

Sandra Luna (SLU)

Ana Barata (ABT)

Cláudia Lima (CAL)

Cliente: Carlos Ferreira (CGF)

Red text on a white background

Description automatically generated

**1DM, Equipa nº 01, No name is still the best name**

Beatriz Coelho – 1240536

Francesco Castioni – 1240539

Sofia Monteiro – 1241156

Nelson Moreira – 1222576

Reconstrução e Identificação de Imagens Utilizando Eigenfaces

Relatório do Projeto de LAPR1

**Janeiro de 2025**

**ÍNDICE**

[Índice de Figuras iii](#_Toc187486290)

[Índice de Tabelas iii](#_Toc187486291)

[1. Introdução iii](#_Toc187486292)

[2. Metodologia de Trabalho iii](#_Toc187486293)

[**2.1 Scrum no desenvolvimento do Projeto** iv](#_Toc187486294)

[**2.2 Planeamento e distribuição de tarefas** v](#_Toc187486295)

[**2.3 Reflexão crítica sobre a dinâmica da equipa** v](#_Toc187486296)

[3. Eigenfaces vii](#_Toc187486297)

[3.1. Decomposição Própria viii](#_Toc187486298)

[3.2. Eigenfaces ix](#_Toc187486299)

[4. Desenvolvimento e Implementação da Aplicação xii](#_Toc187486300)

[4.1. Requisitos xiv](#_Toc187486301)

[4.2. IDE xiv](#_Toc187486302)

[4.3. Repositório e controlo de versões xiv](#_Toc187486303)

[4.4. Planificação xiv](#_Toc187486304)

[4.5. Execução do programa xv](#_Toc187486305)

[4.6. Ficheiros de Entrada e Saída xvi](#_Toc187486306)

[4.7. Leitura do CSV xvii](#_Toc187486307)

[4.8. Escrita do CSV xviii](#_Toc187486308)

[4.9. Métodos desenvolvidos xviii](#_Toc187486309)

[5. Resultados xxi](#_Toc187486310)

[5.1. Apresentação dos resultados xxii](#_Toc187486311)

[5.2 Discussão dos resultados xxii](#_Toc187486312)

[6. Conclusão xxii](#_Toc187486313)

[Referências xxii](#_Toc187486314)

# Índice de Figuras

*Figura 1: Imagem 38 da base de imagens.*

*Figura 2: Transformação da imagem numa matriz 64 x 64.*

*Figura 3: Transformação de uma matriz N×N pixels numa representação vetorial N2×1 (Trivedi, 2009).*

*Figura 4:* *Fórmula para obtenção da Meanface (Trivedi, 2009).*

*Figura 5:*

*Figura 6:*

1. **Introdução**

No âmbito da unidade curricular Laboratório / Projecto I, fomos desafiados a desenvolver uma aplicação informática para a instituição European Intelligence Agency (EIA) que permita representar e identificar indivíduos utilizando eigenfaces (vetores próprios) (Turk & Pentland, 1991) obtidos de um conjunto de fotografias de faces humanas.

Como nos foi transmitido, a EIA possui uma base de imagens de grande dimensão, que não pretende partilhar, e quer encontrar uma representação compacta para esta base de imagens que possa ser distribuída por todas as instituições europeias afiliadas para identificar indivíduos que circulam em cada um dos países onde estas atuam. Para atingir os seus objetivos a EIA pretende que a base do sistema a desenvolver seja a decomposição de uma imagem/matriz em valores e vetores próprios, em particular as eigenfaces. A EIA pretende também um estudo de sensibilidade ao número de eigenfaces (de valores e vetores próprios) a utilizar na decomposição de cada imagem.

Considerando que uma imagem na escala de cinzento pode ser representada através de uma matriz, para facilitar o desenvolvimento e avaliação do projeto, a EIA define três tarefas/funcionalidades a incluir na aplicação:

1. Decomposição Própria (Larson & Falvo, 2009) de uma matriz simétrica;

2. Reconstrução de cada imagem (imagem em escala de cinzento (0-255)) utilizando as eigenfaces (Turk & Pentland, 1991)mais relevantes;

3. Dada uma imagem, identificar a imagem mais próxima/parecida que existe na base de imagens utilizando os pesos de um conjunto de eigenfaces.

Para a execução destas tarefas também é necessário implementar módulos para leitura e escrita de matrizes/imagens guardadas em pastas assim como módulos de cálculo de um conjunto de estatísticas e métricas.

Ao longo deste documento iremos referir sobre os métodos de trabalho utilizados, a dinâmica da equipa, o processo de pesquisa e de desenvolvimento da aplicação, os resultados. Iremos também fazer uma reflexão sobre o trabalho final, entre outros tópicos.

1. **Metodologia de Trabalho**

Esta secção apresenta a abordagem adotada para organizar o fluxo do projeto e gerir a equipa. Destacam-se as metodologias usadas, como *Scrum*, e a importância da colaboração, comunicação e distribuição de funções para assegurar eficiência e o progresso contínuo.

## **2.1 Scrum no desenvolvimento do Projeto**

Para contextualizar, a definição de *Scrum* pode ser dada como: uma leve framework que ajuda pessoas, equipas e organizações a gerar valor por meio de soluções adaptáveis ​​para problemas complexos. (Schwaber & Sutherland, 2020).

A metodologia Scrum é um *framework* ágil que promove a entrega contínua de valor em ciclos curtos e iterativos. Liderado pelo *Scrum Master*, o processo envolve:

1. O *Product Owner*, que prioriza o *backlog* de acordo com as necessidades do projeto.

2. O *Scrum Team*, que desenvolve valor a cada *Sprint*.

3. A revisão e adaptação dos resultados junto aos *stakeholders* para otimizar o próximo ciclo.

Esse modelo iterativo permite flexibilidade, melhoria contínua e rápida resposta a mudanças.

O Scrum tem uma estrutura intencionalmente incompleta, definindo apenas os elementos essenciais para implementar a sua teoria. Baseia-se na inteligência coletiva das pessoas que o aplicam, promovendo adaptações ao contexto. Em vez de fornecer instruções detalhadas, estabelece regras que orientam as interações e os relacionamentos entre os membros da equipa.

