NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTAMENTO DE INFORMÁTICA

GONÇALO FELICIANO

Licenciado em Engenharia informática

CONCEÇÃO E DESENVOLVIMENTO DE UMA APLICAÇÃO IOS PARA ELIMINAÇÃO ASSISTIDA DE FOTOGRAFIAS SEMELHANTES

MESTRADO EM ENGENHARIA INFORMÁTICA

Universidade NOVA de Lisboa
Setembro, 2021

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTAMENTO
DE INFORMÁTICA

CONCEÇÃO E DESENVOLVIMENTO DE UMA APLICAÇÃO IOS PARA ELIMINAÇÃO ASSISTIDA DE FOTOGRAFIAS SEMELHANTES

GONÇALO FELICIANO

Licenciado em Engenharia informática

Orientador: Fernando P. R. Birra
Assistant Professor, FCT | NOVA University Lisbon

Coorientador: João M. S. Lourenço
Associate Professor, FCT | NOVA University Lisbon

Júri

Presidente: Maria Cecília L. Gomes
Assistant Professor, FCT | NOVA University Lisbon

Arguente: Pedro M. F. Centieiro
Magycal

Orientador: Fernando P. R. Birra
Assistant Professor, FCT | NOVA University Lisbon

MESTRADO EM ENGENHARIA INFORMÁTICA

Universidade NOVA de Lisboa
Setembro, 2021

Conceção e Desenvolvimento de uma Aplicação iOS para Eliminação Assistida de Fotografias Semelhantes

Este documento foi gerado com o processador (pdf/Xe/Lua)LaTeXe o modelo NOVAthesis (v6.7.0) [31].

Resumo

A redução de preço das máquinas fotográficas DSLR e a massificação dos smartphones com excelente capacidade fotográfica levou à massificação da fotografia digital. Esta massificação, aliada ao custo quase nulo das fotografias, originou um incremento no número de fotografias capturadas onde, por vezes, o mesmo assunto é capturado múltiplas vezes, dando origem a várias fotografias semelhantes. Com frequência o fotógrafo planeia mais tarde filtrar aquelas fotografias e a suas repetições para preparar uma galeria final. No entanto, esta filtragem nem sempre acontece, seja por falta de tempo ou pela incerteza e insegurança de se estar a eliminar uma imagem que afinal se deveria manter. Não remover estas imagens sem interesse da galeria do dispositivo móvel origina problemas variados, tal como a falta de espaço ou a desorganização da galeria de imagens no smartphone.

Nesta dissertação propõe-se uma arquitetura para uma aplicação iOS para facilitar o processo de singularização de imagens em iPhone. O processo proposto é composto por quatro etapas: criação de conjuntos com imagens semelhantes, ordenação automatizada dos grupos gerados, marcação do grupo como tratado e, por fim, finalização do trabalho, onde as melhores fotografias dos grupos marcados como processados e movidas para a galeria da aplicação, sendo as restantes ocultadas.

Para facilitar o processo de criação dos grupos de imagens semelhantes e de avaliar a sua qualidade técnica, esta dissertação também apresenta um estudo sobre potenciais algoritmos para a realização dessas tarefas.

Palavras-chave: Desenvolvimento de Aplicações Móveis, iOS, Deteção de Imagens Similares, Gestão de Fotografias

Abstract

The price reduction of DSLR cameras and the massification of smartphones with excellent photographic capacity led to the massification of digital photography. This massification, allied to the almost null cost of photographs, resulted in an increase in the number of photographs captured where, sometimes, the same subject is captured multiple times, giving rise to several similar photographs. Often the photographer later plans to filter those photographs and their repetitions to prepare a final gallery. However, this filtering does not always happen, due to the lack of time or to the uncertainty and insecurity associated with eliminating an image that, after all, should be maintained. Not removing these repeated images from the mobile device gallery leads to various problems, such as lack of space or a disorganized image gallery on the smartphone.

This dissertation proposes an architecture for an iOS application to facilitate the process of image singularization on iPhone. The proposed process consists of four steps: creation of clusters with similar images, automated ordering of the photographs in the generated clusters, marking the cluster as processed and, finally, finalizing the work, where the best photographs of the cluster marked as processed and moved to the gallery application, the rest being hidden.

Aiming to ease the process of creating groups of similar images and to assess their technical quality, this dissertation also presents a study of potential algorithms for carrying out these tasks.

Keywords: Mobile Application Development, iOS, Similar Image Detection, Photo Management.

Keywords: Mobile Application Development, iOS, Similar Image Detection, Photo Management.

Índice

Índice de Figuras

Índice de Tabelas

1

Introdução

1.1 Contexto

A evolução da tecnologia na área da fotografia digital, aliada à utilização generalizada de smartphones e à massificação da utilização das redes sociais, que são hoje utilizadas como plataformas de partilha de momentos e/ou promoção de serviços e produtos, conduziram ao aumento do número de pessoas que se tornaram fotógrafos amadores.

A possibilidade de captar fotografias com qualidade aproximada à das câmaras fotográficas através de um telemóvel (sem custos acrescidos), a consciencialização da importância e do impacto que a qualidade das fotografias têm naqueles que irão ser os consumidores, fez com que as pessoas se preocupassem com a apresentação de imagens apelativas para captar a atenção dos outros.

Desta forma, tornou-se frequente e habitual a captura de várias fotografias repetidas, tiradas no mesmo contexto, com o intuito de posteriormente se escolherem as melhores e que estas sejam suficientemente boas para serem guardadas e apresentadas.

1.2 Descrição do problema

Nas câmaras DSLR a falta de espaço de armazenamento não é relevante, pois é resolvida com a substituição de um cartão de memória. Já nos dispositivos móveis o cenário é bem diferente, pois aquando da ocorrência de falta de espaço existem duas opções possíveis de resolução do problema: ou o armazenamento on-line (Dropbox, Google drive, etc.) ou, em alguns casos, a substituição do cartão de memória. No entanto, esta última opção é cada vez menos uma verdadeira opção, pois a tendência atual dos telemóveis é para não aceitarem cartões de memória adicionais.

A quantidade de fotografias repetidas ou idênticas impõe a necessidade de comparar e selecionar a melhor. Na captura DSLR a seleção é feita no pós-processamento, na captura móvel a seleção só poderá ser feita através da comparação imagem a imagem.

Antes deste processo é necessário definir o conceito de Melhor Imagem.

Existem três formas de definir a melhor imagem: critérios definidos pelo fotógrafo, critérios técnicos e a fusão dos dois.

- Critérios definidos pelo fotógrafo — O utilizador define qual a melhor fotografia com base nos seus critérios pessoais ou artísticos;
- Critérios técnicos — São as fotografias tecnicamente corretas (exposição correta, sem estarem tremidas ou desfocadas, com baixos níveis de ruído, bom enquadramento);
- Fusão dos dois — O fotógrafo escolhe as melhores imagens de entre as fotografias tecnicamente corretas.

Para se encontrar a melhor imagem é necessário agrupá-las em grupos de semelhança e sujeitá-las aos critérios do utilizador, sendo este quem define qual a melhor imagem.

Na fotografia móvel e assumindo que utilizamos a galeria nativa para fazer a filtragem, o processo passa por:

1. Abrir uma fotografia e analisá-la, tentando reter tudo o que tem de bom e o que tem de mau;
2. Fechar a fotografia;
3. Abrir uma segunda fotografia e repetir o processo da primeira;
4. Decidir se a segunda é melhor que a primeira apenas confiando nos dados que retivemos;
5. Decidir qual fotografia a remover e remover a mesma.

Este workflow gera alguns potenciais problemas.

- Nas dimensões reduzidas dos smartphones nem sempre se consegue perceber se a fotografia apresenta falhas, tais como a falta de foco, nitidez ou o excesso de ruído;
- Não se consegue, de forma otimizada, comparar duas fotografias lado a lado, de forma a perceber qual delas tem o melhor enquadramento;
- Quando apagamos uma fotografia essa ação é permanente, ou seja, a fotografia é apagada, não nos permitindo recuar na decisão.

Se pensarmos no mundo da fotografia DSLR, o workflow é mais fácil no sentido em que pode ser aberta mais de uma imagem em simultâneo, facilitando o processo de comparação. Sendo ainda possível separar aquelas que se pretende posteriormente remover, permitindo assim uma possibilidade de revisão. As aplicações que fazem a gestão das bibliotecas de imagem permitem aos utilizadores definirem os seus próprios workflows de processamento.

Em ambos os casos a filtragem tem que começar e terminar no mesmo dispositivo. A possibilidade de migração entre dispositivos será um benefício para o utilizador podendo optar por um equipamento que lhe permita maior celeridade e qualidade ao seu trabalho.

O processo de filtragem e escolha das fotografias existentes na galeria nativa do smartphone, para além de tedioso, acarreta riscos para o utilizador, pois pode facilmente vir a apagar uma fotografia que pretende manter, sabendo que esta ação é irreversível, poderá levar o utilizador a não iniciar sequer o processo de seleção.

1.3 Solução proposta

Para dar resposta ao problema levantado na secção 1.2 foi criada a aplicação *photo|uniq*. Esta aplicação tem versões para várias plataformas e está focada no apoio ao utilizador na singularização de imagens. Por outras palavras, o objetivo da aplicação é ajudar o utilizador a selecionar as melhores imagens dentro de um grupo de imagens semelhantes, para ajudar o utilizador na escolha de imagens que poderá remover.

Uma das características que diferencia a *photo|uniq* das demais aplicações no mercado é a possibilidade ser feita a comparação de imagens lado a lado, onde, quando o utilizador foca um motivo numa das imagens, escalando a imagem nesse ponto, a segunda imagem escala no mesmo motivo.

Atualmente já existem implementações do *photo|uniq* para Android e macOS.

Esta dissertação foca-se no desenvolvimento da versão iOS da aplicação *photo|uniq*. A versão iOS está dividida em duas áreas: uma área de trabalho para onde são importadas as imagens de modo a serem trabalhadas. Na área de trabalho as imagens estão organizadas por pastas, sendo o conteúdo das pastas grupos de semelhança, isto assumindo que o utilizador já calculou os grupos de semelhança, e uma galeria contendo todas as imagens, consideradas pelo utilizador como melhores, estas são apenas passadas para a galeria após finalizado o trabalho nas pastas do workspace. A galeria está organizada em álbuns.

Para concretizar a organização descrita foram identificados seis processos/etapas.

1. **Importação de imagens** — Esta é a primeira etapa a ser realizada e é a etapa responsável por importar as imagens da biblioteca do iOS (app Photos) para a aplicação *photo|uniq*. Em ambiente iOS existem três possibilidades para importar imagens: todas as imagens da biblioteca; o conteúdo de uma pasta; e imagens independentes. A aplicação garante que uma imagem não pode ser importada duas vezes.

2. **Criação de grupos de semelhança** — Após as imagens estarem importadas é necessário segmentar as imagens em grupos de semelhança, para tal é necessário calcular os keypoints de todas as imagens e correlacioná-los. Neste ponto a aplicação garante que uma imagem está num e só um grupo de semelhança.

3. **Ordenação dos grupos de semelhança** — Sendo o principal objetivo da aplicação ajudar o utilizador a selecionar a melhor imagem, é necessário de alguma forma

ordenar as imagens. A versão iOS disponibiliza duas formas de ordenação, sendo uma manual, onde cabe ao utilizador ordenar manualmente as imagens, e a outra automatizada, com base num algoritmo pré-definido. Esta última ordenação apenas sugere ao utilizador qual será potencialmente a melhor imagem, cabendo sempre ao utilizador a decisão final.

4. **Comparação de imagens lado a lado** — Para facilitar o utilizador tanto a fazer a ordenação manual das imagens como a confirmar se concorda com a ordenação automática, o *photo|uniq* iOS, dispõe de um assistente para a comparação de imagens que permite fazer a comparação de duas imagens lado a lado, onde quando uma imagem é ampliada a segunda é ampliada no mesmo motivo.

5. **Seleção da melhor imagem** — Após a confirmação por parte do utilizador que imagem ou imagens quer manter cabe a este marcá-las como "a manter";

6. **Finalizar o trabalho** — Quando o utilizador estiver satisfeito com as seleções feitas poderá marcar o grupo de semelhança e a respetiva pasta como processado. Quando o faz, as imagens marcadas como melhores são passadas para a galeria *photo|uniq*, uma galeria local à aplicação que contém apenas as imagens selecionadas pelo utilizador. Podendo numa versão futura da aplicação as seleções feitas serem espelhadas para a biblioteca do equipamento (aplicação Fotografias).

A primeira versão do *photo|uniq* para iOS foi implementada para funcionar totalmente off-line, porém, está preparada para receber um servidor de processamento de imagens, responsável por todo o processamento sobre as imagens onde se pode destacar cálculo e correlação de keypoints, análise de qualidade e deteção de imagens semelhantes.

1.4 Organização do documento

Este documento está organizado em 5 capítulos. O primeiro capítulo, este, visa contextualizar o leitor sobre o tema da dissertação, seguindo-se a apresentação da questão à qual ste trabalho dará resposta, terminado com a descrição sumária da solução proposta.

O segundo capítulo, *Trabalho e tecnologias relacionadas*, começa por apresentar uma análise critica sobre algumas das aplicações para singularização de imagens disponíveis para iOS, após esta análise feita é introduzido o funcionamento da versão iOS da aplicação *photo|uniq*, onde é detalhado como esta executa as principais funcionalidades. Estando apresentadas as principais funcionalidades da aplicação serão descritas as principais alternativas para a sua implementação, tanto ao nível de frameworks como ao nível de algoritmos, por fim são apresentadas diretrizes da Apple para a criação de interfaces gráficas.

No terceiro capítulo, *A aplicação photo|uniq*, serão apresentados os detalhes da implementação da versão iOS da aplicação *photo|uniq*, onde se pode destacar a arquitetura da

aplicação, modelo de dados, é também apresentada a interface e as principais decisões tomadas durante o desenvolvimento da aplicação.

No quarto capítulo, *Validação da solução*, são apresentados e analisados dados sobre o desempenho da aplicação, nomeadamente os custos temporal e energético, e o consumo de memória e processador das principais funcionalidades.

No quinto capítulo, *Considerações finais*, é feito um resumo de tudo o que foi feito, assim como uma análise critica à globalidade do trabalho desenvolvido, terminando com algumas indicações do que poderá ser feito em versões futuras da aplicação.

2

Trabalho e Tecnologias Relacionadas

2.1 Aplicações para singularização de fotografias

2.1.1 Importação de imagens

Uma das funcionalidades mais importantes das aplicações que lidam com imagens, é a forma como estas importam imagens.Nas aplicações analisadas foram detetadas duas possibilidades para a importação de imagens: importação simultânea de todas as imagens, funcionalidade encontrada em todas as aplicações testadas; importação de pastas existentes na aplicação Fotografias. Esta abordagem à importação de imagens apenas foi na Ducl Duplicate Photo Cleaner[1]. Na implementação da Ducl Duplicate Photo Cleaner[1] para importação de pastas, não é possível importar mais de uma pasta em simultâneo. Existe uma terceira abordagem, importação de imagens singulares, abordagem que não foi encontrada em nenhuma das aplicações testadas.

2.1.2 Agrupamento de Imagens

Partindo do pressuposto que o objetivo das aplicações analisadas é singularizar imagens, por outras palavras selecionar ou ajudar o utilizador a selecionar apenas uma imagem dentro de um grupo de semelhança, é necessário em primeiro lugar agrupar as imagens, nas aplicações testadas. Foram detetados vários métodos de agrupamento de imagens, onde se destaca: Agrupamento de imagens similares, onde as imagens similares são colocadas no mesmo grupo, funcionalidade detetada em todas as aplicações testadas, com vários níveis de sucesso; imagens tecnicamente incorretas. É nesta categoria que são colocadas as imagens tremidas ou desfocadas. Esta funcionalidade foi detetada em aplicações como De-dupe[25], Clean Doctor - Clean Storage+[48] ou Cleaner One [45]. Uma das razões que leva ao agrupamento de imagens desfocadas e tremidas num grupo independente é que uma imagem tremida ou desfocada não é facilmente recuperável em pós-processamento; Imagens duplicadas, neste grupo são agrupadas imagens e são cópias exatas entre si, esta funcionalidade apenas foi detetada na Clean Doctor - Clean Storage+ [48]; Capturas de

ecrã, neste grupo são agrupadas todas as capturas de ecrã que o utilizador faz ao longo do tempo, esta funcionalidade foi detetada em aplicações como: Gemini Photos[33], Cleaner One[45] ou De-dupe[25], esta funcionalidade permite juntar todas as capturas de ecrã no mesmo grupo.

Apenas a Ducl Duplicate Photo Cleaner[1] permite parametrizar o algoritmo que define se duas imagens são semelhantes, podendo o utilizador definir a percentagem de similaridade para que duas imagens sejam consideradas semelhantes.

2.1.3 Seleção da melhor imagem

Após as imagens estarem agrupadas é necessário definir a sua ordenação. Após os grupos estarem gerados é necessário definir qual a melhor imagem, sendo uma das estratégias possíveis para que uma aplicação defina qual a melhor imagem é ordenar os grupos de semelhança. Apenas a Clean Doctor - Clean Storage+[48] e Ducl Duplicate Photo Cleaner[1] não indicam qual, para a aplicação é considerada a melhor imagem.

Independentemente de ser a aplicação a selecionar qual a melhor imagem cabe ao utilizador decidir se concorda com essa seleção, podendo este selecionar mais imagens que considere como melhores assim como alterar a seleção automática. Nenhuma das aplicações testada disponibiliza um assistente que ajude o utilizador a verificar se a seleção feita pela aplicação está correta.

2.1.4 Persistência de dados

A análise de imagens é um processo computacionalmente exigente, tanto ao nível de hardware como em consumo de tempo, levando a que as aplicações de análise de imagens, caso nada seja feito para mitigar o consumo de recursos e tempo, tenham potencialmente um fraco desempenho. Esta questão é especialmente relevante nas plataformas moveis, uma vez que apesar de terem cada vez mais recursos, estes ainda são muito limitados e a exigência das aplicações evolui da mesma forma que os recursos evoluem.

Existem várias estratégias para melhorar o desempenho desta gama de aplicações. Um das estratégias mais comum é salvar os valores já calculados, onde ao invés de calcular os valores sempre que são necessários, os valores são calculados apenas uma vez. Esta funcionalidade apenas foi detetada na Gemini Photos [33].

2.1.5 Síntese

Com base na tabela 2.1 e nos pontos anteriores, pode se verificar que as aplicações testadas são de alguma forma limitadas.

Apesar de aplicações como Gemini Photos[33] e De-dupe[25] terem um assistente para a seleção automática da melhor fotografia não é possível parametrizar os critérios que levam a essa seleção, por consequência, podem levar a que a melhor fotografia selecionada pela aplicação não seja a melhor fotografia para o utilizador. Esta falta de parametrização

leva a outra questão, caso a aplicação detete duas fotografias igualmente boas, tem duas formas de atuar: ou deixa as duas por remover ou faz uma seleção aleatória, caso opte por deixar as duas fotografias desmarcadas cabe ao utilizador decidir qual quer manter.

Sabendo que o utilizador deve ter o controlo total sobre o que a aplicação faz, independentemente se teve de selecionar qual é para si a melhor imagem, ou apenas confirmar a seleção automatizada, é importante existir uma workflow otimizado, que o ajude nesse processo. Este processo não foi encontrado em nenhuma das aplicações testadas.

De forma a optimizar o processo de seleção da melhor imagem aplicações como Gemini Photos[33] e Cleaner One [45], segmentam as imagens desfocadas, contudo nenhuma das aplicações testadas segmentou imagens com outras falhas técnicas (ruído elevado, erros de exposição, motion blur).

Em aplicações como Smart Cleaner - Clean Storage[14] aquando da criação dos grupos de semelhança, as imagens consideradas não melhores, são automaticamente selecionadas, mas seguindo a regra que o utilizador deve ter controlo absoluto sobre o que a aplicação faz, esta seleção pode ser alterada. Após a alteração, é sempre possível voltar à seleção original. Nas aplicações que informam o utilizador qual a melhor imagem de cada grupo de semelhanças mas não selecionam automaticamente, é possível selecionar as imagens a remover, tanto de forma automatizada como manual mas se o utilizador começar a fazer a seleção manual e posteriormente optar pela seleção automática, esta sobrepõe-se ás escolhas do utilizador selecionando todas as fotos marcadas como não melhores.

Quando se trabalha numa aplicação móvel, é natural que se façam várias interrupções, seja porque se aproveita os tempos mortos, seja porque a bateria está no fim ou simplesmente porque a aplicação fecha inesperadamente. Sabendo isto, é importante que uma aplicação guarde o seu estado antes de fechar, de forma a permitir que o utilizador nunca perca nenhum dos processamentos anteriormente feitos. De todas as aplicações testadas, esta funcionalidade apenas está disponível na Gemini.

Nenhuma das aplicações testadas permite continuidade multi-dispositivos, ou seja, quando existe continuidade esta é apenas local.

Na maioria das aplicações testadas ao se remover uma imagem esta é removida do dispositivo, não dando ao utilizador a possibilidade de revisão das imagens que irá remover, contudo esta questão, em iOS, não é catastrófica, uma vez que após removidas as imagens podem ser recuperadas até trinta dias depois.

Uma questão importante na performance da aplicação é se esta guarda os dados já processados, por exemplo, os grupos de semelhança já calculados. Na maioria das aplicações, com excepção do Gemini Photos, todos os dados são processados sempre que a aplicação é aberta, ou seja todos os grupos de semelhança têm de ser calculados cada vez que a aplicação é iniciada. Aplicação como Smart Cleaner - Cleanner [14], têm este processo relativamente otimizado, uma vez que vai mostrando os grupos de semelhança à medida que os vai calculando.

Tabela 2.1: Tabela síntese das funcionalidades/limitações.

