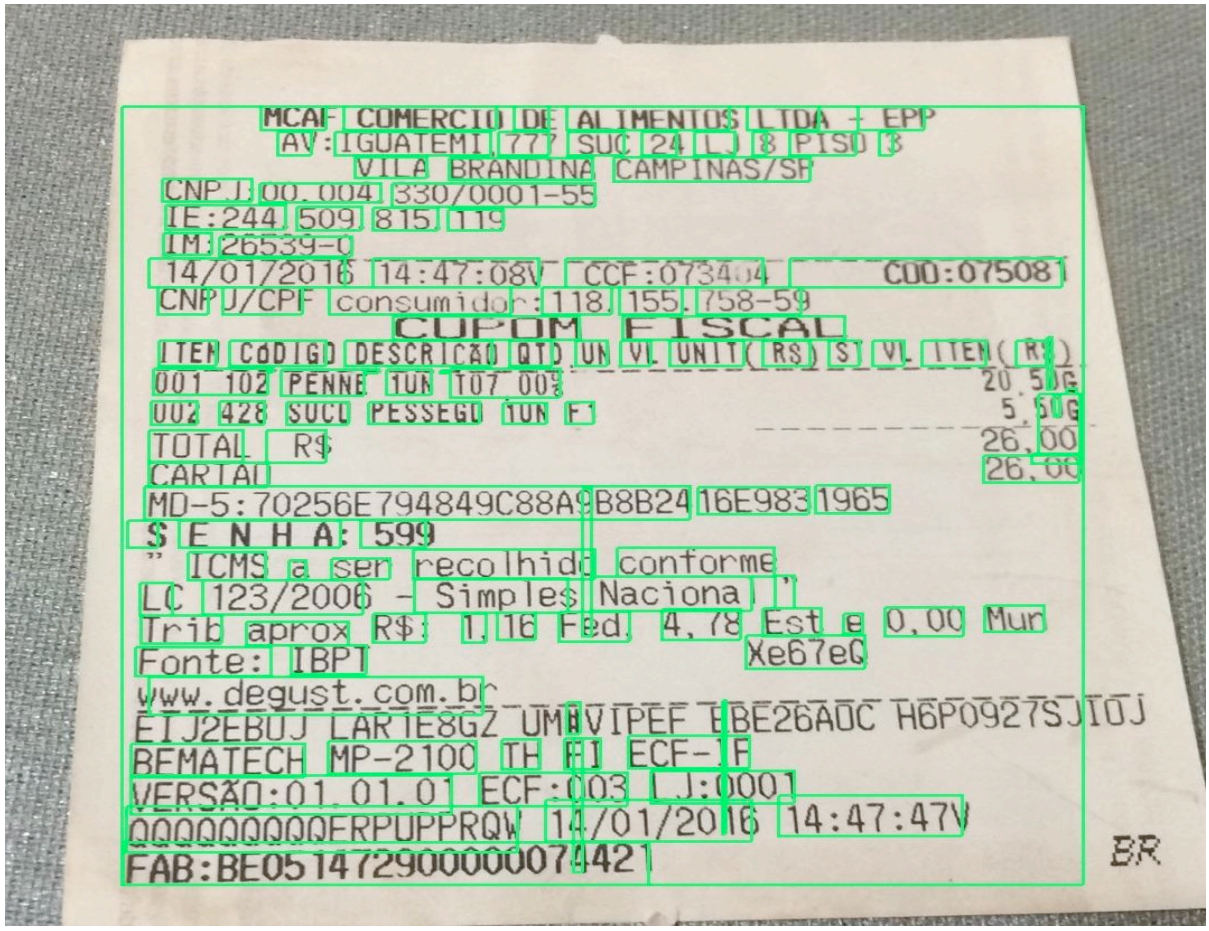
Para utilizar os serviços do *Google Cloud Vision* é necessário primeiro cadastrar uma conta no *Google Cloud* e habilitar o serviço de API. Após esse passo, deve ser gerado uma chave de acesso para que seja possível realizar a autenticação com o serviço *Google Cloud Vision*, na documentação contém um passo-a-passo para essa configuração (GOOGLE, 2017b) . A comunicação com o serviço pode ser feita através de uma requisição *HTTP*, porém existem diversas implementações de abstrações para realizar essa comunicação de forma mais simples, sendo em python a utilizada na proposta desenvolvida.

O texto de uma imagem é obtido pelo Google Vision através da biblioteca *google-cloud-python*. Essa biblioteca contém a implementação das abstrações para utilizar todos os serviços do *Google Cloud*, sendo implementada pela própria equipe da *Google* e disponibilizada em código aberto (GITHUB, 2017) . A primeira etapa é fazer a autenticação do cliente utilizando a função *Client*, que procura na variável de ambiente *GOOGLE_APPLICATION_CREDENTIALS* o caminho do arquivo que armazena as chaves privadas para comunicação com o servidor. Após a autenticação, é necessário indicar a imagem que será utilizada por meio da função *image*, podendo receber pelo parâmetro

source_uri a *url* da imagem, o conteúdo em bytes por *content*, ou também o caminho relativo para o arquivo através do parâmetro *filename.* Por fim, é feito a requisição para o serviço de *OCR* com a chamada de função *detect_text*, retornando uma lista de objetos com os blocos de texto encontrados.

Na Figura 29, é possível visualizar a lista de blocos do retorno do *OCR* em uma imagem de cupom fiscal. Os polígonos em verde representam os blocos encontrados com seus respectivos textos. No entanto, nem todos os blocos de texto são encontrados de forma descritiva, ou seja, campo de um lado e em seguida seu respectivo valor em texto. Em vista disso, não é uma tarefa trivial obter os campos de CNPJ, hora da venda e o total da compra. Nas próximas seções serão apresentadas as heurísticas para extrair esses valores do retorno do *OCR*.

Figura 29 - Retorno da aplicação do *OCR* pelo *Google Vision*. Cada polígono em verde representa um bloco de texto encontrado.



Fonte: desenvolvido pelo autor.

3.3.2 Extração do Campo CNPJ

O valor de CNPJ presente em um cupom fiscal é uma informação sobre o estabelecimento que foi emitido o cupom. De acordo com a Receita Federal (RECEITA FEDERAL, 2017a), o Cadastro Nacional de Pessoa Jurídica (CNPJ) compreende as informações cadastrais das entidades de interesse das administrações tributárias da União, dos Estados, do Distrito Federal e dos Municípios. A partir do CNPJ, é possível fazer uma consulta para derivar diversas outras informações do estabelecimento, como por exemplo o nome, CEP, atividade econômica e outros dados que podem ser consultados na Receita Federal (RECEITA FEDERAL, 2017b).

Para extrair o campo de CNPJ, voltamos para a Figura 29 onde estão todos os blocos de texto encontrados pelo *Google Vision*. O maior bloco da Figura corresponde ao primeiro objeto retornado pela aplicação do *OCR*. Todos os outros blocos encontrados estão contidos neste bloco maior, e seu respectivo texto é a junção de todos os textos da esquerda para direita e de cima para baixo. Portanto, através do maior bloco é obtido todo o texto contido no cupom fiscal, possibilitando o uso de uma expressão regular para extrair o campo e o valor do CNPJ.

Segundo Ramos, Neto e Vega (2009), expressão regular é uma notação alternativa utilizada para representar a classe linguagens mais simples que se conhece de acordo com a Hierarquia de Chomsky, ou seja, as linguagens regulares. Pelo fato de que o texto do cupom fiscal é sempre um texto finito, ou seja, uma linguagem regular, foram utilizadas expressões regulares para extrair o CNPJ. No Código 8 é possível ver a implementação da função *extract,* que aplica duas expressões regulares distintas no texto do cupom fiscal.

Código 8 – Extração do CNPJ do texto de uma imagem de cupom fiscal.

```
from brazilnum.cnpj import validate_cnpj, format_cnpj

SUFIXES = ['CNPJ', 'CPJ', 'CNP', 'CNJ', 'PJ', 'NPJ']
JOINED = [re.compile(sufix + "([0-9]{14})") for sufix in SUFIXES]
NUMBER_PATTERN = r"([0-9]{2}\.[0-9]{3}\.[0-9]{3}[/][0-9]{4}-[0-9]{2})"

def extract(text):
    content = number_pattern(text)
    if not validate_cnpj(content):
        content = self.cnpj_pattern(text)

    return format_cnpj(content)

def number_pattern(text):
```

```
    found = re.search(NUMBER_PATTERN, content)
    return found.group(1)

def cnpj_pattern(text):
    founds = [re.search(pattern, content) for pattern in JOINED]
    return founds[0].group(1)
```

Fonte: desenvolvido pelo autor.

A heurística da função *extract* é primeiro aplicar uma expressão regular que procura pelo padrão de números do CNPJ, ou seja, um padrão dado por *'99.999.999/9999-99'* onde o 9 representa um número de 0 a 9. Após isso, uma validação é feita com o valor encontrando para verificar se foi encontrado de fato um CNPJ válido. Essa validação é feita através da biblioteca *brazilnum*, que contém diversas validações e formatações numéricas de padrões brasileiros. Caso esse valor não seja válido, uma outra expressão regular é aplicada para tentar extrair novamente o CNPJ. A outra expressão procura por qualquer um dos possíveis sufixos especificados no início do Código 8, seguido de uma sequência de 14 valores numéricos. Após o valor ser encontrado por alguns dos sufixos, é feito uma nova validação para verificar se ele também é válido, retornando o valor formatado no padrão de CNPJ.

3.3.3 Extração do Campo Data e Hora da Venda

A data e hora da venda é uma informação que representa o exato momento em que foi efetuada a compra no estabelecimento. Esse campo contém o dia, mês e ano, assim como também a hora e os minutos do fechamento da compra. Sua importância é dada principalmente pelo fato de que outras informações do cupom estão correlacionadas com o tempo, como por exemplo o número de produtos vendidos e o valor da compra.

Semelhante ao campo de CNPJ, a hora da venda também tem um formato característico, facilitando sua extração via expressão regular. Este formato é sempre dado pelo padrão *'99/99/9999 99:99'*, no qual representa a data seguida das horas e minutos. Porém, para aumentar a acurácia da expressão regular, logo antes de ser aplicada o texto é traduzido para um novo texto com apenas números e os caracteres especiais que possam estar contidos no formato de data. Por fim, a data encontrada em formato *string* é enviada para uma função que transforma para o formato de data padrão do *python*, possibilitando comparações e operações com outras datas na análise comparativa.

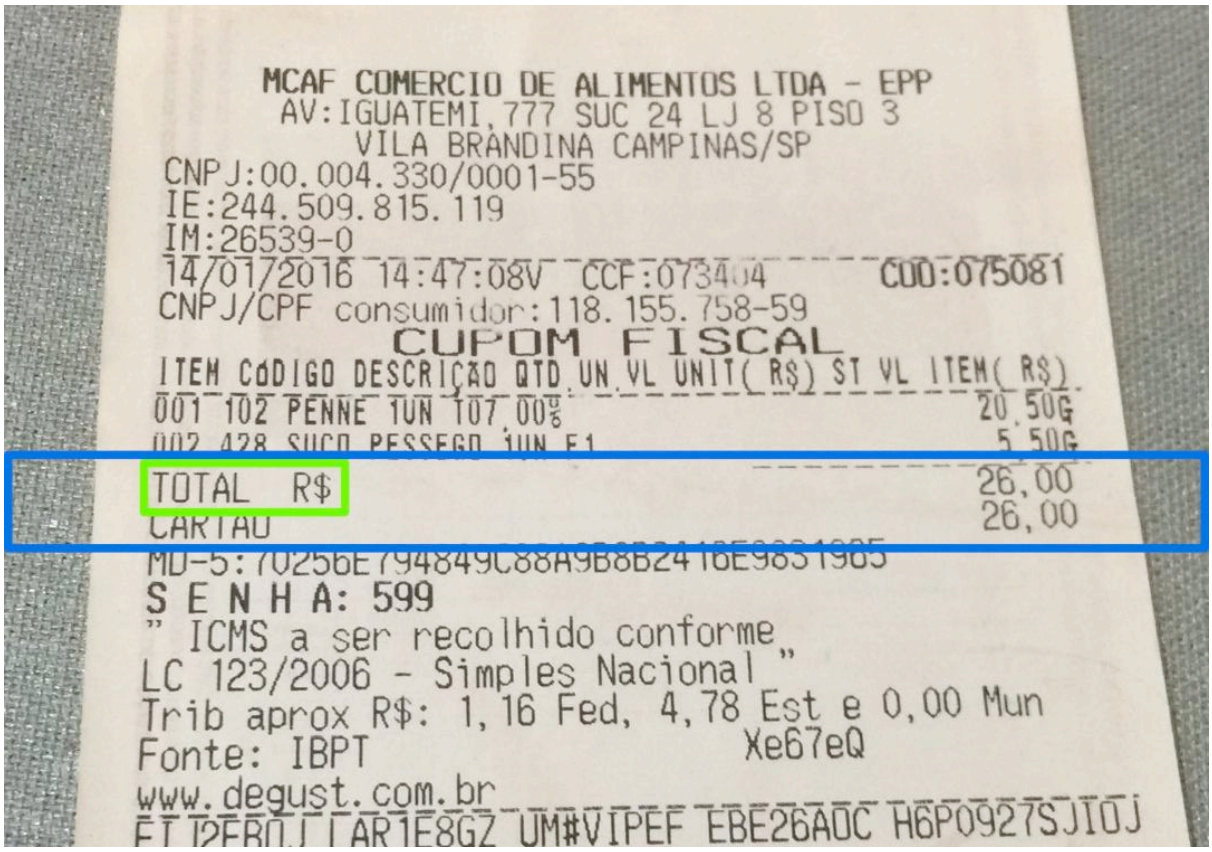
3.3.4 Extração do Campo Valor Total da Compra

O valor total da compra representa o custo total que um determinado consumidor teve no estabelecimento. No cupom fiscal também está contido outros valores de custo, como por exemplo valores de impostos e o preço de cada produto da compra. Esses valores estão representados em um formato numérico *'9,99'*, onde do lado esquerdo pode conter um ou mais valores de unidade e do lado direito sempre dois valores numéricos representando os centavos. Diferente da extração de CNPJ e data, não é um bom método buscar o valor total em todo texto do cupom fiscal, tendo em vista que o texto pode conter muitos valores nesse mesmo formato, porém referentes a produtos ou impostos.

Portanto, a estratégia utilizada foi usar os blocos encontrados para buscar pelo valor total em uma área delimitada dentro do cupom fiscal, aumentando a probabilidade de encontrar um valor que seja de fato o valor total. Isto é, primeiro é utilizado uma expressão regular para buscar na lista de objetos o bloco que contém o campo do valor total. Após encontrar esse bloco, é determinado um limite inferior e superior a partir da posição desse bloco para buscar o valor total. Como pode se observar na Figura 30, o bloco em verde representa o campo de valor total encontrado, já o bloco em azul a área calculada que será utilizada para buscar o valor total, que neste caso é *26,00*.

De acordo com o Código 8, a função *extract* recebe por parâmetro a lista dos blocos de texto e retorna o valor total encontrado. A primeira etapa do algoritmo é buscar o bloco referente ao campo de valor total. Isto é feito através da função *find_total_box*, que recebe por parâmetro a lista de blocos juntamente com a expressão regular *FIELD_REGEX* que será utilizada na busca. Caso esse bloco não seja encontrado o algoritmo retorna uma *string* vazia, o que representa não ter encontrado o campo de valor total. Após isso, a função *find_near_boxes* realize o cálculo dos limites de busca a partir do bloco encontrado, e logo após retorna todos os blocos contidos nessa área. O cálculo dos limites de busca é feito com base de que provavelmente o bloco do valor total estará ao lado ou um pouco abaixo do campo de total.

Figura 30 - Técnica utilizada para extrair o valor total do cupom fiscal. O bloco em verde representa o campo do valor total, já o bloco em azul a área de busca do valor.



Fonte: desenvolvido pelo autor.

Código 8 – Extração do Valor Total dos blocos de texto de uma imagem de cupom fiscal.

```
FIELD_REGEX =
r"(TOTAL|TOTALR(S|$)|R(S|$)TOTAL|VALORPAGO|TOTA|OTAL)"
TOTAL_VALUE_REGEX = r"([1-9][0-9]*)[,.]([0-9]?[0-9]?)"

def extract(boxes):
    total_box = find_total_box(boxes, FIELD_REGEX)
    if not total_box:
        return ''
    near_boxes = find_near_boxes(total_box, boxes)
    total_values = total_value_boxes(near_boxes, TOTAL_VALUE_REGEX)

    return max(total_values)
```

Fonte: desenvolvido pelo autor.

Com os blocos encontrados dentro da área de busca, é necessário filtrar apenas os blocos que contêm o texto no formato do valor total. Essa seleção é feita na função *total_value_boxes*, aplicando a expressão regular *TOTAL_VALUE_REGEX* para filtrar apenas

valores válidos. Por fim, entre os valores selecionados pela expressão regular é retornado o com maior valor numérico entre eles. Apesar da estratégia aplicada demonstrar bons resultados, existem alguns casos onde o cupom fiscal possui também o valor pago em dinheiro, que em muitos casos é maior que o valor total da compra. Isso faz com que o método retorne o valor incorreto, porém é um caso exclusivo que também pode ser resolvido em uma implementação futura.

4 RESULTADOS

O objetivo deste capítulo é apresentar os resultados obtidos pela arquitetura proposta desenvolvida. Esta análise é feita comparando os valores extraídos pelo *OCR* com os valores que realmente estão na foto do cupom fiscal. Cada elemento do conjunto de dados contém a *url* da imagem juntamente com os valores de CNPJ, data e valor total da compra retirados manualmente de cada foto. Os resultados são apresentados em taxa de acerto, ou seja, a razão entre o número de acertos do *OCR* pela quantidade de imagens. Para exemplificar, supondo que o número de elementos do conjunto de dados é de 1000 e a quantidade de acertos do *OCR* para data e hora foi de 650, a taxa de acerto do *OCR* para data e hora é dita de 65%.

Na análise comparativa entre o valor extraído pelo *OCR* e o valor na foto do cupom, foram desenvolvidos métodos distintos de comparação para cada um dos três campos. Isto foi feito para melhorar a taxa de acerto para alguns campos em determinados casos. O comparador do CNPJ apenas verifica a igualdade entre os dois valores. Já o comparador de data e hora verifica se a data extraída está em um intervalo de 2 minutos com a data do cupom fiscal. O cupom fiscal pode conter dois valores de data e hora, o primeiro referente a compra e o segundo ao tempo em que foi impresso. Devido ao fato de que ambas as datas geralmente contêm uma diferença de no máximo 2 minutos, ambas são consideradas válidas para análise. No caso do valor total, a comparação é feita ignorando os valores decimais, pois algumas vezes esse valor é lido em um bloco distinto, dificultando sua extração por completo.