Vários processos, técnicas e métodos podem ser empregues dentro da estrutura. O Scrum envolve práticas existentes ou torna-as desnecessárias. O Scrum torna visível a eficácia relativa da gestão atual, ambiente e técnicas de trabalho, para que melhorias possam ser feitas.

Em suma, a abordagem Scrum consiste num método que organiza o trabalho em etapas para entregar melhoria contínua. Baseia-se num conjunto de regras, valores e práticas que proporcionam uma estrutura clara para trabalho em equipa. Este *framework* também promove a integração, garantindo que o resultado permaneça próximo da realidade e alinhado às necessidades do cliente. Este método propõe a divisão do trabalho em sprints, com reuniões para informar o que é que cada elemento fez, o que falta fazer e eventuais dificuldades encontradas.

A eleição de um Scrum Master é fundamental para o bom funcionamento do trabalho. Este gere a organização, atribuição de tarefas e o tempo do projeto, para que todos os membros consigam cumprir com os seus deveres. Após apresentar o Projeto Integrador à equipa, decidiu-se atribuir a função de Scrum Master ao Francesco Castioni e realizar o projeto em 3 Sprints, cada com duração de uma semana.

A metodologia Scrum foi aplicada ao longo dos 3 sprints deste projeto, sendo que em todos os *sprints* foram introduzidas tarefas num *board* no GitHub, de modo que depois da análise detalhada do enunciado, todos os elementos do grupo tivessem partes de cada uma das funcionalidades que foram implementadas. No âmbito desta metodologia foi também implementado uma política de reuniões semanais, sendo que, não foram feitas reuniões diárias (dailys) mas sim reuniões semanais no final de semana de modo a servir como retrospetiva e *review* do sprint.

## **2.2 Planeamento e distribuição de tarefas**

O primeiro sprint decorreu durante a semana de 9 a 15 de dezembro e teve como principais tarefas estudar a fundo o problema, bem como as bases teóricas por trás dele. O primeiro sprint também teve como objetivo codificar a primeira funcionalidade do problema. A equipa reuniu-se três vezes durante a semana, uma vez presencialmente nas instalações do ISEP e duas vezes online, através da plataforma *Microsoft Teams*®. Também utilizamos o *Github*® (para controlo de versões do código, para acompanhar as tarefas pendentes e para facilitar a organização) e o WhatsApp® (para comunicação). Além disso, a plataforma que utilizamos para o desenvolvimento do código foi *o IntelliJ IDEA*®. Durante a primeira semana foi organizado o canal da Equipe na plataforma Teams e criada uma subpasta chamada “Documentos Internos” (dentro da pasta Documentos) para melhor organização e facilidade de acesso por todos os membros dos documentos compartilhados criados para estudo a teoria por trás do problema, incluindo resumos de temas como Centralização de imagens ou o que a meanface. Foi criado um *Scrum Board* na plataforma GitHub, onde foram listados vários itens do *Backlog* relativos aos objetivos do projeto e às tarefas a serem realizadas para alcançá-los. Os encontros da semana foram registados na seção Diário de bordo e também foi criada uma seção para cada membro compartilhar suas reflexões sobre a semana (o que fizeram, quais as dificuldades que enfrentaram, o que farão a seguir, assiduidade).

Na primeira semana, começámos por desenvolver a funcionalidade de Decomposição Própria, que ficou parcialmente concluída. Esta funcionalidade inclui a impressão de matrizes, a transformação de uma matriz em valores e vetores próprios, a validação do valor de K inserido pelo utilizador para definir a quantidade de valores/vetores próprios a serem considerados, e o cálculo do Erro Absoluto Médio (EAM).

Durante este processo, cada membro da equipa esteve envolvido em todas as etapas do desenvolvimento das funções. Esta abordagem teve como objetivo garantir que todos os integrantes compreendam o funcionamento e possam explicar detalhadamente cada parte desta funcionalidade.

Durante a semana de 16 a 22 de dezembro, iniciamos o segundo sprint com uma reunião presencial para avaliar o progresso do *sprint* anterior e atribuir novas tarefas a cada membro da equipa. Foi necessário reformular o código desenvolvido, uma vez que inicialmente tínhamos utilizado *RealMatrix* da library *commom maths* em todas as *features* desenvolvidas. Além disso, cada membro ficou responsável por finalizar funções da segunda funcionalidade: reconstrução da matriz imagem.

Durante a pausa letiva, os membros desenvolveram testes unitários para as funcionalidades existentes.

Após a pausa letiva, na semana de 6 a 12 de janeiro, no terceiro sprint, o Nelson Moreira ficou encarregue de desenvolver a Funcionalidade 3, enquanto os outros membros ficaram encarregues da redação do relatório e elaboração dos slides. No final da semana, a equipa reuniu-se para testar a aplicação, corrigindo erros como *loops* infinitos no modo interativo, e para rever o relatório.

**2.3 Reflexão crítica sobre a dinâmica da equipa**

Como equipa, consideramos que a nossa dinâmica de desenvolvimento de trabalho teve um papel crucial na obtenção de bons resultados. Desenvolver um contrato justo com antecedência, implementar a metodologia *Scrum*, a utilização de ferramentas como o *Github*®, o *Office 365*®, o *Teams*®, a distribuição adequada de tarefas, reuniões regulares, com uma comunicação clara e que valoriza as diferentes opiniões, preservando o respeito entre os colegas foram fatores essenciais para o sucesso da equipa. Segue-se, abaixo, o ponto de vista de cada membro da equipa, começando pelo *Scrum Master*.

Francesco Castioni: Como *Scrum Master* fui eu que estive em contacto com o cliente e a transmitir os objetivos do deste ao resto da equipa, tal como os apontamentos que a professora Sandra Luna foi referindo nas reuniões com todos os *Scrum Masters*. Ao longo deste projeto, tomei a iniciativa de distribuir as tarefas que foram analisadas pela equipa de acordo com o enunciado, para facilitar que todos tenham trabalho ao longo das 3 de modo que todos tivesse objetivos fossem atingidos.

Relativamente a performance individual, sinto que foi uma experiência enriquecedora e que gostava de voltar a repetir, pois como foi a minha primeira vez como *Scrum Master* posso fazer muito melhor. Quanto ao que foi realizado em Java, tive sempre algumas dificuldades com o tema, sendo que levou a que demorasse algum tempo a conseguir realizar aquilo que me era atribuído, apesar disso tudo aquilo que me propus a fazer foi feito com sucesso. Deste modo, tanto como *Scrum Master* e como *developer* cumpri os objetivos que foram definidos.