Aplicações	Selecção da melhor imagem	Parametrizar algoritmo	Persistência	Deteção de features	Deteção de fotografias tecnicamente incorrectas	Otimização da seleção manaul da melhor fotografia
De-dupe	✓	✗	✗	✗	✓	✗
Clean Doctor Clean Storage+	✗	✗	✗	✗	✗	✗
Ducl duplicate photo cleaner	✗	✓	✗	✗	✗	✗
Cleaner One	✓	✗	✗	✗	✓	✗
Gemini Photos	✓	✗	✓	✗	✓	✗
Smart Cleaner Clean Storage	✓	✗	✗	✗	✗	✗

2.2 A aplicação *photo|uniq*- iOS

2.2.1 Estrutura interna

A aplicação divide-se em dois componentes: Workspace e Galeria *photo|uniq*.

É no workspace a área onde será realizado todo o trabalho de criação de grupos de semelhança por forma a serem posteriormente ordenados com o intuito de selecionar as imagens a manter.

O conteúdo do workspace está organizado em pastas, sendo o conteúdo das pastas organizado em grupos de semelhanças, caso existam imagens que ainda não foram analisadas estas aparecem diretamente na pasta, mas, na verdade, estão dentro de um grupo de semelhanças default, por fim dentro de um grupo de semelhanças está um conjunto de imagens semelhantes entre si.

A galeria *photo|uniq* é o componente para onde são transferidas as imagens após processadas, e está organizado em álbuns, sendo o conteúdo dos álbuns de imagens.

2.2.2 Importação de imagens

Sendo o objetivo da aplicação a singularização de imagens é necessário que o utilizador importe imagens para dentro da aplicação de forma a que as consiga trabalhar.

Partindo do pressuposto que o iOS é um sistema seguro, onde o utilizador tem controlo quase total sobre os dados aos quais uma aplicação pode aceder, o primeiro passo será autorizar a aplicação a aceder ás imagens presentes na biblioteca do iOS.

Existem três possíveis opções de autorização: não autorizado, autorizado a aceder a imagens selecionadas ou autorizado a aceder a todas as imagens. A autorização de acesso à galeria é pedida na primeira abertura da aplicação, caso o utilizador pretenda mudar o nível de autorização, essa alteração apenas poderá ser feita nas definições da aplicação presentes na central de definições do iOS.

Estando autorizado o acesso à biblioteca é possível importar imagens de três formas: importar todas as imagens, onde todas as imagens presentes na biblioteca são importadas para pasta default do workspace; importar as imagens contidas numa pasta criada pelo utilizador, sendo as pastas nativas do iOS ignoradas. Ao ser importada uma pasta, será criada uma pasta no workspace com o mesmo nome e conteúdo da pasta importada, ou importar imagens de forma independente, onde o utilizador pode escolher as imagens que pretende importar, caso sejam importadas imagens individuais, estas serão adicionadas à pasta default.

Em qualquer um dos modos de importação de imagens não é possível importar duas vezes a mesma imagem, caso o utilizador tente importar uma pasta já importada simplesmente não importa. O facto de não importar duas vezes a mesma imagem não significa que não importe duas imagens iguais, a importação de imagens iguais será possível caso existam várias cópias da mesma imagem na biblioteca.

2.2.3 Gerar grupos de semelhança

Estando as imagens devidamente importadas, o utilizador poderá, de imediato, gerar os grupos de semelhança. Ao gerar os grupos de semelhança é criado um grupo individual para cada conjunto de imagens similares, garantindo que não existem grupos de semelhança contidos em outros, da mesma forma que uma imagem está num e só num grupo de semelhança.

Existe um passo opcional antes da geração de grupos de semelhança, caso o utilizador tenha importado todas as imagens ou imagens individuais, caso pretenda, poderá separar as imagens por várias pastas, uma das vantagens desta organização é permitir ao utilizador separar as imagens por categorias, por exemplo, se o utilizador fez uma viagem pela Europa pode separar as imagens por país.

Dentro de cada grupo de semelhança o utilizador pode ordenar as imagens, seja manualmente ou de forma automática; dar início à comparação de imagens lado a lado; mover imagens para dentro do grupo e por fim marcar o grupo como tratado.

2.2.4 Processo seleção da melhor imagem

Nesta secção serão detalhadas as principais etapas do processo de seleção da melhor imagem dentro de um grupo de semelhança, desde a sua ordenação à finalização do processo onde as imagens são movidas para a galeria *photo|uniq*.

As etapas descritas são:

- Ordenação de imagens -Definição da ordenação das imagens. Esta ordenação pode ser automática, onde as imagens são ordenadas como base na sua qualidade ou manual, onde será o utilizador a definir o critério de ordenação.

- Comparação manual de imagens - Permite ao utilizador confirmar que a ordenação definida no ponto anterior está correta através da comparação de pares de imagens lado a lado, sendo possível alterar a ordem das imagens.

- Marcar Imagens como favoritas - Ao marcar uma imagem como favorita garante que quando a pasta e o grupo forem processados, essa imagem é passada para a galeria *photo|uniq*.

- Finalizar a seleção das melhores imagens dentro de um grupo de imagens - A última etapa consiste em marcar um grupo de semelhança como processado de forma a que, quando a pasta onde o grupo está inserido for processado, o grupo seja incluído no processamento.

- Processar uma pasta - ao marcar uma pasta como processada, todo o seu conteúdo previamente processado é movido para a galeria *photo|uniq*.

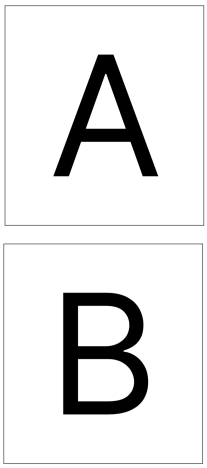


Figura 2.1: Posicionamento das imagens na vista de homografia.

2.2.4.1 Ordenação de imagens com base na sua qualidade

Sendo o objetivo da aplicação *photo|uniq* ajudar o utilizador a selecionar a melhor imagem é necessário ordenar as imagens com base na sua qualidade. Para definir essa ordenação existem duas possibilidades: ordenação automática, recorrendo a algoritmos como o BRISQUE, que atribui um valor global à qualidade de cada imagem, permitindo dessa forma ordenar as imagens, ou recorrendo à ordenação manual, onde o utilizador tem a possibilidade de alterar a posição das imagens atrás de drag and drop, ordenando assim as imagens com base nos seus critérios pessoais.

Ordenar imagens apenas olhando para uma miniatura nem sempre é fácil, e ver imagem a imagem não facilita o processo, de forma a facilitar este processo a *photo|uniq* permite a comparação de imagens lado a lado que, quando se aplica zoom numa das imagens a outra é ampliada no ponto equivalente.

2.2.4.2 Comparação manual imagens

A comparação de imagens lado a lado é o ponto mais distintivo da *photo|uniq*, uma vez que esta funcionalidade não foi encontrada em nenhuma outra aplicação.

Na vista de comparação de imagens lado a lado existem dois estados: com a escala aplicada e sem a escala aplicada, sendo o estado inicial sem escala aplicada.

Sem escala aplicada:

Não tendo escala aplicada é possível ao utilizador mudar as imagens que estão a ser comparadas, sendo garantido que as duas imagens são sempre diferentes.

Quando a escala das imagens tem o valor de um (sem escala), é possível alterar as imagens que estão a ser comparadas, quando uma imagem é alterada, em situação alguma é possível comparar uma imagem com ela própria.

À parte da alteração de imagens outra funcionalidade desta vista é promover ou despromover uma imagem, isto é, assumindo que a imagem de cima é a imagem A e a de baixo é a imagem B 2.1, a imagem A pode ser despromovida em relação a B, e a B pode ser promovida em relação à imagem A. Ao promover uma imagem esta é movida para a posição anterior a imagem A, no caso da despromoção a imagem despromovida é movida à posição seguinte a imagem B.

Por exemplo, se a lista de imagens for:

```
[img1, img2, img3, img4, img5]
```

Assumindo que tem a `img2` na posição A e `img4` na posição B, ao promover a imagem B a lista fica com a ordenação:

```
[img1, img4  img2, img3, img5]
```

já no caso da despromoção da imagem A e lista ficar com a ordenação

```
[img1, img3  img4, img2, img5]
```

Com escala aplicada:

Ao escalar uma imagem, enquanto a escala for superior a um, a imagem à qual se aplica a escala, torna-se a imagem pivot, sendo a outra a imagem subordinada, a qual também escalada na mesma proporção.

A escala é aplicada num ponto selecionado pelo utilizador na imagem pivot, sendo a imagem subordinada ampliada no ponto equivalente da imagem pivot.

2.2.4.3 Marcar Imagens como favoritas

Ao marcar uma imagem como favorita dá indicação à aplicação para a manter quando o grupo for marcado como processado.

Quando uma imagem é marcada como favorita esta será passada para a primeira posição no grupo de semelhança, caso um grupo tenha mais de uma favorita a primeira imagem será sempre a última marcada como favorita, No caso de uma imagem ser desmarcada como favorita esta passa para a primeira posição das não favoritas

2.2.4.4 Finalizar a seleção das melhores imagens dentro de um grupo de imagens

Estando todas as imagens favoritas de um grupo de semelhança marcadas é necessário indicar que o trabalho no grupo em questão já está terminado, para tal é necessário marcar um grupo como finalizado.

Quando um grupo é marcado como finalizado, todas as imagens não favoritas, ou seja, as que o utilizador pretende descartar são ocultadas, caso o utilizador mude de ideias e pretenda fazer novas alterações ao grupo de semelhança, por exemplo, alterar as imagens favoritas, é possível desmarcar o grupo de semelhança como finalizado, sendo as imagens ocultas (não favoritas), mostradas novamente.

2.2.4.5 Processar uma pasta

O processamento de uma pasta garante que todas as imagens marcadas como favoritas dentro dessa pasta serão passadas para a galeria *photo|uniq*.

Este processamento só afeta os grupos de semelhança marcados como finalizados, sendo os restantes ignorados até serem marcados como finalizados e a pasta processada novamente.

Durante o processamento das imagens estas são movidas para a galeria *photo|uniq* para um álbum com o mesmo nome da pasta no workspace.

2.2.5 Definições

Nas aplicações iOS as definições das aplicações podem estar localizadas na aplicação ou na aplicação definições. Existem razões para a seleção de cada uma das opções, uma das razões que levam a que as definições estejam dentro da própria aplicação é estas serem alteradas com regularidade de forma a alterar o funcionamento da aplicação enquanto está a correr, por seu lado caso as definições sejam apenas configurações da aplicação, como é o caso do seu idioma, as definições podem estar na aplicação de definições. Estas duas abordagens podem ser utilizadas em simultâneo.

No caso da aplicação *photo|uniq*, uma vez que as definições são apenas configurações da aplicação, que após definidas não existe necessidade de serem alteradas, as definições estão na aplicação de definições do iOS.

2.3 Frameworks de visão computacional

2.3.1 OpenCV

O OpenCV[38] é uma biblioteca Open source de visão computacional e machine learning [38], foi oficialmente lançada em 1999, contudo apenas em 2006 viu a sua primeira versão final [47]. O OpenCV foi originalmente escrito em C++ e posteriormente criadas interfaces para Java, Python, Matlab.

Sendo o openCV uma biblioteca de visão computacional pode ser usado na análise e processamento de imagens, no caso do *photo|uniq* pode ser usado, entre outras funcionalidades, para deteção de keypoints e respectiva correlação entre eles.

Apesar de estar escrita em C++ existe uma build do openCV para iOS, esta build apresenta dois problemas: não é diretamente compatível com Swift como tal deve ser usada em ficheiros Objective-C++ e posteriormente acedidas pela aplicação em Swift; o que obriga à utilização da framework AssetsLibrary [41] que se encontra deprecated [3] tendo sido substituída pela framework PhotoKit.

Tal como referido anteriormente para se usar o openCV é necessário criar uma classe Objective-C++ (*.mm) o respetivo header, e criar o bridging header [7] de forma a ser possível utilizar código Objective-C para Swift.

2.3.2 Apple Vision

Em 2017 a Apple lançou a framework de visão computacional com facilidade de integração com a framework de aprendizagem automática da Apple o Core ML, o framework Vision [11] permite executar uma variedade de tarefas relacionadas com imagens ou vídeos. No âmbito deste trabalho é possível fazer a análise de similaridade com base em Feature Points [2], permitindo assim computar a diferença entre duas imagens. Recorrendo a class VNHomographicImageRegistrationRequest [12] é possível gerar a matriz de homografia de forma a alinhar o conteúdo de duas imagens.

Para a seleção da melhor imagem, como visto anteriormente, é necessário analisar o foco, motion blur, estimação de ruído e análise de cor. Para análise de cor, tanto a sua temperatura como a exposição da imagem, a mesma pode ser inferida através da análise de histogramas, que podem ser obtidos recorrendo a classe Histogram da framework Accelerate [5], já no campo da deteção de foco e motion blur pode-se recorrer ao modelo de aprendizagem automática para classificar as imagens [4], os classificadores necessários, são facilmente treinados, recorrendo ao assistente do Xcode criado para o efeito.

2.3.3 Discussão

Tanto o openCV com o vision são tecnologias de visão computacionais competentes cada uma com seus prós e contras. Devido à sua maturidade o openCV tem maior quantidade de documentação disponível , sendo uma tecnologia mais madura e amplamente utilizada certamente qualquer problema que exista, já alguém o enfrentou como tal torna-se mais simples achar a solução a qualquer problema que possa surgir durante a implementação.

Um dos contras do OpenCV é ter de ser programado em Objective-C, como tal não é possível salvar diretamente alguns dos valores gerados pelo OpenCV numa aplicação swift tendo de ser necessário criar um wrapper que transforme um valor obtido pelo OepnCV num tipo de dados válido em Swift, não obstando a necessidade de criação wrapper, o swift tem uma velocidade de execução mais rápida que o Objective-C[24][16].

Sendo o Vision uma framwork de um nível mais alto dispõe de várias funcionalidades já implementadas que no openCV teriam de ser feitas manualmente, porém o facto de já estarem pré-implementados não permitem presonalização.

Uma das grandes vantagens do OpenCV e a sua maturidade é a quantidade de documentação disponível. No caso do visoin foi concebido para ser executado em dispositivo Apple.

2.4 Algoritmos relevantes para aplicação *photo|uniq*

2.4.1 Deteção de keypoints

SIFT [32] ou Scale-Invariant Feature Transform é um algoritmo patenteado, para deteção de features e cálculo dos respetivos descritors. Este algoritmo tem quatro etapas:

seleção do scale space, localização dos keypoints, atribuir a orientação aos keypoints, calcular os descritores.

Para a seleção do scale space são calculados potenciais keypoints em diferentes escalas da imagem. O objetivo do scale space é replicar o conceito que um objeto só pode ser escalado até um certo ponto, por exemplo, um cubo de açúcar é perfeitamente visível numa mesa, mas numa galáxia ele é inexistente.

O scale space de uma imagem consiste na aplicação de vários níveis de blur a várias escalas de uma imagem, cada escala terá sempre metade do tamanho da anterior.

Tendo o scale space definido é necessário calcular o DOG (Difference of Gaussian kernel), gerando um novo conjunto de imagens. O conjunto de imagens gerado pelo DOG será utilizado para a localização dos potenciais keypoints. Para se achar um potencial keypoints cada píxel é comparado com os seus vizinhos, tanto no próprio nível como nos níveis acima e a baixo, caso seja uma extremidade é um potencial keypoint. O facto de serem utilizadas várias escalas da mesma imagem torna o algoritmo invariante à escala.

Para localizar os keypoint reais é utilizada uma aproximação do Harris Corner Detector.

De forma a tornar o algoritmo também invariante à rotação, para isso é definida uma vizinhança à volta de cada keypoint onde dependendo da escala e da magnitude do gradiente a direção é calculada.

Para o cálculo dos descritores é definida uma vizinhança de $16x16$ tendo o keypoint centrado, sendo essa vizinhança subdividida em blocos de $4x4$, em seguida para cada bloco $4x4$ será criada um histograma. Uma vez que, existem 8 direções possíveis será gerado um array com 128 bits. De forma a tornar esta abordagem invariante à rotação é necessário subtrair a rotação do keypoint a todas as orientações calculadas.

SURF [13] ou Speeded Up Robust Feature é um algoritmo patenteado para deteção de features e cálculo dos respetivos descriptores. Este algoritmo é executado em duas etapas: deteção de Feature e cálculo dos descriptores para cada feature.

Para a deteção de features o SURF utilizada o Fast-Hessian Detector, para tal cálcula o determinante da matriz de Hessian, este determinante é utilizado para detetar mudanças à volta de um ponto, onde caso existam mudanças acentuadas o ponto em análise é considerado uma feature. De forma a ser possível calcular o determinante da matriz de Hessian, é necessário aplicar um filtro Gaussian (Gaussian kernel) em seguida terá de ser calculada a derivada de segunda ordem[46].

De forma a garantir a invariância em relação ao tamanho da imagem o SURF utiliza uma pirâmide de escalas, onde a resolução varia de acordo com o nível da escala. Uma vez que o SURF utiliza boxed filter não necessita de aplicar o filtro iterativamente a todos os níveis, podendo ser aplicado os filtros devidamente escalados diretamente nas imagens.

O cálculo dos descritores em SURF é feito em duas etapas: selecionar uma orientação para o keypoint que possa ser reproduzida e calcular o descritor sendo este devolvido num vetor de 128bits

ORB [44] ou Oriented FAST and Rotated BRIEF é um algoritmo não patenteado e como tal é de utilização livre para deteção Features em imagens. Este algoritmo é executado em duas etapas, na primeira são detetados keypoints, e na segunda são calculados os descritores.

Para a deteção de keypoints é utilizada uma otimização do FAST, onde é adicionada a orientação e invariância da escala.

Para definir se um dado pixel p é um keypoint válido o FAST compara o brilho de dezasseis pontos em redor de p , estes dezasseis pontos devem formar um círculo à volta do ponto a ser analisado. O resultado da comparação é dividido em três grupos: mais brilhante que p, menos brilhante que p e com o mesmo brilho que p. Para p ser um keypoint válido é necessário que oito dos dezasseis pixeis que o circundam sejam mais ou menos brilhantes que p.

De forma a tornar o ORB invariante a escala, antes da deteção dos keypoints e grada uma pirâmide de escalas, onde cada nível da pirâmide corresponde a uma escala diferente da mesma imagem, e ao detetar os keypoints em todos os níveis da pirâmide torna o algoritmo invariante a escala.

Após os keypoints estarem localizados é necessário calcular a orientação de cada keypoint, este cálculo é feito com base nas alterações da intensidade dos seus vizinhos, após este cálculo os keypoints são rodados para a sua forma canónica de forma a tornar o algoritmo invariante à rotação.

Para o cálculo dos descriptores o ORB utiliza uma variante do BRIEF, o rBRIEF, que a torna invariante à rotação. O rBRIEF cria um vetor binário com 128 a 512 bits para keypoints localizados pelo FAST.

2.4.1.1 Discussão

Os três algoritmos descritos fazem o que lhes compete, detetam keypoints e extraem os descritores para cada keypoint, porém o SURF e o SIFT são patenteados e como tal não podem ser utilizados livremente, ao contrário do ORB que é open source.

Sabendo que os três algoritmos são eficientes, detetam keypoints, extraíem os descritores. Os três algoritmos podem não detetar os mesmos keypoints, mas essa diferença prende-se com a forma como eles são calculados, e não necessariamente com a sua qualidade, no campo dos descritores os três algoritmos geram um vetor com 128 bits para cada keypoint, uma vez mais poderá diferir devido à forma com são gerados.

Podendo os três algoritmos ser parametrizados de forma semelhante em openCV um dos pontos que mais diferencia estes três algoritmos é a sua complexidade e velocidade.

Ao nível de desempenho, tanto em velocidade como em qualidade de keypoints detetados, de acordo com Karami, Prasad e Shehata [REF] o ORB é o algoritmo mais rápido porém o que apresenta genericamente melhores resultados é o SIFT, contudo imagens com rotação de 90.º o ORB e o SURF têm melhor desempenho.

2.4.2 Correlação de imagens

FLANN [17] é um e uma biblioteca que implementa um conjunto de algoritmos para a correlação de imagens, ou seja, para fazer a correspondência entre os keypoints detetados numa imagem base com os keypoints detetados na imagem de destino, detetando assim os keypoints equivalentes.

Este conjunto de algoritmo tem como filosofia Approximate Nearest Neighbors, ou seja, todas as correlações detetadas não são necessariamente corretas, mas aplicando o teste de Low Ratio é possível filtrar os resultados obtidos de forma a se obter apenas correlações reais (entre dois pontos realmente semelhantes)

O facto de deste algoritmo não detetar apenas correlações reais, mas também correlações aproximadas, torna o algoritmo mais rápido e com uma utilização de recursos de hardware otimizada.

Brute Force Matching [26] é o método de correlação de imagens mais simples que existe. A sua filosofia é simples e consiste em calcular a distância euclidiana entre todos as feature de uma imagem e as feature de outra imagem[18] [23].

O facto de existir uma correspondência entre duas features não significa que essa correspondência seja uma correspondência válida. Dois pontos são considerados semelhantes caso a sua distância euclidiana seja inferior a um determinado critério.

2.4.3 Calculo de imagens semelhantes

O cálculo do grau de semelhança entre imagens é um processo complexo composto por três etapas:

Extração de informação das imagens - Nesta etapa são extraídas informações de imagens com o objetivo de calcular o quão diferentes são as duas imagens.Estas informações podam ser, por exemplo, um preceptual hash ou um conjunto de features;

Cálculo da distância entre duas imagens - após serem extraídas as informações de duas imagens a etapa seguinte é calcular distância entre as imagens para tal é necessário definir o algoritmo que será utilizado para o cálculo dessa distância.

Agrupamento de imagens semelhantes - tendo a distância entre as imagens calculadas é necessário agrupa-las em conjuntos de imagens semelhantes, de forma a tornar o agrupamento o mais eficiente possível no âmbito da *photo|uniq* foi utilizado um algoritmo d agrupamento espacial.

2.4.3.1 Extração de informação de imagem

Existem vários métodos para a extração de informações de imagens de forma a que a distância entre duas imagens possa ser calculada, por outras palavras, calcular o quão diferentes são duas imagens.