Os resultados são apresentados em etapas, demonstrando a eficiência de cada técnica aplicada no conjunto de imagens e também em cada *cluster*. Na apresentação dos resultados por *cluster*, são utilizados os gerados pelos algoritmos *K-means* e *Spectral Clustering*. Por fim, também são apresentados os resultados das técnicas selecionadas por *clusters*, isto é, a aplicação de técnicas distintas em cada *cluster* com objetivo de melhorar a taxa de acerto. Essa análise tomou como base a arquitetura proposta, onde cada técnica é selecionada para

um ou mais *clusters*. Para fins de análise comparativa dos resultados, o critério de seleção de uma técnica para um determinado *cluster* é pelo qual se obteve o melhor resultado.

4.1 ANÁLISE DAS IMAGENS SEM ALTERAÇÕES

Antes de demonstrar o resultado das técnicas desenvolvidas, primeiramente foi analisado os resultados obtidos pelo envio da imagem diretamente para o serviço de *OCR*, ou seja, sem nenhuma alteração. Isso pôde ser feito através do Código 7, utilizando o parâmetro *source_uri* para enviar a *url* da imagem para o *Google Vision*.

No Quadro 7 é possível visualizar os resultados da aplicação do *OCR* para cada campo extraído. A primeira linha do Quadro denota a taxa de acerto dos campos em todo o conjunto de dados, isto é, as 3200 imagens de cupons fiscais. As demais linhas representam a análise separada pelos *clusters* gerados pelo algoritmo *K-means*, visível na Figura 24.

Quadro 7 - Resultados da aplicação do *OCR* nas imagens sem alterações, segmentado pelos *clusters* gerados no algoritmo *K-means*.

Cluster	CNPJ	Valor Total	Data e Hora	Quantidade
Sem *cluster*	75,78125%	55,03125%	69,21875%	3200
Cluster 0	78,08012%	60,84656%	71,12622%	1323
Cluster 1	74,67043%	42,84369%	68,36158%	1062
Cluster 2	88,09523%	61,90476%	76,19047%	42
Cluster 3	81,25000%	71,15384%	75,00000%	208
Cluster 4	69,55752%	57,87610%	63,71681%	565

Fonte: desenvolvido pelo autor.

Interpretando os resultados do Quadro, fica claro a existência de uma variação na taxa de acerto de um *cluster* para o outro. Isso ocorre pois cada *cluster* possui imagens com características diferentes.

As imagens dos *clusters* 2 e 3 são as que possuem alta resolução de *pixels*, o que pode ser utilizado como justificativa para os bons resultados em relação aos outros *clusters*. Porém, já as imagens do *cluster* 4 não demonstraram um bom resultado nas taxas de acerto, tendo em vista que são imagens que contêm de modo geral um baixo número de *pixels* e também um baixo nível de contraste. Em relação aos demais *clusters*, 0 e 1, as taxas de

acertos foram próximas ao resultado em todo conjunto de dados, exceto pelo *cluster* 1 que não obteve uma boa taxa de extração de valor total.

O Quadro 8 mostra os resultados separados pelos *clusters* resultantes da aplicação do *Spectral Clustering*, podendo ser visualizado na Figura 26. De um modo geral, todos os *clusters* obtiveram um bom resultado na extração dos três campos, com exceção do *cluster* 0 que obteve resultados inferiores. O *cluster* 4 também não conseguiu extrair o campo valor total de modo eficiente, do mesmo jeito que o *cluster 1* do algoritmo *K-means*.

Quadro 8 - Resultados da aplicação do *OCR* nas imagens sem alterações, segmentado pelos *clusters* gerados no algoritmo *Spectral Clustering*.

Cluster	CNPJ	Valor Total	Data e Hora	Quantidade
Sem *cluster*	75,78125%	55,03125%	69,21875%	3200
Cluster 0	69,08396%	52,29007%	62,21374%	262
Cluster 1	78,14166%	54,60776%	72,04874%	1313
Cluster 2	87,80487%	60,97560%	75,60975%	41
Cluster 3	74,13366%	69,67821%	65,96534%	808
Cluster 4	75,12886%	41,10824%	69,84536%	776

Fonte: desenvolvido pelo autor.

4.2 ANÁLISE DAS TÉCNICAS DESENVOLVIDAS

Esta seção tem como objetivo apresentar os resultados da aplicação das técnicas desenvolvidas na arquitetura proposta, realizando uma análise comparativa entre elas. Por fim, será apresentado qual a melhor técnica que pode ser aplicada em cada *cluster*, junto com o impacto disso na taxa de acerto dos resultados finais.

4.2.1 Técnica de Realces

Como descrito no capítulo anterior, a Técnica de Realces aplica uma série de realces em uma determinada imagem. Os realces aplicados são ajustes de nitidez, brilho e contraste, e por fim é modificado o balanço de cor. Na Figura 27 é possível ver um exemplo da aplicação dessa técnica. Os dados do Quadro 9 demonstram os resultados obtidos pela aplicação da

Técnica de Realces com o algoritmo *K-means*, possibilitando uma análise comparativa com os resultados do *OCR* sem alterações na imagem, ou seja, referentes ao Quadro 7.

Quadro 9 - Resultados da aplicação do *OCR* nas imagens com a Técnica de Realces, segmentado pelos *clusters* gerados no algoritmo *K-means*.

Cluster	CNPJ	Valor Total	Data e Hora	Quantidade
Sem *cluster*	71,06250%	56,03125%	69,03125%	3200
Cluster 0	73,31821%	59,18367%	70,14361%	1323
Cluster 1	67,23163%	48,11676%	68,17325%	1062
Cluster 2	88,09523%	50,00000%	61,90476%	42
Cluster 3	81,73076%	70,67307%	75,00000%	208
Cluster 4	67,78761%	58,58407%	66,37168%	565

Fonte: desenvolvido pelo autor.

Analisando primeiro os resultados em todo o conjunto de dados, isto é, a primeira linha do Quadro 9, fica evidente que a taxa de acerto do CNPJ e da data e hora foi inferior, quando comparada com o do Quadro 7. Entretanto, a taxa de acerto do valor total foi um pouco melhor, 1% a mais que a aplicação sem alterar as imagens. Ao realizar uma análise por *clusters*, é possível ver melhor em quais conjuntos de imagens ocorreu uma melhora dos resultados, assim como também os que pioraram.

Tanto o *cluster* 0 quanto o *cluster* 2 tiveram uma taxa de acerto inferior em relação aos resultados sem aplicação da técnica, portanto não é uma boa estratégica aplicar a Técnica de Realces nesses dois conjuntos de imagens. Os resultados do *cluster* 3 não apresentaram grandes diferenças, apesar de ter sido o único que obteve alguma melhora na extração do CNPJ. Já o *cluster* 1 obteve resultados diferentes, pois a taxa de acerto do valor total obteve uma melhora significativa de aproximadamente 6%, porém ao mesmo tempo a aplicação da técnica piorou a extração do CNPJ. Caso a aplicação que utilize a arquitetura proposta tenha preferência pela extração de valor total, utilizar a Técnica de Realces no *cluster* 1 presume-se que seja uma boa alternativa.

Os melhores resultados da aplicação dessa técnica podem ser vistos no *cluster* 4, onde apesar de ter uma pequena perda na extração de CNPJ, tanto a taxa de acerto de valor total quanto a de data e hora obtiveram melhoras em relação aos resultados do Quadro 7. Portanto, aplicar a Técnica de Realces nas imagens classificadas no *cluster* 4 pelo algoritmo

K-means, ou seja, baixa resolução de *pixels* e baixo nível de contraste, pode ser uma boa estratégia para aumentar a taxa de acerto na extração de valor total e data e hora.

No Quadro 10 também são apresentados os resultados da aplicação da técnica, com a diferença que os *clusters* são os gerados pelo algoritmo *Spectral Clustering*.

Quadro 10 - Resultados da aplicação do *OCR* nas imagens com a Técnica de Realces, segmentado pelos *clusters* gerados no algoritmo *Spectral Clustering*.

Cluster	CNPJ	Valor Total	Data e Hora	Quantidade
Sem *cluster*	71,06250%	56,03125%	69,03125%	3200
Cluster 0	67,93893%	52,67175%	63,74045%	262
Cluster 1	74,71439%	56,35948%	71,59177%	1313
Cluster 2	87,80487%	48,78048%	60,97560%	41
Cluster 3	69,30693%	65,71782%	66,95544%	808
Cluster 4	66,88144%	46,90721%	69,07216%	776

Fonte: desenvolvido pelo autor.

De um modo geral, não houve muita diferença em relação aos resultados do Quadro 9, ou seja, do algoritmo *K-means*. Na maioria dos casos, a taxa de acerto de algum campo é melhorada, porém do mesmo jeito a taxa de outro campo é reduzida. O *cluster* 1 e 4 apresentaram resultados interessantes em relação a extração do valor total, contudo a taxa de acerto de CNPJ desses mesmos *clusters* teve um decréscimo mais significativo.

4.2.2 Técnica de Binarização

Esta técnica é responsável por aplicar a binarização no conjunto de imagens, definida a partir do cálculo de um limiar. Tanto a aplicação da binarização quanto o cálculo do limiar podem ser visualizados nos Códigos 5 e 6 respectivamente, assim como um exemplo da aplicação da técnica na Figura 28. O Quadro 11 demonstra os resultados da aplicação da Técnica de Binarização com os *clusters* do algoritmo *K-means*, esses resultados são analisados e comparados com os resultados das imagens sem alterações e com a Técnica de Realces, ou seja, Quadros 7 e 9, respectivamente.

Através dos resultados da aplicação da Técnica de Binarização em todo o conjunto de imagens, isto é, primeira linha do Quadro 11, é possível ver que assim como a Técnica de Realces, houve uma perda em relação a extração do CNPJ e data e hora. Entretanto, a taxa de

acerto no campo valor total foi consideravelmente superior comparando com os resultados dos Quadros 7 e 9.

Quadro 11 - Resultados da aplicação do *OCR* nas imagens com a Técnica de Binarização, segmentado pelos *clusters* gerados no algoritmo *K-means*.

Cluster	CNPJ	Valor Total	Data e Hora	Quantidade
Sem *cluster*	68,87500%	58,87500%	64,00000%	3200
Cluster 0	69,31216%	61,14890%	65,53287%	1323
Cluster 1	72,50470%	60,92278%	67,32580%	1062
Cluster 2	88,09523%	50,00000%	64,28571%	42
Cluster 3	78,84615%	62,98076%	65,38461%	208
Cluster 4	55,92920%	48,84955%	53,62831%	565

Fonte: desenvolvido pelo autor.

A partir dos resultados segmentados em *clusters*, observa-se também que, que praticamente todos os *clusters* obtiveram taxas de acerto inferiores nos campos CNPJ e data e hora, comparados com as imagens sem alterações e a Técnica de Realces. Em relação ao *cluster* 0, a taxa de acerto do campo valor total foi superior comparando com os resultados anteriores, porém foi inferior na extração de CNPJ e data e hora.

O principal resultado da aplicação da Técnica de Binarização foi em relação ao *cluster* 1, pois melhorou em aproximadamente 18% a extração do valor total comparado com as imagens sem alterações. Além disso, as perdas na extração de CNPJ e data e hora não foram tão significativas, sendo até melhor que a Técnica de Realces no CNPJ. Portanto, aplicar a Técnica de Binarização no *cluster* 1 gerado pelo algoritmo *K-means*, é de fato uma boa alternativa para melhorar a extração do valor total, sem perder muito nos outros campos.

Analisando os dados do Quadro 12, ou seja, aplicação da Técnica de Binarização utilizando os *clusters* do algoritmo *Spectral Clustering*, é possível perceber que dois *clusters* tiverem resultados positivos. Diferente do algoritmo *K-means*, onde apenas o *cluster* 1 apresentou bons resultados, com os *Spectral Clustering* os *clusters* 1 e 4 apresentaram boas taxas de acerto no campo valor total, sem muitas perdas no CNPJ e data e hora.

O *cluster* 4 teve um aumento de 20% em relação às imagens sem alterações, e aproximadamente 15% em relação a Técnica de Realces. Contudo, os demais *clusters* tiveram

resultados bem inferiores, não sendo uma boa alternativa utilizar a Técnica de Binarização nesses casos.

Quadro 12 - Resultados da aplicação do *OCR* nas imagens com a Técnica de Binarização, segmentado pelos *clusters* gerados no algoritmo *Spectral Clustering*.

Cluster	CNPJ	Valor Total	Data e Hora	Quantidade
Sem *cluster*	68,87500%	58,87500%	64,00000%	3200
Cluster 0	52,29000%	45,80152%	46,94656%	262
Cluster 1	74,40974%	60,92916%	69,23076%	1313
Cluster 2	87,80487%	48,78048%	63,41463%	41
Cluster 3	60,89108%	57,92079%	57,54950%	808
Cluster 4	72,42268%	61,34020%	67,65463%	776

Fonte: desenvolvido pelo autor.

4.2.3 Técnica de Compressão

O funcionamento da Técnica de Compressão é um pouco diferente das Técnicas de Realces e Binarização, pois ao invés de aplicar algum tipo de filtro na imagem, é explorado a compressão *JPEG*. Resumidamente, a técnica aplica uma nova compressão na imagem, porém utilizando um parâmetro de qualidade no maior nível possível. A implementação da técnica está no Código 6, onde o nível de qualidade na compressão *JPEG* é fixado no máximo, ou seja, em 100. A partir dos resultados da Técnica de Compressão, é feita uma análise comparativa com todas as outras técnicas descritas anteriormente, assim como também com as imagens sem alterações.

O Quadro 13 demonstra os resultados da aplicação da técnica com os *clusters* do algoritmo *K-means*. Realizando uma análise comparativa com as imagens sem alterações e também com as técnicas anteriores, a aplicação da Técnica de Compressão mostrou taxas de acerto relativamente superiores. Analisando a aplicação da técnica em todo o conjunto de imagens, a taxa de acerto dos três campos foi superior, com o valor total chegando a aproximadamente 13% a mais que na Técnica de Binarização. Além disso, foi a única técnica que conseguiu melhorar a taxa de acerto dos três campos simultaneamente, onde ambos ficaram com taxas de acerto superiores a 70%.

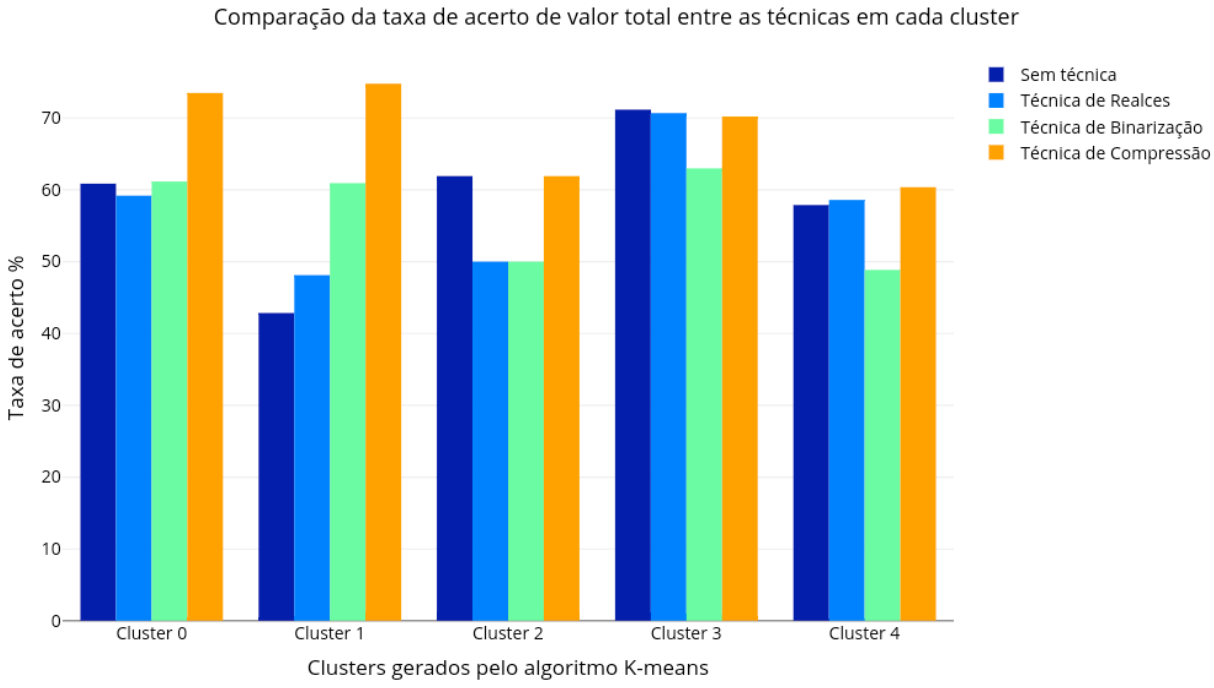
O Quadro 13 - Resultados da aplicação do *OCR* nas imagens com a Técnica de Compressão, segmentado pelos *clusters* gerados no algoritmo *K-means*.

Cluster	CNPJ	Valor Total	Data e Hora	Quantidade
Sem *cluster*	79,5%	71,21875%	72,3125%	3200
Cluster 0	80,87679%	73,46938%	72,26001%	1323
Cluster 1	81,63841%	74,76459%	74,67043%	1062
Cluster 2	88,09523%	61,90476%	69,04761%	42
Cluster 3	83,17307%	70,1923%	77,40384%	208
Cluster 4	70,26548%	60,35398%	66,37168%	565

Fonte: desenvolvido pelo autor.

Analisando os resultados da taxa de acerto do valor total no gráfico da Figura 31, na maior parte dos casos os *clusters* da Técnica de Compressão obtiveram resultados superiores em relação aos gerados pelas Técnicas de Realces e Binarização, e pelas imagens sem alterações. Com exceção do *cluster* 3, que obteve uma taxa de acerto de valor total um pouco inferior comparando com as imagens sem alterações, porém comparando com os demais o resultado foi igual ou superior.

Figura 31 – Gráfico de comparação da taxa de acerto de valor total entre as técnicas desenvolvidas. Resultados separados pelos clusters do algoritmo *K-means*.