Quanto à dinâmica da equipa, aplicamos o modelo que funcionou no primeiro projeto, em que existe uma divisão prévia das tarefas que são preciso ser feitas, reuniões regulares para discutir o estado do projeto e a procura da discussão constante de dúvidas ou preocupações de cada um dos elementos do grupo, todas estas ações levam a que consigamos funcionar da melhor maneira possível. Todos os membros cumpriram com aquilo que lhes foi atribuído. Não podia estar mais satisfeito com a forma como encaramos este desafio e o levamos a sua conclusão.

Beatriz Coelho: Durante o desenvolvimento deste projeto, enfrentei diversos desafios a nível individual, particularmente no que diz respeito ao nível de conhecimentos em Java e à compreensão de alguns conceitos de álgebra linear requeridos para a realização das tarefas. Apesar dessas dificuldades, consegui atingir os objetivos que me foram propostos, cumprindo as tarefas dentro dos prazos estabelecidos e demonstrando uma atitude proativa, especialmente na organização das tarefas.

Em termos de dinâmica de equipa, mantivemos o grupo que já tínhamos formado no módulo de *Skills*, uma decisão que considero ter sido positiva. Apesar de sermos diferentes em vários aspetos, conseguimos encontrar um equilíbrio eficaz enquanto equipa e complementarmo-nos. No módulo anterior, definimos uma metodologia de trabalho funcional: estabelecemos uma divisão clara de tarefas com deadlines específicas para cada membro e realizamos reuniões periódicas para acompanhar o progresso. Fora das reuniões, existe um diálogo semi-constante e uma discussão saudável, com troca de ideias e resolução de problemas em conjunto, quer por mensagens de texto, áudios e vídeos.

Considero que a forma como trabalhámos enquanto equipa foi um dos principais fatores que contribuíram para o sucesso do projeto. A comunicação aberta e a capacidade de ouvir diferentes perspetivas assim como o esforço individual de cada membro, permitiram-nos manter o foco nos objetivos e garantir que todas as tarefas fossem realizadas de forma eficiente.

Nelson Moreira: Para mim, como o membro com maior responsabilidade na parte do código, o maior desafio foi traduzir as exigências do enunciado para a implementação em código. Sinto que, com o que aprendi ao longo do desenvolvimento, poderia ter alcançado um resultado melhor. Caso tivesse a oportunidade de realizar o projeto de novo, tenho a certeza de que conseguiria obter um resultado mais satisfatório.

Relativamente à nossa equipa, ao longo de todo o processo, demonstrámos um elevado nível de responsabilidade e empatia uns pelos outros, algo que considero ser um dos aspetos mais importantes no trabalho em grupo. Um dos nossos pontos fortes foi a diversidade de conhecimentos que cada um de nós possui em várias áreas. Isso permitiu-nos trabalhar de forma dinâmica, com a divisão das tarefas a ser feita de acordo com os pontos fortes e especialidades de cada membro.

Apesar dos desafios que surgiram ao longo do desenvolvimento, acredito que não teria conseguido realizar um trabalho melhor em qualquer outro grupo. Senti que cada membro da equipa se dedicou ao máximo. Todos contribuíram de diversas formas para o sucesso do projeto e, sem a colaboração de um de nós, tenho a certeza de que não teríamos sido capazes de concluir o trabalho com êxito.

Sofia Monteiro: No meu ponto de vista, este projeto foi bastante desafiante. Senti algumas dificuldades a nível de conhecimentos de java e de álgebra, mas fui superando essas dificuldades depois de bastante pesquisa.

Fui cumprindo com as minhas tarefas, que me foram atribuídas pelo *Scrum Master*, e com os deadlines discutidos em equipa. Essa organização permitiu manter um ritmo consistente de trabalho, garantindo que o progresso fosse alcançado sem comprometer a qualidade do resultado final. Também estive presente em todas reuniões, quer nas presenciais quer nas online. Estas foram momentos cruciais para alinharmos expectativas, esclarecer dúvidas e ajustarmos o planeamento conforme necessário.

Acredito que o ambiente colaborativo melhorou significativamente a minha contribuição pessoal para o projeto. A equipa manteve uma comunicação aberta, e eu pude partilhar as minhas ideias e contribuir para as discussões confortavelmente. Isso fez-me sentir valorizada e motivada a participar ativamente. A divisão clara das tarefas e as reuniões regulares também me ajudaram a permanecer no caminho certo e garantir que meu trabalho estivesse alinhado com as metas do grupo. No geral, sinto que a dinâmica do grupo proporcionou uma oportunidade para crescimento pessoal e profissional.

1. **Eigenfaces**

O reconhecimento facial através de *eigenfaces* é uma das técnicas mais conhecidas e utilizadas no processamento de imagens e inteligência artificial. Esta técnica baseia-se na decomposição própria, que tem por base conceitos fundamentais da Álgebra Linear, nomeadamente valores e vetores próprios.

## 3.1. Decomposição Própria

A decomposição própria de uma matriz simétrica consiste no processo de decompor a matriz nos seus valores e vetores próprios.

As matrizes simétricas possuem características que tornam a decomposição mais simples, nomeadamente:

* Valores próprios reais;
* Vetores próprios ortogonais entre si, ou seja, perpendiculares ao espaço;
* Possibilidade de diagonalização, já que, de acordo com o Teorema Espectral, qualquer matriz simétrica com entradas reais pode ser diagonalizada através de uma matriz ortogonal, P.

Em termos mais simples, isto significa que a matriz original pode ser expressa como o produto de uma matriz ortogonal, P, uma matriz diagonal, D e a transposta da matriz ortogonal, PT. Tem como formula:

A =PDPT, sendo,

* P a matriz cujas colunas são os vetores próprios ortonormais de A.
* D a matriz diagonal que contém os valores próprios de A.
* PT a transposta da matriz ortogonal P.

Na prática, esta diagonalização facilita o estudo das propriedades da matriz, uma vez que todos os valores importantes ficam concentrados na diagonal principal da matriz D. Além disso, a matriz ortogonal P utilizada na decomposição é composta por vetores próprios ortonormais da matriz original A, garantindo que a transformação preserva distâncias e ângulos no espaço.