Perceptual Hashing é uma classes de algoritmos que produz Perceptual hash com um fingerprint de uma imagem. Ao contrário dos hash criptográficos onde uma pequena mudança gera um hash completamente diferente, o que apenas permite perceber se os hash, por consequência os dados aos quais o hash se refere são iguais ou diferentes. Já nos Perceptual hashs uma pequena mudança nos gera um hash semelhante, o que permite perceber se a diferença entre os dados é grande ou pequena [30].

Existem vários algoritmos para o cálculo de Perceptual Hashs onde se pode destacar o aHash (averger hash), pHash (Perceptive Hash) e dHash (difference hash).

aHash ou Average Hash representa a média aritmética das baixas frequências de uma imagem, isto é importante, uma que com as altas frequências é possível obter os detalhes, já no caso das baixas é possível obter a estrutura da imagem [28].

Algoritmicamente o aHash é composto por 5 etapas: reduzir a imagem para 8x8; converter a imagem para preto e branco; calcular a média das cores; computação dos bits (um se o valor estiver a cima da média e zero se estiver a baixo); criação do hash, onde os 64 bits obtidos são transformados num inteiro de 64 bits[28]. A primeira e segunda etapas podem ser invertidas, sendo em primeiro lugar a imagem convertida para preto e branco, e só depois reduzida[30].

dHash ou diference hash é um algoritmo que calcula o hash com base em gradientes. O dHash trabalha com a diferença entre pixeis adjacentes, o que permite identificar a direção relativa do gradiante[27].

Algoritmicamente o dHash é composto por quatro etapas: redução do tamanho da imagem para 8 x 9 ; transforma a imagem em escala cinza; calcular a diferença entre pixeis onde se o valor do pixel x for inferior ao valor em $x+1$, o próximo bit do hash será 1 caso contrário será 0 [30].

pHash ouPerceptive Hash tem um funcionamento semelhante ao aHash, mas ao invés de comparar cores compara frequências obtidas através da aplicação de discrete cosine transform [15] [30].

Algoritmicamente o pHash é o mais complexo dos algoritmos analisados sendo composto por seis etapas: conversão da imagem para preto e branco posterior redução para 32 x 32, calcular a discrete cosine transform de forma a transformar a imagem num conjunto de frequências e scalars; reduzir os resultados obtidos pelo DCT, ficando apenas com as frequências mais baixas localizadas na matriz 8x8 localizada no canto superior esquerdo da matriz gerada pelo DCT; calcular a media do DCT; criação do hash com 64 bits com base na matriz 8x8 e na madia do DCT, onde se o valor estiver acima da média será atribuído o valor de 1 caso contrário será 0[15]; por fim e necessário transformar o valor obtido na etapa anterior num inteiro de 64 bits.

Vision Feature Prints é um componente da framework Vision da Apple, o que torna a implementação do algoritmo proprietária. Consiste numa rede classificação (classification network) previamente treinada de forma a conseguir extrair os descritores de uma imagem. A framework devolve uma lista com os descritores de cada imagem de forma a ser possível calcular a distância euclidiana com o intuito de se perceber o quanto as imagens estão próximas.

2.4.3.2 Métricas de distância

Existem várias métricas para o cálculo da distância entre dois pontos, sendo as abordadas nesta dissertação a Distância Euclidiana e Distância de Hamming.

A Distancia euclidiana é uma métrica utilizada para calcular a distância entre dois pontos no espaço euclidiano. Esta distância pode ser recorrendo as coordenadoras cartesianas dos pontos e ao teorema de Pitágoras, já a Distância de Hamming é uma métrica utilizada para comparar sequências de dados de igual dimensão, onde o resultado da comparação será o número de elementos que variam entre as duas sequências. O cálculo da distância de Haming tem uma complexidade de $\mathcal{O}(n)$

A distância euclidiana e a distância de Hamming têm propriedades semelhantes; o seu valor é simétrico, ou seja, a diferença entre A e B será a mesma que entre B e A; O resultado da distância entre dois elementos será sempre um valor positivo se os elementos diferirem, ou zero se os elementos forem iguais; quanto menor o valor da distância mais semelhantes são os elementos.

2.4.3.3 Agrupamento de imagens

DBSCAN [43] é um algoritmo de clustering por densidade com noção de ruído e transitividade, ou seja, nem todos os elementos têm de ser adicionados a um cluster, sendo assim possível encontrar elementos que não se enquadram em nenhum cluster, no caso da *photo|uniq* o ruído serão as imagens consideradas Únicas.

Uma das vantagens do DBSCAN em ralação a algoritmos como o K-means é não ser necessário indicar quantos grupos devem ser formados, uma vez que não é necessário indicar quantos grupos são formados o DBSCAN, além da lista de valores a agrupar, a versão utilizada recebe apenas três parâmetros:

- **epsilon** - Limiar para que dois elementos sejam considerados próximos e por consequente serem adicionados ao mesmo grupo;
- **minPts** - Número mínimo de elementos que um grupo pode ter;
- **Função para calcular a distância entre dois elementos** - Função que recebe dois elementos do tipo da lista a singularizar e devolve a distância entre esses dois elementos. Esta função pode ser padronizada ou definida pelo programador.

O DBSCAN devolve uma lista com os conjuntos de elementos gerados e uma lista com o ruído (elementos não agrupados).

Para gerar os grupos o DBSCAN seleciona aleatoriamente um elemento (*corepoint*), caso esse elemento tenha na sua vizinhança, a uma distância inferior ao epsilon, mais de $minPts - 1$ elementos , de forma a que o total de elementos dentro do limiar incluindo o ponto selecionado no mínimo minPts, será formado um grupo com esses elementos, caso não seja será selecionado um novo candidato a *corepoint*. Devido à transitividade se $A- > B- > C$ (A é semelhante a B e B é semelhante a C) então $A- > C$. Após os primeiros elementos serem adicionados ao cluster é necessário expandir o grupo para tal é necessário alterar o *corepoint* passando a ser o elemento mais próximo do *corepoint* original, onde tal como na primeira iteração é verificado se na vizinhança existem pelos menos minPts a contar com o *corepoint* caso existam os elementos são adicionados ao grupo. Este processo repete-se até serem percorridos todos os pontos do grupo.

Um dos pontos negativos do DBSCAN é este não ser determinístico na geração dos grupos, dependendo dos elementos selecionados nas primeiras iterações, visto que após um elemento ser adicionado a um grupo este será ignorado nas pesquisas seguintes mesmo que o novo grupo seja mais adequado para o elemento em questão, porém os elementos considerados como ruído serão sempre considerados como ruído.

2.4.4 Análise de qualidade

Image Quality Assessment é uma classe de algoritmos para a estimação de qualidade de uma imagem. Estes algoritmos recebem uma imagem e devolvem a sua classificação de qualidade.

Existem três categorias de algoritmo de Image Quality Assessment: Full-Reference, Reduced-Reference e No-Reference.

Os algoritmos Full-Reference recebem uma imagem com qualidade, esta imagem irá servir de base para a classificação das restantes imagens, os algoritmos de Reduced-Reference em vez de receberem uma imagem recebem informações sobre a imagem, são estas informações que vai determinar a qualidade das restantes imagens, por fim os algoritmos No-Reference recebem apenas a imagem a avaliar a qualidade.

No âmbito desta dissertação são analisados apenas algoritmos que dependam somente da própria imagem.

2.4.4.1 BRISQUE

BRISQUE[35] ou Referenceless Image Spatial Quality Evaluator é um algoritmo No-Reference para a estimação da qualidade de uma imagem.

Sendo um algoritmo sem referências é necessário definir o conceito de qualidade para o mesmo. Esta definição é importante uma vez que a noção de qualidade pode variar de projeto para projeto, e a noção de qualidade é dada a partir de um modelo de treino.

Este algoritmo classifica as imagens numa escala de 0 a 100 onde 0 é a melhor classificação e 100 a pior, e pode ser divido em três etapas: extrair Natural Scene Statistic, calcular o vetor de Features e por fim estimar a qualidade.

2.4.4.2 Workflow

Nos trabalhos anteriores foi proposto por Felismino [19] e Mano [34] um workflow para análise automatizada da qualidade de imagens. Este workflow é composto por várias etapas que vão analisando a qualidade de imagem, são elas: deteção de foco, estimação do motion blur, estimação de ruído e análise de cor, onde em cada uma das etapas a imagem é classificada nessa critério.

De forma a tornar este método viável é necessário definir o conceito de qualidade e a escala sobre a qual a imagem irá ser classificada. Um dos métodos mais simples de definir o conceito de qualidade é atribuir um peso a cada uma das etapas, garantindo que o somatório dos pesos nunca passe de um.

O valor obtido em cada etapa, deve ser uniformizado, sendo que a uniformização garante que o resultado de cada etapa está sempre no mesmo intervalo de valores.

2.4.4.3 Discussão

Os dois métodos descritos para a análise de qualidade têm abordagens distintas, cada um com as suas vantagem e desvantagens.

Uma das principais vantagens da análise de qualidade recorrendo ao workflow é a sua facilidade de parametrização, seja por parte do programador ou por parte do utilizador final; outra vantagem é a possibilidade de expansão do algoritmo, caso se pretenda adicionar uma nova etapa é relativamente simples; por outro lado, uma das principais desvantagens é a complexidade de execução deste executar este workflow na totalidade, por exemplo se existirem cinco etapas, será necessário analisar a mesma imagem cinco vezes.

O BRISQUE por sua vez não é tão parametrizável, mas é possível alterar a noção de qualidade do algoritmo alterando o modelo de treino. No campo das vantagens uma das principais é que a imagem apenas tem de ser analisada uma vez de forma a se obter o sua classificação de qualidade.

2.4.5 Matriz de homografia

Sendo um dos pontos-chave da *photo|uniq* a comparação de imagens lado a lado centradas em pontos equivalentes, é necessário converter as coordenadas de um ponto numa imagem nas coordenadas do ponto equivalente na segunda imagem.

Uma das técnicas utilizadas para fazer esse mapeamento é recorrer à matriz de homografia entre as duas imagens. A matriz de Homografia é uma matriz 3 x 3 que mapeia um ponto num plano para o ponto equivalente em outro plano. Na figura 2.2 pode ser

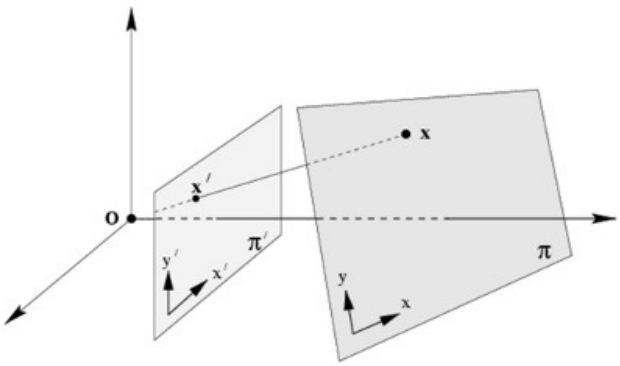


Figura 2.2: Mapeamento da Homografia [39].

visto um exemplo do mapeamento entre um ponto no plano π para o seu equivalente no plano π' no caso da aplicação *photo|uniq*, π e π' podem ser vistos com duas imagens semelhantes.

Caso o sistema de coordenadas dos dois planos esteja homogeneizado este mapeamento pode ser escrito como $x' = Hx$, onde H é a matriz de homografia que define o mapeamento entre os dois pontos, e x e x' são o ponto na imagem base e o seu ponto equivalente na imagem de destino.

Se assumirmos que x' pode ser definido como (x', y', z'), x pode ser definido como (x, y, z), e que H representa matriz de homografia entre x e x', a equação que transforma x em x' pode ser definida como:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{2.1}$$

A equação 2.1 define o mapeamento de $x \Rightarrow x'$ tendo sido calculada a matriz de homografia com esse objetivo, caso se pretenda mapear um ponto de $x' \Rightarrow x$ se a matriz de homografia $x \Rightarrow x'$ já estiver calculada poderá ser utilizada o inverso dessa matriz sendo a equação: 2.2.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}^{-1} \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} \tag{2.2}$$

2.5 Frameworks para criação da interface

2.5.1 UIKit

O UIKit é uma das frameworks para a criação de layouts em ambiente Apple, esta framework deriva do AppKit (framework para a criação de layouts para macOS. O UIKit

está disponível desde 2008 com o aparecimento do appStore na versão 2 do iOS o que a torna uma tecnologia bastante madura, levando a que exista muito mais documentação, tutoriais e certamente qualquer problema que possa ocorrer o que já aconteceu com outras pessoas.

Esta framework está desenhada para ser utilizada com o padrão de desenho MVC, onde o modelo de dados não liga diretamente a vista tendo os dados que passar por um controller. Para se criar uma interface e associá-la a um modelo de dados são necessários três ficheiros: um modelo de dados, uma vista e um controller.

O modelo de dados um conjunto de ficheiros Swift com as classes, bases de dados.

Uma vista está num ficheiro de interface, este pode ser .storyboard ou .xib, em iOS genéticamente é utilizado um ficheiro .storyboard. Para se abrir ficheiros de interface utiliza-se a Interface Builder, uma aplicação dentro do Xcode para a criação de interfaces. Uma vista num storyboard é criada através de darg and drop, onde é possível posicionar os elementos no local pretendido, sendo necessário definir regras tanto para o posicionamento como para as dimensões dos elementos gráficos, estas regras servirão para que o layout se adapte a todas as resoluções, estas regras podem ser definidas graficamente ou por código.

Um storyboard pode conter todas as vistas de uma aplicação, e quando é necessário fazer a ligação entre vistas, essencialmente para navegação, esta ligação também é feita graficamente, o que num projeto de grandes dimensões com várias vistas, se pode tornar complexo de entender o que está ligado e onde.

O storyboard é a representação gráfica de ficheiro XML, que devido à sua complexidade é quase impossível escrever manualmente.

O controller é o elo entre a vista e o modelo, tal como o modelo o controller também é uma classe swift. Esta classe é responsável por toda a interação do utilizador com a interface gráfica. Para que seja possível interagir com elementos gráficos, seja alterar as suas propriedades ou o conteúdo, ou mesmo para executar alguma ação é necessário criar referências no controller e associá-las aos elementos gráficos, a Interface Builder disponibiliza um mecanismo que facilita a criação destas referências. Estas referências são uma das principais fontes de erros ligados ao storyboard, uma vez que é permitido fazer mais de uma referência do mesmo tipo ao mesmo elemento, e caso seja removida a referência do lado do código sem ser removida do lado da interface provoca um erro.

2.5.2 Swift UI

SwiftUI é a mais recente framework para a criação de interfaces para dispositivos Apple, lançada com o iOS 13 em 2019. Uma vez que é uma framework muito recente ainda não há muita documentação sobre ela, como tal ainda poderão surgir muitos novos problemas.

Esta framework foi pensada para o padrão MVVM, onde a vista está separada do modelo estando à vista associado um view model. Uma das características das vistas é estas serem reativas, ou seja, reagem às alterações feitas no view model, isto é, quando

um valor do view model é alterado se estiver a ser utilizado na vista este também será alterado.

Outro conceito importante é o de estado, onde caso uma variável de estado tenha algum presente na interface quando a variável for alterada, o valor da vista também será alterado automaticamente.

Tal como com o UIKit o modelo é o conjunto de classes que contém a lógica de negócio da aplicação.

A vista é a interface gráfica, que no caso do swiftUI a interface é declarativa e escrita inteiramente em swift o que a torna mais rápida de executar, e mais simples de programar, podendo ser totalmente criada em código, além da possibilidade de criar a interface por código também é possível criar a interface por drag and drop, podendo, em ambos os casos, interface ser simulada em tempo real.

O conteúdo de uma vista seja ela a principal ou uma sub-vista é organizado em stacks (verticais, horizontais ou profundidade), o que faz com que não seja necessário autolayout para que a interface se adapte a qualquer resolução. Sendo uma framework comum a todos os dispositivos Apple torna mais simples a criação interfaces gráficas para aplicações multi-plataforma.

Devido a ser uma framework tão recente ainda não tem tantos componentes disponíveis como o UIKit, mas esta questão pode ser contornada utilizando vistas UIKit em vistas swiftUI.

2.5.3 Discussão

Tanto o swiftUI como o UIKit cumprem a sua tarefa, criar uma interface gráfica para Aplicações iOS, tendo cada uma com os seus prós e contras.

O UIKit tem como principal vantagem a sua maturidade, o que garante que a framework tenha todas as funcionalidades implementadas, assim como tem muito mais suporte da comunidade, por seu lado o swiftUI minimiza o facto de não ter todos os componentes do UIKit permitindo que sejam utilizadas vistas UIKit dentro uma vista swiftUI. Ambas permitem a criação de interfaces recorrendo ao drag and drop, mas o swiftUI possibilita a alteração da vista através de código de forma mais simples .

Para uma aplicação se adaptar qualquer resolução utilizando o UIKit é necessário definir as regras do autolayout, por seu lado, uma vez que uma interface swiftUI é construída a base de stacks, tornando autolayout desnecessário.

Uma vez que em swiftUI é totalmente declarativo, não é necessário um controller para ligar a vista ao modelo de dados o que faz com que não exista o erro com as referências dos elementos da interface no código.

Sendo o swiftUI pensado para o MVVM funciona com variáveis de estado, onde quando uma variável é alterada o seu valor na interface é atualizado.

No campo da velocidade de renderização, uma vez que com swiftUI é possível criar interfaces totalmente em Swift ao contrário do UIKit que o código da interface é XML, o

swiftUI renderiza as interfaces mais rapidamente.

2.6 Persistência de dados em iOS

Em iOS existem cinco formas nativas com propriedades e utilizações distintas de salvar dados localmente no dispositivo, sendo elas: Core Data, user defualts e SQLit.

2.6.1 Core Data

Core Data é a Framework mais comum para a persistência local em iOS sendo usada pela grande maioria das aplicações que necessitam de salvar dados localmente. O Core Data foi inicialmente disponibilizada no iOS 3, e tem vindo a evoluir ao longo das versões do iOS.

O Core Data é uma Framework de persistência própria da Apple, que organiza os objetos da aplicação em grafo, abstraindo o processo de salvar e ler dados. Esta Abstração evita que o programador lide com a complexidade de aceder e lidar com a base de dados manualmente. Sabendo que existe uma abstração torna-se claro que o Core Data salva os dados numa base dados relacionais, no caso do iOS os dados são salvos num base dados SQLit.

A abstração do Core Data permite que os dados sejam facilmente manipulados ao nível do objeto, onde cada objeto representa uma entidade.

Sendo o core data uma framework Apple é natural que esta tenha uma relação otimizada com as demais frameworks. Uma das otimizações mais interessantes é a sua relação com o SwiftUI, em que, quando um objeto salvo em core data, é visível, na interface gráfica, é alterado, seu valor na interface gráfica é automaticamente atualizado, não sendo necessário utilizar modificadores de estado quando se cria interfaces gráficas em SwiftUI.

2.6.2 User Defaults

Os User defaults são o método de persistência mais rudimentar existente em iOS, foi lançado junto com o iOS 1. Uma das grandes vantagens dos User defaults é a simplicidade da sua estrutura. Um User defaults não passa de um conjunto chave /valor e sendo um conjunto chave/valor ao estilo de dicionário, torna-os limitados no tipo de dados que suportam. Apesar de rudimentar, os User defaults são de extrema utilidade quando é necessário salvar dados e não temos acesso a outro meio de persistência, isso acontece caso use a share extention. Tal como o nome indica o principal objetivo dos User defaults é guardar os dados defaults, por exemplo, o nome do utilizador ou um token de início de sessão.

2.6.3 SQLit

Toda a persistência relacional em iOS é feita em SQLit. Como referido anteriormente o Core Data está assente em SQLit, ou seja, todos os dados das suas classes estão realmente em SQLit.

De forma genérica SQLit é uma base de dados relacional leve, que genericamente tem um único ficheiro. Apesar de estar assente apenas num ficheiro, disponibiliza as principais funcionalidades de uma base de dados relacional tradicional.

Uma das vantagens do seu uso é que não necessita de um gestor de bases de dados, uma vez que toda a base de dados está apenas num ficheiro sendo mais simples de fazer o seu backup.

2.6.4 Discussão

Todos os mecanismos de persistência podem e coexistem dentro da mesma aplicação uma vez que cada um tem a sua utilidade, onde para guardar os dados globais de uma aplicação pode ser utilizado o Core Data, esta escolha prende-se com a sua capacidade de lidar diretamente com objetos, podendo em alguns casos o programador abstrair-se tanto do salvamento dos dados como do seu carregamento, sendo o salvamento feito com uma linha de código e o carregamento automático quando a aplicação abre. No caso dos User defualts, estes podem ser utilizados para guardar dados de personalização da aplicação, sendo estes dados mais simples que todos os dados gerados pelo modelo de dados, não faz sentido guardar estes dados com uma framework tão poderosa como o core data. Sendo plists listas de propriedades, são utilizados para guardar informações fulcrais para o bom financiamento da aplicação, outra utilidade das plists é guardarem parâmetros de default de métodos, uma das grandes vantagens dos plists é permitir que os dados estejam em ficheiros dedicados, podendo estes ser alterados na interface do Xcode. O SQLit mesmo que não seja diretamente utilizado será sempre utilizado, pois o Core Data é uma abstração do acesso ao SQLit.

2.7 Ambiente de desenvolvimento

2.7.1 Ambiente integrado de desenvolvimento Apple: Xcode

O Xcode é muito mais do que um ambiente integrado de desenvolvimento, é um conjunto de programas que permitem não só escrever o programa, mas também, testar as aplicações desenvolvida, seja no simulador ou num equipamento físico, fazer o profiling da sua aplicação de forma a facilitar a otimização do código escrito, uma vez que na ferramenta de profiling e possível perceber onde a aplicação esta a gastar mais tempo e distribuir a aplicação seja entre os membros da equipa de desenvolvimento, seja para utilizadores externos, entre outras.