Fonte: desenvolvido pelo autor.

Os resultados da aplicação da Técnica de Compressão com o algoritmo *Spectral Clustering* estão apresentados no Quadro 14. De um modo geral, alterar o algoritmo de "clusterização" com essa técnica não teve um efeito tão significativo, pelo fato de que seus resultados são positivos em praticamente todos os casos. Assim como no *K-means*, a taxa de acerto do campo data e hora foi inferior no *cluster* 2 comparado com as imagens sem alterações, porém com a diferença de que no *Spectral Clustering* a extração de valor total no *cluster* 3 também foi inferior.

Tendo em vista que os resultados apresentados estão avaliando a taxa de acerto do serviço externo *Google Vision*, não é uma tarefa trivial justificar o porquê dos resultados da Técnica de Compressão serem superiores. Além de não ser possível ver uma diferença visual na imagem com a aplicação da técnica, o serviço do *Google Vision* é um modelo de aprendizagem de máquina pelo qual não é possível ter acesso. Esse fator costuma ser conhecido como *caixa-preta*, onde não há informações sobre o que acontece com a imagem ao ser enviada para o *OCR*.

No entanto, como sabemos que é um modelo de aprendizado de máquina, é provável que o mesmo seja sensível ao conjunto de dados que foi utilizado para treino. Por esse motivo, alterar a compressão da imagem pode refletir em uma interpretação diferente no modelo usado pelo *Google Vision*. Lembrando que, como é uma *caixa-preta*, essas afirmações são apenas uma hipótese para justificar os bons resultados da Técnica de Compressão.

Quadro 14 - Resultados da aplicação do *OCR* nas imagens com a Técnica de Compressão, segmentado pelos *clusters* gerados no algoritmo *Spectral Clustering*.

Cluster	CNPJ	Valor Total	Data e Hora	Quantidade
Sem *cluster*	79,50000%	71,21875%	72,31250%	3200
Cluster 0	70,22900%	57,63358%	64,50381%	262
Cluster 1	83,01599%	73,64813%	76,16146%	1313
Cluster 2	87,80487%	60,97560%	68,29268%	41
Cluster 3	74,38118%	69,43069%	67,07920%	808
Cluster 4	81,57216%	74,09793%	74,09793%	776

Fonte: desenvolvido pelo autor.

4.2.4 Técnicas Selecionadas por *Cluster*

Nesta seção será demonstrado como seriam os resultados na arquitetura proposta, ou seja, selecionando as técnicas a serem aplicadas em cada *cluster*. O critério de seleção de qual técnica será utilizada em um *cluster* foi escolhido com base em qual *cluster* a determinada técnica teve as melhores taxas de acerto. Essa seleção foi feita para exemplificar os resultados da arquitetura, porém como posto na Figura 20, a escolha da técnica a ser aplicada deve ser feita antes do envio das imagens para o *OCR*.

Considerando os resultados relativamente superiores da Técnica de Compressão, a análise das técnicas selecionadas por *cluster* irá utilizar apenas os resultados das imagens sem alterações e das Técnicas de Realces e Binarização. Contudo, caso os resultados dessas técnicas em um determinado cluster tenham sido inferiores em relação às imagens sem alterações, não será aplicada nenhuma técnica neste *cluster*. O algoritmo de cluster estabelecido para apresentar os resultados foi o *K-means*, ou seja, os resultados serão referentes aos Quadros 7, 9 e 11.

No Quadro 15 é possível ver qual técnica foi selecionada para cada *cluster*, baseando-se no critério de melhor taxa de acerto. Para os *clusters* 0, 2 e 3 nenhuma técnica foi aplicada tendo em vista que ambas as Técnicas de Realces e Binarização obtiveram resultados inferiores em comparação as imagens sem alterações. No *cluster* 1 foi selecionado a aplicação da Técnica de Binarização, onde de fato obteve seu melhor resultado. Já o último *cluster*, número 4, a técnica escolhida foi a Técnica de Realces, na qual obteve uma considerável melhora na taxa de acerto do valor total e na data e hora.

Quadro 15 - Seleção das técnicas de PDI para os *clusters* do *K-means*, juntamente com seu respectivo resultado na aplicação do *OCR*.

Cluster	Técnica	CNPJ	Valor Total	Data e Hora	Quantidade
Cluster 0	Nenhuma	78,08012%	60,84656%	71,12622%	1323
Cluster 1	Binarização	72,50470%	60,92278%	67,32580%	1062
Cluster 2	Nenhuma	88,09523%	61,90476%	76,19047%	42
Cluster 3	Nenhuma	81,2500%	71,15384%	75,00000%	208
Cluster 4	Realces	67,78761%	58,58407%	66,37168%	565

Fonte: desenvolvido pelo autor.

Aplicando a Técnica de Binarização no cluster 1 e a Técnica de Realces no *cluster* 4, é possível obter os resultados do *OCR* em todo o conjunto de imagens. No Quadro 16 são apresentados esses resultados, nos quais foram obtidos através do Quadro 15. Realizando primeiro uma análise comparativa com as imagens sem alterações, ou seja, primeira linha do Quadro 7, houve um aumento significativo na extração do valor total, assim como pequena perda na taxa de acerto do CNPJ. Porém, como essa perda foi relativamente pequena em relação ao ganho na extração do valor total, pode-se considerar que os resultados foram melhores do que não aplicar nenhuma técnica nas imagens.

Comparando os resultados do Quadro 16 com a aplicação das Técnicas de Realces e Binarização em todo conjunto de imagens, isto é, primeira linha dos Quadros 9 e 11 respectivamente, os resultados foram superiores em todos os campos. Portanto, em relação ao escopo definido de comparação, selecionar técnicas por cluster apresentou o melhor resultado da análise.

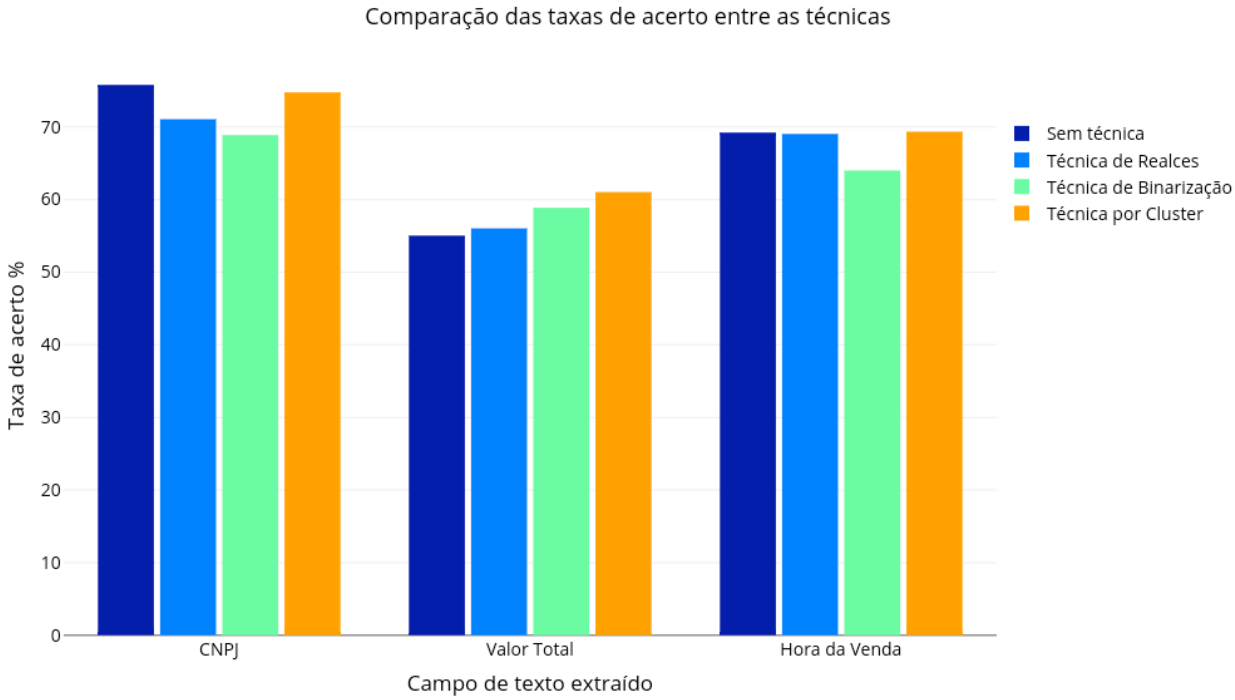
Quadro 16 - Resultado do *OCR* utilizando as técnicas selecionadas no Quadro 15.

CNPJ	Valor Total	Data e Hora	Quantidade
74,75000%	61,03125%	69,34375%	3200

Fonte: desenvolvido pelo autor.

Para visualizar melhor essa comparação, na Figura 32 é apresentado um gráfico de comparação entre as taxas de acerto da aplicação do *OCR* no conjunto de imagens. Cada grupo de barras representa um campo extraído, sendo que a barra roxa é o resultado das imagens sem aplicação de técnica, a azul a Técnica de Realce, a barra verde a Técnica de Binarização e por último a barra laranja demonstra a seleção de técnica por cluster. Por fim, esses resultados obtidos comprovam que a arquitetura proposta pode ser efetivamente uma boa estratégia para melhorar a extração de dados de imagens de cupons fiscais via aplicação de *OCR*.

Figura 32 – Gráfico de comparação das taxas de acerto de CNPJ, Valor Total e Hora da Venda dos resultados sem aplicar técnica, Técnicas de Realces e Binarização, e técnica selecionada por cluster.

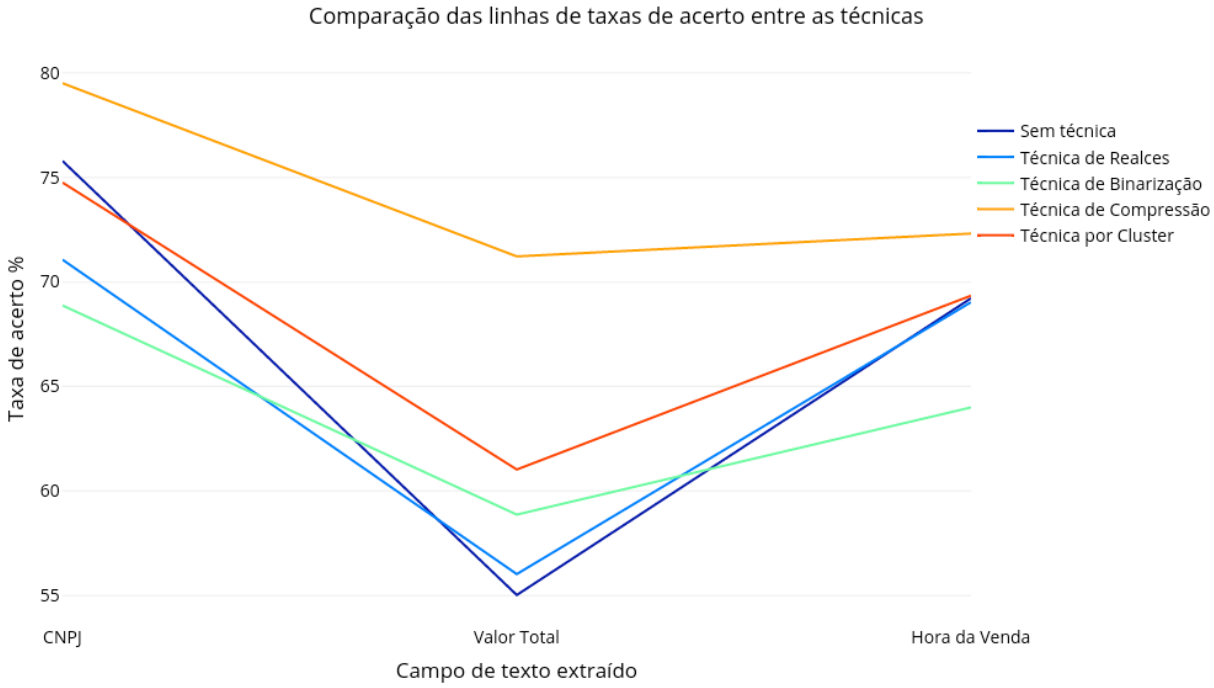


Fonte: desenvolvido pelo autor.

Finalizando a análise dos resultados, na Figura 33 é possível ver a linha de taxas de acerto para cada resultado obtido. Estas linhas representam a proximidade entre as taxas de acerto dos três campos extraídos. Exemplificando, em aplicações onde é necessário utilizar ambos os três campos, a qualidade na extração de dados via *OCR* será relativa a menor taxa de acerto obtida. Ou seja, visualizando a linha roxa da aplicação sem técnica, mesmo tendo uma boa taxa de acerto em CNPJ e data e hora, a baixa taxa no valor total faz com que ela não seja uma boa alternativa para esse tipo de aplicação.

Contudo, analisando as linhas laranja e amarela das Técnicas de Compressão e por Cluster, elas tendem a ser mais parecidas com uma reta perpendicular ao eixo da taxa de acerto, significando que as taxas de acerto entre os três campos são próximas. Concluindo, ao invés de apenas olhar para uma taxa alta em um determinado campo, é importante também analisar como uma determinada técnica obteve resultados em todos os campos extraídos, sendo mais eficiente nesses tipos de aplicações.

Figura 33 – Gráfico das linhas de taxas de acerto de CNPJ, Valor Total e Hora da Venda em todos os resultados obtidos.



Fonte: desenvolvido pelo autor.

5 CONSIDERAÇÕES FINAIS

Retomando ao objetivo geral proposto, isto é, analisar o impacto de técnicas de "clusterização", processamento de imagens e *OCR* na extração de textos de imagens de cupons fiscais, é possível concluir que o mesmo foi alcançado com êxito. Analisando também os objetivos específicos juntamente aos métodos, fica mais claro visualizar como foi atingido o objetivo geral.

O primeiro objetivo específico foi analisar o estado da arte em algoritmos de "clusterização", processamento de imagens e *OCR*. Esse objetivo foi amplamente abordado na parte teórica do trabalho, ou seja, nas fundamentações teóricas. A seção 2.1 aborda os fundamentos e o estado da arte em algoritmos de "clusterização", assim como os algoritmos clássicas. Na seção 2.2, é exemplificado os fundamentos de imagens digitais, assim como as técnicas mais utilizadas de processamento de imagens. Os princípios básicos das aplicações de *OCR* e também seus algoritmos clássicos são discutidos na seção 2.3.

Em análise do segundo objetivo, isto é, propor uma arquitetura para transformar imagens de cupons fiscais em texto através de análise em *cluster*, processamento de imagens e

OCR, é possível ver a arquitetura na Figura 20. No início do capítulo 3 está descrito a proposta de arquitetura, assim também como seu funcionamento ao decorrer das seções do capítulo. Os próximos três objetivos estão relacionados às etapas da arquitetura proposta, em outras palavras, aplicar análise em *cluster*, técnicas de PDI e por fim o *OCR*, nos quais também foram abordados no capítulo 3.

O terceiro objetivo proposto é especificar os requisitos e as técnicas para análise em cluster de imagens de cupons fiscais, pelo qual é abordado na seção 3.1. Tendo em vista que foi especificado tanto os atributos a serem utilizados na análise como os algoritmos aplicados, pode-se concluir que esse objetivo também foi alcançado. Os dois atributos selecionados para análise foram o número de *pixels* e o nível de contraste da imagem, já as três técnicas de "clusterização" foram o *K-means*, *DBSCAN* e o *Spectral Clustering*. Através dos resultados obtidos pelos algoritmos, também é conclusivo que utilizando os dois atributos selecionados, apenas o *K-means* e o *Spectral Clustering* demonstraram bons resultados.

A próxima etapa da arquitetura proposta é aplicar as técnicas de PDI nas imagens, na qual é especificada pelo quarto objetivo. Esse objetivo consiste em determinar o conjunto de técnicas a serem utilizadas, e de acordo com a seção 3.2, foram propostas três técnicas. A primeira técnica desenvolvida aplica uma sequência de realces nas imagens, sendo que um exemplo pode ser visto na Figura 27. No Código 5 é possível ver a implementação da segunda técnica, que aplica um filtro de binarização. Um exemplo de aplicação da técnica é posto na Figura 28. A Técnica de Compressão foi a última do conjunto de técnicas propostas, onde é explorado a compressão de imagens no formato *JPEG*. Portanto, podemos concluir que o objetivo de especificar as técnicas de PDI para imagens de cupons fiscais foi atingido.

O quinto objetivo é a proposta de analisar qual algoritmo de *OCR* pode ser utilizado com eficiência em imagens de cupons fiscais que, portanto, é retratado na seção 3.3. Primeiramente foi levantado os requisitos para utilizar o serviço de *OCR* do *Google Vision*, isso quer dizer, configuração de acesso e custos operacionais. Em seguida, também foi feita uma análise sobre o retorno em blocos de texto do *OCR*, propondo análises em heurísticas para extrair os campos de CNPJ, data e hora e valor total.

Através do último objetivo, ou seja, analisar os resultados obtidos na arquitetura proposta, é possível verificar a eficiência de cada etapa da arquitetura. Todos os resultados foram descritos no capítulo 4. Esses resultados abordam tanto o envio para o *OCR* das imagens sem alterações, como também as imagens com as técnicas aplicadas. Nas Técnicas de Realces e Binarização foi possível perceber uma limitação na melhora da taxa de acerto

dos campos de texto. Os resultados mostram que é possível melhorar a extração de dados com essas técnicas, porém isso é limitado em *clusters* específicos das imagens.

No final do capítulo 4 também é feito uma análise em relação a qual técnica poderia ser aplicada em cada *cluster* para melhorar a eficiência na extração de dados. Portanto, como pode ser visto no Quadro 16, é possível concluir que a arquitetura proposta pode de fato melhorar a extração de dados de imagens de cupons fiscais.

Ainda analisando os resultados obtidos, a Técnica de Compressão demonstrou a existência do cenário onde uma determinada técnica pode ter resultados positivos independentemente de qual *cluster* ela foi aplicada. Consequentemente, nesse caso a etapa de análise em *cluster* pode não ser uma boa alternativa, pelo fato de que a técnica pode ser aplicada em todo o conjunto de imagens. Porém, pode-se apenas concluir isso no escopo desenvolvido neste projeto, pois caso seja concebida uma nova técnica que apresente resultados melhores em determinados *clusters*, utilizar a arquitetura proposta pode ser uma boa alternativa.