Valores próprios e Vetores próprios

Os valores próprios (λ) podem ser definidos como escalares associados a uma transformação linear que representam a matriz A, definindo a intensidade do efeito da transformação numa direção específica, por outras palavras, o quanto a transformação expande ou contrai os vetores ao longo de certas direções no espaço.

A equação para encontrar os valores próprios de uma matriz A é dada por:

**det (A- λI) = 0** (Larson & Falvo, 2009)

Onde:

* A é a matriz que representa a transformação linear.
* λ é o valor próprio.
* I é a matriz identidade, de mesma ordem que A.

Vetores próprios (ν) são vetores não nulos que, ao serem aplicadas transformações lineares (A) permanecem com a mesma direção, ou seja, apenas aumentam ou diminuem o tamanho, não sofrendo rotação ou distorção. Tal como explica a equação:

**A ν = λ ν** (Larson & Falvo, 2009)

Em que:

* A é a matriz que representa a transformação linear.
* λ é o valor próprio.
* **ν** é a vetor próprio não nulo.

A compreensão destes conceitos é essencial no contexto do reconhecimento facial, pois são os valores e vetores próprios que permitem reduzir a quantidade de dados necessários para representar uma imagem. Isto é possível ao concentrar-se nas direções mais relevantes dos dados, ou seja, nas características mais significativas da imagem, o que torna o processo de reconhecimento mais eficiente e preciso.

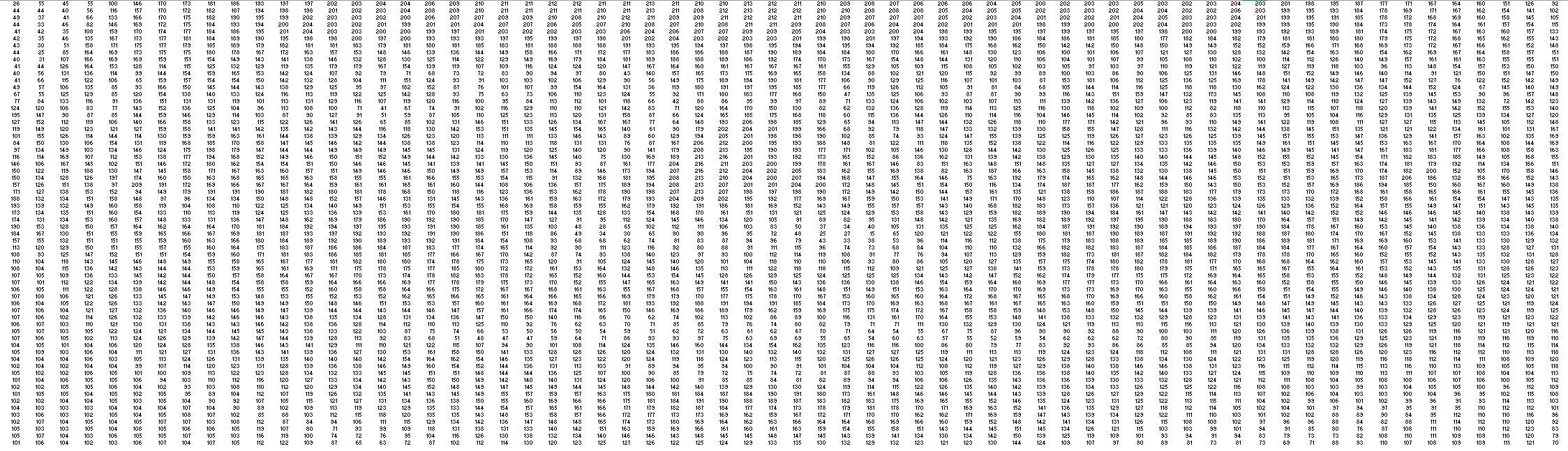
## 3.2. Eigenfaces

As *eigenfaces* são obtidas por meio da decomposição em valores e vetores próprios, resultante da transformação de uma imagem em representações matemáticas que facilitam sua análise. De acordo com *Kshirsagar et al*., “as eigenfaces são as componentes principais de uma distribuição de faces ou, de forma equivalente, os vetores próprios da matriz de covariância do conjunto de imagens de faces, em que uma imagem com N x N pixels é considerada um ponto (ou vetor) num espaço N²-dimensional”. (Kshirsagar et al., 2011)

O principal objetivo das *eigenfaces* é transformar uma imagem digital numa representação numérica baseada em escalas de cinza, onde cada pixel assume um valor dentro do intervalo [0, 255]. Essa conversão permite que cada pixel da imagem seja tratado como uma variável num vetor de alta dimensão, facilitando o processamento e o reconhecimento de padrões faciais.

Por exemplo, a Figura 1 mostra a imagem 38 da base de dados disponibilizada no Moodle, que contém imagens com resolução de 64 x 64 pixels. Cada pixel da imagem é convertido num valor correspondente à sua intensidade em escala de cinza. Essa imagem pode ser representada como uma matriz de 4096 elementos (64 x 64). A Figura 2 ilustra a transformação da imagem 38 nessa matriz, onde cada posição contém um valor entre 0 e 255, representando a intensidade do pixel.

Após a decomposição em valores próprios, a matriz é reduzida a um conjunto de componentes principais, conhecidos como *eigenfaces*, que representam as características mais relevantes da imagem original. Esta técnica permite reduzir a dimensão das imagens e ao mesmo tempo preservar as informações essenciais para o reconhecimento facial.

**Figura 1:** Imagem 38 da base de imagens. **Figura 2:** Transformação da imagem numa matriz 64 x 64.

“Muito do trabalho de reconhecimento facial é focado em detetar características individuais, como o nariz, olhos, boca e cabeça, bem como a relação entre estas características” (Turk & Pentland, 1991). O objetivo é que, ao introduzir uma nova imagem, seja possível identificar e avaliar as suas características em relação ao conjunto já existente.

Para identificar as características gerais presentes nas faces, cada imagem é transformada num vetor, e calcula-se a face média (MeanFace), que representa a média de todas as imagens da base de dados. Esse processo consiste em somar os vetores de todas as imagens base e dividir o resultado pelo número total de imagens. A face média reflete as características comuns da base de dados, funcionando como um ponto de referência para a análise das variações individuais de cada imagem em relação a essa média.