Cada uma das funcionalidades mencionadas recorre a uma aplicação distinta dentro do Xcode. Para testar a aplicação pode-se utilizar o simulador. Esta aplicação simula o iOS, contudo utiliza os recursos do Mac onde estiver a correr, o que não a torna própria para testes de desempenho real, apenas para teste de desempenho relativo, ou seja, perceber se uma implementação correr de forma mais eficiente que outra. Além da diferença de recurso existem outras diferenças onde se pode destacar: algumas APIs e recursos de equipamento físico não estão disponíveis no simulador, porém, o uso do simulador tem algumas vantagens e razões para ser utilizado, onde se destaca a possibilidade de simular condições como: modo escuro, localização GPS especifica, forçar erros, entre outras.

Para a análise de desempenho é utilizado o instruments, uma das diferenças em relação à ferramenta de profiling da Apple é que esta permite mostrar dados não relacionados entre si, mas ligados ao desempenho da aplicação, numa só vista.

Com o recurso ao instruments, é possível entre outras coisas: analisar o impacto que uma aplicação tem sobre a memória ou analisar o impacto que a aplicação tem no processador.

2.7.2 Linguagem de programação Swift

Nesta secção será apresentada a linguagem Swift. Linguagem atual de desenvolvimento para dispositivos Apple.

Swift é uma linguagem de programação tornada pública pela Apple em 2014 para ser usada como linguagem principal de desenvolvimento para os seus dispositivos.

Inicialmente o Swift foi lançado como uma linguagem proprietária, mas em 2015 este panorama mudou quando a Apple tornou a linguagem open source, sob a licença Apache 2.0 com uma Runtime Library Exception [9] tornando o Swift disponível para todas as plataformas, mas com suporte oficial apenas para Mac e Linux [9]. Com o facto de a linguagem se ter tornado open source apareceram várias outras utilizações para o Swift, como, por exemplo, para desenvolvimento web.

Uma das vantagens do Swift é que cerca de 42,5% das bibliotecas estão escritas em Swift [24].

Com a versão 5.0 do Swift foi atingida a ABI Stability [24]. Esta linguagem apresenta algumas funcionalidades interessantes onde se podem destacar:

- **Inferência de tipos** - Em Swift em muitas ocasiões não é necessário indicar explicitamente o tipo de uma variável, uma vez que o compilador consegue inferir o tipo de dados da variável através do seu valor;

- **Tuplos** - São variáveis que podem conter mais de um valor, estes valores podem ser de tipos diferentes;

- **Structs** - Em Swift as Structs são estruturas de dados bastante poderosas, diferem das classes apenas em três aspetos:1- São tratadas como valores ao invés das classes

que são tratadas como referência; 2- não podem herdar de outras classes; 3-não podem ser desinicializadas;

- **Extensões** - Possibilidade de adicionar funcionalidades a classes já existentes, sem necessidade de recorrer a uma estrutura baseada em herança, outra utilização interessante para extensões está na organização do código onde podemos separar blocos de código por funcionalidade;

- **Enums** - Em Swift os enums são mais expressivos uma vez que podem conter um valor associado ao nome;

- **Observers** - Com frequência é necessário executar uma ação sempre que uma variável é alterada, em Swift podemos usar Observers. Um Observer tem a possibilidade de executar código antes e depois de uma variável ser alterada, quando a variável é inicializada o Observer não é executado;

- **Lazy Property** - Uma propriedade lazy é uma propriedade que só é instanciada quando é acedida pela primeira vez. As propriedades lazy devem ser usadas sempre que o seu valor dependa de fatores externos ou quando o seu valor é computacionalmente exigente de ler ou gerar. Outra característica interessante das propriedades lazy é que estas podem referir-se à própria instância da classe;

- **Computed Properties** - São propriedades que não têm valor, no entanto, dispõe de um getter e setter por forma a apresentarem valores de outra variável. As variáveis computadas têm diversas utilidades, ocultando o setter que permite criar variáveis só de leitura, isto é, encapsular outra variável;

- **Opcionais** - Em Swift por defeito as variáveis não podem ser nulas, mas por vezes é necessário que uma variável tenha o valor nulo. Usamos opcionais, para que se consiga obter variáveis com valor nulo. Um opcional é uma variável que pode ter dois estados: "Não tem valor" ou "Tem valor e é X" onde X representa o valor.

Em Swift os argumentos dos métodos podem ter labels, o que permite que existam nomes internos e externos para argumentos de métodos, onde o nome interno apenas é conhecido no método, já o externo só é conhecido na assinatura do método quando este é chamado. Caso não seja definida a label, o nome interno e externo é igual.

2.7.3 Convenções

Tal como todas as linguagens de programação o Swift têm um conjunto de convenções para a escrita de código. Estas convenções estão descritas no guia para a criação de APIs, e serão apresentadas nesta secção assim como serão seguidas no desenvolvimento do protótipo final.

Em Swift um dos princípios base na atribuição dos nomes tanto para propriedades como para métodos é a clareza. Os nomes devem ser explícitos, o que faz com que seja mais importante serem explícitos que curtos.

Tal como a maioria das linguagens Orientadas a objetos o Swift utiliza UpperCamelCase para classes e protocolos, e lowerCamelCase para tudo o resto.

Existem várias regras para a criação de nomes considerados explícitos, sendo uma das principais: a evitar ambiguidades, ou seja, devemos incluir no nome todas as palavras necessárias por forma a garantir que ao ser lido o código este é percetível, por outro lado, devem ser excluídas palavras redundantes, uma vez que estas são desnecessárias para garantir as desambiguidades nos nomes, outra regra que ajuda a evitar a ambiguidade nos nomes é estes refletirem o seu papel no programa.

À parte das questões de ambiguidade existem outras diretrizes para atribuição de nomes a métodos: os nomes devem ser estilo inglês: x.insert(y, at: z), que podem ser lidos como se se tratasse de uma frase, ao contrário de x.insert(y, z).

Os nomes dos métodos devem ter em conta o seu efeito, se este recai sobre a própria instância, ou se devolve um novo valor, caso o resultado do método recaia sobre a própria instância, o seu nome deve ser um verbo imperativo, por exemplo: x.sort(), caso devolva um novo valor, deve-se usar sufixos como “ed e “ing”, por exemplo: z = x.sorted().

Ainda no campo dos nomes, tanto para métodos como para propriedades, sempre que possível devem ser usados termos simples, evitando expressões demasiado científicas, por exemplo: deve ser usado o termo pele em vez de epiderme, contudo, caso seja para remover ambiguidades podem ser usados termos técnicos.

Tanto nos nomes dos métodos como das propriedades devem ser evitadas abreviaturas.

2.7.4 Sistema operativo móvel da Apple: iOS

iOS é o sistema operativo para telefones móveis Apple. O iOS foi lançado no ano de 2007 aquando o lançamento do primeiro iPhone, contudo apenas foi possível a empresas externas a Apple fazerem as suas aplicações em 2008 quando foi lançada a appStore.

O sistema e constituição por quatro camadas: Core OS, Core Services, Media Services, Cocoa Touch.

- **Core OS** - a camada mais baixa é a responsável por fazer a ligação entre a parte física do equipamento e a parte lógica.

- **Core Service** - é esta camada que disponibiliza os recursos base para a criação de aplicações.

- **Media Services** - é camada responsável pela implementação de todas as frameworks de ligadas ao processamento multimédia.

- **Cocoa Touch** - é a camada responsável tanto pela interface gráfica como pela forma como o utilizador interage com a aplicação.

2.8 Diretrizes para a criação de interfaces gráficas

Para desenvolver interfaces gráficos para aplicações iOS, existem regras muito rígidas.

As *Human Interface Guidelines* (HIG) definem as regras base, que os criadores das interfaces gráficas devem seguir, para criarem interfaces para dispositivos Apple. As HIG estão divididas em quatro blocos, iOS tvOS, macOS e watchOS.

Uma vez que, com este trabalho se pretende desenvolver uma aplicação para iOS, as restantes serão ignoradas.

2.8.1 Conceitos gerais

Para a criação da identidade gráfica da aplicação, a Apple aconselha, serem seguidos os princípios de design [6].

- **Aesthetic Integrity** — O quão bem a interface gráfico está ligado à funcionalidade [6] e que o design das interfaces devem seguir os princípios do design gráfico [20];

- **Consistencia** — Uma aplicação consistente é aquela que implementa padrões do sistema [6], o que torna os elementos base da aplicação similares às restantes aplicações iOS;

- **Manipulação direta** — A manipulação direta consiste na possibilidade de o utilizador poder manipular elementos no ecrã, quer seja por rodar o equipamento, ou seja, usando gestos;

- **Feedback** — Todas as ações que são feitas na aplicação devem ter feedback, por exemplo, se se carrega num botão, este deve ter resposta imediata, por forma a ser percetível que uma ação foi iniciada [21], posteriormente deve ser mostrado o resultado da ação, ou pelo menos, a informação de que esta foi concluída [6];

- **Metáforas** — De forma a facilitar o uso da aplicação, devem ser usadas metáforas visuais (usar uma ideia ou objeto que represente outro conceito [22]), isto acontece porque com o uso de metáforas torna o uso da aplicação mais intuitivo;

- **User control** — Ser o utilizador controlar a aplicação e não o contrário [6], no caso da *photo|uniq* a aplicação apenas sugere, que fotografias devem ser removidas, cabendo ao utilizador decidir se estas são realmente removidas.

Além dos princípios já analisados é necessário entender conceitos como:

- **Arquitetura da aplicação** — Refere-se à parte gráfica e aos seus componentes. Loading é um dos componentes que, quando uma ação demora o utilizador deve estar ciente do tempo que falta para esta estar concluída. Modality, técnica que permite mostrar conteúdo em cima de outro, deve ter sempre forma de voltar ao conteúdo original, sem efetuar qualquer alteração. Navegação, define o estilo de navegação usado na aplicação, quando esta é iniciada a primeira vez, apresenta um ecrã de Boas-vindas. Pedidos de permissão, só devem ser solicitados os necessários. Nas definições, a aplicação deve surgir pronta a funcionar, sem ser necessária qualquer configuração;

- **Interação do utilizador** — Existem várias formas de o utilizador interagir com a aplicação. No âmbito deste trabalho as formas de interação com aplicações iOS [6] serão utilizadas: autenticação com impressão digital ou faceID, inserção de dados, feedback e gestos;

- **Cor** — A cor é um elemento fundamental em qualquer aplicação, deve-se usar uma paleta de cores limitada que conjugue com as cores do logótipo. Não devemos usar as mesmas cores para elementos estáticos e não estáticos. Outro ponto importante na cor está relacionado com a forma como as diversas culturas interpretam cada cor. Geralmente o ponto mais importante no uso da cor é manter-se a consistência, se um botão de confirmação é amarelo num ponto, todos os botões de confirmação devem ser amarelos.

2.8.2 Diretrizes Apple

Nesta secção serão apresentadas algumas das diretrizes da Apple para a criação de interfaces gráficos.

A grande maioria das aplicações iOS usa quatro categorias de componentes:

- **Barras** — Este elemento tem como propósito mostrar ao utilizador onde ele está, bem como facilitar a navegação na aplicação;

- **Vistas de Conteúdo** - Tal como o nome indica é o local onde o conteúdo da aplicação deve ser mostrado;

- **Controles** — São todos os elementos que executam ações;

- **Vistas Temporárias** — São vistas que não estão sempre visíveis ou que disponibilizam alguma informação ao utilizador.

O conteúdo de uma aplicação deve ser claro e de fácil interação. Para que o conteúdo seja claro e fácil de interagir, podem ser implementadas estratégias como: usar o espaço negativo, ou seja, deve existir espaço entre os elementos; a cor deve ser utilizada para simplificar a interface, não devemos usar um leque de cores muito variado; usar as fonts

do sistema, de forma a garantir a legibilidade; sempre que possível usar botões sem bordas.

Com a variação de tamanhos e orientações dos dispositivos Apple, todas as aplicações devem implementar Auto Layout, para que as aplicações possam ser executadas em todos os dispositivos e em todas as orientações, exceto se a aplicação tiver de correr numa determinada orientação, caso isto aconteça, a aplicação deve suportar as duas versões da orientação. No caso da vertical, deve ser possível utilizar a aplicação com botão Home em cima ou em baixo. Quando uma aplicação abre, deve abrir na orientação correta.

Por forma a garantir melhor experiência de utilização devem ser evitadas mudanças ao layout, sem sentido.

Um dos fatores mais importante numa aplicação é a experiência de utilização. Para que a experiência de utilização seja a melhor possível, quando uma aplicação inicia deve abrir diretamente a aplicação, ou seja, não se deve utilizar ecrãs de boas vidas, exceto se forem estritamente necessários. Durante o uso da aplicação não devem ser pedidos dados ao utilizador que possam ser recolhidos por outras fontes, da mesma forma que não é aconselhado pedir dados de configuração ao utilizador, ao invés disso, a aplicação deve ser configurada para o perfil apresentado pela maioria dos utilizadores. É também importante na experiência de utilização retardar o mais possível o pedido de login.

Sabendo que uma aplicação iOS não pode ser fechada por código, mas sim através do iOS, obriga a que a aplicação esteja sempre preparada para ser fechada, salvando sempre que possível os dados, de forma a que o utilizador não tenha de repetir tudo o que foi feito anteriormente.

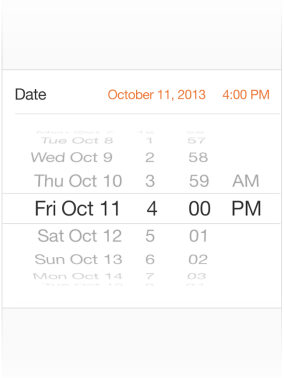
No campo da interatividade, nas barras, tanto barras de ferramentas como barras de navegação, os botões não têm bordas, nos botões disponíveis na vista de conteúdo só devem ter fundo ou contorno se for estritamente necessário.

Sabendo que a principal forma de interação com as aplicações iOS são os gestos; toque; entre outros, deve-se usar os gestos para tornar a experiência de utilização de uma aplicação mais imersiva. Sabendo que o iOS tem vários gestos pré-definidos, há ,sempre que possível, usar esses gestos, contudo caso seja necessário criar um gesto personalizado deve existir um caminho alternativo para realizar a ação que o gesto alternativo iria gerar. Apesar de ser possível criar novos gestos esta ação deve ser evitada.

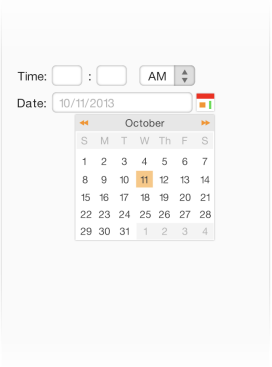
Sempre que possível o feedback ao utilizador deve ser incorporado na interface e os alertas devem ser evitados.

Como já referido anteriormente sempre que possível deve se recolher todas as informações a partir do iOS, contudo existem situações em que isso não é possível e têm de ser solicitados dados ao utilizador, estes inputs devem ser o mais simples possível.

Dois dos principais mecanismos de comunicação de uma aplicação, com utilizador são, a cor e tipografia. No campo da cor, o designer deve garantir que as cores escolhidas funcionam bem entre si e que existe contraste, tanto na separação da barra de navegação como na aplicação em si de forma a que esta possa ser utilizada em todas as condições de luz.



(a) Componente otimizado para touch screen



(b) Componente não otimizado para touch screen

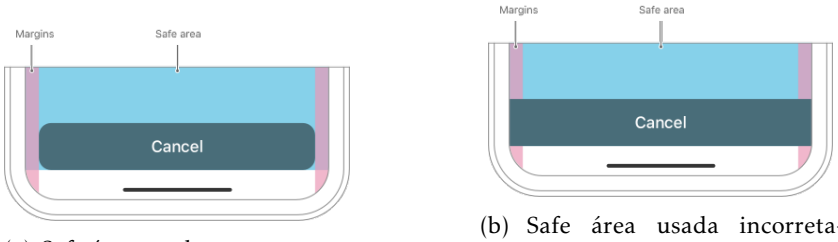
Figura 2.3: Exemplos de Componentes [10].

Uma das questões mais delicadas no uso da cor é o daltonismo, a maioria dos daltónicos tem dificuldade em distinguir o vermelho do verde. Para evitar este problema, uma aplicação não deve depender apenas do verde e do vermelho para distinguir estados.

No campo da tipografia devemos usar o tamanho da font de forma a definir a importância do conteúdo, quanto maior mais importante, se as fonts têm diversos tamanhos o designer tem de garantir que a font é legível em todos os tamanhos. Por uma questão de coerência não é aconselhado utilizar um leque muito grande de fonts na mesma aplicação.

A Apple define alguns critérios ao nível de tamanho e organização para os seus elementos gráficos [10].

- O texto nunca deve ser inferior a 11pts;
- O texto na barra de navegação deve ter 17pts;
- Qualquer elemento com o qual o utilizador possa interagir tem de ter pelos menos 44pt de lado menor;
- O conteúdo deve ser visto sem ser necessário fazer zoom ou fazer scroll na horizontal;
- Devem ser usados elementos desenhados para touch screens(Exemplo:2.3);
- Quando se lida com texto, deve ser evitado o baixo contraste e garantindo espaçamento entre linhas e parágrafos;
- Devem ser usadas imagens de alta resolução e evitar a distorção das mesmas;
- Os elementos devem ser alinhados de forma a mostrar a relação entre eles;
- Conteúdo interativo deve estar dentro da safe área (Exemplo:2.4).



(a) Safe área usada corretamente

(b) Safe área usada incorretamente

Figura 2.4: Exemplos de uso da Safe área [6].

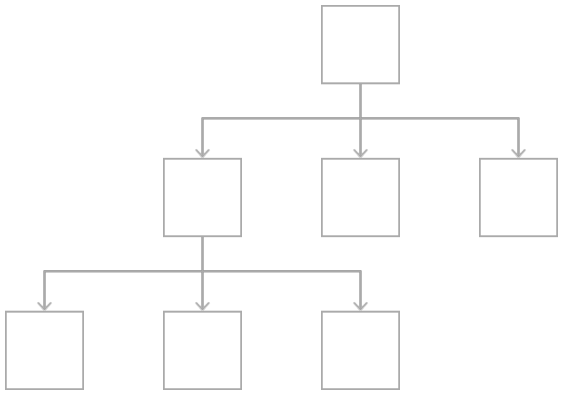


Figura 2.5: Navegação hierarquia.

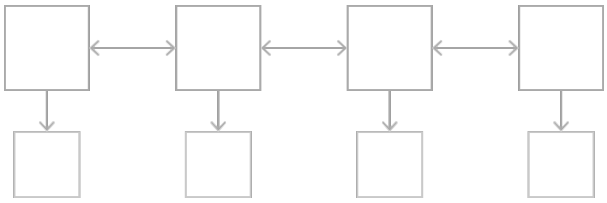


Figura 2.6: Navegação Plana.

2.8.3 Directrizes para a Navegação

Enquanto a navegação da aplicação for de encontro às suas expectativas, o utilizador comum, não se apercebe de como esta está estruturada. Com base nesta ideia, cabe ao programador estruturar a aplicação de forma que a navegação nunca seja notada.

A Apple sugere três possíveis padrões de navegação em iOS:

- **Navegação hierarquia**(ver Figura:2.5) — Padrão de navegação com vários níveis, permite ao utilizador, fazer uma escolha por cada nível, fazendo essa escolha, muda de nível, este processo repete-se até se chegar ao conteúdo de destino. Para se chegar a um conteúdo diferente é necessário retroceder e selecionar opções diferentes;

- **Navegação plana**(ver Figura::2.6)- Padrão de navegação com poucos níveis, permite mudar de vista sem se alterar o nível;

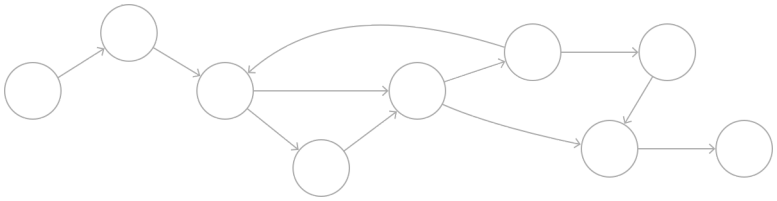


Figura 2.7: Content-Driven ou Experience-Driven.

- **Content-Driven ou Experience-Driven**(ver Figura::2.7)- Padrão de navegação sem níveis, permite ao utilizador navegar livremente pela aplicação.

Independentemente do padrão de navegação selecionado, é importante garantir que o caminho do ponto A de uma aplicação para o ponto B é claro e fácil de seguir, de forma a que o utilizador nunca se sinta perdido na aplicação. Uma das técnicas utilizadas para que o utilizador nunca se perca na aplicação é indicar sempre onde ele está, de onde veio, e que opções tem para navegar. Os elementos que permitem efetuar a navegação dentro de uma aplicação, quando se estrutura a navegação dentro de uma aplicação, sempre que possível, deve-se aos componentes standard, podendo personalizar os componentes escolhidos. O facto de se usar componentes standard como base, aumenta a familiaridade do utilizador com a aplicação bem como a perceção de como esta trabalha.

Todas as aplicações devem funcionar da forma mais fluida possível. Uma das possibilidades de incrementar a fluidez, é a implementação de gestos. Os gestos não devem ser usados como único meio de navegação, mesmo que exista a possibilidade de interagir pela aplicação apenas com gestos, deve existir sempre a possibilidade de interagir pelos meios tradicionais.

3

A aplicação *photo|uniq*

3.1 Arquitetura

A implementação da versão iOS da aplicação *photo|uniq*, foi dividida em quatro grandes componentes: Workspace, Galeria de imagens, processamento e persistência (fig: 3.1). Estes componentes subdividem-se em vários sub componentes que serão responsáveis pelas principais funcionalidades de cada grande componente.