A principal dificuldade encontrada no desenvolvimento do projeto foi implementar as técnicas de PDI, pois era necessário verificar como o *OCR* do *Google Vision* iria reagir às alterações feitas. Além disso, enviar todo o conjunto de imagens para *OCR* para obter as taxas de acerto era uma tarefa que demandava um certo tempo pelas limitações de rede. Em média, cada imagem enviada tomava um tempo de 2 a 3 segundos, considerando que a aplicação da técnica era rápida. Por fim, o maior aprendizado desse projeto foi buscar a solução para um problema comum no mercado através de conceitos teóricos ensinados durante a graduação de ciências da computação.

5.1 TRABALHOS FUTUROS

De acordo com a arquitetura proposta, seria possível utilizar qualquer análise em *cluster*, assim como também determinar o próprio conjunto de técnicas de PDI, e por fim, definir qual algoritmo de *OCR* seria utilizado para extrair o texto. Sendo assim, outros atributos poderiam ser utilizados para análise em *cluster*, assim como também outros algoritmos de “clusterização”.

Existe uma variedade de possíveis algoritmos que podem ser aplicados para alterar uma imagem, possibilitando o desenvolvimento de diversas outras técnicas de PDI. Dessa

maneira, para trabalhos futuros outras técnicas podem ser concebidas e testadas na arquitetura, avaliando novos resultados na extração de texto.

Além disso, o serviço de *OCR* poderia ser substituído pelo *Microsoft Vision API,* que também oferece o reconhecimento de caracteres via API REST. Isso permitiria uma nova análise de resultados, em que possivelmente, poderia alterar em qual cluster uma determinada técnica deveria ser aplicada. Novas heurísticas na extração dos campos de texto também podem ser desenvolvidas a fim de melhorar a taxa de acerto.

Contudo, o principal trabalho futuro talvez seja investigar os resultados obtidos pela Técnica de Compressão, isto é, compressão de imagens. Desenvolver outros tipos de compressões como *PNG*, *WEBP* e outros formatos com o objetivo de analisar novamente os resultados. Em vista disso, possivelmente criar um modelo de aprendizagem de máquina menor, com a finalidade de testar compressões de diversas outras bibliotecas. Por fim, é também preciso verificar artigos publicados com esse mesmo escopo, a fim de obter respostas mais conclusivas.

REFERÊNCIAS

Aldenderfer, M., Blashfield, R. *Cluster Analysis.* Sage Publications, Beverly Hills, USA, 1984.

Bin Mohamad, Ismail; Dauda Usman (2013). "Standardization and Its Effects on K-Means Clustering Algorithm" (PDF). Research Journal of Applied Sciences, Engineering and Technology.

CAMBRIDGE in COLOR. Sharpening: Unsharp Mask. Disponível em: <http://www.cambridgeincolour.com/tutorials/unsharp-mask.htm>. Acesso em: 04/10/2017.

CAMBRIDGE in COLOR. CAMERA HISTOGRAMS: TONES & CONTRAST. Disponível em: <http://www.cambridgeincolour.com/tutorials/histograms1.htm>. Acesso em: 05/10/2017.

Celina Abar. http://www.pucsp.br/~logica/Fuzzy.htm, PUC-SP/2004.

Charles Arthur. China drives smartphone growth - and low prices. Disponível em: https://www.theguardian.com/technology/2013/nov/14/china-smartphone-android-google-iphone. Acesso em: 16.09.2017.

CONFAZ. Conselho Nacional de Política Fazendária. Disponível em: < https://www.confaz.fazenda.gov.br/> Acesso em: 12/10/2017.

Dana, H. B., Christopher M. B. (1982). *Computer Vision*. Prentice Hall. ISBN 0-13-165316-4
Line Eikvil. *Optical Character Recognition.* https://www.nr.no/~eikvil/*OCR*.pdf, 1993.

Duda, R.,Hart, P. *Pattern Classification and Scene Analysis.* John Wiley & Sons, NY, USA, 1973.

Er. Neetu Bhatia. *Optical Character Recognition Techniques: A Review.* International Journal of Advanced Research in Computer Science and Software Engineering. ISSN: 2277 128X. Volume 4, Issue 5, May 2014.

FILHO, O.; NETO, H. Processamento Digital de Imagens. São Paulo, Brazil: Ed. Brasport, 1999. 331 p.

GITHUB. tesseract-*OCR*. Disponível em: <https://github.com/tesseract-*OCR*/tesseract/> . Acesso em: 03.04.2017.

GITHUB. Google Cloud Python Client. Disponível em: <https://github.com/GoogleCloudPlatform/google-cloud-python> . Acesso em: 11/10/2017.

GONZALEZ, R. C.; WOODS, R. E. Digital Image Processing. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001.

GOOGLE. Cloud Vision API. Disponível em: <https://cloud.google.com/vision/> . Acesso em: 03.04.2017.

GOOGLE. Authenticating to the Cloud Vision API. Disponível em: <https://cloud.google.com/vision/docs/auth?hl=pt-br>. Acesso em: 11/10/2017b.

Han, J., Kamber, M *Data Mining: Concepts and Techniques.* Morgan Kaufmann Publishers, San Mateo, CA, 2000.

Huang, T. (1996-11-19). Vandoni, Carlo, E, ed. *Computer Vision : Evolution And Promise* (PDF). 19th CERN School of Computing. Geneva: CERN. pp. 21–25. doi:10.5170/CERN-1996-008.21. ISBN 978-9290830955.

MacKay, David (2003). "Chapter 20. An Example Inference Task: Clustering" (PDF). *Information Theory, Inference and Learning Algorithms*. Cambridge University Press. pp. 284–292. ISBN 0-521-64298-1. MR 2012999.

MacQueen, J. B. (1967). *Some Methods for classification and Analysis of Multivariate Observations*. Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability. **1**. University of California Press. pp. 281–297. MR 0214227. Zbl 0214.46201.

Martin Ester, Hans-Peter Kriegel, Jorg Sander, Xiaowei Xu (1996). *A Density-Based Algorithm for Discovering Clusters*. Institute for Computer Science, University of Munich. KDD96-037.

MICROSOFT. Computer Vision API. Disponível em: <https://www.microsoft.com/cognitive-services/en-us/computer-vision-api>. Acesso em: 03.04.2017.

Nicu Sebe; Ira Cohen; Ashutosh Garg; Thomas S. Huang (3 June 2005). *Machine Learning in Computer Vision*. Springer Science & Business Media. ISBN 978-1-4020-3274-5.

OPENCV. Image file reading and writing, version 3.3.0. Disponível em: <http://docs.opencv.org/3.3.0/d4/da8/group__imgcodecs.html>. Acesso em: 08/10/2017.

Pang-Ning Tan, Michael Steinbach, Vipin Kumar. *Introduction to Data Mining.* Addison-Wesley, 2005. ISBN: 0321321367.

PEDRINI, H.; SCHWARTZ, W. Análise de Imagens Digitais: Princípios, Algoritmos e Aplicações. São Paulo, Brazil: Ed. Thomson Learning, 2007. 528 p. ISBN 978-85-221-0595-3.

PILLOW. Image Enhance Module. Disponível em: <https://pillow.readthedocs.io/en/4.3.x/reference/ImageEnhance.html> . Acesso em: 05/10/2017.

RAMOS, M. V. M., NETO, J. J., VEJA, I. S., Linguagens Formais: Teoria, Modelagem e Implementação. Ed. Bookman, 2009.

RECEITA FEDERAL. Informações Gerais sobre o CNPJ. Disponível em: <http://idg.receita.fazenda.gov.br/orientacao/tributaria/cadastros/cadastro-nacional-de-pessoas-juridicas-cnpj/informacoes-gerais-sobre-o-cnpj>. Acesso em: 12/10/2017a.

RECEITA FEDERAL. Emissão de Comprovante de Inscrição e de Situação Cadastral. Disponível em: <http://www.receita.fazenda.gov.br/pessoajuridica/cnpj/cnpjreva/cnpjreva_solicitacao2.asp> Acesso em: 12/10/2017b.

Richard Szeliski (30 September 2010). *Computer Vision: Algorithms and Applications*. Springer Science & Business Media. pp. 10–16. ISBN 978-1-84882-935-0.
Trevor, H., Robert, T., Jerome, F. (2009). *The Elements of Statistical Learning - Data Mining, Inference, and Prediction, Second Edition.* Springer-Verlag New York. ISBN: 978-0-387-84858-7.

Richa Goswami; O.P. Sharma. *A Review on Character Recognition Techniques.* International Journal of Computer Applications (0975 – 8887). Volume 83 – No 7, December 2013.

SCIKIT-LEARN. Module cluster, K-means. Disponível em: <http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html#sklearn.cluster.KMeans>. Acesso em: 30/09/2017.

SCIKIT-LEARN. Module cluster, DBSCAN. Disponível em: <http://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html#sklearn.cluster.DBSCAN> . Acesso em: 30/09/2017.

SCIKIT-LEARN. Module cluster, Spectral Clustering. Disponível em: <http://scikit-learn.org/stable/modules/generated/sklearn.cluster.SpectralClustering.html#sklearn.cluster.SpectralClustering>. Acesso em: 01/10/2017.

SCIKIT-IMAGE. Module exposure, check if image is low contrast. Disponível em: <http://scikit-image.org/docs/dev/api/skimage.exposure.html#skimage.exposure.is_low_contrast>. Acesso em: 26/09/2017.

SCKIT-IMAGE. Tutorial Examples - Image Thresholding. Disponível em: <http://scikit-image.org/docs/stable/auto_examples/xx_applications/plot_thresholding.html#sphx-glr-auto-examples-xx-applications-plot-thresholding-py>. Acesso em: 07/10/2017a.

SCIKIT-IMAGE. Image Filters - Local Thresholding. Disponível em: <http://scikit-image.org/docs/stable/api/skimage.filters.html#skimage.filters.threshold_local>. Acesso em: 07/10/2017b.

Ulrike Von Luxburg (2007). A Tutorial on Spectral Clustering. 10.1.1.165.9323.
Andrew Y. Ng and Michael I. Jordan and Yair Weiss (2001). On Spectral Clustering: Analysis and an algorithm. ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, 849--856. MIT Press. 10.1.1.19.8100.

WIKIPEDIA. Tesseract (software). Disponível em: <https://en.wikipedia.org/wiki/Tesseract_(software)>. Acesso em: 03.04.2017.

WIKIPEDIA. Brightness. Disponível em <https://en.wikipedia.org/wiki/Brightness>. Acesso em: 04/10/2017.

WIKIPEDIA. Color balance. Disponível em <https://en.wikipedia.org/wiki/Color_balance>. Acesso em: 05/10/2017.

Anexo A – Artigo

Análise e Classificação de imagens para aplicação de OCR em cupons fiscais

José Victor Feijó de Araujo[1], **Elder Rizzon Santos**[1], **Alexandre Golçalves Silva**[1]

[1]Instituto de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)
Florianópolis – SC – Brazil

victor.feijo@grad.ufsc.br, {elder, alexandre.goncalves}@ufsc.br

Abstract. *The proposal suggested by this work was to analyze the impact of a classification model, followed by image processing techniques and OCR for text extraction of tax coupons. This model had as proposition to classify the images of coupons in subgroups and then apply image processing techniques selected for each group with its proper characteristics, finally extracting text of these images through an OCR algorithm. Thus, methods have been developed to classify images into clusters using clustering algorithms. Were also proposed three techniques to be applied, each one applying different algorithms to change the images. These images were sent to the Google OCR Vision service, where it was possible to extract the text of images in blocks. The results of the developed model were evaluated by comparing the accuracy rate of OCR with the actual text values present in the tax coupons. Positive results were obtained using the developed model, improving the extraction of the total purchase value by approximately 6%.*

Resumo. *A proposta sugerida por este trabalho foi de analisar o impacto de um modelo de classificação, seguido de técnicas de PDI e OCR para extração de texto em cupons fiscais, classificando-os em subgrupos. Técnicas selecionadas de PDI foram aplicadas para cada grupo com suas devidas características, por fim extraindo texto dessas imagens através de um algoritmo de OCR. Sendo assim, foram desenvolvidos métodos para classificar as imagens em clusters utilizando algoritmos de "clusterização". Também foram propostas três técnicas de PDI, a primeira aplicando uma série de realces, a segunda uma binarização adaptativa e a terceira técnica utilizando a compressão de dados JPEG. Essas imagens foram enviadas para o serviço de OCR do Google Vision, onde foi possível extrair o texto das imagens em formato de blocos. Os resultados do modelo desenvolvido foram avaliados comparando a taxa de acerto do OCR com os valores de texto reais presentes nos cupons fiscais. Foram obtidos resultados positivos utilizando o modelo desenvolvido, melhorando a extração do valor total da compra em aproximadamente 6%.*

1. Introdução

As aplicações que utilizam visão computacional vêm se popularizando cada vez mais com os avanços tecnológicos em processamento de imagens. Além desses avanços, os dispositivos com câmeras de boa qualidade tiveram uma significativa redução de custos nos últimos anos, aumentando o acesso a essas tecnologias. Entretanto, mesmo com

esses avanços ainda é uma tarefa difícil o processamento automático de certos tipos de documentos. Dependendo da qualidade da imagem, os textos identificados podem não ser corretos, e essa margem de erro pode ser um fator decisivo para utilização dessas tecnologias na identificação de texto automática em documentos.

Diante disso, este trabalho tem como proposta analisar e classificar imagens de cupons fiscais para extração de conteúdo em texto, utilizando técnicas de análise em cluster, processamento de imagens (PDI) e reconhecimento óptico de caracteres (OCR).

A análise em cluster tem sido identificada como uma tarefa essencial em mineração de dados [Han e Kamber 2000]. Por um ponto de vista de alto nível, a "clusterização" pode ser definida como a segmentação de uma população heterogênea em um número maior de subgrupos homogêneos [ALDENDERFER e BLASHFIELD 1984]. Portanto, nesta proposta foi utilizado um modelo de "clusterização" de imagem no qual é factível verificar e separar claramente as propriedades de cada grupo de imagens. A partir desses grupos há a possibilidade de analisar as características semelhantes e usar algoritmos de PDI específicos para cada grupo classificado.

De acordo com Pedrini e Schwartz (2007), visão computacional é a área de pesquisa que procura auxiliar a resolução de problemas altamente complexos, buscando simular a interpretação humana e a habilidade do ser humano em tomar decisões de acordo com as informações contidas na imagem. Utilizando um algoritmo de "clusterização" para agrupar as imagens possibilita que o algoritmo de PDI seja mais eficiente em seu direcionado cluster, pois as imagens podem ter diferentes resoluções, tamanho, luminosidade entre outras propriedades.

A extração de textos através de imagens de cupons fiscais foi realizada neste trabalho utilizando um procedimento de reconhecimento de padrões chamado de Reconhecimento Óptico de Caracteres (Optical Character Recognition, OCR). A análise qualitativa, que consiste em tratar com precisão e legibilidade dos resultados da extração de dados foi feita com base na taxa de acerto dos campos definidos que o OCR conseguiu identificar comparando com os dados reais no cupom fiscal.

2. Fundamentação Teórica

2.1. Análise em cluster

De acordo com Han e Kamber (2000), a análise em cluster é o processo que consiste em agrupar um conjunto de objetos físicos ou abstratos em classes similares de objetos. Já o cluster é uma coleção de objetos que são similares nessa mesma coleção e não similares a objetos em diferentes clusters. Diferente de classificação, a análise em cluster não utiliza classes pré-definidas e exemplos de dados para treino. Portanto, é uma forma de aprendizado por observação das características semelhantes entre os objetos [HAN e KAMBER 2000]. Existem diversos algoritmos de clusterização, sendo que cada um pode performar de diferentes maneiras dependendo do conjunto de dados a ser aplicado.

O **K-means** é um algoritmo de clusterização particionado que tem como objetivo buscar o número de clusters K especificado pelo usuário, no qual os clusters são representados por centróides. Para gerar os clusters e classificar os dados, o algoritmo faz uma comparação entre cada valor de cada dado por meio da distância, geralmente utilizando a distância euclidiana. Conforme o algoritmo vai iterando, o valor de cada centróide é

refinado pela média dos valores de cada atributo de cada ocorrência que pertence a este centróide. Com isso, o algoritmo gera *K* centróides e coloca os dados de acordo com sua distância dos centróides. Na Figura 1 é possível ver um exemplo das iterações do algoritmo com um conjunto de dados em duas dimensões de 40 pontos e *K=2*.

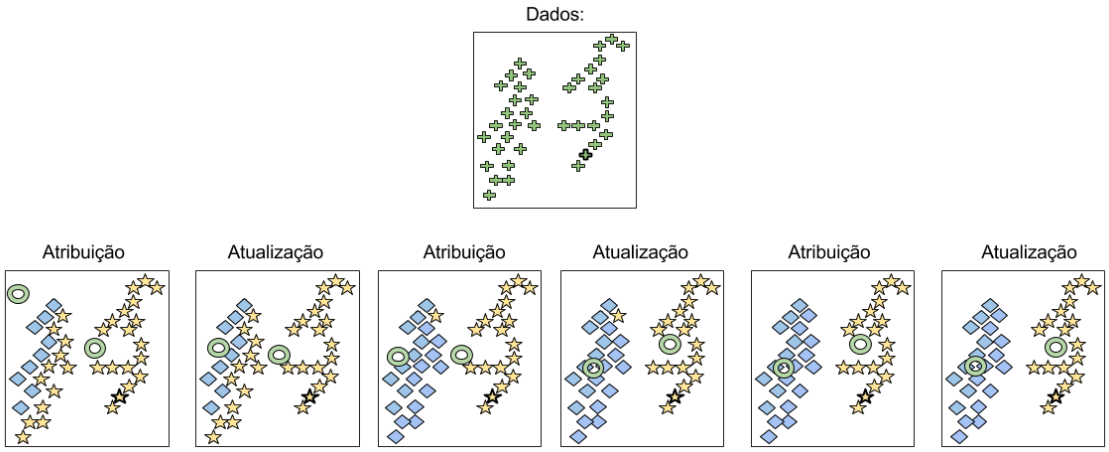


Figure 1. Algoritmo K-means aplicado a um conjunto de 40 dados e K=2

Outro algoritmo clássico de análise em cluster é o **DBSCAN**, que é um algoritmo que depende da noção de densidade em clusters, no qual foi criado para lidar com clusters de tamanhos arbitrários [Ester, Kriegel, Sander e Xu 1996]. Tan, Steinbach, Kumar (2005) descrevem o algoritmo informalmente como: quaisquer pontos centrais que são próximos o suficiente (com uma distância Eps entre eles) são colocados no mesmo cluster. Na execução do algoritmo o usuário precisa selecionar dois parâmetros, *Eps* e *MinPts*. Na Figura 2 é possível ver a aplicação do algoritmo em um conjunto de dados com 3000 pontos de duas dimensões, utilizando *Eps=10* e *MinPts=4*.