A Figura 3 ilustra o processo de transformação de uma imagem Ii numa matriz N x N, que é depois concatenada numa representação vetorial Com dimensão N2 x 1. Esse vetor corresponde à representação linear da imagem, facilitando o cálculo da *MeanFace* (através da fórmula ilustrada pela Figura 4) e a consequente comparação entre imagens.

A close-up of a diagram

Description automatically generated

Figura 3: Transformação de uma matriz N×N pixels numa representação vetorial N2×1 (Trivedi, 2009).

Figura 4: Fórmula para obtenção da **Meanface** (Trivedi, 2009).

O processo de *eigenfaces* consiste em identificar características gerais presentes em um conjunto de imagens, reduzindo o esforço computacional necessário para analisar cada imagem individualmente.

Essa técnica exige uma base de imagens inicial, que permite identificar padrões comuns em rostos humanos. Como refere (Trivedi, 2009) esse deve ser o primeiro passo do processo de criação de *eigenfaces*.

Após o cálculo da face média ( e a centralização de imagens através de é possível obter a matriz de covariancia , através da fórmula , aplicada a todas as imagens presentes na base de dados. A matriz de covariancia é normalmente definida por em que o A (Trivedi, 2009).

Embora esta fórmula tradicional seja válida, ao programar em Java, existe uma forma mais eficiente de calcular a matriz de covariância. Em vez de calcular AAT, que resulta numa matriz com dimensões equivalentes ao número de píxeis das imagens, utilizamos C=ATA. Com esta abordagem, o cálculo é realizado apenas com base no número de imagens da base de dados, em vez do número de pixéis, reduzindo significativamente o esforço computacional.

Assim, a matriz de covariancia tem como objetivo é eliminar as características gerais, mantendo apenas as caracteristicas que são diferentes em cada uma das imagens, subtraindo para isso a MeanFace de cada imagem ao vetor correspondente.

Após a criação das *eigenfaces*, qualquer imagem nova é projetada nesse espaço de características gerais, permitindo identificar suas particularidades em relação à base de dados.

Essa estratégia otimiza o processo de reconhecimento facial, pois cria um subespaço que contém as características comuns a todas as imagens da base, facilitando a comparação com novas imagens introduzidas. Isso evita a necessidade de analisar cada rosto individualmente, reduzindo o processamento necessário.

Após calcular os pesos associados às imagens, é possível determinar o número de K vetores próprios a serem utilizados na reconstrução da imagem. Esse valor define o nível de detalhe da imagem reconstruída. Por exemplo, se uma imagem for representada por 40 vetores próprios, o utilizador pode escolher qualquer valor de K entre 1 e 40. Quanto maior o K, mais próxima a reconstrução será da imagem original.

A precisão da reconstrução varia com o número de imagens na base de dados e com o valor de K selecionado. As características gerais da base são projetadas num subespaço comum, e qualquer nova imagem é projetada nesse mesmo espaço, sendo comparada com todas as imagens da base. Cada imagem tem um peso associado, e a nova imagem é avaliada com base nesses pesos. A identificação facial é feita através da distância euclidiana, comparando os pesos da nova imagem com os pesos existentes, identificando a imagem mais próxima na base.

### 3.2.1. Alternativas às Eigenfaces: Fisherfaces

Durante a nossa pesquisa, encontramos outros algoritmos de redução dimensional que, assim como as *eigenfaces*, permitem otimizar o reconhecimento facial. Um exemplo são as *fisherfaces*, que se diferenciam por focar nas variações específicas de cada indivíduo, como pose, iluminação e expressão facial.

De acordo com (David Bissi, 2018), as *fisherfaces* destacam essas variações, aumentando a separabilidade entre diferentes indivíduos. Enquanto as *eigenfaces* identificam apenas características gerais, as fisherfaces são mais robustas a variações nas condições das imagens.

Segundo o mesmo autor, as *eigenfaces* apresentam uma taxa de acerto de 69,09%, mas com maior probabilidade de falsos positivos. Já as *fisherfaces* atingem uma taxa de acerto de 71,27%, sem apresentar falsos positivos.

Na opinião do grupo, o uso de *fisherfaces* poderia ser uma melhoria no algoritmo de reconhecimento facial utilizado neste projeto, pois traria maior precisão e confiabilidade ao processo de identificação.

1. **Desenvolvimento e Implementação da Aplicação**

Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente

Figura 5:

## 4.1. Requisitos

Considerando o enunciado que nos foi fornecido e as indicações do cliente, o projeto a desenvolver no corrente ano letivo é uma aplicação informática para a instituição European Intelligence Agency (EIA) que permita representar e identificar indivíduos utilizando eigenfaces (vetores próprios) (Turk & Pentland, 1991) obtidos de um conjunto de imagens/fotografias de faces humanas. A EIA possui uma base de imagens de grande dimensão, que não pretende partilhar, e quer encontrar uma representação compacta para esta base de imagens que possa ser distribuída por todas as instituições europeias afiliadas para identificar indivíduos que circulam em cada um dos países onde estas atuam. Para atingir os seus objetivos a EIA pretende que a base do sistema a desenvolver seja a decomposição de uma imagem/matriz em valores e vetores próprios, em particular as eigenfaces. A EIA pretende também um estudo de sensibilidade ao número de eigenfaces (de valores e vetores próprios) a utilizar na decomposição de cada imagem.

Considerando que uma imagem na escala de cinzento pode ser representada através de uma matriz, para facilitar o desenvolvimento e avaliação do projeto, a EIA define três tarefas/funcionalidades a incluir na aplicação:

1. Decomposição Própria (Larson, 2012; Ghaoui L., 2024) de uma matriz simétrica;

2. Reconstrução de cada imagem (imagem em escala de cinzento (0-255)) utilizando as eigenfaces (Turk & Pentland, 1991) mais relevantes;

3. Dada uma imagem, identificar a imagem mais próxima/parecida que existe na base de imagens utilizando os pesos de um conjunto de eigenfaces.

Para a execução destas tarefas e também necessário implementar módulos para leitura e escrita de matrizes/imagens guardadas em pastas e módulos de cálculo de um conjunto de estatísticas e métricas.