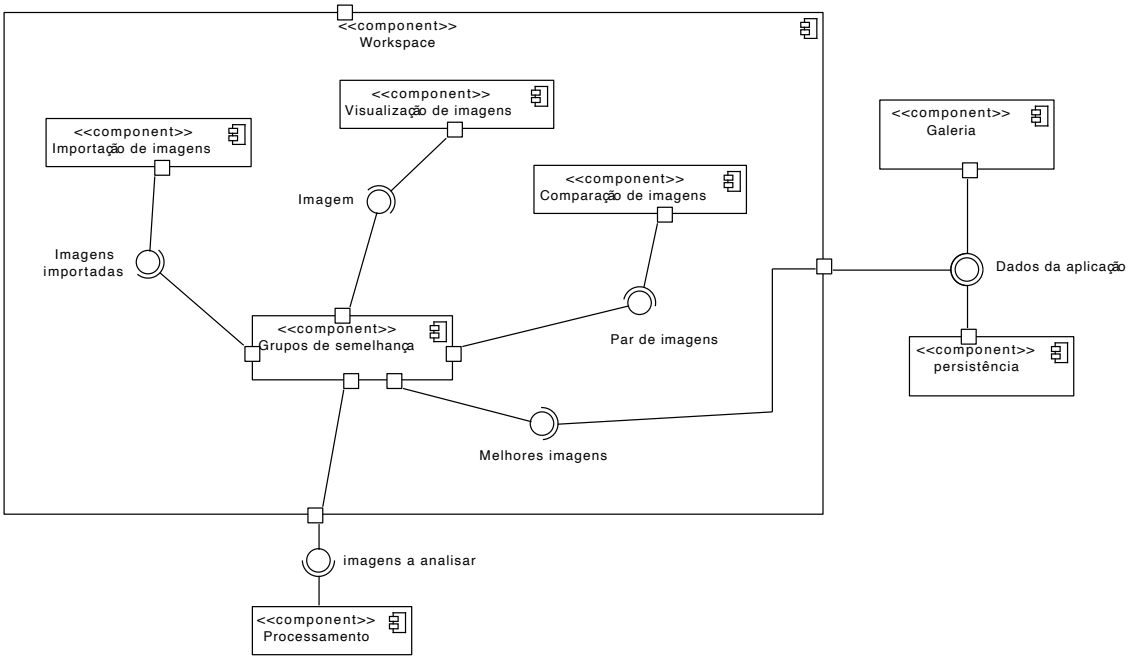


Figura 3.1: Modelo da arquitetura da aplicação.

O workspace é o componente principal da aplicação e aquele para onde o utilizador é redirecionado quando abre a aplicação, é neste componente que o utilizador consegue importar as imagens para que estas sejam tratadas, é também no workspace que o utilizador

dá início a todos os processamentos que é possível fazer na aplicação (geração de grupos de semelhança, ordenação, seja ela manual ou automática, e comparação de imagens lado a lado.

O componente da galeria de imagens ao contrário do workspace não tem sub componentes, servindo apenas como montra das imagens que o utilizador decidiu manter.

No caso do componente de processamento este pode ser considerado o coração da aplicação. É neste componente que são executados todos os processamentos sobre imagens, incluindo: geração dos grupos de semelhança; cálculo de keypoints de todas as imagens; correlação de imagens; análise de qualidade; cálculo de matriz de homografia. Na primeira versão iOS do *photo|uniq* está implementado localmente em OpenCV vs e Apple vision. Para definir este componente foi definida uma interface de forma a facilitar a sua substituição, seja por uma versão local mais otimizada, ou por uma versão remota onde o processamento estaria a cargo de um servidor externo ao equipamento.

Independentemente do local onde é feito o processamento das imagens é necessário persistir, nesta implementação os dados são sempre salvos localmente.

3.2 Modelo de dados

Nesta secção será apresentado o modelo de dados que será salvo no equipamento, sendo os restantes atributos não referidos.

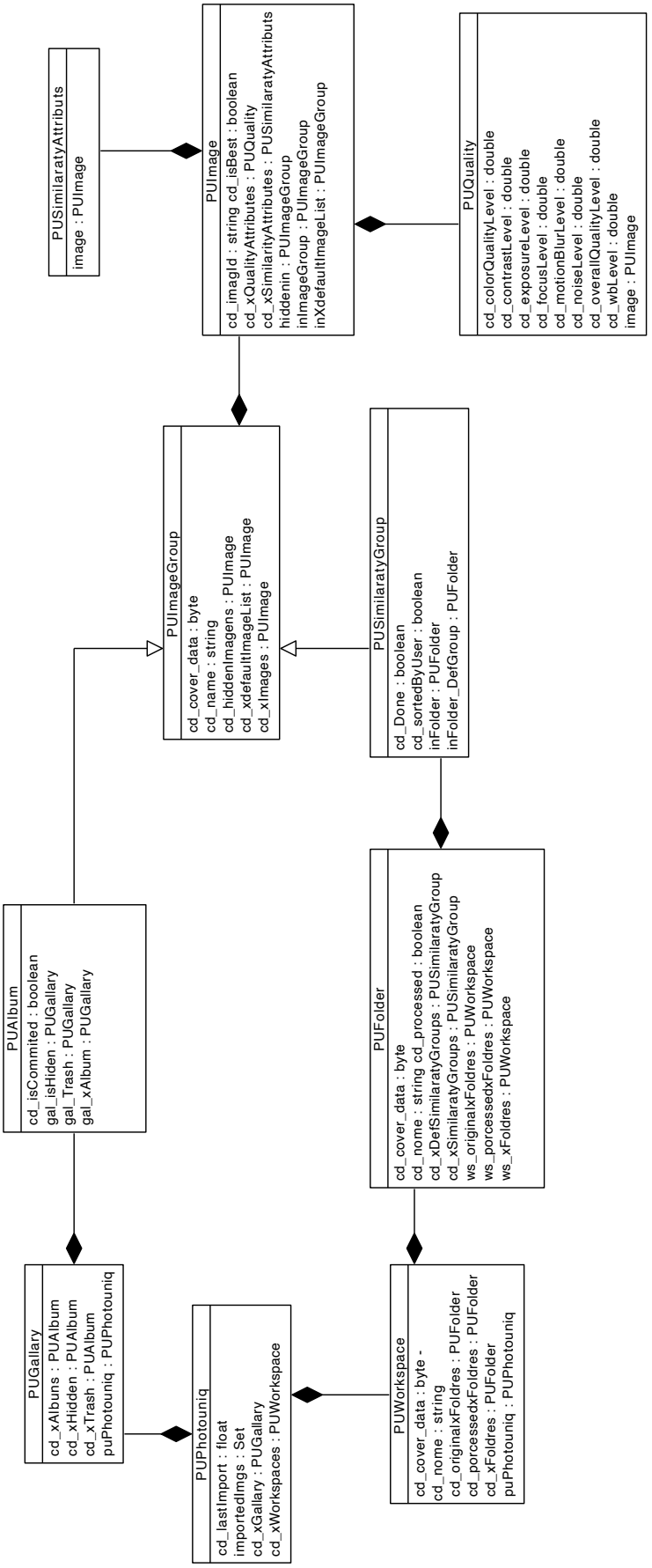


Figura 3.2: Modelo de dados da aplicação.

O modelo de dados da versão iOS *photo|uniq* (fig: 3.2) está dividido em 11 classes: PUGallery, PUAlbum, PUPhotouniq, PUWorkspace ,PUFolder, PUImageGroup,PUQuality, PUImage, PUSimilaratyGroup, PUSimilaraty, PUState.

PUPhotouniq - Está é a classe base da aplicação, é responsável por gerir todos os workspaces e galerias, da mesma forma que é responsável pela importação das imagens.

PUWorkspace — Esta classe representa um workspace. O Workspace é constituído por uma lista de pastas (xFolderes) pela sua imagem principal (cover) e um identificador único. Esta classe é responsável por gerir a lista de pastas existentes no workspace assim como adicionar conteúdo as pastas.

PUFolder - Esta classe representa uma pasta no ambiente de trabalho. É constituída por um grupo de semelhança predefinido (defaultSimilaratyGroups), um conjunto de grupos de semelhança (cd xSimilaratyGroups), o seu nome() , imagem principal(), e a indicação se a pasta já foi totalmente processada().

Esta classe é responsável por gerir os seus grupos de semelhança, executando todas as operações sobre eles, onde se inclui a geração de grupos de semelhança , a partir das imagens contidas no grupo de semelhança predefinido.

PUImageGroup -Classe abstrata que representa um grupo de imagem, sejam elas semelhantes ou não. Esta classe contém apenas uma lista de imagens e um nome para grupo, sendo responsável por adicionar e remover imagens da lista de imagens.

PUSimilaratyGroup -Esta classe herda de PUImageGroup representa e gere grupos de semelhança. Além das operações base, como adicionar ou remover imagens de um grupo de semelhança é também responsável pela ordenação automática das imagens e, na homografia, pela promoção e despromoção de imagens.

PUImage - Representa uma imagem, tendo como atributos a própria imagem, os atributos de semelhança (xSimilarityAttributes:PUSimilaratyAttributs) atributos de qualidade (xQualityAttributes : PUQuality) e atributos de estado(cd xStateAttributes:PUState).

PUQuality - Representa os atributos de qualidade da imagem, os atributos desta classe quantificam todos os atributos de qualidade da imagem, sendo o overallQualityLevel o atributo que representa a qualidade geral da imagem.

PUSimilaraty - Representa os atributos de semelhança, onde se inclui os atributos necessários para calcular grupos de semelhança: descritores (descriptors) e keypoints (keypoints). Para o descritor foi utilzado o tipo UIImage uma vez que os descritores cálculados pelo openCV são armazenados num cv::Mat, e existe uma conversão do openCV de Mat para UIImage e de UIImage para Mat. Esta classe também guarda todas as matrizes e correspondências entre si e a imagem à qual está associada e todas as suas imagens semelhantes.

PUState - Representa o estado da imagem, esta classe tem atributos que indicam se a imagem está num grupo de semelhança, se está marcada como melhor, se está oculta, ou se está no lixo.

PUGallery - Esta classe representa a galeria de imagens que o utilizador decide manter, é composta por uma lista de álbuns.

PUAlbum - Esta classe herda de PUImageGroup e representa um álbum na galeria, este álbum contém apenas imagens marcadas como melhores pelo utilizador, estas imagens apenas são transferidas para um álbum após o utilizador dar o trabalho como finalizado tanto no grupo de semelhança como na pasta onde o grupo se encontra.

Visto que foi utilizado Core Data para a persistência dos dados, todas as propriedades não nativas têm uma propriedade inversa, onde quando uma propriedade é alterada a sua inversa também será alterada.

Um dos exemplos da relação de propriedades inversa é PUPhotouniq.cdxGallary e PUGallary.puPhotouniq.

O nome das classes foi definido seguindo o espírito da nomenclatura dos objetos iOS, onde as primeiras letras representam a framework ao qual dizem respeito, no caso do *photo|uniq*, o nome de todas as classes próprias da aplicação têm o prefixo *PU* . Para a nomenclatura dos atributos é seguida a mesma filosofia, onde todos os tributos que estão salvos em core data têm o prefixo de *cd*.

Em todos os casos em que existe uma imagem principal, e esta não tenha sido definida a aplicação assume que essa imagem é a primeira imagem.

3.3 UI/UX

3.3.1 Navegação

A versão iOS do *photo|uniq* tem uma fusão entre estrutura plana e hierarquia. A estrutura plana divide os principais componentes da aplicação, o workspace e a galeria photouniq. Foi adotada esta estrutura de forma a possibilitar ao utilizar alterar facilmente entre o workspace, o componente principal da aplicação, e a galeria, onde após os grupos de semelhança serem marcados como finalizados, o utilizador poderá ver por si, as imagens consideradas como melhores .

Dentro de cada componente, a estrutura é hierarquia, semelhante à estrutura de pastas num computador, esta estrutura possibilita ao utilizador organizar as imagens em pastas, de forma a simplificar o trabalho da seleção das melhores imagens. O workspace tem 4 níveis, estando organizado em pastas, permite ao utilizador dividir as suas imagens em grandes grupos, por exemplo, uma pasta pode conter todas as fotos de Lisboa, outra pasta todas as fotos de Sintra. Dentro de cada pasta existem grupos de semelhança. Caso as imagens não tenham ainda sido tratadas, estas são apresentadas ao nível da pasta, após

o cálculo dos grupos de semelhança, é criado mais um nível, onde são apresentados os grupos de semelhança e dentro de cada grupo de semelhança estão as imagens similares.É neste ponto que o utilizador poderá fazer a ordenação da imagem. Existe sempre um nível abaixo das imagens onde é possível ver a imagem em ponto grande.

3.3.2 Desenho da interface gráfica

A interface gráfica de uma aplicação é um dos seus componentes mais importantes, visto que será através desta que o utilizador consegue interagir com a aplicação.

A interface de uma aplicação não técnica, como é o caso da *photo|uniq*, deve ser simples e intuitiva, de forma a que qualquer utilizador consiga utilizar a aplicação na sua primeira utilização, para isso é necessário que a interface seja simples evitando ruído visual desnecessário, uma das estratégias para garantir a simplicidade é, sempre que possível, manter-se o padrão gráfico e de interação do sistema em que a aplicação está a correr.

Figura 3.3: Base da interface da aplicação .

A versão iOS da *photo|uniq* tem cinco vistas todas com a mesma estrutura base (fig: 3.3). Esta estrutura e composta por uma barra superior, a barra de navegação, uma área central onde será apresentado o conteúdo seja para ser visualizado (galeria) ou trabalhado (Workspace) e em baixo uma tab bar cuja finalidade é alternar entre o workspace e a galeria *photo|uniq*.

3.3.2.1 Workspace

Vista do workspace 3.4 Vista inicial da aplicação. Nesta vista a barra de navegação permite adicionar uma nova pasta (lado esquerdo), sendo possível, posteriormente, enviar imagens para a pasta criada de forma a serem trabalhadas, e no lado direto existe o

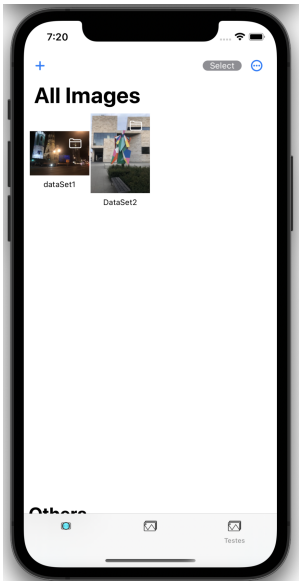


Figura 3.4: Vista do Workepace.

menu de opções, que neste nível apenas possibilita que sejam importadas imagens. É possível importar imagens de três formas distintas: importar todas as imagens, sendo estas adicionadas a uma pasta pré-definida; importar imagens singulares (selecionar que imagens se pretende importar) sendo também adicionadas à pasta pré-definida e importar um álbum da aplicação Fotografias, sendo criada uma pasta com o nome do álbum e um botão para ativar o modo de seleção, permitindo assim ao utilizador fundir ou remover pastas. Na área de trabalho após serem importadas imagens apenas é visível a lista de pastas já criadas, sendo utilizada como capa a primeira imagem do 1 grupo de semelhança .

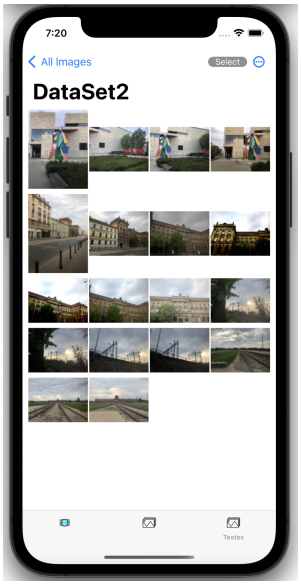


Figura 3.5: Vista de uma pasta.

Vista de uma Pasta 3.5 Nesta vista a barra de navegação, do lado esquerdo, não permite adicionar novos elementos, mas sim recuar para o nível anterior, do lado direito mantém o menu de opções3.6 e ativação do modo de seleção. Na Vista de pastas as opções disponíveis são:

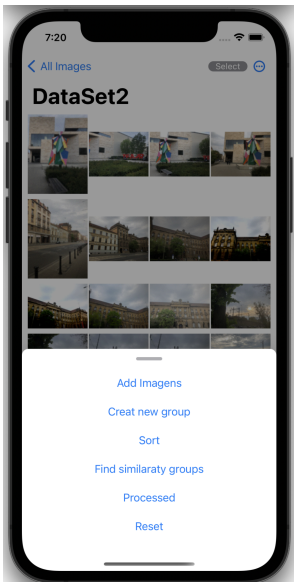


Figura 3.6: Exemplo da apresentação do menu de opções.

Adicionar uma nova imagem a pasta - Permite mover para a pasta selecionada uma imagem de outra pasta na aplicação;

Criar um grupo de semelhança vazio - permite criar um grupo de semelhança vazio;

Ordenar automaticamente todos os grupos - ordena todas as imagens em todos os grupos de semelhança. Está ordenação é feita com base no algoritmo de ordenação automatizado;

Gerar grupos de semelhança - analisa as imagens de forma a gerar os grupos de semelhança, imagens não agrupadas são adicionadas ao mesmo grupo de imagens, evitando assim a criação de grupos em excesso;

Marcar o grupo como processado - finaliza o trabalho nos grupos de semelhança marcados como processados.Ao finalizar o trabalho as imagens aprovadas pelo utilizador são enviadas para a galeria *photo|uniq*;

Reset Fazer o reset ao grupo ou fazer o reset os grupos de semelhança já criados são desfeitos.

A área de trabalho de uma pasta pode ter imagens não agrupadas, estado inicial, todas as imagens agrupadas, sendo apresentados os conjuntos de imagens semelhantes,

ou grupos de semelhança e imagens não agrupadas. É possível existirem imagens não agrupadas e grupos de semelhança caso tenham sido adicionadas novas imagens após a geração dos grupos de semelhança.

Vista de um grupo de semelhança Tal como no nível anterior, nesta vista do lado esquerdo da barra de navegação é possível recuar um nível e do lado direito o utilizador tem o botão de seleção e o menu de opções. As opções disponíveis num grupo de semelhança são: comparação lado a lado caso apenas exista uma imagem esta opção está desativada,ordenar apenas o grupo aberto, adicionar imagens,já importadas, ao grupo de semelhança e marcar o grupo como processado. Quando um grupo é marcado como processado as imagens não marcadas como melhores são ocultadas, caso o grupo seja desmarcado como processado as imagens são novamente apresentadas.

Figura 3.7: Exemplo da apresentação de uma imagem individual.

Vista de uma imagem individual 3.7 Esta vista é a responsável por apresentar ao utilizador uma imagem individual. Nesta vista o utilizador pode navegar pelas fotografias, na pasta ou grupo de semelhança. A barra de navegação permite apenas voltar ´à vista anterior, no menu de opções, que se encontra na parte inferior da vista é possível marcar a imagem como favorita, aceder as opções sobre a imagem (onde é possível entre outras funcionalidades enviar a imagem para outro grupo de semelhança).

Comparação lado a lado 3.8 Nesta vista o utilizador consegue comparar duas imagens, que quando é aplicado zoom o utilizador irá focar um ponto equivalente nas duas imagens. Quando a imagem não tem escala aplicada em baixo de cada imagem é apresentada a posição da imagem na lista de imagens, mas quando é aplicado zoom esse valor, assim

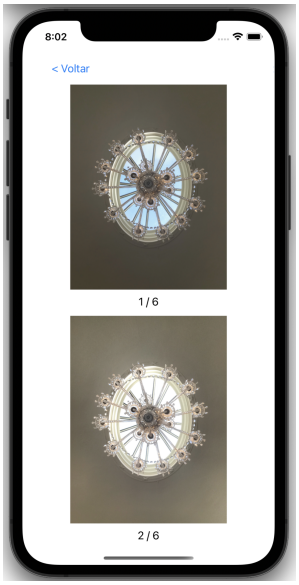


Figura 3.8: Exemplo da apresentação da comparação lado a lado (homografia).

como a barra de navegação é ocultado, possibilitando não só que a imagem tenha mais espaço para ser analisada como ajuda o utilizador a perceber que a imagem tem zoom.

3.3.2.2 Galeria *photo|uniq*

componente responsável por apresentar, depois de processadas, as melhores imagens para o utilizador, este componente, nesta primeira versão é apenas responsável por mostrar as melhores imagens. Na vista principal da galeria a barra de navegação apenas permite criar álbuns. Na Vista de álbuns apenas é possível ver as melhores imagens.

3.4 Implementação

Nesta secção são descritas algumas das decisões tomadas durante o processo de desenvolvimento da aplicação.

3.4.1 Criação dos grupos de semelhança

A geração grupos de imagens semelhantes é o coração da aplicação, ou seja, a funcionalidade principal.

Na primeira versão do *photo|uniq* para iOS este processo divide-se em duas etapas: extração de informações das imagens recorrendo a framework Vision da Apple e agrupamento de imagens, onde foi utilizada a implementação do DBSCAN disponibilizada pala NSHipster [37].

Para a extração das informações foi seguida a estratégia dos algoritmos de hash referidos nas secção 2.4.3.1 onde uma imagem é reduzida para uma miniatura de forma a que os elementos que não são o foco da imagem (ruído visual) possam ser descartados.

Este conceito baseia-se na ideia que se um elemento tem destaque com uma resolução de $5000x5000$ também terá destaque com uma resolução 50 x 50, já se um elemento considerado ruído visível a $5000x5000$, a probabilidade de ser visível numa resolução de $50x50$ é reduzida. Como tal para a extração de informação de uma imagem esta foi reduzida para $50x50$, que além de descartar o ruído visual aumenta a performance da aplicação, pois quanto menor for a imagem a analisar mais rápido será o processo de extração de informações.

A implantação do DBSCAN utilizada está dividida em duas etapas: cálculo da distância entre todos os pares de imagens e agrupamento das imagens semelhantes.

Para o cálculo da distância (diferença) entre duas imagens é utilizada a informação extraída de cada imagem e calculada a distância entre as informações que representam o conteúdo de cada imagem. O cálculo da distância devolve um valor com vírgula flutuante, onde quanto mais próximo de zero for esse valor mais similares são as imagens, se a distância entre duas imagens for zero as imagens são iguais. Estando a distância entre todas as imagens calculada, a próxima etapa será definir o conceito de imagens semelhantes, de forma a ser possível agrupar as imagens com um dado critério. Após testes realizados com quatro conjuntos de imagens com características distintas foi definido que a distância admissível estaria entre 9 e 10. No caso da distância ser 9, iria gerar poucos falsos positivos (imagens agrupadas que não o deveriam ter sido), por outro lado, irá gerar mais falsos negativos (imagens consideradas únicas quando, na verdade, existem imagens semelhantes), já no caso da distância ser 10 irá aumentar o número de falsos positivos e diminuir o número de falsos negativos, acima de 10 em alguns conjuntos de imagens o algoritmo gera demasiados falsos positivos.