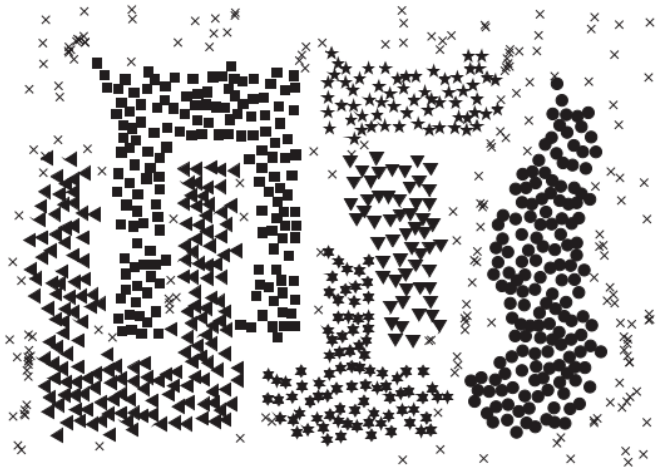


Figure 2. Algoritmo DBSCAN aplicado a um conjunto de 3000 dados com EPS=10 e MinPts=4

Pelo fato de o DBSCAN utilizar a definição de densidade, o torna relativamente resistente a ruídos, podendo lidar com clusters de tamanhos e formatos arbitrários. Por

exemplo, ele pode encontrar clusters que um algoritmo como o K-means não conseguiria encontrar. Entretanto, o DBSCAN tem dificuldade de lidar com clusters com uma grande variedade de densidades.

Outro algoritmo também utilizado é o **Spectral Clustering**, que é uma generalização dos métodos de agrupamento padrões, projetado para funcionar em situações onde os clusters não têm apenas um formato circular ou elíptico [Hastie, Tibshirani e Friedman 2009]. A ideia principal é construir uma matriz que representa as relações locais de vizinhança sobre as observações. Existem diversas maneiras de definir a matriz de similaridade e seu grafo que representa o comportamento local, sendo o *K-nearest-neighbor* o mais popular. Também é necessário calcular a matriz Laplaciana, podendo ser não-normalizada, normalizada, entre outras [VON LUXBURG 2007].

2.2. Processamento digital de imagens

O processamento digital de imagens pode ser definido como um conjunto de técnicas para capturar, representar e transformar imagens utilizando um computador. A aplicação dessas técnicas permite extrair e identificar informações das imagens e melhorar a qualidade visual de aspectos estruturais, simplificando a percepção humana e a interpretação automática por meio de máquinas [PEDRINI e SCHWARTZ 2007]. Uma imagem digital pode ser representada através de uma matriz bidimensional, na qual cada pixel da imagem corresponde a um elemento da matriz. Na Figura 3 é possível ver um exemplo de representação matricial de uma imagem, sendo que uma pequena região destacada é formada por números inteiros que representam o nível de cinza dos pixels da imagem.

(a)

120	138	120	151	139
110	129	129	139	146
150	138	137	138	129
137	129	129	128	137
146	125	131	132	145

(b)

Figure 3. Representação matricial; (a) imagem; (b) níveis de cinza correspondentes a região destacada da imagem

O histograma de uma imagem pode ser representado por um gráfico indicando o número de pixels na imagem para cada nível de cinza [PEDRINI e SCHWARTZ 2007]. Analisando o histograma de uma imagem é possível obter uma indicação de sua qualidade quanto ao nível de contraste e quanto ao seu brilho médio, ou seja, se a imagem é predominante clara ou escura [FILHO e NETO 1999]. A Figura 4 apresenta dois tipos diferentes de histogramas que são frequentemente encontrados em imagens comuns. O

histograma da Figura 4(a) apresenta uma concentração de pixels nos valores baixos de cinza, o que corresponde a uma imagem predominantemente escura. Já o histograma da Figura 4(b) os pixels estão concentrados próximo ao limite superior da escala de cinza, caracterizando uma imagem clara [FILHO e NETO, 1999].

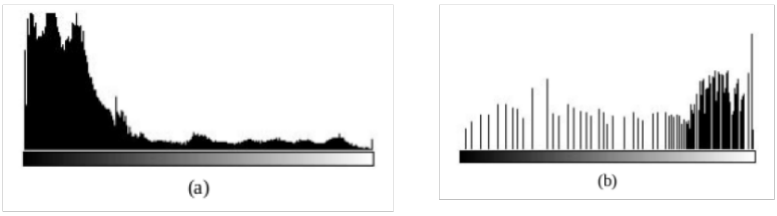


Figure 4. Representação matricial; (a) imagem; (b) níveis de cinza correspondentes a região destacada da imagem

Embora o histograma de uma imagem proporciona diversas características sobre ela (nível de cinza mínimo, médio e máximo, predominância de pixels claros ou escuros etc.), outras características qualitativas somente podem ser extraídas dispondo-se da imagem propriamente dita [FILHO e NETO, 1999].

2.3. Reconhecimento óptico de caracteres (OCR)

O OCR é um método de reconhecimento de padrões que pertence a família de técnicas de identificação automática [EIKVIL 1993]. Existem diversas técnicas de identificação automática, na qual cada uma satisfaz às necessidades de sua área de aplicação. Um sistema típico de OCR consiste em diversos componentes, podendo variar de acordo com a aplicação e a metodologia utilizada. Na Figura 5 podem ser vistos uma série de componentes envolvendo as etapas do reconhecimento de caracteres.

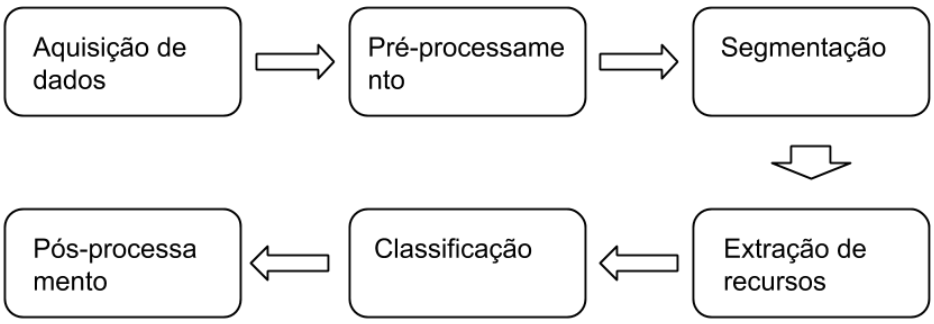


Figure 5. Sequência de etapas em um sistema OCR

Um dos sistemas de OCR mais conhecidos no mercado é o *Tesseract*, uma aplicação de OCR mantida pela *Google* e por diversos outros desenvolvedores como *software livre*. Embora o *Tesseract* seja uma aplicação de OCR gratuita, o resultado de sua aplicação em imagens digitais de documentos pode não conter todas as informações desejadas. Entretanto, existem outras aplicações modernas que conseguem desempenhar bons resultados na extração de texto de imagens, como por exemplo o *Google Cloud Vision API*

e o *Microsoft Vision API*. Essas aplicações utilizam modelos de aprendizado de máquina capazes de extrair diversas características da imagem e possivelmente bons resultados na extração de texto.

3. Proposta de análise e classificação

A proposta de análise e classificação foi desenvolvida com base na concepção de que o OCR aplicado em uma imagem é um algoritmo de uso geral, que pode ser utilizado tanto para extrair os dados de um cupom fiscal quanto para extrair o número da placa de um carro, por exemplo. Contudo, considerando que as imagens a serem analisadas são de cupons fiscais, é possível modificar as suas características, visando obter melhora nos resultados do OCR. Diante disso, foi desenvolvida uma arquitetura em que as imagens percorrem pelas seguintes etapas: algoritmo de clusterização, técnica de processamento de imagem e aplicação do OCR. Essa metodologia tem como objetivo analisar as imagens em subgrupos (clusters), e para cada um desses subgrupos aplicar uma técnica de PDI antes da aplicação do OCR. De acordo com o fluxograma da Figura 6, é possível observar cada etapa em que a imagem percorre na metodologia desenvolvida.

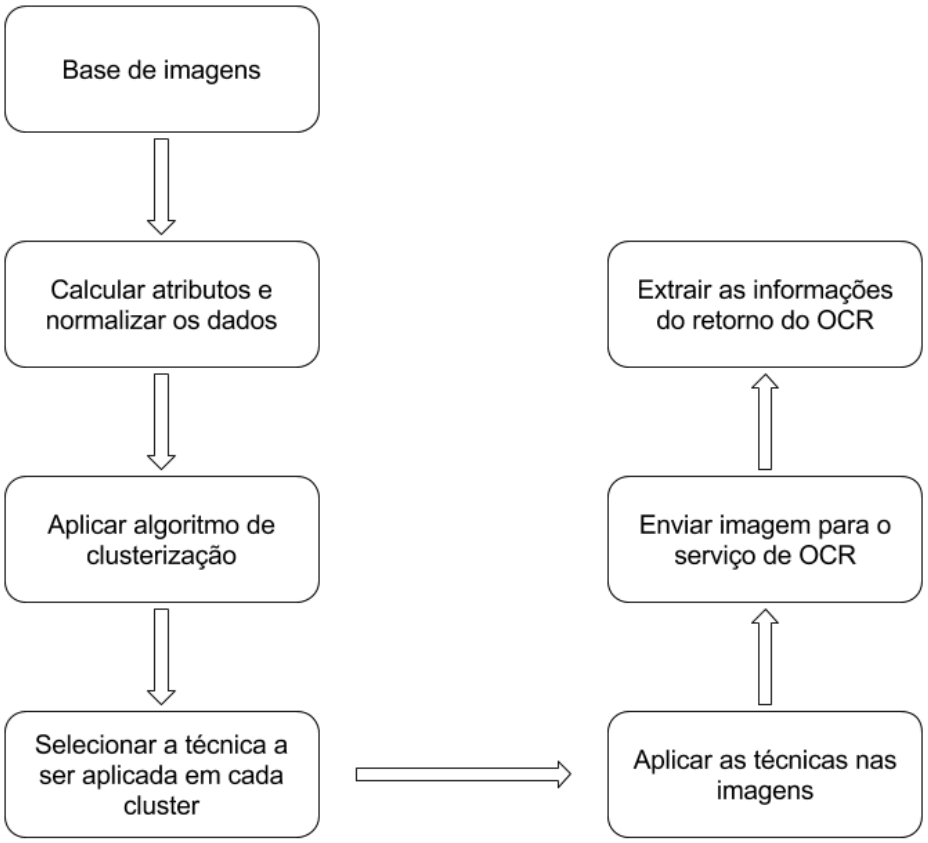


Figure 6. Fluxograma da arquitetura desenvolvida

No desenvolvimento da metodologia proposta, foi utilizado um conjunto de dados com 3200 imagens de cupons fiscais. O conjunto de dados foi fornecido pela empresa SumOne, que utilizava as imagens em suas aplicações de mercado. Na Figura 21, é

possível ver alguns exemplos de imagens extraídas desse conjunto. Essas imagens foram obtidas através de distintos smartphones, por isso elas variam a resolução, qualidade da câmera, iluminação e diversos aspectos. Cada elemento do conjunto de dados contém uma url que indica o endereço de onde a imagem está armazenada, e outros três campos que foram extraídos manualmente: CNPJ, valor total e a data e hora que foi efetuada a compra.

Figure 7. Exemplos de imagens do conjunto de dados

3.1. Análise em cluster

O propósito da etapa de análise em cluster é separar as imagens com características semelhantes em subgrupos, analisando-os para aplicação de técnicas de PDI direcionadas. No desenvolvimento da metodologia para alcançar esse objetivo, foram estabelecidos quatro requisitos: seleção dos atributos, normalização, aplicação do algoritmo e visualização dos resultados. As imagens do conjunto de dados possuem diversos atributos que podem ser utilizados para realizar a "clusterização", porém é necessário selecionar apenas os que possam gerar bons resultados para uma futura análise e aplicação das técnicas de PDI. Foram selecionados dois atributos na proposta desenvolvida: o número de pixels e o nível de contraste de cada imagem.

O atributo número de pixels de uma imagem é obtido através da quantidade de pixels que a imagem possui, ou seja, levando em consideração que a imagem é uma matriz de pixels, esse valor é a multiplicação do número de linhas pelo número de colunas. O atributo pode ser considerado interessante para criação dos clusters pelo fato de que o conjunto de dados contêm imagens de diferentes câmeras, contendo uma boa distribuição desse valor. O segundo atributo utilizado, o nível de contraste, tem como finalidade medir o grau de contraste de uma imagem. Os atributos selecionados contêm distintas ordens de magnitude, e para que isso não afete os resultados dos algoritmos de clusterização, eles foram normalizados para obter valores em ordens semelhantes.

Considerando todos os dados devidamente pré-processados, a próxima etapa é

aplicar de fato um algoritmo que realize a distribuição do conjunto de dados em clusters. Na proposta realizada, o escopo foi limitado a análise de três algoritmos de clusterização, sendo eles: K-means, DBSCAN e Spectral Clustering. Cada um desses algoritmos utiliza um método distinto para verificar a semelhança entre os dados, diante disso foi realizada uma análise comparativa entre eles para determinar a qualidade dos clusters gerados. Na Figura 8 é possível ver o gráfico com os resultados do algoritmo K-means, que foi aplicado com o número de clusters fixado em 5. Já na Figura 9 está representado os clusters gerados pelo algoritmo DBSCAN utilizando *EPS=0.13* e *MinPts=300*. O último algoritmo aplicado também pode ser visto na Figura 10, onde assim como o K-means também foi fixado o número de clusters em 5. Por fim, na Tabela 1 é possível ver os clusters gerados por cada algoritmo com seus respectivos números de imagens.

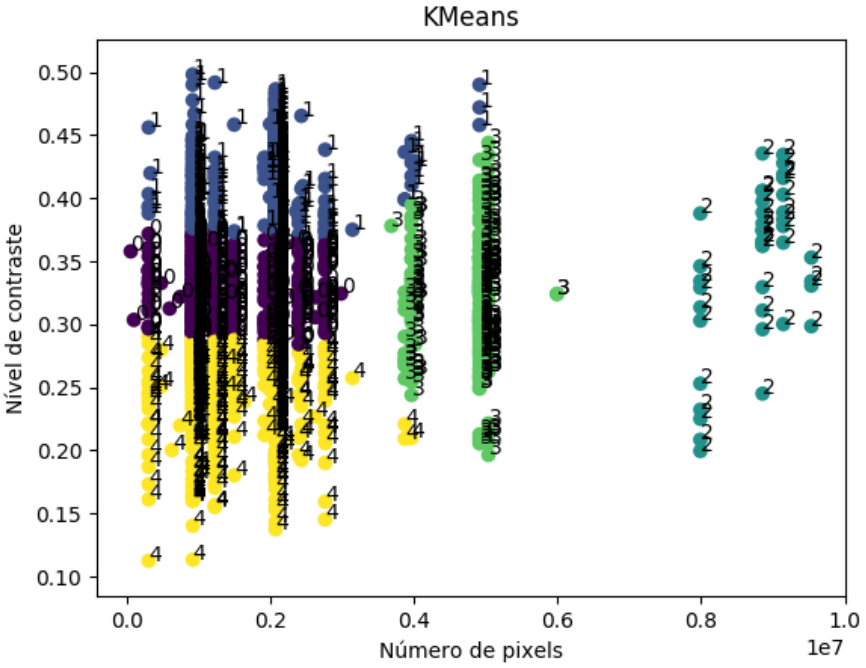


Figure 8. Gráfico representativo dos clusters gerados utilizando o algoritmo K-Means no conjunto de imagens

Table 1. Clusters gerados por cada algoritmo aplicado com suas respectivas quantidades de imagens

Cluster	K-means	Spectral	DBSCAN
0	1323	262	2957
1	1062	1313	243
2	42	41	0
3	208	808	0
4	565	776	0

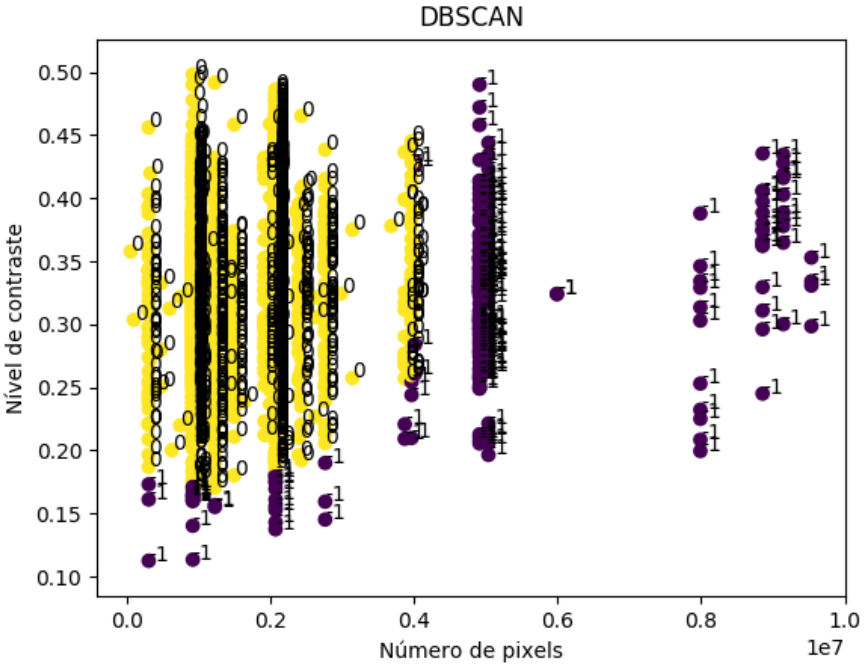


Figure 9. Gráfico representativo dos clusters gerados utilizando o algoritmo DBSCAN no conjunto de imagens

Fazendo uma análise comparativa entre os clusters gerados pelo K-Means, e pelo DBSCAN, fica evidente que o primeiro algoritmo separou os clusters de maneira mais eficiente. Além disso, o K-means utilizou melhor o atributo nível de contraste para distinguir os clusters, possibilitando analisar clusters com baixo contraste e baixa resolução de pixel e também clusters de alto contraste e baixa resolução de pixels. Comparando os resultados do Spectral Clustering com o K-means, é possível concluir que ele obteve um bom resultado na geração dos clusters. Em conclusão, os resultados de ambos são relevantes para analisar a aplicação das técnicas de PDI que serão descritas na próxima seção.