## 4.2. IDE

O IDE utilizado como já foi referido foi o *IntelliJ IDEA*®, este software permite que, o código seja desenvolvido de forma fácil, partilhado utilizando o terminal e a linguagem git, e de modo geral para a linguagem de programação Java é aquele que faz mais sentido utilizar. Além disso foi aquele que foi utilizado durante todo o semestre, sendo que, é aquele com que o grupo está mais familiarizado.

## 4.3. Repositório e controlo de versões

Com facilidade em utilizar o git, sendo que já tinha sido usado no primeiro projeto para a cadeira de LAPR1 e com uma formação suplementar por parte no NEI-ISEP só faria sentido utilizá-lo de novo. Assim sendo, e sempre com o *IntelliJ IDEA*® em mente e utilizado o GitHub não só como controlo de versões, mas também como ScrumBoard cada commit feito, para um total de 200 commits, estes permitem a que todos os que estão envolvidos no projeto consigam ver aquilo que foi mudado e a mensagem que vem com as mudanças isto facilita a que caso seja implementado algum erro, ou seja preciso de voltar a algo que foi anteriormente, haja essa possibilidade.

Assim, o controlo de versões e a linguagem git são essenciais para qualquer trabalho de desenvolvimento de software que inclua vários membros.

## 4.4. Planificação

Considerando o número significativo de métodos necessários para implementar as três funcionalidades, elaboramos um diagrama para ilustrar, de maneira geral, os principais constituintes da aplicação:

Uma imagem com texto, captura de ecrã, design

Descrição gerada automaticamente

Figura 6-

## 4.5. Execução do programa

A execução da aplicação pode ser realizada a partir do terminal, utilizando um ficheiro no formato .jar. Inicialmente, o programa apresenta um menu para o utilizador escolher o modo de execução:

switch (escolha) {  
 case 0:  
 *ficheiroInterativo*(); // Modo interativo já implementado.  
 break;  
 case 1:  
 System.*out*.println("Digite os parâmetros para o modo não interativo no formato:");  
 System.*out*.println("-f X -k Y -i Z -d W nome\_ficheiro\_saida.txt");  
 sc.nextLine(); // Consumir a nova linha após o número.  
 String input = sc.nextLine().trim();  
 *executeNonInteractiveMode*(input.split(" ")); // Chama o modo não interativo.  
 break;  
 case 2:  
 System.*out*.println("Saindo...");  
 break;  
 default:  
 System.*out*.println("Escolha inválida. Tente novamente.");  
}

Caso o utilizador opte pelo modo interativo (escolha 0), o programa entra num segundo *switch*, que resulta num menu permitindo ao utilizador a escolha entre as três funcionalidades. A cada seleção, o utilizador fornece os parâmetros necessários, como o caminho para os ficheiros ou o número de *eigenfaces* a utilizar.

Caso o utilizador opte pelo modo não interativo, os parâmetros devem ser fornecidos diretamente como argumentos na linha de comandos e o programa processa as funcionalidades automaticamente, sem a necessidade de interação constante.

## 4.6. Organização do projeto

Uma imagem com texto, captura de ecrã, software, Software de multimédia

Descrição gerada automaticamente

Figura:

A Figura mostra a organização

De forma a facilitar a organização e o acesso aos ficheiros e tornar por isso o programa mais eficiente, foram utilizadas extensões e criados diretórios específicos para cada tipo de dados.

### 4.6.1. Extensões de ficheiros

O programa utiliza duas extensões principais para os ficheiros:

* **.csv**: Utilizada para armazenar matrizes de dados.
* **.jpg**: Utilizada para armazenar imagens reconstruídas e autovetores em formato gráfico.

Estão definidas no código da seguinte forma:

final static String *CSV\_FILE\_EXTENSION* = ".csv";  
final static String *JPG\_FILE\_EXTENSION* = ".jpg";

### 4.6.2. Diretórios Utilizados

* “CSVdatabase”: Contém ficheiros CSV que representam matrizes de imagens originais utilizadas no reconhecimento facial.
* “ImagensReconstruídas”: Armazena as imagens reconstruídas a partir das matrizes.
* “Identificacao”: Guarda a imagem identificada como a mais semelhante à imagem fornecida para comparação.
* “Eigenfaces”: Contém os vetores próprios calculados durante o processo.

final static String *IMAGE\_PATH* = "app/CSVdatabase";  
final static String *RECONSTRUCTEDIMAGES\_OUTPUT\_PATH* = "app/ImagensReconstruídas";  
final static String *IDENTIFIED\_IMAGE\_OUTPUT\_PATH* = "app/Identificacao";  
final static String *EIGENFACES\_OUTPUT\_PATH* = "app/Eigenfaces";

### 4.6.3. Nomes de Ficheiros de Entrada

Os ficheiros de entrada devem estar no formato CSV e são lidos a partir do diretório CSVdatabase. O programa filtra apenas os ficheiros com a extensão ".csv" ao listar os ficheiros de entrada, conforme o código abaixo:

files = folder.listFiles((dir, name) -> name.endsWith(".csv"));

### 4.6.4. Nomes de Ficheiros de Saída

Os ficheiros de saída são gerados automaticamente com base no nome dos ficheiros de entrada. Para cada imagem reconstruída ou vetor próprio calculado, o programa cria ficheiros de saída nos formatos CSV e JPG. Estes ficheiros são posteriormente armazenados nos diretórios correspondentes, pelo processo explicado em 4.8.1..

*writeCSV*(reconstructedImageMatrix, fileNameNoExtension, *RECONSTRUCTEDIMAGES\_OUTPUT\_PATH*);  
*writeCSV*(reconstructedImageForJPGMatrix, fileNameNoExtension + "\_transformada", *RECONSTRUCTEDIMAGES\_OUTPUT\_PATH*);  
*writeCSV*(eigenFaceMatrix, fileNameNoExtension, *EIGENFACES\_OUTPUT\_PATH*);  
*writeCSV*(eigenFaceImageVector, fileNameNoExtension + "\_transformada", *EIGENFACES\_OUTPUT\_PATH*);  
*writeArrayAsImage*(reconstructedImageForJPGMatrix, *RECONSTRUCTEDIMAGES\_OUTPUT\_PATH* + "/" + fileNameNoExtension + *JPG\_FILE\_EXTENSION*);  
*writeArrayAsImage*(eigenFaceImageMatrix, *EIGENFACES\_OUTPUT\_PATH* + "/" + fileNameNoExtension + *JPG\_FILE\_EXTENSION*);

## 4.7. Leitura do CSV

Os ficheiros CSV (*Comma-Separated Values*) são amplamente utilizados para armazenar e organizar grandes volumes de dados de forma simples e estruturada. Tal como o nome indica, estes ficheiros são delimitados por separadores, como vírgula (","), o que facilita a manipulação dos seus dados de forma manual ou automática.