Utilizando o DBSCAN para agrupar imagens, obriga a que o limiar para considerar duas imagens semelhantes seja mais rígido, dado que o DBSCAN agrupa as imagens de forma transitiva, garantindo que as imagens contidas no grupo são semelhantes, não a todas as imagens contidas no grupo, mas para ser adicionada ao grupo basta ser semelhante apenas uma das imagens contidas no grupo. O facto de o algoritmo ser transitivo obriga a que o limiar para considerar duas imagens semelhantes seja mais baixo, de forma a minimizar a existência de falsos positivos.

Estando definido o conceito de imagem semelhante para a aplicação a etapa seguinte será agrupar de forma transitiva as imagens semelhantes.

Após o agrupamento estar concluído o algoritmo devolve uma lista com as imagens sem duplicados e uma lista com todos os grupos de semelhança gerados.

3.4.2 Criação da interface

A framework para a criação da interface gráfica é das escolhas mais importantes e das que se deve ter mais ponderação, na fase inicial do desenvolvimento de uma aplicação visto que após a aplicação estar desenvolvida, ou num estado avançado, é complexo mudar por completo a framework selecionada.

Em iOS, como mencionado no ponto, existem atualmente duas frameworks nativas para a criação de interfaces o UIKit e o swiftUI.

Para a desenvolvimento da interface da versão iOS da *photo|uniq* optou-se pelo swiftUI, que não sendo a tecnologia mais madura traz outras vantagens como: não ser necessário definir regras para que o layout se ajuste a todas as resoluções; as interfaces gráficas programadas de forma declarativa diretamente em swift torna a sua renderização mais rápida em comparação com UIKit; caso os dados salvos em core data sofram alterações as vistas swiftUI são atualizadas automaticamente, o mesmo acontece com as variáveis de estado, que quando são alteradas, essa alteração leva a que a vista seja atualizada.

O facto de o swiftUI ainda estar em desenvolvimento leva a que nem tudo o que existe em UIKit esteja já implementado em swiftUI, sedo necessário utilizar vistas UIKit na interface desenvolvida em swiftUI.

3.4.3 Persistência de dados

Sabendo que uma das estratégias para melhorar o desempenho da aplicação, evitando que esta faça cálculos desnecessários é, sempre que possível, persistir o resultado de todos os cálculos, como base neste conceito torna-se imperativo que existe um mecanismo de persistência na *photo|uniq* para iOS.

Tal como visto no ponto 3.4.3, existem diversas formas de salvar dados em iOS e no caso do *photo|uniq* optou-se por utilizar o Core Data como sistema de base de dados geral da aplicação onde são salvos todos os dados gerados pela aplicação (imagens importadas, grupos de semelhança gerados, entre outros). Esta escolha foi feita com base na simplicidade de integração e utilizaçãoe velocidade do core data assim como na existência da automática da interface quando existem alterações no conteúdo da salvo no Core Data.

Alguns dos algoritmos utilizados requerem parametrização. De acordo com as boas práticas valores diretamente no código são desaconselhados, de forma a evitar esse procedimento, nesta versão da aplicação foram utilizados ficheiros de propriedades para armazenar esses dados permitindo alterar esses valores sem necessidade de alterar nada no código. Outra utilização dos ficheiros de propriedades é nas definições, uma vez que estas estão localizadas fora da aplicação é necessário que de alguma forma a aplicação tenha acesso aos valores definidos nas opções, para que tal seja possível também é utilizado um ficheiro propriedades.

3.4.4 Correlação de imagens

A correlação de imagens é o processo de fazer keypoints de uma imagem A corresponderem a keypoints de uma Imagem B. Esta correlação nada tem a ver com criação de grupos de semelhança, sendo apenas feita após os grupos estarem gerados e possibilitará o cálculo da matriz de homografia para que seja possível comparar duas imagens semelhantes com foco em pontos equivalentes.

A correlação de imagens é um processo composto por duas etapas:

Deteção de keypoints Corresponde ao processo de deteção de pontos que se destacam numa imagem, seja no ponto onde existe uma mudança súbita de contraste ou uma aresta.

Existem vários algoritmos para deteção de keyPoints(sec 2.4.1), dos algoritmos analisados nesta dissertação a escolha para a implementação na versão iOS da *photo|uniq* foi o ORB visto que este ao contrário do SURF e SIFT é de código aberto, é tendencialmente mais rápido sedo os resultados equivalentes aos restantes;

Correlação keypoints a correlação de keyPoints é o processo de fazer corresponder os keypoints de uma imagem aos keypoints equivalentes de outra imagem. Nem todas as correspondências são correspondências reais, ou seja os algoritmos de correlação geram sempre falsos positivos sendo necessário aplicar o Lowe's Ratio test de forma a reduzir, ou mesmo eliminar os falsos positivos.

Dos algoritmos analisados optou por se utilizar o BRUTEFORCE, visto que o FLANN não é compatível com os descritores obtidos através do ORB.

3.4.5 Análise de qualidade das imagens

Como demonstrado na secção 2.4.4 existem várias opções para se calcular de forma automática a qualidade de uma imagem, de forma a ser possível propor ao utilizador uma ordenação baseada na qualidade. Na a primeira versão iOS da *photo|uniq* foi selecionado o BRISQUE [35] [40].

Esta opção foi selecionada com o intuito de reduzir a complexidade de configuração, visto que o BRISQUE funciona de forma autónoma, ou seja, sem que o utilizador tenha necessidade de parametrizar, porém, é necessário fornecer modelos de treino para que o algoritmo ganhe a noção de qualidade. No âmbito desta dissertação foi utilizado o modelo de treino fornecido no exemplo da utilização do algoritmo fornecidos pelo openCV. Este modelo de treino foi treinado com a base de dados LIVE-R2 [29] e avaliado com a base de dados TID2008 [42].

4

Validação da Solução

Neste capítulo será apresentada a validação funcional e não funcional da solução. Com a validação funcional pretende-se avaliar o desempenho da aplicação, analisando a qualidade dos grupos gerados assim como a ordenação automática proposta. Por seu lado com a validação não funcional pretende-se analisar como o consumo de recursos — tempo, RAM, processador e energia — são afetados com o aumento do número de fotografias.

4.1 Validação funcional

Sendo o objetivo da aplicação *photo|uniq* ajudar o utilizador a selecionar as melhores imagens de entre um grupo de imagens semelhantes, podemos afirmar que os principais algoritmos da aplicação são o algoritmo de geração de grupos de semelhança e o algoritmo de ordenação automática de grupos de semelhança.

Nesta secção será feita a validação funcional dos algoritmos mencionados anteriormente com o objetivo de se perceber se a aplicação cumpre os requisitos e, caso não o faça, avaliar que pontos necessitam ser melhorados.

4.1.1 Geração de grupos de semelhança

Para a validação funcional da geração de grupos de semelhança foram criados dois data sets, e foi pedido a 5 utilizadores para criarem grupos de semelhança para cada data set. Os data sets utilizados distinguem-se por o Data Set 1A.1 ser, em teoria, relativamente óbvio de agrupar, sendo o Data Set 2A.1 mais subjetivo.

Estando os grupos de semelhança criados pelos voluntários, estes são comparados com os gerados pela aplicação.

De forma a ser possível comparar os resultados obtidos pela aplicação com os resultados gerados pelos voluntários, é necessário definir uma estratégia eficiente para realizar essa comparação. A estratégia de comparação utilizada para validar o algoritmo de geração de pares da versão iOS da *photo|uniq* foi a comparação de pares, ou seja, contabilizar todos os pares nos grupos, tanto os gerados pela aplicação como os criados pelos voluntários e contabilizar, os pares corretos, incorretos (falsos positivos) e ausentes (falsos

negativos), tendo sempre por base os grupos gerados pelos voluntários. De forma a garantir que os dados são uniformizados, foi calculada a percentagem para cada um dos casos.

Partindo do data set "1, 2, 3, 4, 5, 6, 7, 8, 9, 10", se os voluntários criarem dois grupos:"2, 4, 6, 8, 10", e "1, 3, 5, 7, 9", se o algoritmo gerar 4 grupos"1, 2, 3" ; "4, 6, 10" ; "8" ; "5, 7, 9".

O resultado seria:

- Total de pares: "2, 4"; "2, 6"; "2, 8"; "2,10"; "4, 6"; "4, 8"; "4,10"; "6, 8"; "6,10"; "8, 10"; "1, 3"; "1, 5"; "1, 7"; "1, 9"; "3, 5"; "3, 7"; "3, 9"; "5, 7"; "5, 9"; "7, 9- 20 pares
- Pares corretos: "1, 3" ; "4, 6" ; "4, 10" ; "6, 10" ; "5, 7" ; "5, 9" ; "7, 9" - 7 pares;
- Falsos Positivos: "1, 2"; "2, 3" - 2 pares;
- Falsos Negativos: "1, 5" ; "1, 7" ; "1, 9" ; "3, 5" ; "3, 7" ; "3, 9" ; "2, 4" ; "2, 6" ; "2, 8" ; "2, 10" ; "4, 8" ; "6, 8" ; "8, 10" - 13 pares.

Os resultados obtidos nos testes com utilizadores (tabela 4.1) foram inesperados, uma vez que os utilizadores agruparam as imagens de uma forma mais lata, agrupando as imagens maioritariamente por assunto e não por semelhança, o que fez com que, por exemplo, todas as imagens de um dado edifício sejam adicionadas ao mesmo grupo independentemente do seu grau de semelhança. Com esta abordagem, na maioria dos casos, aconteceu que todas as fotografias foram agrupadas, não existindo imagens sem duplicados.

O facto de todas as imagens do mesmo assunto estarem no mesmo grupo, tem impacto na análise de desempenho da aplicação. Se assumirmos que os grupos gerados pela aplicação contêm imagens de alguma forma semelhantes, e como tal do mesmo assunto, a probabilidade de serem gerados falsos positivos (pares de imagens mal agrupadas) é baixa, por outro lado, uma vez que os grupos são maiores, contendo imagens do mesmo assunto independentemente de serem semelhantes ou não, de acordo com os utilizadores existem muitos falsos negativos (pares de imagens não agrupados quando deviam ter sido).

Tabela 4.1: resultados da análise funcional da geração de grupos.

	Data Set 1	**Data Set 2**
Falso Negativo	21	29
Falso Positivo	0	0
Pares Corretos	19	8

No Data Set 1 a aplicação cria três grupos com imagens semelhantes e um de imagens sem duplicados, no grupo 1 foram adicionadas duas imagens, no grupo 2 foram adicionadas 6 imagens, e no grupo 3 foram adicionadas 3 imagens, totalizando um total de 19 pares. No caso do Data Set 2 a aplicação também gera três grupos de imagens similares

e um de imagens sem duplicados, onde no grupo 1existem quatro imagens, no grupo 2 duas imagens e no grupo 3 duas imagens, sendo dez imagens consideradas como únicas pela aplicação, totalizando um total de oito pares. Para o efeito de análise do desempenho da aplicação na geração de grupos de semelhança (deteção de falsos positivos, falsos negativos e pares corretamente gerados), o conjunto de imagens únicas não foi tido em consideração.

Sabendo que o utilizador agrupou as imagens de forma mais lata, agrupando todas as imagens, que de uma forma ou outra estão relacionadas, fez com que todos os pares gerados pela aplicação sejam válidos, ou seja, não existindo falsos positivos (pares gerados incorretamente), por outro lado, uma vez que os grupos criados pelos utilizadores são de espectro mais amplo, leva a que estes tenham mais imagens, o que leva à existência de mais falsos negativos, ou seja, imagens não agrupadas corretamente, podendo essas imagens estar divididas por vários grupos ou serem consideradas únicas.

4.1.2 Ordenação automática

Para a validação funcional da ordenação foi realizado um teste sintético e um teste funcional com utilizadores reais.

Para a validação sintética foram selecionadas 4 imagens com características distintas e aplicadas as transformações: exposição: $-0,5$ e $+0,5$; blur: $1,5$; $3,5$ e 10. O Objetivo deste teste é perceber como o algoritmo de ordenação automática se comporta com imagens com características distintas.

No caso da validação funcional, com utilizadores, foram utilizados quatro data sets, sendo um deles composto pela mesma imagem modificada digitalmente, e os três restantes data sets reais. Para a realização destes testes foi pedido a cinco utilizadores que ordenassem as imagens.

Como metodologia para a validação funcional da ordenação automática, visto que não existiu consenso entre os utilizadores, foi calculada para cada grupo, a posição média de cada imagem sendo posteriormente as imagens ordenados com base na posição média obtida, por fim e de forma a validar a qualidade do algoritmo proposto é comparada a posição obtida nos testes com os resultados da aplicação.

4.1.2.1 Validação sintética

Com a validação sintética do algoritmo de ordenação pretende-se verificar o comportamento do algoritmo quando aplicado a imagens com características distintas, sendo assim possível perceber os pontos fracos do algoritmo da forma que foi parametrizado

Segundo a tabela 4.2 a parametrização do BRISQUE utilizada no desenvolvimento da versão iOS da *photo|uniq* é coerente na análise do blur, onde em todas as imagens testadas a ordenação é a mesma e correta, estando as imagens ordenadas por intensidade do blur, por outro lado, se a análise se focar na avaliação da exposição, o algoritmo já não é coerente, onde a ordenação proposta não foi a mesma em todos os casos.

Tabela 4.2: Resultados do BRISQUE.

	Base	**Exposição** (−0,5)	**Exposição** (+0,5)	**Blur** (1,5)	**Blur** (3,5)	**Blur** (10)
Imagem 1 (B.1.1)	9.7381	8.9805	10.3709	57.4843	76.939	78.8337
Imagem 2 (B.1.2)	22.8891	23.4020	22.2513	51.8938	66.1775	70.9680
Imagem 3 (B.1.3)	22.5069	20.6187	20.2060	60.8675	79.7380	81.3826
Imagem 4 (B.1.4)	8.8630	7.8428	5.8804	73.3680	96.5491	100.0000

O facto de a ordenação por exposição não ser a mesma não significa necessariamente que o algoritmo é ineficiente ou que gera resultados incorretos, visto que a análise de exposição de uma imagem é subjetiva, por outras palavras, se as imagens base não estão todas eximiamente expostas, pequenas alterações na exposição tanto para cima como para baixo podem melhorar a imagem. Isto acontece visto que, em teoria, quanto mais detalhe for possível distinguir numa imagem mais bem exposta esta estará.

O conceito de boa exposição é algo bastante complexo de definir visto que existem muito mais fatores além da quantidade de detalhe percetível na imagem.

4.1.2.2 Validação de qualidade

Os resultados obtidos nos testes de ordenação podem ser vistos e consultados de forma sintética nas tabelas

- Data Set 1B.2.1 (sintético com uma imagem modificada digitalmente) - tabela 4.3;
- Data Set 2B.2.2 (com imagens praticamente iguais) - tabela 4.4;
- Data Set 3B.2.3 (real com imagens urbanas) - tabela 4.5;
- Data Set 4B.2.4 (real com imagens de natureza) - tabela 4.6.

Como pode ser constatado pela análise das tabelas da ordenação a ordenação proposta pela aplicação nunca é 100% de acordo com a proposta pelos utilizados, porem nos Data Sets 3 e 4 a melhor imagem para os utilizadores é a melhor imagem para a aplicação, no caso do Data Set 1 as duas primeiras imagens são invertidas, estando separadas por uma distância de 3,5%, a aplicação considera a foto 1 do Data Set 2 3,5% melhor que a foto 2, a melhor imagem para os utilizadores.

Com este teste foi também possível comprovar que o BRISQUE com a configuração utilizada, não é eficiente na análise da exposição de fotografias

Existem duas possibilidades para avaliar o desempenho do algoritmo de ordenação automática: comparar apenas a melhor imagem para o algoritmo com a melhor imagem para os utilizadores ou comparar os resultados globais do algoritmo, os obtidos na aplicação versus os obtidos através dos testes com utilizadores. Caso se analise apenas a seleção da melhor imagem, a aplicação tem resultados decentes, existindo concordância entre os

resultados da aplicação e a proposta dos utilizadores em duas de quatro possibilidades (tabela 4.5 e 4.6) , com uma terceira onde a primeira e segunda fotografias estão trocadas (tabela 4.4), estando separadas por 3,5 pontos em 100. Por outro lado se avaliarmos o desempenho do algoritmo de forma global a sua performance é mais fraca visto que o resultado da ordenação proposto pela aplicação nunca é igual ao obtido através dos testes com utilizadores.

No contexto da *photo|uniq*, visto que o seu objetivo é singularizar imagens, ou seja, idealmente manter apenas uma imagem por cada grupo de semelhança, pode-se concluir que a performance da ordenação tem resultados positivos, uma vez que para três dos quatro data sets analisados a melhor imagem para os utilizadores é a melhor imagem para a aplicação ou tem uma classificação próxima da melhor.

4.2 Análise de escalabilidade temporal

Nesta secção será feita uma análise de escalabilidade de forma a ser possível não só obter o custo relativo de processar mais uma imagem mas também validar se esse custo está relacionado com a complexidade algorítmica do algoritmo utilizado.

No âmbito da análise da escalabilidade temporal a complexidade interna dos algoritmos utilizados foi ignorada. Foi possível ignorar a complexidade interna dos algoritmos visto que estes podem ser vistos como uma caixa negra, onde tem como parâmetro de entrada, a imagem e de saída o seu resultado, seja ele as feature prints ou a avaliação da qualidade geral da imagem. Na análise de escalabilidade foi possível ignorar a complexidade interna do algoritmo uma vez que os valores de entrada são sempre uma imagem redimensionada para a mesma resolução em pixels. O facto dos parâmetros de entrada serem semelhantes toda a complexidade dos algoritmos de análise de qualidade e de extração de feature prints seja linear $\mathcal{O}(n)$, estando apenas dependente do número de imagens, por seu lado a complexidade do algoritmo de agrupamento, no caso da versão iOS do *photo|uniq* é o DBSCAN, a complexidade teórica e $\mathcal{O}(n^2)$.

Como mencionado no ponto 2.7.1, o simulador não deve ser utilizado apenas para análise de desempenho relativo, tendo de ser realizada a análise de desempenho real num dispositivo físico. Sendo o objetivo desta análise de escalabilidade perceber o custo temporal relativo de processar mais uma imagem. Todos os testes para a análise de escalabilidade temporal foram realizados num simulador do iPhone 12 Pro max a correr o iOS 14.5, usando como host um iMac i9 com 32 Gb de RAM.

No contexto da análise de escalabilidade, a importação de imagens é o processo em que a aplicação vai à biblioteca do telefone (Aplicação Fotografias) aceder e copiar para si as referências das imagens que pretende importar. Na importação não está incluído o tempo que a aplicação demora a carregar efetivamente uma imagem mas sim o tempo que a aplicação demora a aceder à biblioteca do equipamento e copiar as referencias para as imagens.

Tabela 4.3: Resultados da ordenação do Data Set 1.

	Ordenação nos testes	**Ordenação da Aplicação**	**Score da Aplicação**
Fotografia 1	3	1	18,902
Fotografia 2	2	2	20,027
Fotografia 3	1	3	24,505
Fotografia 4	5	4	69,787
Fotografia 5	4	5	96,699

Tabela 4.4: Resultados da ordenação do Data Set 2.

	Ordenação nos testes	**Ordenação da Aplicação**	**Score da Aplicação**
Fotografia 1	2	1	31,847
Fotografia 2	1	2	35,367
Fotografia 3	4	3	35,579
Fotografia 4	5	4	35,901
Fotografia 5	3	5	46,379
Fotografia 6	6	6	87,416

Tabela 4.5: Resultados da ordenação do Data Set 3.

	Ordenação nos testes	**Ordenação da Aplicação**	**Score da Aplicação**
Fotografia 1	1	1	7,135
Fotografia 2	4	2	14,454
Fotografia 3	2	3	37,628
Fotografia 4	2	4	47,009
Fotografia 5	5	5	52,001

Tabela 4.6: Resultados da ordenação do Data Set 4.

	Ordenação nos testes	**Ordenação da Aplicação**	**Score da Aplicação**
Fotografia 1	1	1	15,509
Fotografia 2	3	2	19,956
Fotografia 3	2	3	22,057
Fotografia 4	5	4	39,254
Fotografia 5	4	5	50,264

A complexidade teórica da importação de imagens é: no pior dos casos, $\mathcal{O}(n^2)$, visto que após importar uma imagem, é verificado se a imagem a importar ja se encontra na aplicação, caso já exista na aplicação, a imagem será descartada, por seu lado se for uma imagem nova, sem ainda ter sido importada, será adicionada à lista de imagens; e $\mathcal{O}(n)$ no melhor dos casos. Este caso acontece quando se importa pela segunda vez a primeira imagem da lista.

Para a realização da análise da escalabilidade e posterior comparação com a complexidade teórica foram importados, data sets com 10, 100, 500, 1000, 2000, 5000, 7500 e 10000, sendo cada um dos data sets testados dez vezes.

Os data sets referidos foram analisados em dois cenários distintos: o primeiro onde todas as imagens são únicas (sem terem sido previamente importadas); e um segundo cenário onde são importadas metade das imagens, sendo, em seguida, importadas novamente, de forma a garantir que 50% das imagens já se encontram devidamente importadas

De acordo com os testes realizados, com os resultados resumidos no gráfico 4.1, é possível comprovar que, no caso da importação de imagens sempre diferentes (linha azul), a complexidade da importação é quadrática ($\mathcal{O}(n^2)$). Este cenário tem a complexidade quadrática visto que ao importar uma imagens esta é comparada com todas as imagens já importadas, sendo no final adicionada à lista de imagens

No segundo cenário metade das imagens são importadas, sendo estas posteriormente re-importadas a complexidade é linear, isto acontece uma vez que ao se importar uma imagens já importada é necessário percorrer a lista e sendo todas as imagens importadas duas vezes, o acesso a lista será sempre linear

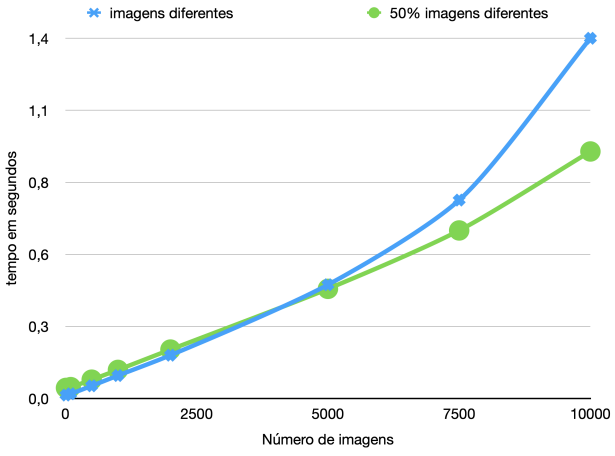


Figura 4.1: Escalabilidade da importação de imagens.