3.2. Técnicas de PDI

Neste passo, o conteúdo da imagem é alterado através da aplicação de algum algoritmo, podendo ter o objetivo de ajustes de brilho e contraste, filtros adaptativos ou até compressões na imagem. Isto posto, é possível desenvolver diversas técnicas para serem aplicadas no conjunto de imagens, mais especificamente, aplicando uma técnica distinta para cada cluster gerado ou a mesma técnica em demais clusters. Na proposta desenvolvida, foram elaboradas três técnicas: Técnica de Realces, Técnica de Binarização e Técnica de Compressão. Cada uma dessas técnicas utiliza um algoritmo distinto para alterar a imagem, permitindo analisar os resultados da aplicação de uma determinada técnica em um ou mais clusters.

A primeira técnica desenvolvida, nomeada Técnica de Realces, aplica um conjunto de quatro algoritmos de realce em imagens, sendo eles: Sharpness, ajuste de brilho, ajuste de contraste e balanceamento de cor. O objetivo desses algoritmos é alterar as imagens de modo que a aplicação do OCR consiga extrair de forma mais eficiente os dados

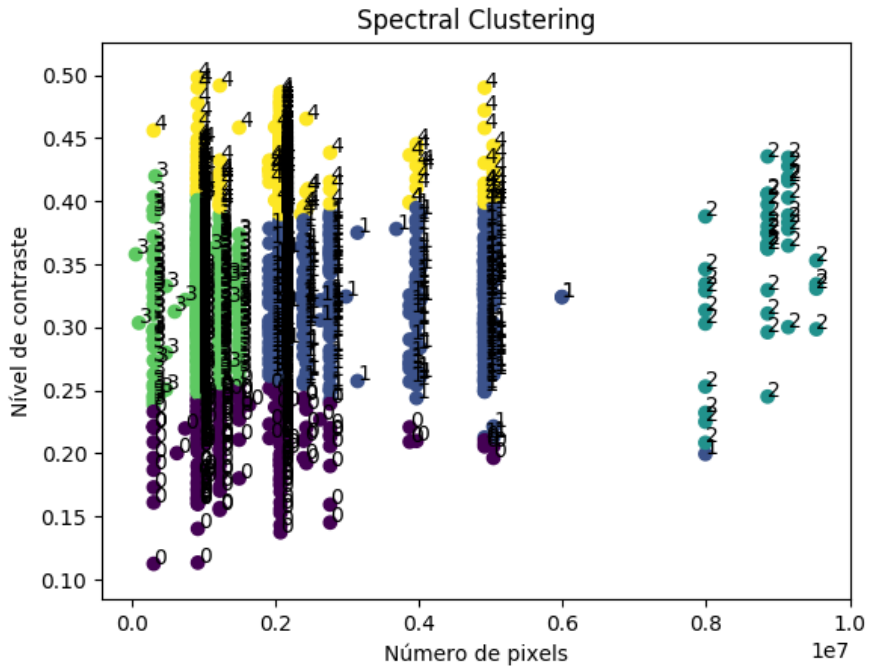


Figure 10. Gráfico representativo dos clusters gerados utilizando o algoritmo Spectral Clustering no conjunto de imagens

do cupom fiscal. Na Figura 11 é possível visualizar um exemplo da aplicação da Técnica de Realces em uma imagem de cupom fiscal. Na esquerda está a imagem original, e na direita o resultado da aplicação da técnica.

Fazendo uma análise comparativa entre as duas imagens, é possível observar que na imagem original as cores do papel do cupom estão meio amarelas e manchadas, não dando contraste aos tons de pretos do texto. Porém, na imagem com a aplicação da técnica, a cor do papel é normalizada em branco e os tons de preto do texto contêm um melhor destaque, ficando mais nítido e facilitando a leitura dos valores.

A segunda técnica desenvolvida, Técnica de Binarização, tem como objetivo aplicar um algoritmo de limiarização (thresholding) na imagem, separando suas regiões em fundo e objeto, ou seja, no escopo de cupons fiscais em papel e texto. Conforme Filho e Neto (1999), a forma mais simples de limiarização consiste na bipartição do histograma, convertendo os pixels cujo tom de cinza é maior ou igual a um certo valor de limiar em brancos e os demais em pretos. Se a imagem tiver um fundo relativamente uniforme, pode ser aplicado a limiarização global. No entanto, se houver uma grande variação na intensidade do fundo, aplicar a limiarização local é mais adequado para obter melhores resultados. É possível ver um exemplo da aplicação da Técnica de Binarização na Figura 12, onde na esquerda é a imagem original e no lado direito o resultado da técnica.

Mesmo com a presença de sombra na imagem original, a aplicação da limiarização local obteve um bom resultado. De um modo geral, o resultado da técnica foi uma imagem com apenas o texto do cupom fiscal, seu fundo perdeu destaque pois foi normalizado em branco e é possível visualizar facilmente o texto em preto.

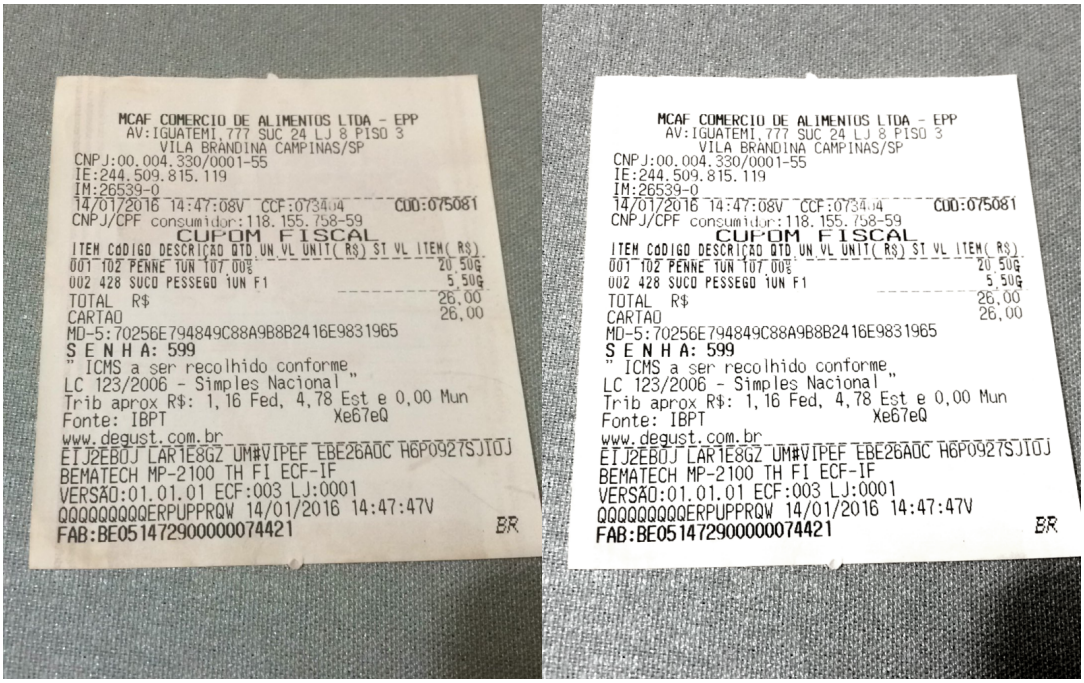


Figure 11. Demonstração da aplicação da Técnica de Realces em uma imagem. Na esquerda Imagem original, e na direita o resultado da técnica

A terceira e última técnica desenvolvida apresenta um funcionamento um pouco diferente da técnicas anteriores, ao invés de aplicar métodos de filtro ou realce na imagem, é explorado a compressão de dados. O objetivo dessa técnica é utilizar a compressão não para diminuir o tamanho do arquivo final da imagem, mas na verdade o contrário, aumentando o seu tamanho. Como todas as imagens do conjunto de dados são de câmeras de smartphones, em algum momento já foi aplicada alguma compressão JPEG, ou seja, seu tamanho em bytes já está reduzido. Portanto, o algoritmo recebe uma imagem com um tamanho de N bytes, e aplica a compressão JPEG com parâmetro de qualidade máximo.

Mesmo parecendo contra intuitivo utilizar a compressão de imagens para aumentar o tamanho da imagem final, a aplicação dessa técnica no conjunto de imagens mostrou resultados positivos em relação à extração de dados via OCR. Na seção de resultados, são apresentados todos os resultados da aplicação dessa técnica nos clusters, comparando com a aplicação das Técnicas de Realces e Binarização.

3.3. Aplicação do OCR

O serviço de OCR utilizado foi o Google Cloud Vision API, uma aplicação que permite aos desenvolvedores entender o conteúdo de uma imagem por meio do encapsulamento de poderosos modelos de aprendizado de máquina em uma API REST. Entre os diversos recursos do Google Vision, os principais são: Detecção Facial, Atributos de imagens, Detecção na Web, Detecção de pontos de referência e Reconhecimento óptico de caracteres. O recurso utilizado no desenvolvimento foi o reconhecimento óptico de caracteres (OCR), que extrai textos de uma imagem com compatibilidade para um extenso grupo de idiomas, além do suporte a identificação automática de idioma. Na Figura 13, é possível visualizar a lista de blocos do retorno do OCR em uma imagem de cupom fiscal. Os polígonos em verde representam os blocos encontrados com seus respectivos textos. No

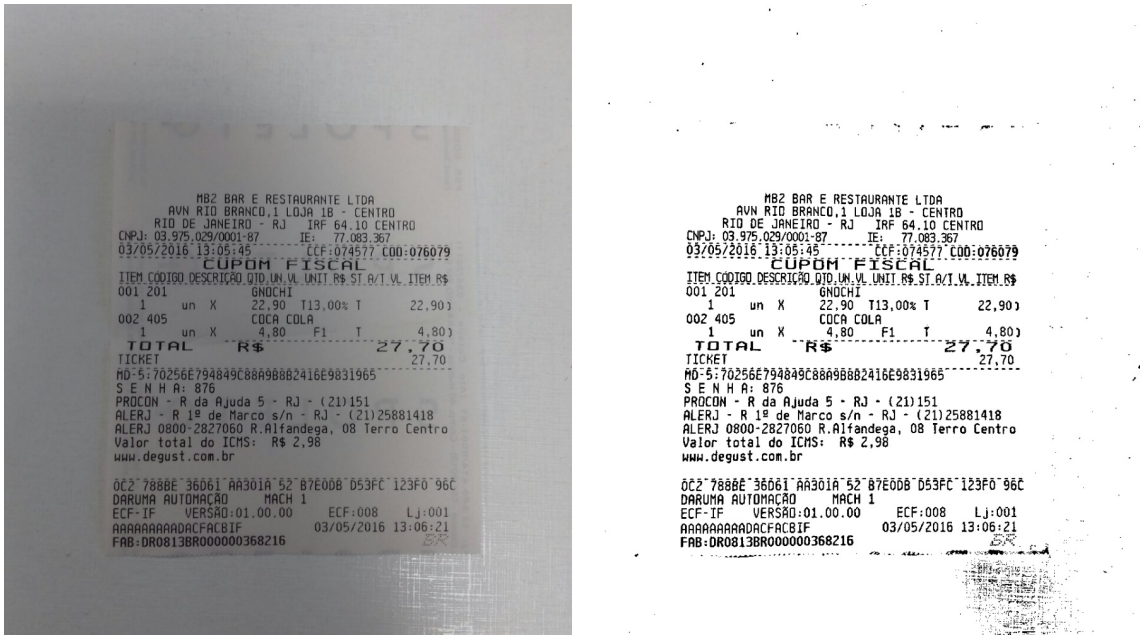


Figure 12. Demonstração da aplicação da Técnica de Realces em uma imagem. Na esquerda Imagem original, e na direita o resultado da técnica

entanto, nem todos os blocos de texto são encontrados de forma descritiva, ou seja, campo de um lado e em seguida seu respectivo valor em texto. Em vista disso, não é uma tarefa trivial obter os campos de CNPJ, hora da venda e o total da compra.

A heurística para extrair o valor do CNPJ é primeiro aplicar uma expressão regular que procura pelo padrão de números do CNPJ, ou seja, um padrão dado por '99.999.999/9999-99' onde o 9 representa um número de 0 a 9. Após isso, uma validação é feita com o valor encontrando para verificar se foi encontrado de fato um CNPJ válido. Semelhante ao campo de CNPJ, a hora da venda também tem um formato característico, facilitando sua extração via expressão regular. Este formato é sempre dado pelo padrão '99/99/9999 99:99', no qual representa a data seguida das horas e minutos. Diferente da extração de CNPJ e data, não é um bom método buscar o valor total em todo texto do cupom fiscal, tendo em vista que o texto pode conter muitos valores nesse mesmo formato, porém referentes a produtos ou impostos.

Portanto, a estratégia utilizada foi usar os blocos encontrados para buscar pelo valor total em uma área delimitada dentro do cupom fiscal, aumentando a probabilidade de encontrar um valor que seja de fato o valor total. Isto é, primeiro é utilizado uma expressão regular para buscar na lista de objetos o bloco que contém o campo do valor total. Após encontrar esse bloco, é determinado um limite inferior e superior a partir da posição desse bloco para buscar o valor total. Apesar da estratégia aplicada demonstrar bons resultados, existem alguns casos onde o cupom fiscal possui também o valor pago em dinheiro, que em muitos casos é maior que o valor total da compra.

4. Resultados

O objetivo deste capítulo é apresentar os resultados obtidos pela arquitetura proposta desenvolvida. Esta análise é feita comparando os valores extraídos pelo OCR com os valores que realmente estão na foto do cupom fiscal. Cada elemento do conjunto de dados contém

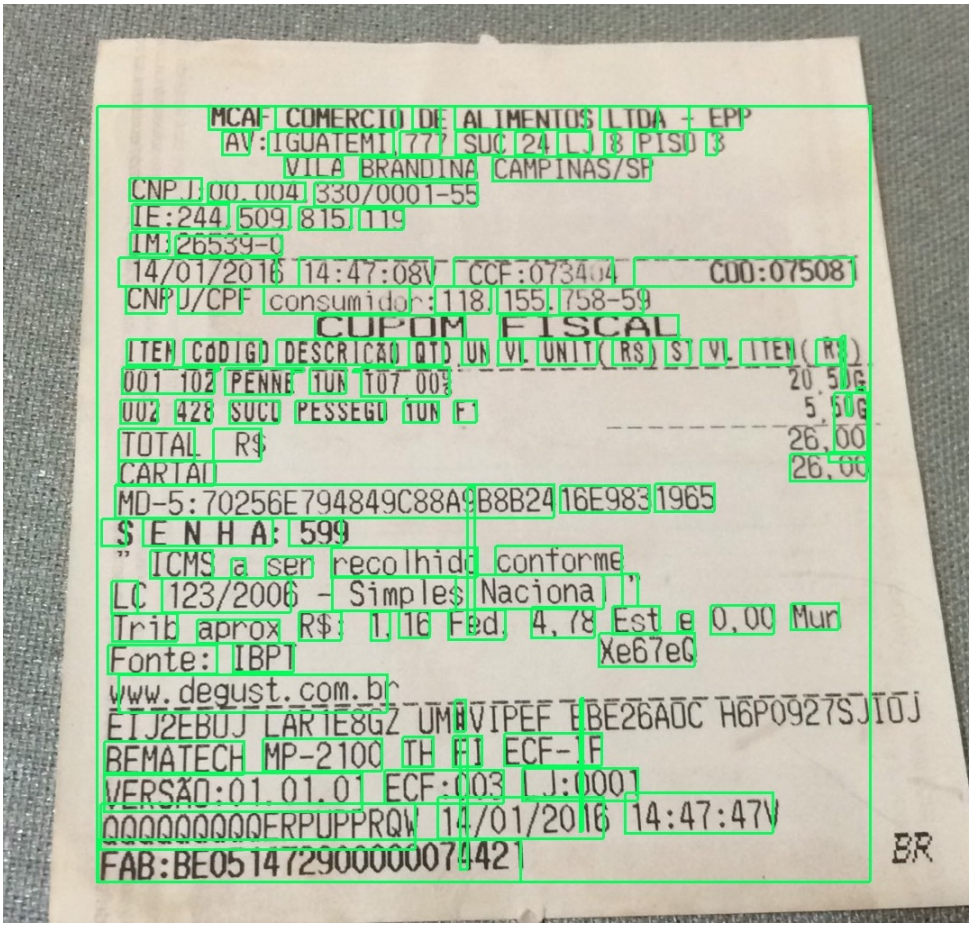


Figure 13. Retorno da aplicação do OCR pelo Google Vision. Cada polígono em verde representa um bloco de texto encontrado.

a url da imagem juntamente com os valores de CNPJ, data e valor total da compra retirados manualmente de cada foto. Os resultados são apresentados em taxa de acerto, ou seja, a razão entre o número de acertos do OCR pela quantidade de imagens. Para exemplificar, supondo que o número de elementos do conjunto de dados é de 1000 e a quantidade de acertos do OCR para data e hora foi de 650, a taxa de acerto do OCR para data e hora é dita de 65%. Na apresentação dos resultados por cluster, são utilizados os gerados pelo algoritmo K-means. Por fim, também são apresentados os resultados das técnicas selecionadas por clusters, isto é, a aplicação de técnicas distintas em cada cluster com objetivo de melhorar a taxa de acerto.

4.1. Análise das imagens sem alterações

Antes de demonstrar o resultado das técnicas desenvolvidas, primeiramente foi analisado os resultados obtidos pelo envio da imagem diretamente para o serviço de OCR, ou seja, sem nenhuma alteração. Na tabela 2 é possível visualizar os resultados da aplicação do OCR para cada campo extraído. A primeira linha do Quadro denota a taxa de acerto dos campos em todo o conjunto de dados, isto é, as 3200 imagens de cupons fiscais. As demais linhas representam a análise separada pelos clusters gerados pelo algoritmo K-means, visível na Figura 8.