Na aplicação desenvolvida, utilizámos ficheiros CSV para armazenar os valores dos pixels das imagens. Em que cada linha do ficheiro representa uma linha de pixels, sendo os valores separados pelo delimitador ",". Para importar estes ficheiros para o programa em Java, foi desenvolvido o método readCSV, que permite a leitura dos dados contidos no ficheiro e a sua conversão para uma matriz bidimensional (double[][]), permitindo a sua utilização nos métodos subsequentes.

A primeira tarefa do método é verificar se o ficheiro existe no diretório indicado. Se o ficheiro não for encontrado, é lançada uma exceção *FileNotFoundException*. Caso o ficheiro seja encontrado, o método avança, para a contagem do número de linhas:

while (fileReader.hasNextLine()) {  
 fileReader.nextLine();  
 rowSize++;  
}  
fileReader.close();

De seguida lê a primeira linha, remove espaços extra e, com o separador “,” divide a linha lida num array de strings. Posto isto, determina o número de colunas com base no número de elementos da linha e cria a matriz bidimensional com o número de linhas (rowSize) e colunas (columnSize). Caso o ficheiro esteja vazio é impressa uma mensagem de erro.

if (fileReader.hasNextLine()) {  
 String line = fileReader.nextLine().trim();  
 String[] arrayLine = line.split(",");  
 columnSize = arrayLine.length;  
  
 dataCSV = new double[rowSize][columnSize];

Através de um ciclo for começa a preencher a primeira linha do array criado no passo anterior. Como anteriormente tínhamos *strings*, o método converte em *double* já que este será o formato utilizado pelos métodos subsequentes.

for (int i = 0; i < columnSize; i++) {  
 dataCSV[0][i] = Double.*parseDouble*(arrayLine[i].trim());  
}

Após preencher a primeira linha, o método preenche as restantes através de um novo ciclo for, seguindo o mesmo processo até preencher a matriz com todos os dados.

for (int i = 1; i < rowSize; i++) {  
 line = fileReader.nextLine().trim();  
 arrayLine = line.split(",");  
 for (int j = 0; j < columnSize; j++) {  
 dataCSV[i][j] = Double.*parseDouble*(arrayLine[j].trim());  
 }  
}

## 4.8. Escrita do CSV

## 4.9. Métodos desenvolvidos

Durante o desenvolvimento da aplicação, criámos um total de 43 métodos, distribuídos entre métodos principais e auxiliares. Estes métodos foram implementados para cobrir as três principais funcionalidades do programa: **matrixDecomposition**, **imageReconstruction** e **imageIdentification**.

Para uma explicação mais detalhada, decidimos focar-nos nos métodos principais que desempenham papéis fundamentais no cálculo e manipulação de dados, como getEigenVectors, createMeanFace, centralizeMatrix, reconstructImage, createCovMatrix e imageWeights. Estes métodos são responsáveis por operações essenciais, como o cálculo de vetores prórpios, a centralização de matrizes, a reconstrução de imagens e a criação da matriz de covariância e a determinação dos pesos das imagens.

Além dos métodos principais, implementámos vários métodos auxiliares, como arrayToMatrix, matrixVectorMultiplication, e setColumnVector, que auxiliam na realização de tarefas específicas ao longo do programa, contribuindo para a modularização e eficiência do código.

### 4.9.1. createMeanFace

O método createMeanFace tem como objetivo calcular a imagem média (mean face) a partir de um conjunto de imagens representado por uma matriz bidimensional (imageMatrix), tendo por isso como parâmetro de entrada double[][] imageMatrix, matriz em que cada coluna representa uma imagem, com cada uma das linhas a corresponder ao pixel específico de cada imagem.

Assim sendo, é pretendido calcular a média de cada pixel e para isso:

São inicializadas duas variáveis locais:

* sum, que irá armazenar a soma dos valores dos mesmos pixels em diferentes imagens;
* average, que será utilizado para calcular a média desses valores.

De seguida, o método utiliza dois ciclos for para percorrer a matriz:

* O primeiro ciclo externo percorre cada linha da matriz, ou seja, cada pixel específico ao longo das imagens.
* O segundo ciclo interno percorre cada coluna da matriz, ou seja, todas as imagens que contêm aquele pixel.

Dentro do ciclo interno:

* A soma dos valores dos pixels é acumulada na variável sum.
* Após percorrer todas as imagens, é calculada a média dividindo a soma pelo número de imagens, sendo esse valor armazenado no vetor meanVector.

No final, o método retorna o vetor meanVector, que representa a média de cada pixel ao longo das imagens fornecidas.

public static double[] createMeanFace(double[][] imageMatrix) {  
 double sum, average;  
 int images = imageMatrix[0].length;  
 int pixels = imageMatrix.length;  
 double[] meanVector = new double[pixels];  
  
 for(int i = 0; i < pixels; i++) {  
 sum = 0;  
 for(int j = 0; j < images; j++) {  
 sum+= imageMatrix[i][j];  
 }  
 average = sum / images;  
 meanVector[i] = average;  
 }  
  
 return meanVector;  
}

### 4.9.2. centralizedMatrix

O método centralizedMatrix tem como objetivo centralizar uma matriz de imagens subtraindo a imagem média (*mean face*) de cada imagem do conjunto de dados. Em termos práticos, isto significa calcular a diferença entre cada imagem e a média dos seus valores de pixels, eliminando componentes comuns presentes em todas as imagens, como diferenças na iluminação ou fundo das imagens.

Este método:

1. Determina o número de linhas (pixels) e colunas (imagens) da matriz de entrada imageMatrix, através da aplicação da propriedade length.
2. Utiliza o método createMeanFace para calcular a imagem média, que tal como explicado em 4.7.2. é um vetor que contém a média de cada pixel ao longo de todas as imagens.
3. Cria uma matriz bidimensional centralizedMatrix para armazenar as imagens centralizadas.
4. Utiliza ciclos for para percorrer cada imagem (coluna) da matriz de entrada:

* Obtém a coluna correspondente à imagem.
* Subtrai a imagem média à coluna, resultando numa imagem centralizada.
* Insere a imagem centralizada na matriz centralizedMatrix.