4.2.1 Carregamento de imagens

Por carregamento de imagens entende-se o processo de colocar as imagens na memória, sendo estas lidas a partir da aplicação fotografias. Esta análise é importante uma vez que

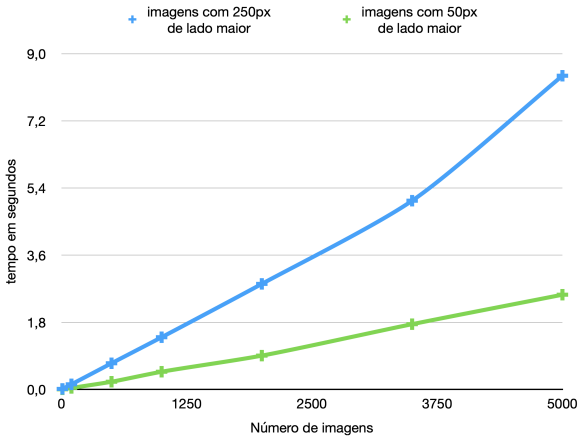


Figura 4.2: Escalabilidade do carregamento de imagens.

o carregamento das imagens é uma das etapas base de todas as principais funcionalidades da aplicação.

No carregamento de imagens existem dois cenários possíveis: quando a aplicação está a gerar os grupos de semelhança, onde as imagens têm todas a mesma resolução; e todos os outros algoritmos onde as imagens têm a sua resolução original, podendo esta variar,

Sabendo que, aquando da geração de grupos de semelhança todas as imagens são carregadas com a mesma dimensão, a complexidade teórica do carregamento de imagem, neste caso e é de $\mathcal{O}(n)$.

Conforme os testes realizados, com resultados resumidos no gráfico 4.2,a complexidade da importação das imagens é $\mathcal{O}(xn)$), onde x representa o declive da reta. O valor de x está diretamente relacionado com a dimensão da imagem, onde quanto mais pequena for a imagem mais rápido esta é carregada.

4.2.2 Extração de informações

A extração de informações de uma imagem é o processo de análise de uma imagem de forma ser possível calcular os feature prints com o objetivo de posteriormente ser possível calcular a distância entre os feature prints de duas imagens com o objetivo de perceber o quão semelhantes às imagens são, onde quanto mais baixa for a distância entre os feature prints mais semelhantes são as imagens.

Sabendo que para a extração de informações todas as imagens são redimensionadas para a mesma resolução, e as imagens apenas são analisadas uma vez considera-se a complexidade teórica do algoritmo de extração de informações de $\mathcal{O}(n)$, onde n representa o número de elementos na lista.

Foi analisada a escalabilidade na extração de informações em dois cenários, com imagens diferentes e sempre com a mesma imagem:Com estes dois cenários pretende-se perceber se a complexidade das imagens afeta a escalabilidade.

De acordo como o gráfico 4.3 que representa a média dos tempos para a extração de informações, ambos os casos são perfeitamente lineares como diferença mínima, como tal pode se concluir que para a escalabilidade não é relevante a complexidade da imagem.

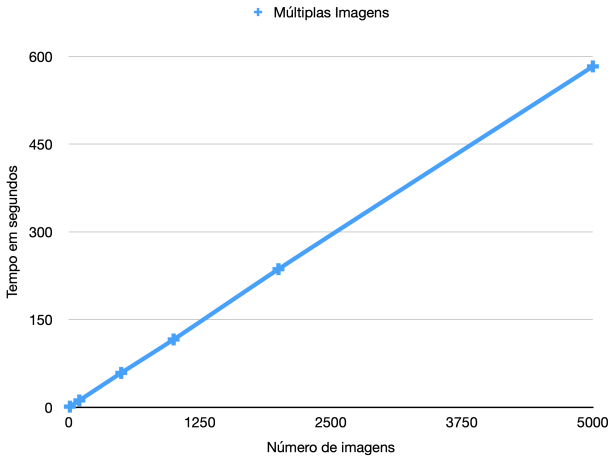


Figura 4.3: Escalabilidade do agrupamento de imagens.

4.2.3 Agrupamento de imagens semelhantes

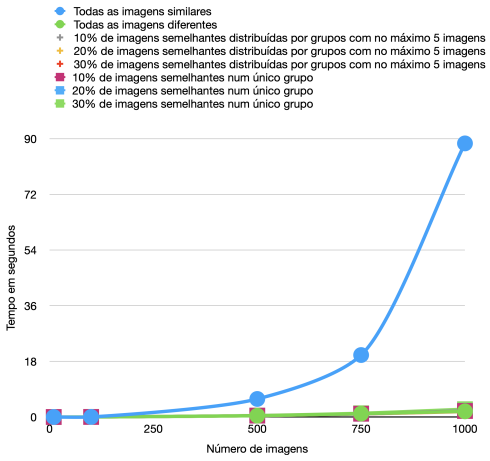


Figura 4.4: Escalabilidade do agrupamento de imagens.

O algoritmo selecionado para o agrupamento de imagens semelhantes é o DBSCAN sendo a sua complexidade teórica, no pior dos casos $\mathcal{O}(n^2)$, contudo caso seja utilizada uma estrutura de indexação otimizada e o número de elementos for baixo a complexidade pode ser reduzida para $\mathcal{O}(n \log n)$ [36].

Para a análise de escalabilidade foram considerados os cenários: Todas as imagens são semelhantes, nenhuma das imagens é semelhante, 10%, 20% e 30% de imagens semelhantes, sendo criado um grupo para percentagem e com a mesma percentagem de

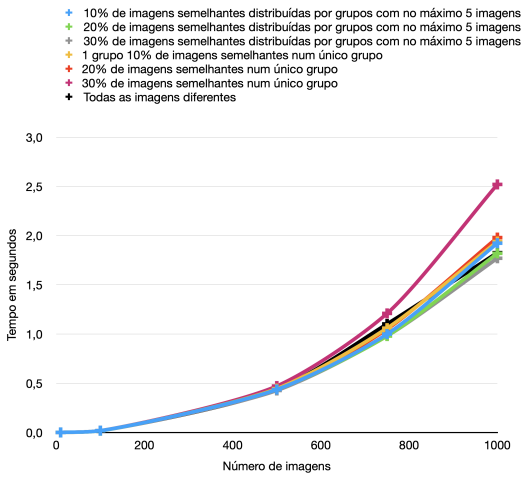


Figura 4.5: Ampliação da escalabilidade do agrupamento de imagens).

imagens semelhantes mas formando múltiplos grupos com um máximo de cinco imagens por grupo.

Para realizar este teste o nível de semelhança entre as imagens foi definido artificialmente, nos primeiros dois casos, alterado o limiar para considerar imagens semelhantes, nos restantes garantindo que as primeiras imagens têm as mesmas informações para terem uma distância de 0.

Os resultados dos testes de escalabilidade realizados ao DBSCAN estão resumidos no gráfico 4.4 visto que, o tempo de agrupamento de um conjunto com todas as imagens semelhantes se destaca, achatando os demais numa única linha, optei por fazer uma ampliação da zona achatada, que pode ser consultada no gráfico 4.5.

Ao analisar o primeiro gráfico (gráfico: 4.4) é possível constatar que se todas as imagens do grupo forem similares entre si o algoritmo tem a pior performance possível, piorando de forma quadrática com o aumento do número de imagens, nos testes realizados este aumento começa-se a destacar pelas 250 imagens.

Analisando o segundo gráfico (gráfico: 4.5), onde é possível analisar como os restantes casos escalam, sendo também quadráticos, porém não com tanta intensidade. Neste gráfico é possível perceber a tendência que quantas mais imagens semelhantes entre si existirem pior será a performance do algoritmo, isto pode ser constatado ao se analisar a curva que representar 30% de imagens semelhantes entre si, que se encontra destacada. Nos restantes casos a performance é semelhante.

Esta perda de desempenho prende-se com a implementação do DBSCAN utilizada onde, a lista de imagens é percorrida uma primeira vez de forma a ser possível criar um estrutura com cada imagem associada a uma label, designada por lista de pontos, estando a lista de pontos devidamente criada, esta é percorrida até não haver imagens na lista. Durante cada interação da lista é selecionada uma imagem pivot e são procuradas as imagens semelhantes a essa imagem, sendo as imagens consideradas semelhantes adicionadas a uma nova lista de imagens semelhantes, e removidas da lista de pontos.

Com a lista de imagens semelhantes criada, o algoritmo vai iterar pela lista de imagens semelhantes procurado para cada imagens por todas as imagens semelhante que ainda se encontram na lista principal. Em ambos os caso quando uma imagem e considerada semelhante a a imagem pivot é removida do grupo principal e adicionada a grupo de imagens semelhantes.

Com base nesta análise é possível afirmar que quanto mais imagens semelhantes entre si o grupo de imagens tiver, tendencialmente a performance será pior. Nos testes realizados a perda de performance torna-se mais notória quando existem 30% de imagens semelhantes entre si e pelo menos 800 imagens no conjunto de imagens a analisar. No extremo oposto está o grupo com 30% de imagens divididas por múltiplos grupos que têm uma performance marginalmente melhor que os restantes casos.

Após esta análise pode-se afirmar que a complexidade real da implantação do DBSCAN utilizada é de $\mathcal{O}(n^2)$.

4.2.4 Ordenação automática

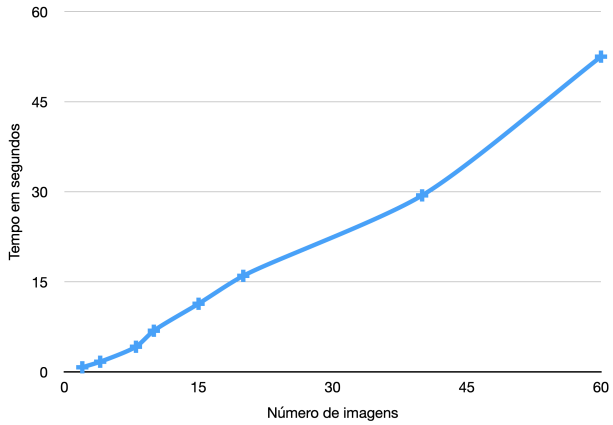


Figura 4.6: Escalabilidade do BRISQUE.

A ordenação das imagens é composta por duas etapas: análise de qualidade global da imagem com o BRISQUE e ordenação da lista de imagens. Para ordenação da lista de imagens é utilizado o algoritmo de ordenação do Swift que segundo a documentação tem uma complexidade de $\mathcal{O}(n \log n)$.[8]. Por seu lado o BRISQUE, uma vez que as imagens são analisadas aproximadamente com a mesma dimensão a sua complexidade teórica será de $O\mathcal{O}(n)$, tornando a complexidade teórica da ordenação $\mathcal{O}(n \log n)$.

Para a análise de escalabilidade foi medido o tempo que o BRISQUE, de forma isolada, que demora a processar conjuntos de imagens. Foram definidos conjuntos com 2, 4, 8 10, 15, 20, 40 e 60 imagens.

Segundo os testes realizados, resumidos no gráfico 4.6, a complexidade real da versão do BRISQUE utilizada na primeira versão iOS do *photo|uniq* é linear $\mathcal{O}(n)$, o que torna

a complexidade global da ordenação de $\mathcal{O}(n \log n)$, visto que sempre que se utilize um algoritmo composto por vários subalgoritmos a complexidade geral do algoritmo será sempre a pior complexidade de entre todos os subalgoritmos.

4.3 Análise de escalabilidade de recursos

Nesta secção será analisado o impacto que a aplicação *photo|uniq* tem nos recursos do equipamento. Para a realização dos testes foi utilizada a ferramenta de análise do debuger do Xcode, onde é possível ver o resumo da utilização da RAM, CPU, GPU, e o impacto energético. De forma a que estes testes possam ser o mais fidedigno possível, foi utilizado um equipamento real, um iPhone SE 1Gen. O foco desta análise centra-se na utilização da RAM, CPU, GPU, e impacto energético.

De todas as funcionalidades da aplicação, a análise de escalabilidade de recursos, centra-se na geração de grupos de semelhança e na sua ordenação. Foram escolhidas estas duas funcionalidades visto que são as potencialmente mais exigentes ao nível de recursos.

Para a realização destes testes foram definidos data sets com caraterísticas distintas para a geração de grupos e para a ordenação.

Para a geração de grupos de semelhança foram criados conjuntos com 100, 500, 1000 e 1500 imagens a simular data sets reais. No caso da ordenação foram criados dois conjuntos de imagens, um com todas as imagens de igual resolução e outro com várias resoluções.

4.3.1 Geração de grupos

Sendo o desta análise perceber o custo relativo do processamento de mais imagem, ou seja, perceber como o aumento de imagens a processar tem impacto na utilização de recursos do equipamento, esta analise foca-se nos consumos máximos e não no perfil de consumo. De forma a realizar estes testes forma definidos grupos com 100, 500, 1000 e 1500 imagens.

A RAM é um recurso limitado, como tal é importante que a aplicação faça um uso da RAM o mais eficiente possível. No caso da versão iOS do *photo|uniq* o consumo de RAM 4.7 acompanha a complexidade do algoritmo de geração de grupos, sendo em ambos os caso, exponenciais.

Independentemente do número de imagens existe um padrão no consumo de RAM. Quando a aplicação não tem imagens, a RAM utilizada ronda os 20Mb, após a importação de imagens a aplicação necessita mais RAM, isto acontece uma vez que tendo as imagens importadas, independentemente de estarem carregadas ou não, a aplicação irá sempre necessitar de mais memória. O segundo aumento de RAM acontece quando a pasta com as imagens é aberta, onde são carregadas miniaturas das imagens, de seguida existe um aumento abrupto do consumo de RAM, este acontece quando a framework Metal é ativada. Após a ativação da framework é iniciado o processo de análise das imagens e finaliza com a geração de grupos que têm impacto na RAM quando existem muitos grupos ou grupos

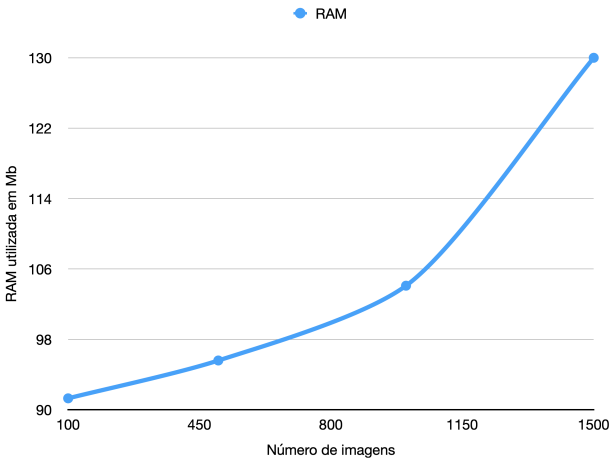


Figura 4.7: Utilização da RAM em pico.

muito grandes, esse aumento é especialmente notório nos testes realizados com 1500 imagens.

Para a extração de informações das imagens é utilizada a framework Vision, que por sua vez utiliza a framework metal para a realização dos cálculos sobre imagens. Sendo o Vision uma framework do sistema, é o sistema que gere a sua alocação e respetiva libertação de memória. No caso da aplicação *photo|uniq* o sistema aloca memória durante o setup da framework e não a liberta.

A RAM é utilizada para guardar temporariamente os dados a serem processados, mas o processamento propriamente em si, é realizado num processador seja o CPU ou GPU. Visto que para a geração de grupos de semelhança é utilizada a framework Vision da Apple, e esta recorre ao Metal para o processamento das imagens, o processo está otimizado, sendo a maior parte do processamento realizado no processador gráfico 4.8.

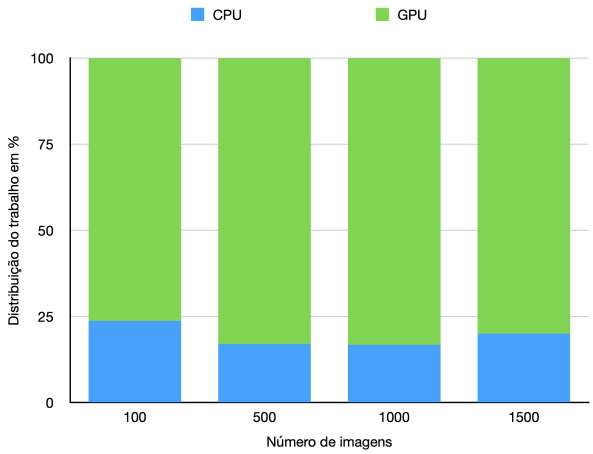


Figura 4.8: Distribuição do trabalho entre RAM e CPU.

Na geração de grupos, o GPU é responsável pelo cálculo das informações das imagens

com o objetivo de serem posteriormente calculadas a distância entre as imagens cabendo ao CPU fazer o cálculo da distância entre as imagens de forma a se perceber se duas imagens estão próximas ou não.

Com a análise da utilização do CPU é possível constatar que durante o processamento das imagens (extração das informações) a sua utilização é relativamente baixa rondando os 30% independentemente do número de imagens a tratar, existindo no final do processamento um pico para os 96%, esse pico acontece na mesma altura que existe o pico na utilização da RAM, quando as imagens únicas são agrupadas.

Sendo esta versão da *photo|uniq* para um dispositivo móvel é importante perceber o consumo energético da aplicação durante as etapas mais exigentes. Neste caso, durante a geração de grupos onde quanto mais imagens forem analisadas maior o impacto energético [Tabela consumo energia].

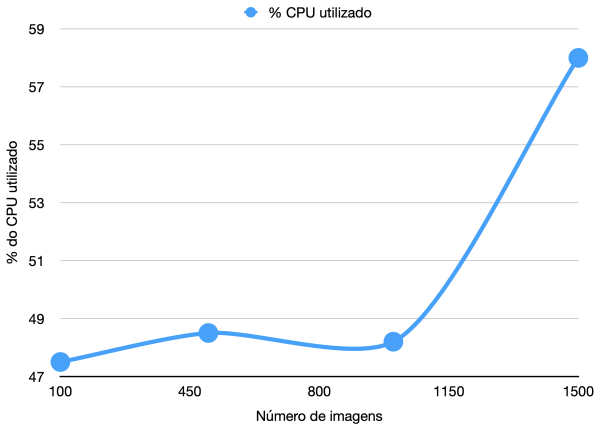


Figura 4.9: Utilização do CPU em pico.

4.3.2 Ordenação de grupos

O processo de ordenação de imagens é composto por duas etapas: cálculo da qualidade da imagem e posterior ordenação com base no valor da qualidade. A teste realizados veem este processo com um único passo, sendo por isso o processo analisado como um todo.

Com esta análise pretende-se perceber a forma como o aumento de imagens, e a variação da sua resolução influencia o consumo de recursos do equipamento. Foi analisado o impacto que a variação da resolução tem no consumo de recursos uma vez que o algoritmo utilizado analisa uma imagem de cada vez, e em teoria, libertando os recursos após cada análise.

Uma vez que o algoritmo utilizado analisa uma imagem de cada vez leva a que e que o BRISQUE analisa todos os pixies de uma imagem, leva a que em teoria, o principal fator que afeta a consumo de recurso (RAM e processador), será a resolução efetiva da imagem. Com o objetivo de confirmar esta teoria a aplicação foi testada com data sets

de duas imagens e dezasseis imagens, com baixa resolução (640×480), média resolução (1920×1280) , alta resolução (2400×1600) e muito alta resolução (5042 × 3151), dando origem aos grupos:

- 16 imagens com baixa resolução;
- 2 imagens com baixa resolução;
- 16 imagens com média resolução;
- 2 imagens com média resolução;
- 16 imagens com alta resolução;
- 2 imagens com alta resolução;
- 16 imagens com muito alta resolução;
- 2 imagens com muito alta resolução.

Em todos os casos as imagens têm nativamente a resolução indicada, não sendo redimensionadas pela aplicação.

Como metodologia de teste da aplicação, o algoritmo de ordenação foi executado diversas vezes em todos os grupos, estando os resultados em utilização da RAM resumidos no gráfico: 4.10, em que é possível perceber que, quanto mais resolução tiver a imagem e consequentemente maior dimensão física, maior será o consumo de RAM, não sendo afetada pela quantidade de imagens, onde o consumo de RAM nos testes com duas e com dezasseis imagens é semelhante.

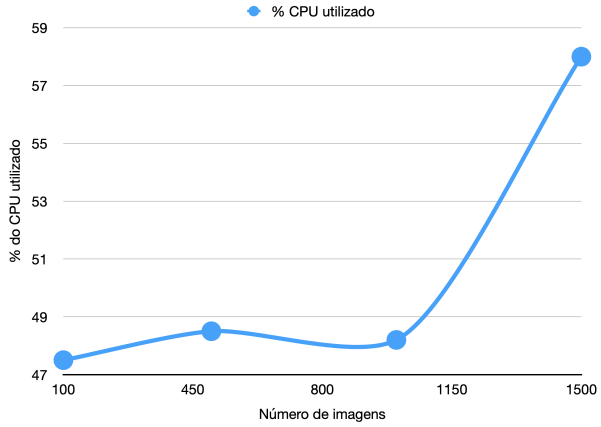


Figura 4.10: Máximo de RAM utilizada na ordenação.