As imagens dos clusters 2 e 3 são as que possuem alta resolução de pixels, o

Table 2. Resultados da aplicação do OCR nas imagens sem alterações, segmentado pelos clusters gerados no algoritmo K-means

Cluster	CNPJ	Valor Total	Data e Hora	Quantidade
Sem *cluster*	75,78125%	55,03125%	69,21875%	3200
Cluster 0	78,08012%	60,84656%	71,12622%	1323
Cluster 1	74,67043%	42,84369%	68,36158%	1062
Cluster 2	88,09523%	61,90476%	76,19047%	42
Cluster 3	81,25000%	71,15384%	75,00000%	208
Cluster 4	69,55752%	57,87610%	63,71681%	565

que pode ser utilizado como justificativa para os bons resultados em relação aos outros clusters. Porém, já as imagens do cluster 4 não demonstraram um bom resultado nas taxas de acerto, tendo em vista que são imagens que contêm de modo geral um baixo número de pixels e também um baixo nível de contraste.

4.2. Análise das técnicas desenvolvidas

Esta seção tem como objetivo apresentar os resultados da aplicação das técnicas desenvolvidas na arquitetura proposta, realizando uma análise comparativa entre elas. Por fim, será apresentado qual a melhor técnica que pode ser aplicada em cada cluster, junto com o impacto disso na taxa de acerto dos resultados finais. Os dados da Tabela 3 demonstram os resultados obtidos pela aplicação da Técnica de Realces com o algoritmo K-means.

Table 3. Resultados da aplicação do OCR nas imagens com a Técnica de Realces, segmentado pelos clusters gerados no algoritmo K-means

Cluster	CNPJ	Valor Total	Data e Hora	Quantidade
Sem *cluster*	71,06250%	56,03125%	69,03125%	3200
Cluster 0	73,31821%	59,18367%	70,14361%	1323
Cluster 1	67,23163%	48,11676%	68,17325%	1062
Cluster 2	88,09523%	50,00000%	61,90476%	42
Cluster 3	81,73076%	70,67307%	75,00000%	208
Cluster 4	67,78761%	58,58407%	66,37168%	565

A Tabela 4 demonstra os resultados da aplicação da Técnica de Binarização em todo o conjunto de imagens.

A Tabela 5 demonstra os resultados da aplicação da técnica de Compressão com os clusters do algoritmo K-means. Realizando uma análise comparativa com as imagens sem alterações e também com as técnicas anteriores, a aplicação da Técnica de Compressão mostrou taxas de acerto relativamente superiores. Analisando a aplicação da técnica em todo o conjunto de imagens, a taxa de acerto dos três campos foi superior, com o valor

Table 4. Resultados da aplicação do OCR nas imagens com a Técnica de Binarização, segmentado pelos clusters gerados no algoritmo K-means

Cluster	CNPJ	Valor Total	Data e Hora	Quantidade
Sem *cluster*	68,87500%	58,87500%	64,00000%	3200
Cluster 0	69,31216%	61,14890%	65,53287%	1323
Cluster 1	72,50470%	60,92278%	67,32580%	1062
Cluster 2	88,09523%	50,00000%	64,28571%	42
Cluster 3	78,84615%	62,98076%	65,38461%	208
Cluster 4	55,92920%	48,84955%	53,62831%	565

total chegando a aproximadamente 13% a mais que na Técnica de Binarização. Além disso, foi a única técnica que conseguiu melhorar a taxa de acerto dos três campos simultaneamente, onde ambos ficaram com taxas de acerto superiores a 70%.

Table 5. Resultados da aplicação do OCR nas imagens com a Técnica de Compressão, segmentado pelos clusters gerados no algoritmo K-means

Cluster	CNPJ	Valor Total	Data e Hora	Quantidade
Sem *cluster*	79,5%	71,21875%	72,3125%	3200
Cluster 0	80,87679%	73,46938%	72,26001%	1323
Cluster 1	81,63841%	74,76459%	74,67043%	1062
Cluster 2	88,09523%	61,90476%	69,04761%	42
Cluster 3	83,17307%	70,1923%	77,40384%	208
Cluster 4	70,26548%	60,35398%	66,37168%	565

4.3. Técnicas selecionadas por cluster

Nesta seção é demonstrado como seriam os resultados na arquitetura proposta, ou seja, selecionando as técnicas a serem aplicadas em cada cluster. O critério de seleção de qual técnica será utilizada em um cluster foi escolhido com base em qual cluster a determinada técnica teve as melhores taxas de acerto. Considerando os resultados relativamente superiores da Técnica de Compressão, a análise das técnicas selecionadas por cluster irá utilizar apenas os resultados das imagens sem alterações e das Técnicas de Realces e Binarização.

Para os clusters 0, 2 e 3 nenhuma técnica foi aplicada tendo em vista que ambas as Técnicas de Realces e Binarização obtiveram resultados inferiores em comparação as imagens sem alterações. No cluster 1 foi selecionado a aplicação da Técnica de Binarização, onde de fato obteve seu melhor resultado. Já o último cluster, número 4, a técnica escolhida foi a Técnica de Realces, na qual obteve uma considerável melhora na taxa de acerto do valor total e na data e hora. Na Tabela 6 são apresentados os resultados da aplicação dessas técnicas nos determinados clusters.

Table 6. Resultado do OCR utilizando as técnicas selecionadas por cluster

CNPJ	Valor Total	Data e Hora	Quantidade
74,75000%	61,03125%	69,34375%	3200

Para visualizar melhor essa comparação, na Figura 14 é apresentado um gráfico de comparação entre as taxas de acerto da aplicação do OCR no conjunto de imagens. Cada grupo de barras representa um campo extraído. Por fim, esses resultados obtidos comprovam que a arquitetura proposta pode ser efetivamente uma boa estratégia para melhorar a extração de dados de imagens de cupons fiscais via aplicação de OCR.

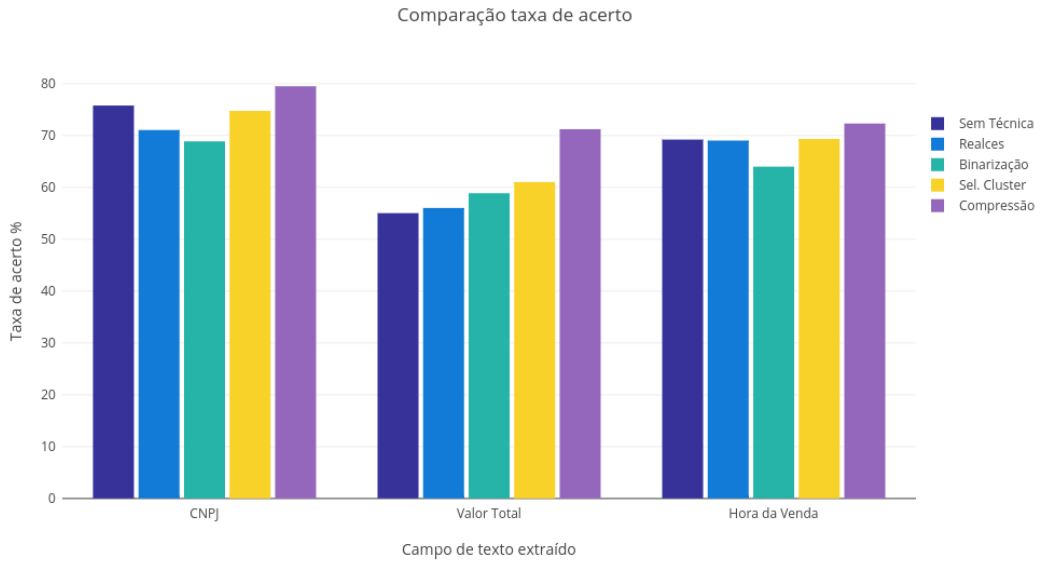


Figure 14. Gráfico de comparação das taxas de acerto de CNPJ, Valor Total e Hora da Venda dos resultados

5. Conclusões

A partir do impacto de técnicas de clusterização, processamento de imagens e OCR na extração de textos de imagens de cupons fiscais, é possível concluir que o resultado esperado do presente trabalho foi alcançado com êxito.

Para desenvolvimento do trabalho, primeiramente aplicou-se uma proposta de arquitetura para transformar imagens de cupons fiscais em texto através da análise em cluster, processamento de imagens e por fim OCR.

No processo de divisão dos clusters por atributos semelhantes, dois atributos selecionados para análise foram o número de pixels e o nível de contraste da imagem, já as três técnicas de "clusterização" foram o K-means, DBSCAN e o Spectral Clustering. Através dos resultados obtidos pelos algoritmos, é conclusivo que utilizando os dois atributos selecionados, apenas o K-means e o Spectral Clustering demonstraram bons resultados.

A próxima etapa da arquitetura proposta consiste em aplicar as técnicas de PDI nas imagens. Dentre técnicas utilizadas, a Técnica de Compressão foi a última do conjunto de

técnicas, onde é explorado a compressão de imagens no formato JPEG, e foi a que obteve melhor resultado.

Também é feito uma análise em relação a qual técnica poderia ser aplicada em cada cluster para melhorar a eficiência na extração de dados. Portanto, como pode ser visto na Tabela 6, é possível concluir que a arquitetura proposta pode de fato melhorar a extração de dados de imagens de cupons fiscais.

Em relação aos trabalhos futuros, existe uma variedade de possíveis algoritmos que podem ser aplicados para alterar uma imagem, possibilitando o desenvolvimento de diversas outras técnicas de PDI. Dessa maneira, para trabalhos futuros outras técnicas podem ser concebidas e testadas na arquitetura, avaliando novos resultados na extração de texto. Além disso, o serviço de OCR poderia ser substituído pelo Microsoft Vision API, que também oferece o reconhecimento de caracteres via API REST. Isso permitiria uma nova análise de resultados, em que possivelmente, poderia alterar em qual cluster uma determinada técnica deveria ser aplicada.

Novas heurísticas na extração dos campos de texto também podem ser desenvolvidas a fim de melhorar a taxa de acerto. Contudo, o principal trabalho futuro talvez seja investigar os resultados obtidos pela Técnica de Compressão, isto é, compressão de imagens. Desenvolver outros tipos de compressões como PNG, WEBP e outros formatos com o objetivo de analisar novamente os resultados.

6. Referências

Aldenderfer, M., Blashfield, R. Cluster Analysis. Sage Publications, Beverly Hills, USA, 1984.

Dana, H. B., Christopher M. B. (1982). Computer Vision. Prentice Hall. ISBN 0-13-165316-4 Line Eikvil. Optical Character Recognition. https://www.nr.no/ eikvil/OCR.pdf, 1993.

Filho, O.; Neto, H. Processamento Digital de Imagens. São Paulo, Brazil: Ed. Brasport, 1999. 331 p.

Han, J., Kamber, M Data Mining: Concepts and Techniques. Morgan Kaufmann Publishers, San Mateo, CA, 2000.

GOOGLE. Cloud Vision API. Disponível em: https://cloud.google.com/vision/ . Acesso em: 03.04.2017.

Martin Ester, Hans-Peter Kriegel, Jorg Sander, Xiaowei Xu (1996). A Density-Based Algorithm for Discovering Clusters. Institute for Computer Science, University of Munich. KDD96-037.

Pang-Ning Tan, Michael Steinbach, Vipin Kumar. Introduction to Data Mining. Addison-Wesley, 2005. ISBN : 0321321367.

Pedrini, H.; Schwartz, W. Análise de Imagens Digitais: Princípios, Algoritmos e Aplicações. São Paulo, Brazil: Ed. Thomson Learning, 2007. 528 p. ISBN 978-85-221-0595-3.

Trevor, H., Robert, T., Jerome, F. (2009). The Elements of Statistical Learning - Data Mining, Inference, and Prediction, Second Edition. Springer-Verlag New York. ISBN:

978-0-387-84858-7.

Ulrike Von Luxburg (2007). A Tutorial on Spectral Clustering. 10.1.1.165.9323. Andrew Y. Ng and Michael I. Jordan and Yair Weiss (2001). On Spectral Clustering: Analysis and an algorithm. ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, 849–856. MIT Press. 10.1.1.19.8100.

WIKIPEDIA. Tesseract (software). Disponível em: https://en.wikipedia.org/wiki/Tesseract(software). Acesso em: 03.04.2017.

Anexo B – Código Fonte

Link para o código no github, juntamente com a base de imagens utilizadas: https://github.com/victorfeijo/visionary.

google_vision_ocr.py

```
from google.cloud import vision
from tinydb import TinyDB
import urllib.request


class GoogleVisionOcr():
	def __init__(self, images=[]):
	self.client = vision.Client()
	self.images = images
	self.db = TinyDB('database/db.json')

	def process_batch(self, save_table):
	count = 3200 - len(self.images)
	print(len(self.images))

	if len(self.images) + count != 3200:
		raise ValueError

	for image in self.images:
		print('=== Image count {} ==='.format(count))

		self._process_and_save(image, self.db.table(save_table), count)
		count = count + 1

	def _process_and_save(self, image, table, count):
	print('=== Processing image {} ==='.format(image['url']))

	raw_text = ''
	text_objects = []

	print('=== Getting image content from url ===')
	content = urllib.request.urlopen(image['url']).read()
	vision_image = self.client.image(content=content)
	print('=== Detecting text from vision ===')
	texts = vision_image.detect_text()

	for text in texts:
		coordinates = [(v.x_coordinate, v.y_coordinate) for v in
text.bounds.vertices]
		text_objects.append({'annotation': text.description, 'top_left':
coordinates[1], 'top_right': coordinates[0], 'bottom_left': coordinates[2],
'bottom_right': coordinates[3]})
	raw_text = texts[0].description

	print('=== Inserting data to database ===')
	table.insert({'url': image['url'], 'raw_text': raw_text, 'text_object':
text_objects})
	print('=== Done image===')
```

ocr_comparator.py

```
import arrow
from dateutil.tz import tzoffset


class OcrComparator:
	@staticmethod
	def cnpj(original, ocr):
	return original == ocr

	@staticmethod
	def total_value(original, ocr):
	original = float(original)
	ocr = float(ocr or 0)

	return ocr-1 <= original <= ocr+1

	@staticmethod
	def sale_time(original, ocr):
	if ocr == '':
		return False

	original_time = arrow.get(original)
	tz_hours = OcrComparator._time_zone_hour(original_time)
	original_time = original_time.replace(tzinfo='00:00')
	original_time = original_time.shift(hours=tz_hours)

	ocr_time = arrow.get(ocr)

	return original_time < ocr_time.shift(minutes=+5) and original_time >
ocr_time.shift(minutes=-5)

	def _time_zone_hour(arrow_date):
	if arrow_date.tzinfo == tzoffset(None, -10800):
		return -3

	return -2
```

ocr_result.py

```
from ocr.extractor.cnpj_extractor import CNPJExtractor
from ocr.extractor.total_value_extractor import TotalValueExtractor
from ocr.extractor.sale_time_extractor import SaleTimeExtractor


class OcrResult():
	def __init__(self, image):
	self.image = image

	def values(self):
	if self.image is None:
		return {}
```

```
        return {
                'cnpj': CNPJExtractor(self.image['raw_text']).extract(),
                'total_value':
TotalValueExtractor(self.image['text_object']).extract(),
                'sale_time': SaleTimeExtractor(self.image['raw_text']).extract(),
        }
```

cnpj_extractor.py

```
import re
from brazilnum.cnpj import validate_cnpj, format_cnpj


class CNPJExtractor:
        SUFIXES = ['CNPJ', 'CPJ', 'CNP', 'CNJ', 'PJ', 'NPJ']
        NUMBER_PATTERN = r"([0-9]{2}\.[0-9]{3}\.[0-9]{3}[/][0-9]{4}-[0-9]{2})"

        def __init__(self, text):
        self.text = text

        def extract(self):
        content = self._number_pattern()

        if not validate_cnpj(content):
                content = self._cnpj_pattern()

                if not validate_cnpj(content):
                return None

        return format_cnpj(content)

        def _number_pattern(self):
        content = re.sub('o', '0', self.text)
        content = re.sub(r"[^0-9.,;\/-]", '', content)
        content = re.sub(r"[,;]", '.', content)

        found = re.search(CNPJExtractor.NUMBER_PATTERN, content)

        if not found:
                return ''

        return found.group(1)

        def _cnpj_pattern(self):
        content = re.sub(r"[^A-Z$0-9\n]", '', self.text)
        content = re.sub('\n ', '', content)

        founds = [re.search(pattern, content) for pattern in self._join_regex()]
        founds = [found for found in founds if found is not None]

        if len(founds) == 0:
                return ''

        return founds[0].group(1)
```

```
	def _join_regex(self):
	return [re.compile(sufix + "([0-9]{14})") for sufix in
CNPJExtractor.SUFIXES]
```

sale_time_extractor.py

```
import re
from datetime import datetime


class SaleTimeExtractor:
	def __init__(self, text):
	self.text = text

	def extract(self):
	content = re.sub(r"A", '4', self.text)
	content = re.sub(r"O", '0', content)
	content = re.sub(r"g", '9', content)
	content = re.sub(r"[^0-9:/-]", '', content)
	found = re.search(self._sale_time_regex(), content)

	if found is None:
		return ''

	return self._parse_sale_time(found)

	def _sale_time_regex(self):
	return
r"([0-9]{2})[\/7-]([0-9]{2})[\/7-]([0-9]{4})([0-9]{2}):([0-9]{2})?:([0-9]{2})?"

	def _parse_sale_time(self, found):
	try:
		sale_time = datetime(int(found.group(3)), int(found.group(2)),
				int(found.group(1)), int(found.group(4)),
				int(found.group(5)), int(found.group(6)))

		return sale_time.strftime("%Y-%m-%dT%H:%M:%S-0000")
	except:
		return ''
```

total_value_extractor.py

```
import re


class TotalValueExtractor:
	def __init__(self, text_obj):
	self.text_obj = text_obj

	def extract(self):
	total = self.total_box()
```

```
        if len(total) == 0:
                return ''

        founds_y = self.find_on_y(total[0])
        total_values = self.total_on_founds(founds_y)

        if len(total_values) == 0:
                founds_x = self.find_on_x(total[0])
                total_values = self.total_on_founds(founds_x)

                if len(total_values) == 0:
                return ''

        return max(total_values)

        def total_box(self):
        total_regex = r"(TOTAL|TOTALR(S|$)|R(S|$)TOTAL|VALORPAGO|TOTA|OTAL)"
        total = [box for box in self.text_obj if re.match(total_regex,
box['annotation']) is not None]

        return total

        def find_on_x(self, total):
        found_values = []

        for box in self.text_obj:
                if (total['top_left'][0]-20 <= box['top_left'][0] <=
total['top_left'][0]+80) and (total['bottom_right'][0]-20 <=
box['bottom_right'][0] <= total['bottom_right'][0]+80):
                found_values.append(box)

        return found_values

        def find_on_y(self, total):
        found_values = []

        for box in self.text_obj:
                if (total['top_left'][1]-20 <= box['top_left'][1] <=
total['top_left'][1]+80) and (total['bottom_right'][1]-20 <=
box['bottom_right'][1] <= total['bottom_right'][1]+80):
                found_values.append(box)

        return found_values

        def total_on_founds(self, founds):
        found_values = [value['annotation'] for value in founds if
re.fullmatch(r"([1-9][0-9]?[0-9]?)[,.]([0-9]?[0-9]?)", value['annotation'])]
        found_values = [re.sub(r"[^0-9,.]", '', value) for value in found_values]
        found_values = [re.sub(r",", '.', value) for value in found_values]
        found_values = [float(value) for value in found_values]

        return found_values
```

cluster.py

```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.cluster import KMeans, DBSCAN, SpectralClustering

from skimage import exposure, io
from skimage.color import rgb2gray
from skimage.util.dtype import dtype_range, dtype_limits

from tinydb import TinyDB, Query
from ocr.ocr_result import OcrResult
from ocr.ocr_comparator import OcrComparator


def contrast_ratio(image, lower_percentile=1, upper_percentile=99):
	image = np.asanyarray(image)
	if image.ndim == 3 and image.shape[2] in [3, 4]:
	image = rgb2gray(image)

	dlimits = dtype_limits(image, clip_negative=False)
	limits = np.percentile(image, [lower_percentile, upper_percentile])

	return (limits[1] - limits[0]) / (dlimits[1] - dlimits[0])

def calc_cluster_data():
	db = TinyDB('./database/db.json')
	table = db.table('original_data')
	image_data = table.all()

	cluster1_table = db.table('cluster1_data')
	counter = 0
	for data in image_data:
	print('Processing image: ', counter)
	image = io.imread(data['url'])
	n_pixel = image.size / 3
	contrast = contrast_ratio(image)

	print('Number of pixels: ', n_pixel)
	print('Contrast ratio: ', contrast)

	data.update({'n_pixel': n_pixel, 'contrast_ratio': contrast})
	cluster1_table.insert(data)

	counter = counter + 1

def normalize(x_values):
	max0 = x_values[:, 0].max()
	min0 = x_values[:, 0].min()
	max1 = x_values[:, 1].max()
	min1 = x_values[:, 1].min()

	return np.array([[(value[0] - min0)/(max0 - min0), (value[1] -
min1)/(max1 - min1)] for value in x_values])
```

```
def plot_cluster(x_values, y_values):
	# Map clusters to colors

	plt.scatter(x_values[:, 0], x_values[:, 1], c=y_values)
	plt.title('KMeans')
	plt.ylabel('Nível de contraste')
	plt.xlabel('Número de pixels')

	for idx, cluster in enumerate(y_values):
	plt.annotate(str(cluster), (x_values[idx][0], x_values[idx][1]))

	plt.show()

def clusterize(cluster_data, algorithm):
	x_values = np.array([[data['n_pixel'], data['contrast_ratio']] for data
in cluster_data])
	normalized_x_values = normalize(x_values)
	y_values = []

	if algorithm == 'kmeans':
	y_values = KMeans(n_clusters=5,
init='k-means++').fit_predict(normalized_x_values)
	elif algorithm == 'dbscan':
	y_values = DBSCAN(eps=0.13,
min_samples=300).fit_predict(normalized_x_values)
	elif algorithm == 'spectral':
	y_values =
SpectralClustering(n_clusters=5).fit_predict(normalized_x_values)
	else:
	raise AttributeError

	plot_cluster(x_values, y_values)

	return y_values

def save_cluster(table, data, texts, clusters):
	for idx, d in enumerate(data):
	if clusters[idx] == -1:
		d.update({'cluster': '1', 'raw_text': texts[idx]['raw_text']})
	else:
		d.update({'cluster': str(clusters[idx]), 'raw_text':
texts[idx]['raw_text']})

	table.insert_multiple(data)

def overall_ratio(image_data):
	image_amount = len(image_data)
	ocr_results = [OcrResult(image).values() for image in image_data]

	return {
	'cnpj_ratio': len([data for idx, data in enumerate(image_data) if
OcrComparator.cnpj(data['cnpj'], ocr_results[idx]['cnpj'])])/image_amount,
	'total_value': len([data for idx, data in enumerate(image_data) if
OcrComparator.total_value(data['total_value'],
ocr_results[idx]['total_value'])])/image_amount,
	'sale_time': len([data for idx, data in enumerate(image_data) if
OcrComparator.sale_time(data['sale_time'],
```

```
ocr_results[idx]['sale_time']])]/image_amount,
        }


if __name__ == "__main__":
        db = TinyDB('./database/original.json')
        # table = db.table('cluster_data')
        # clusters = clusterize(table.all(), 'kmeans')
        # save_cluster(db.table('cluster1_original'), table.all(),
db.table('original_text').all(), clusters)
        table = db.table('cluster3_ocr')
        Cluster = Query()

        image_data = table.all()
        cluster0 = table.search(Cluster.cluster == '0')
        cluster1 = table.search(Cluster.cluster == '1')
        cluster2 = table.search(Cluster.cluster == '2')
        cluster3 = table.search(Cluster.cluster == '3')
        cluster4 = table.search(Cluster.cluster == '4')

        print('--- Overal rating ---- ', len(image_data))
        print(overall_ratio(image_data))

        print('--- Cluster 0 ---- ', len(cluster0))
        print(overall_ratio(cluster0))

        print('--- Cluster 1 ---- ', len(cluster1))
        print(overall_ratio(cluster1))

        print('--- Cluster 2 ---- ', len(cluster2))
        print(overall_ratio(cluster2))

        print('--- Cluster 3 ---- ', len(cluster3))
        print(overall_ratio(cluster3))

        print('--- Cluster 4 ---- ', len(cluster4))
        print(overall_ratio(cluster4))
```

filter1.py

```
import random
import urllib.request
from tinydb import TinyDB, Query
from google.cloud import vision
from PIL import ImageFilter, ImageEnhance, Image
from io import BytesIO
from ocr.ocr_result import OcrResult
from ocr.ocr_comparator import OcrComparator


def apply_and_send(samples, save_table, filter_data, count):
        print('=== Still lefting {} ==='.format(3200 - count))

        for idx, sample in enumerate(samples):
        print('=== Applying filter on image {} ==='.format(idx + count))
```

```
        content = urllib.request.urlopen(sample['url']).read()
        image = Image.open(BytesIO(content))
        filtered = filter1(image, filter_data)

        print('=== Getting image content from url ===')
        buffer = BytesIO()
        filtered.save(buffer, format='JPEG')
        client = vision.Client()
        vision_image = client.image(content=buffer.getvalue())
        print('=== Detecting text from vision ===')
        texts = vision_image.detect_text()

        raw_text = ''
        text_objects = []
        for text in texts:
                coordinates = [(v.x_coordinate, v.y_coordinate) for v in
text.bounds.vertices]
                text_objects.append({'annotation': text.description, 'top_left':
coordinates[1], 'top_right': coordinates[0], 'bottom_left': coordinates[2],
'bottom_right': coordinates[3]})

        try:
                raw_text = texts[0].description
        except:
                pass

        print('=== Inserting data to database ===')
        save_table.insert({'url': sample['url'],
                           'cnpj': sample['cnpj'],
                           'total_value': sample['total_value'],
                           'sale_time': sample['sale_time'],
                           'filter_data': filter_data,
                           'raw_text': raw_text,
                           'text_object': text_objects})

def filter1(image, filter_data):
        enhancer = ImageEnhance.Sharpness(image)
        filtered = enhancer.enhance(filter_data['sharpness'])
        enhancer = ImageEnhance.Brightness(filtered)
        filtered = enhancer.enhance(filter_data['brightness'])
        enhancer = ImageEnhance.Contrast(filtered)
        filtered = enhancer.enhance(filter_data['contrast'])
        enhancer = ImageEnhance.Color(filtered)
        filtered = enhancer.enhance(filter_data['color'])

        return filtered

def overall_ratio(images):
        amount = len(images)
        ocr_results = [OcrResult(image).values() for image in images]

        return {
        'cnpj_ratio': len([data for idx, data in enumerate(images) if
OcrComparator.cnpj(data['cnpj'], ocr_results[idx]['cnpj'])])/amount,
        'total_value': len([data for idx, data in enumerate(images) if
OcrComparator.total_value(data['total_value'],
ocr_results[idx]['total_value'])])/amount,
```

```
        'sale_time': len([data for idx, data in enumerate(images) if
OcrComparator.sale_time(data['sale_time'],
ocr_results[idx]['sale_time'])])/amount,
        }

def compare_results(original, filtered):
        print('=== Original Ratio ===')
        print(overall_ratio(original))

        print('=== Filtered Ratio ===')
        print(overall_ratio(filtered))


if __name__ == "__main__":
        # original = TinyDB('./database/original.json')
        db = TinyDB('./database/filter1.json')
        Cluster = Query()

        # original_images = original.table('original_ocr').all()
        table = db.table('cluster1_ocr')

        filter_data = {
        'sharpness': 1.3,
        'brightness': 1.3,
        'contrast': 1.6,
        'color': 0.2
        }

        # filtered_content = apply_and_send(original_images, filtered,
filter_data, 0)
        # compare_results(original_images, filtered)

        cluster0 = table.search(Cluster.cluster == '0')
        cluster1 = table.search(Cluster.cluster == '1')
        cluster2 = table.search(Cluster.cluster == '2')
        cluster3 = table.search(Cluster.cluster == '3')
        cluster4 = table.search(Cluster.cluster == '4')

        print('--- Cluster 0 ---- ', len(cluster0))
        print(overall_ratio(cluster0))

        print('--- Cluster 1 ---- ', len(cluster1))
        print(overall_ratio(cluster1))

        print('--- Cluster 2 ---- ', len(cluster2))
        print(overall_ratio(cluster2))

        print('--- Cluster 3 ---- ', len(cluster3))
        print(overall_ratio(cluster3))

        print('--- Cluster 4 ---- ', len(cluster4))
        print(overall_ratio(cluster4))
```

filter2.py

```
import imutils
import numpy as np
import urllib.request

from tinydb import TinyDB
from io import BytesIO
from skimage.filters import threshold_local, threshold_sauvola
from PIL import ImageOps, Image

from filter_utils import apply_and_send, overall_ratio

def block_size(image):
	if image.size <= 2073600.0:
	return 75
	elif image.size <= 4915200.0:
	return 95
	elif image.size <= 8856576.0:
	return 131
	else:
	return 151

def filter2(ocr_image):
	print(ocr_image['url'])
	# Get image from url and open in PIL
	content = urllib.request.urlopen(ocr_image['url']).read()
	image = Image.open(BytesIO(content))

	# Transform PIL image to grayscale array
	gray = ImageOps.grayscale(image)
	gray_arr = np.asarray(gray)

	block = block_size(gray_arr)

	# Apply local thresh on image
	local_thresh = threshold_local(gray_arr, block, offset=10)
	binary = gray > local_thresh
	binary = binary.astype("uint8") * 255

	# Open image on PIL again to encode and save as JPEG bytes
	binary_img = Image.fromarray(binary)
	import pdb; pdb.set_trace()  # XXX BREAKPOINT
	buffer = BytesIO()
	binary_img.save(buffer, format='JPEG')

	return buffer.getvalue()

def compare_results(original, filtered):
	print('=== Original Ratio ===')
	print(overall_ratio(original))

	print('=== Filtered Ratio ===')
	print(overall_ratio(filtered))

if __name__ == "__main__":
```

```
	db = TinyDB('database/filter2.json')

	images = db.table('sample_150_images').all()[3:150]
	filtered = db.table('sample_75_4')

	apply_and_send(filter2, images, filtered, len(filtered))
	compare_results(images, filtered.all())
```

filter3.py

```
import imutils
import cv2
import numpy as np
import urllib.request

from tinydb import TinyDB
from io import BytesIO
from google.cloud import vision
from ocr.ocr_result import OcrResult
from ocr.ocr_comparator import OcrComparator

from filter_utils import apply_and_send, overall_ratio

def filter3(ocr_image):
	print(ocr_image['url'])
	image = imutils.url_to_image(ocr_image['url'])

	img_bytes = cv2.imencode('.jpg', image, [cv2.IMWRITE_JPEG_QUALITY,
100])[1]
	if (img_bytes.size / 1000000) >= 4:
	img_bytes = cv2.imencode('.jpg', image, [cv2.IMWRITE_JPEG_QUALITY,
98])[1]

	return img_bytes.tostring()

def compare_results(original, filtered):
	print('=== Original Ratio ===')
	print(overall_ratio(original))

	print('=== Filtered Ratio ===')
	print(overall_ratio(filtered))

if __name__ == "__main__":
	original = TinyDB('database/original.json')
	filter3 = TinyDB('database/filter3.json')

	apply_and_send(filter3, images, len(table_part3))
	compare_results(images, filtered.all())
```

filter_utils.py

```
from google.cloud import vision
from ocr.ocr_result import OcrResult
from ocr.ocr_comparator import OcrComparator

def apply_and_send(filter_method, samples, save_table, count):
	for idx, sample in enumerate(samples):
	print('=== Applying filter on image {} ==='.format(idx + count))
	filtered = filter_method(sample)

	client = vision.Client()
	vision_image = client.image(content=filtered)
	print('=== Detecting text from vision ===')
	texts = vision_image.detect_text()

	raw_text = ''
	text_objects = []
	for text in texts:
		coordinates = [(v.x_coordinate, v.y_coordinate) for v in
text.bounds.vertices]
		text_objects.append({'annotation': text.description, 'top_left':
coordinates[1], 'top_right': coordinates[0], 'bottom_left': coordinates[2],
'bottom_right': coordinates[3]})

	try:
		raw_text = texts[0].description
	except:
		pass

	print('=== Inserting data to database ===')
	save_table.insert({'url': sample['url'],
				'cnpj': sample['cnpj'],
				'total_value': sample['total_value'],
				'sale_time': sample['sale_time'],
				'raw_text': raw_text,
				'text_object': text_objects})

def overall_ratio(images):
	amount = len(images)
	ocr_results = [OcrResult(image).values() for image in images]

	return {
	'cnpj_ratio': len([data for idx, data in enumerate(images) if
OcrComparator.cnpj(data['cnpj'], ocr_results[idx]['cnpj'])])/amount,
	'total_value': len([data for idx, data in enumerate(images) if
OcrComparator.total_value(data['total_value'],
ocr_results[idx]['total_value'])])/amount,
	'sale_time': len([data for idx, data in enumerate(images) if
OcrComparator.sale_time(data['sale_time'],
ocr_results[idx]['sale_time'])])/amount,
	}
```

result_analysis.py

```
from tinydb import TinyDB
from filter_utils import overall_ratio


def query_cluster(images, clusterized, cluster='0'):
	return [img for idx, img in enumerate(images) if
clusterized[idx]['cluster'] == cluster]


def print_analysis(original, cluster0, cluster1, cluster2, cluster3, cluster4):
	print('--- All images ---', len(original))
	print(overall_ratio(original))

	print('--- Cluster 0 ---- ', len(cluster0))
	print(overall_ratio(cluster0))

	print('--- Cluster 1 ---- ', len(cluster1))
	print(overall_ratio(cluster1))

	print('--- Cluster 2 ---- ', len(cluster2))
	print(overall_ratio(cluster2))

	print('--- Cluster 3 ---- ', len(cluster3))
	print(overall_ratio(cluster3))

	print('--- Cluster 4 ---- ', len(cluster4))
	print(overall_ratio(cluster4))


def original_cluster(cluster_table='cluster1_original'):
	original = TinyDB('./database/original.json')

	images = original.table('original_ocr').all()
	clusterized = original.table(cluster_table).all()

	cluster0 = query_cluster(images, clusterized, cluster='0')
	cluster1 = query_cluster(images, clusterized, cluster='1')
	cluster2 = query_cluster(images, clusterized, cluster='2')
	cluster3 = query_cluster(images, clusterized, cluster='3')
	cluster4 = query_cluster(images, clusterized, cluster='4')

	print_analysis(images, cluster0, cluster1, cluster2, cluster3, cluster4)


def filterA_cluster(cluster_table='cluster1_original'):
	original = TinyDB('./database/original.json')
	filterA = TinyDB('./database/filter1.json')

	images = filterA.table('filter1_data').all()
	clusterized = original.table(cluster_table).all()

	cluster0 = query_cluster(images, clusterized, cluster='0')
	cluster1 = query_cluster(images, clusterized, cluster='1')
	cluster2 = query_cluster(images, clusterized, cluster='2')
```

```
        cluster3 = query_cluster(images, clusterized, cluster='3')
        cluster4 = query_cluster(images, clusterized, cluster='4')

        print_analysis(images, cluster0, cluster1, cluster2, cluster3, cluster4)


def filterB_cluster(cluster_table='cluster1_original'):
        original = TinyDB('./database/original.json')
        filterB = TinyDB('./database/filter2.json')

        images = filterB.table('filter2_ocr').all()
        clusterized = original.table(cluster_table).all()

        cluster0 = query_cluster(images, clusterized, cluster='0')
        cluster1 = query_cluster(images, clusterized, cluster='1')
        cluster2 = query_cluster(images, clusterized, cluster='2')
        cluster3 = query_cluster(images, clusterized, cluster='3')
        cluster4 = query_cluster(images, clusterized, cluster='4')

        print_analysis(images, cluster0, cluster1, cluster2, cluster3, cluster4)


def filterC_cluster(cluster_table='cluster1_original'):
        original = TinyDB('./database/original.json')
        filterC = TinyDB('./database/filter3.json')

        images = filterC.table('filter3_ocr').all()
        clusterized = original.table(cluster_table).all()

        cluster0 = query_cluster(images, clusterized, cluster='0')
        cluster1 = query_cluster(images, clusterized, cluster='1')
        cluster2 = query_cluster(images, clusterized, cluster='2')
        cluster3 = query_cluster(images, clusterized, cluster='3')
        cluster4 = query_cluster(images, clusterized, cluster='4')

        print_analysis(images, cluster0, cluster1, cluster2, cluster3, cluster4)


if __name__ == "__main__":
        print('--- ORIGINAL SEND ---')
        original_cluster(cluster_table='cluster1_original')

        print('--- FILTER A ---')
        filterA_cluster(cluster_table='cluster3_original')

        print('--- FILTER B ---')
        filterB_cluster(cluster_table='cluster1_original')

        print('--- FILTER C ---')
        filterC_cluster(cluster_table='cluster1_original')
```