Ao contrário da geração de grupos que utiliza a framework Vision a análise de qualidade é feita recorrendo à biblioteca opneCV, como tal não está otimizada, utilizando apenas o CPU para o processamento das imagens.

Por seu lado a utilização do processador é afectada pela quantidade de imagens, tendo em ambos os casos, com dezasseis imagens e com 2 imagens, a utilização do processador um crescimento logarítmico, com tendência a que, quanto maior forem as imagens mais próximo será a percentagem de utilização do processador, para quantidades diferentes de imagens, o mesmo pode ser analisado no gráfico 4.11 que representa a evolução da utilização do processador para as diversas resoluções, onde se pode constatar que quando as imagens são de baixa resolução a diferença entre as percentagens nos data sets com duas e dezasseis imagens é aproximadamente de 22,5% já quando a imagem tem uma resolução muito alta a diferença é de apenas 0,5%

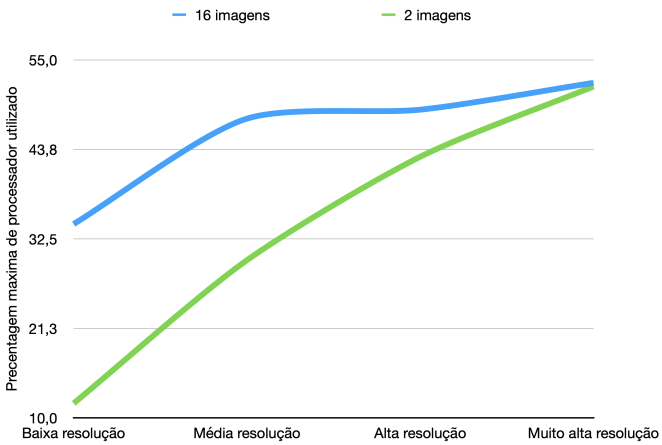


Figura 4.11: Máximo de RAM utilizada na ordenação.

5

Considerações Finais

5.1 Conclusão

Nesta dissertação foi apresentada uma proposta para o desenvolvimento de uma aplicação iOS para a singularização de fotografias. Para que esta singularização seja possível é necessário em primeiro lugar agrupar as imagens em grupos de imagens similares e posteriormente ordenar as fotografias para que seja possível manter a melhor imagem. Na versão atual da aplicação todo o processamento é feito localmente, mas estando em aberto a possibilidade de, numa versão futura, o processamento seja realizado por um servidor remoto.

Apesar de o processamento remoto potencialmente ter vantagens consideráveis na velocidade de análise das imagens, este acarreta também aspetos alguns negativos, de onde se pode destacar a dependência da ligação à rede e o tempo necessário para o envio de imagens para o servidor e a receção dos valores processados.

Durante o desenvolvimento foi possível concluir que a deteção de imagens semelhantes é um processo complexo. Se se pensar que dois humanos podem não concordar se duas imagens são ou não semelhantes, é fácil reconhecer que é complicado definir um critério 100% correto. Para tentar mitigar esta situação, para a versão iOS optou-se por se tentar chegar a um meio-termo, onde para algumas pessoas irão existir falsos negativos, para outras irão existir falsos positivos.

Definir se duas imagens são semelhantes é um processo complexo, mas a sua ordenação é conceptualmente mais complexo, visto que duas imagens quase iguais são, para a maioria das pessoas, semelhantes, por seu lado se se pedir para indicar qual a melhor garantidamente que não irá haver consenso, dado que grupos distintos de pessoas irão analisar a imagem com base em critérios diferentes, ordenando as mesmas imagens de forma distinta.

Não é possível afirmar que a análise de desempenho é fidedigna visto que é possível os utilizadores que realizaram os testes possam não ter entrado no espírito da aplicação, não entendendo o conceito de imagens similares, o que os levou a agrupar as imagens de forma mais lata, o que resulta mais falsos positivos.

5.2 Trabalho futuro

Os próximos passos no desenvolvimento da versão iOS da aplicação *photo|uniq* devem contemplar o melhoramento da interface gráfica, nomeadamente quando se ativa o modo de seleção, e a forma como a aplicação carrega as imagens para que a sua apresentação seja mais fluida.

Para o processamento de imagens existem três potenciais otimizações:

Paralelização dos processamentos. Esta abordagem deve ser aplicada se o objetivo for para que a aplicação se mantenha totalmente local;

A utilização de um servidor para a realização dos processamentos. Com esta abordagem a aplicação fica dependente de internet;

Uma solução hibrida. Neste caso, se não existir serviço de internet a aplicação processa as imagens localmente, mas se a internet estiver disponível os processamentos poderão ser realizados por um servidor.

Bibliografia

[1] Ali ASLAN. *Ducl Duplicate Photos Cleaner*. 2020. URL: https://apps.apple.com/us/app/ducl-duplicate-photos-cleaner/id1462955662 (acedido em 31 de jan. de 2020) (ver pp. 6, 7).

[2] Apple Inc. *Analyzing Image Similarity with Feature Print*. 2019. URL: https://developer.apple.com/documentation/vision/analyzing_image_similarity_with_feature_print (acedido em 10 de jan. de 2020) (ver p. 15).

[3] Apple Inc. *AssetsLibrary*. 2019. URL: https://developer.apple.com/documentation/assetslibrary (acedido em 10 de jan. de 2020) (ver p. 14).

[4] Apple Inc. *Creating an Image Classifier Model*. 2019. URL: https://developer.apple.com/documentation/createml/creating_an_image_classifier_model (acedido em 10 de jan. de 2020) (ver p. 15).

[5] Apple Inc. *Histogram*. 2019. URL: https://developer.apple.com/documentation/accelerate/histogram (acedido em 10 de jan. de 2020) (ver p. 15).

[6] Apple Inc. *Human interface guidelines*. 2019. URL: https://developer.apple.com/design/human-interface-guidelines/ (acedido em 26 de nov. de 2019) (ver pp. 31, 32, 35).

[7] Apple Inc. *Importing Objective-C into Swift*. 2019. URL: https://developer.apple.com/documentation/swift/imported_c_and_objective-c_apis/importing_objective-c_into_swift (acedido em 10 de jan. de 2020) (ver p. 14).

[8] Apple Inc. *sort()*. 2019. URL: https://developer.apple.com/documentation/swift/array/1688499-sort (acedido em 10 de jan. de 2021) (ver p. 60).

[9] Apple Inc. *Swift.org and Open Source*. 2019. URL: https://swift.org/about/#swiftorg-and-open-source (acedido em 26 de nov. de 2019) (ver p. 28).

[10] Apple Inc. *UI Design Do's and Don'ts*. 2019. URL: https://developer.apple.com/design/tips/ (acedido em 11 de dez. de 2019) (ver p. 34).

[11] Apple Inc. *Vision*. 2019. URL: https://developer.apple.com/documentation/vision (acedido em 10 de jan. de 2020) (ver p. 15).

[12] Apple Inc. *VNHomographicImageRegistrationRequest*. 2019. URL: https://developer.apple.com/documentation/vision/vnhomographicimageregistrationrequest#overview (acedido em 10 de jan. de 2020) (ver p. 15).

[13] H. Bay, T. Tuytelaars e L. Van Gool. "SURF: Speeded Up Robust Features". Em: *Computer Vision – ECCV 2006*. Ed. por A. Leonardis, H. Bischof e A. Pinz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 404–417. ISBN: 978-3-540-33833-8 (ver p. 16).

[14] BPMobile. *Smart Cleaner - Clean Storage*. 2020. URL: https//apps.apple.com/us/app/smart-cleaner-clean-storage/id1194582243 (acedido em 17 de fev. de 2020) (ver p. 8).

[15] Cotin. *Image Similarity Comparison*. 2018. URL: https://cotin.tech/Algorithm/ImageSimilarityComparison (acedido em 10 de jan. de 2021) (ver p. 19).

[16] Elena. *Swift vs. Objective-C: What language to Choose in 2021?* 2021. URL: https://gbksoft.com/blog/swift-vs-objective-c/ (acedido em 2 de jun. de 2021) (ver p. 15).

[17] "Fast Approximate Nearest Neighbors With Automatic Algorithm Configuration". Em: *Proceedings of the Fourth International Conference on Computer Vision Theory and Applications* (2009). DOI: 10.5220/0001787803310340 (ver p. 18).

[18] *Feature Matching*. URL: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_matcher/py_matcher.html (ver p. 18).

[19] M. Felismino. "Conceção e Desenvolvimento de uma Aplicação Android para Eliminação Assistida de Fotografias Repetidas". Tese de mestrado. Caparica: Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, 2019 (ver p. 22).

[20] Foraker Labs. *Aesthetic integrity*. 2019. URL: http://www.usabilityfirst.com/glossary/aesthetic-integrity/ (acedido em 26 de nov. de 2019) (ver p. 31).

[21] Foraker Labs. *Feedback*. 2019. URL: http://www.usabilityfirst.com/glossary/feedback/ (acedido em 26 de nov. de 2019) (ver p. 31).

[22] Foraker Labs. *Metaphor*. 2019. URL: http://www.usabilityfirst.com/glossary/metaphor/ (acedido em 26 de nov. de 2019) (ver p. 31).

[23] D. Gada. *Feature Matching using Brute Force Matcher*. 2019. URL: https://thedevnotebook.wordpress.com/2019/03/13/feature-matching-using-brute-force-matcher/ (ver p. 18).

[24] I. Hassan. *Swift vs Objective-C in 2019*. 2019. URL: https://medium.com/swiftify/swift-vs-objective-c-comparison-32aba9dad4e3 (acedido em 11 de dez. de 2019) (ver pp. 15, 28).

[25] Hyundong. *De-dupe*. 2019. URL: https://apps.apple.com/us/app/de-dupe/id1457634880 (acedido em 10 de jan. de 2020) (ver pp. 6, 7).

[26] A. Jakubovic e J. Velagic. "Image Feature Matching and Object Detection Using Brute-Force Matchers". Em: set. de 2018, pp. 83–86. DOI: 10.23919/ELMAR.2018.8534641 (ver p. 18).

[27] N. Krawetz. *Kind of Like That*. 2013. URL: http://www.hackerfactor.com/blog/index.php?/archives/432-Looks-Like-It.html (acedido em 10 de jan. de 2021) (ver p. 19).

[28] N. Krawetz. *Looks Like It*. 2011. URL: http://www.hackerfactor.com/blog/index.php?/archives/432-Looks-Like-It.html (acedido em 10 de jan. de 2021) (ver p. 19).

[29] Z. L.Cormack H.R.Sheikh e A. Bovik. *Live image quality assessment database release 2*. 2005. URL: http://live.ece.utexas.edu/research/quality (acedido em 1 de jan. de 2021) (ver p. 49).

[30] M. Liberatore. *COMPSCI 590K: Advanced Digital Forenics Systems | Spring 2020*. 2020. URL: https://people.cs.umass.edu/~liberato/courses/2020-spring-compsci590k/lectures/09-perceptual-hashing/ (acedido em 10 de jan. de 2021) (ver p. 19).

[31] J. M. Lourenço. *The NOVAthesis LaTeX Template User's Manual*. NOVA University Lisbon. 2021. URL: https://github.com/joaomlourenco/novathesis/raw/master/template.pdf (ver p. ii).

[32] D. G. Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". Em: *International Journal of Computer Vision* 60.2 (2004), pp. 91–110. DOI: 10.1023/b:visi.0000029664.99615.94 (ver p. 15).

[33] Macpaw. *Gemini Photos*. 2020. URL: https://macpaw.com/gemini-photos (acedido em 17 de fev. de 2020) (ver pp. 7, 8).

[34] F. Mano. "A Computing and Storage Server Infrastructure for a Mobile Application". Tese de mestrado. Caparica: Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, 2019 (ver p. 22).

[35] A. Mittal, A. K. Moorthy e A. C. Bovik. "No-Reference Image Quality Assessment in the Spatial Domain". Em: *IEEE Transactions on Image Processing* 21.12 (2012), pp. 4695–4708. DOI: 10.1109/tip.2012.2214050 (ver pp. 21, 49).

[36] Nidhi e K. A. Patel. "An Efficient and Scalable Density-based Clustering Algorithm for Normalize Data". Em: *Procedia Computer Science* 92 (2016), pp. 136–141. DOI: 10.1016/j.procs.2016.07.336 (ver p. 58).

[37] NSHipster. *Nshipster/dbscan: Density-based spatial clustering of applications with noise*. 2019. URL: https://github.com/NSHipster/DBSCAN (acedido em 10 de jan. de 2020) (ver p. 46).

[38] OpenCV. *OpenCV*. 2019. URL: https://opencv.org/about/ (acedido em 10 de jan. de 2020) (ver p. 14).

[39] opencv. *Basic concepts of the homography explained with code*. 2019. URL: https://docs.opencv.org/master/d9/dab/tutorial_homography.html (acedido em 10 de jan. de 2021) (ver p. 23).

[40] opencv. *cv::quality::QualityBRISQUE Class Reference*. URL: https://docs.opencv.org/master/d8/d99/classcv_1_1quality_1_1QualityBRISQUE.html (acedido em 10 de jan. de 2021) (ver p. 49).

[41] OpenCV dev team. *OpenCV iOS Hello*. 2019. URL: https://docs.opencv.org/2.4/doc/tutorials/ios/hello/hello.html#opencvioshelloworld (acedido em 10 de jan. de 2020) (ver p. 14).

[42] N. Ponomarenko et al. "TID2008 - A Database for Evaluation of Full-Reference Visual Quality Assessment Metrics". Em: *Advances of Modern Radioelectronics* 10 (jan. de 2009), pp. 30–45 (ver p. 49).

[43] A. Ram et al. "A Density Based Algorithm for Discovering Density Varied Clusters in Large Spatial Databases". Em: *International Journal of Computer Applications* 3.6 (2010), pp. 1–4. DOI: 10.5120/739-1038 (ver p. 20).

[44] E. Rublee et al. "ORB: An efficient alternative to SIFT or SURF". Em: *2011 International Conference on Computer Vision*. 2011, pp. 2564–2571. DOI: 10.1109/ICCV.2011.6126544 (ver p. 17).

[45] Trend Micro, Incorporated. *CleanerOne*. 2019. URL: https://apps.apple.com/sa/app/cleaner-one/id1156773866 (acedido em 3 de fev. de 2020) (ver pp. 6–8).

[46] D. Tyagi. *Introduction to SURF (Speeded-Up Robust Features)*. 2019. URL: https://medium.com/data-breach/introduction-to-surf-speeded-up-robust-features-c7396d6e7c4e (acedido em 10 de jan. de 2021) (ver p. 16).

[47] Wikipedia. *OpenCV*. 2019. URL: https://en.wikipedia.org/wiki/OpenCV (acedido em 10 de jan. de 2020) (ver p. 14).

[48] W. Zhe. *Clean Doctor - Clean Storage+*. 2019. URL: https://apps.apple.com/pt/app/clean-doctor-clean-storage/id855008026 (acedido em 10 de jan. de 2020) (ver pp. 6, 7).

A

Data Sets para a geração de grupos

A.1 DataSet - 1














A.2 DataSet - 2

















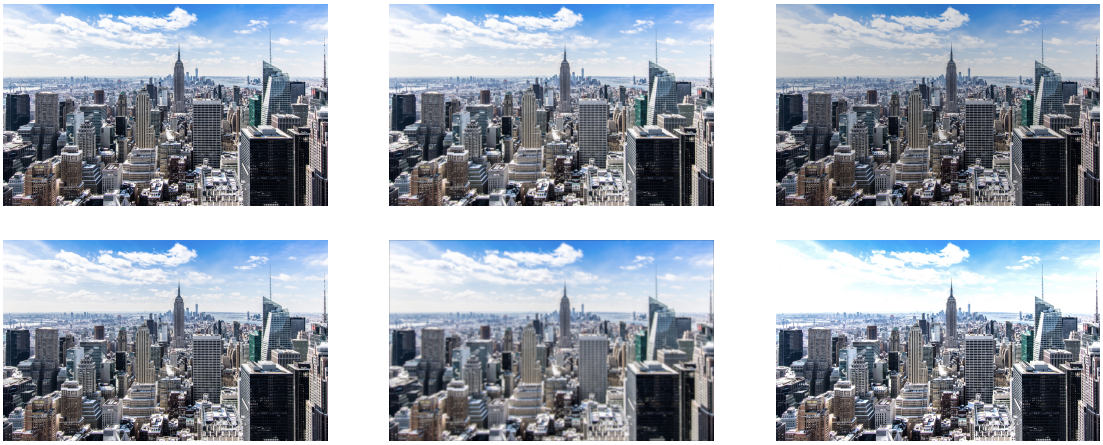
B

Data Sets para a ordenação

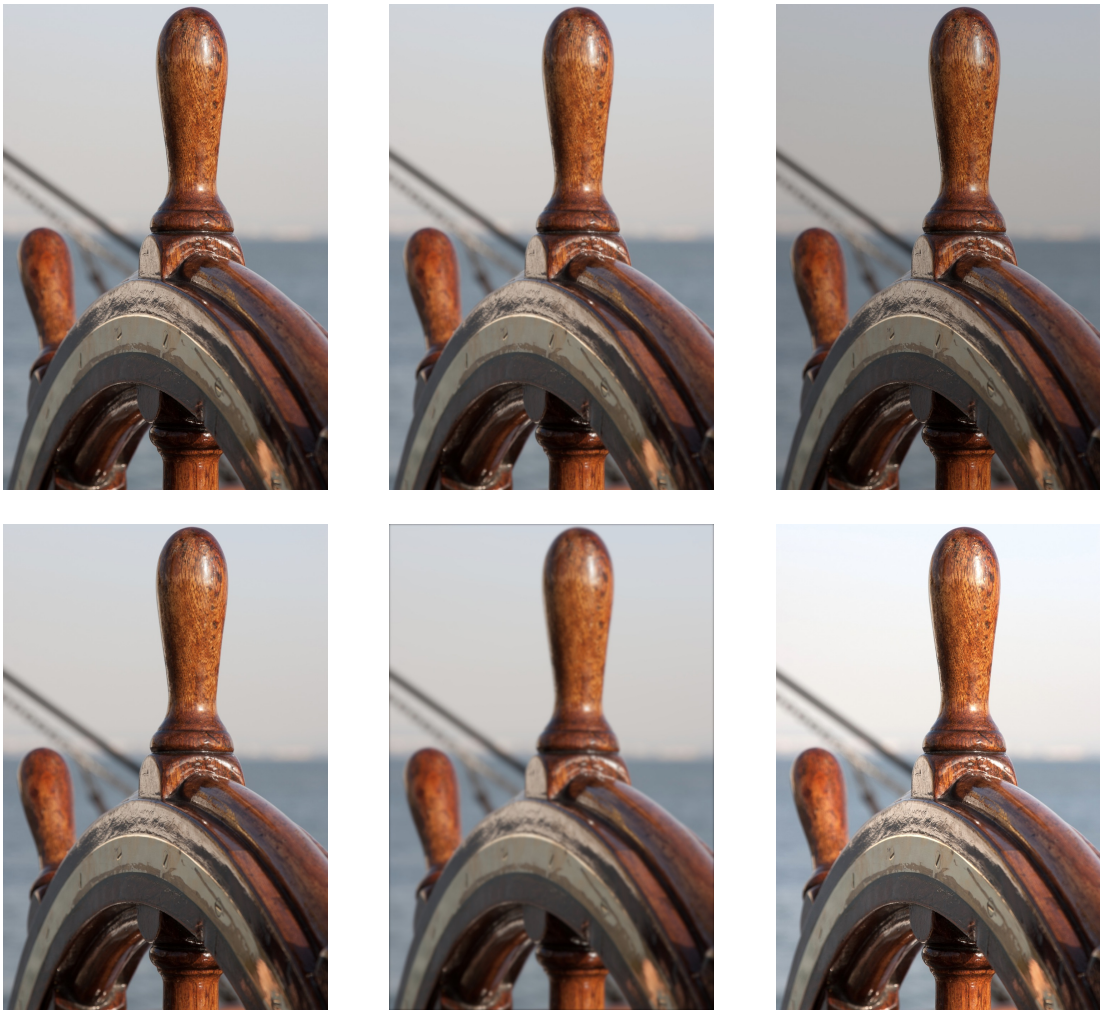
B.1 Data Sets sintéticos

B.1.1 Data Set 1

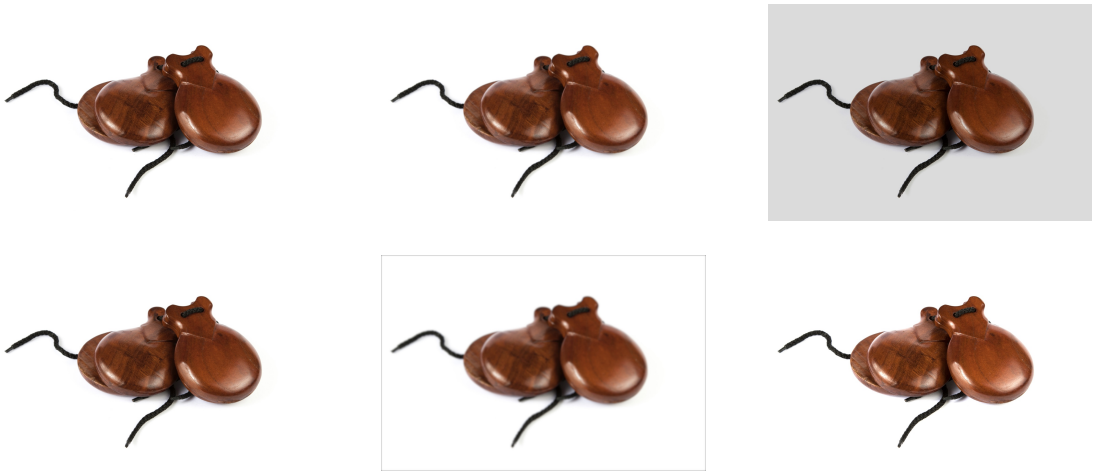
B.1.2 Data Set 2



B.1.3 Data Set 3



B.1.4 Data Set 4



B.2 Data Sets funcionais

B.2.1 Data Set 1

B.2.2 Data Set 2



B.2.3 Data Set 3





B.2.4 Data Set 4