1. Por fim, retorna a matriz centralizedMatrix.

public static double[][] centralizeMatrix(double[][] imageMatrix) {  
 int rows = imageMatrix.length;  
 int cols = imageMatrix[0].length;  
 double[] meanVector = *createMeanFace*(imageMatrix);  
 double[][] centralizedMatrix = new double[rows][cols];  
  
 for(int i = 0; i < imageMatrix[0].length; i++) {  
 double[] column = *getColumnVector*(imageMatrix, i);  
 double[] centralizedColumn = *subtractVectors*(column, meanVector);  
 *setColumnVector*(centralizedMatrix, centralizedColumn, i);  
 }  
 return centralizedMatrix;  
}

### 4.9.3. getEigenVectors

O método getEigenVectors calcula os vetores próprios de um conjunto de imagens armazenado numa matriz bidimensional matrix (que passa como parâmetro).

Para isso, inicialmente é determinado o número de linhas (pixels) e colunas (imagens) do conjunto de dados da matrix através da propriedade length. O método chama *centralizeMatrix* para centralizar a matriz de imagens matrix, e esta é armazenada na matriz bidimensional centeredMatrix. Posteriormente, é aplicado o método *createCovMatrix* à *centeredMatrix*,de forma a criar a matriz de covariância(*covArray*). A matriz de covariância mede a correlação entre os diferentes pixels das imagens, sendo essencial para identificar onde ocorre maior variação nos dados.

De forma a utilizar o objeto EigenDecomposition da biblioteca Apache Commons Math, é necessário converter a matriz de covariância para o formato RealMatrix, o que é realizado através do método auxiliar arrayToMatrix. Posteriormente à conversão, é possível utilizar EigenDecomposition para obter a matriz vetores próprios, eigenVectorsMatrix, através de eig.getV. O resultado é convertido de volta para um array bidimensional (double[][]) usando o método matrixToArray.

Após calcular os vetores próprios de ATA, o método faz a transformação necessária para obter os vetores próprios da matriz AAT, através do método auxiliar transformedEigenVectors.

Como as matrizes ATA e AAT têm os mesmos valores próprios, mas com dimensões diferentes (ATA tem dimensão 40x40 e AAT 4096x4096), é necessário transformar os vetores obtidos para obter os vetores próprios da matriz original AAT. Este cálculo é feito multiplicando a matriz A pela coluna de vetores próprios obtida de ATA, seguindo a fórmula A\*vi=ui, onde vi são os vetores próprios de ATA e ui são os vetores próprios transformados.

Seguidamente, os vetores próprios transformados são armazenados na matriz transformedEigenVectors.

O ciclo for dentro do método aplica a transformação a cada vetor próprio de ATA, calcula o vetor correspondente em AAT, normaliza-o e armazena-o na matriz transformedEigenVectors através do método setColumnVector (que coloca cada vetor próprio numa coluna da matriz).

public static double[][] getEigenVectors(double[][] matrix) {  
 int rows = matrix.length;  
 int cols = matrix[0].length;  
 double[][] centeredMatrix = *centralizeMatrix*(matrix);  
 double[][] covArray = *createCovMatrix*(centeredMatrix);  
 RealMatrix covMatrix = *arrayToMatrix*(covArray);  
 EigenDecomposition eig = new EigenDecomposition(covMatrix);  
 RealMatrix eigenVectorsMatrix = eig.getV();  
 double[][] eigenVectors = *matrixToArray*(eigenVectorsMatrix);  
  
 //Transformar os vetores próprios para A \* A^T  
 double[][] transformedEigenVectors = new double[rows][cols]; // Nova matriz para os vetores próprios transformados  
  
 for (int i = 0; i < cols; i++) {  
 double[] v\_i = *getColumnVector*(eigenVectors, i); // Obter o i-ésimo autovetor de A^T \* A  
 double[] u\_i = *matrixVectorMultiplication*(centeredMatrix, v\_i); // Calcular u\_i = A \* v\_i  
 double norm = *computeNorm*(u\_i); // Calcular a norma de u\_i  
  
 if (norm != 0) {  
 u\_i = *vectorScalarDiv*(u\_i, norm); // Normalizar o vetor  
 }  
  
 *setColumnVector*(transformedEigenVectors, u\_i, i); // Armazenar o vetor próprio transformado  
 }  
  
 return transformedEigenVectors;  
}

### 4.9.4. reconstructImages

O método reconstructImages reconstrói imagens a partir de uma matriz de imagens centralizadas, utilizando uma base de vetores próprios (eigenvectors), uma imagem média (meanFace) e um número ( k ) de componentes principais. Este segue os seguintes passos:

### 4.9.5. createCovMatrix

A função *createCovMatrix* é um passo intermedio no nosso projeto, esta função utilizada dados centralizados que vem da funcao *centralizeMatrix*. Este passo intermedio é importante pois é esta funcão que utiliza todas as imagens compara a relação entre todas. Esta funcão utiliza outras funções como a *transposedMatrix* (troca de linhas com colunas) que vai transpor a matriz, que quando é multiplica pela sua original com as regras de algebra e com a função *multipliedMatrix* o seu tamanho vai ser comprimido para o tamanho da quantidade de imagens disponiveis na base de imagens, facilitando que sejam econtradas relações entre as variaveis. A ultima funcação necessaria é o scalarMultiply que é utiliza para normalizar a matriz de covariancia.

public static double[][] createCovMatrix(double[][] centeredMatrix) {  
 double[][] covMatrix;  
 double[][] A\_T = *transposedMatrix*(centeredMatrix);  
 int N = centeredMatrix[0].length;  
  
 covMatrix = *multipliedMatrix*(A\_T, centeredMatrix);  
 return *scalarMultiply*(covMatrix, (double)1/(N));  
}

### 4.9.6. imageWeights

### 4.9.7.compareEuclideanDistance

A função *compareEuclideanDistance* integra a Funcionalidade 3 e tem como objetivo calcular a distância euclidiana entre os pesos da imagem introduzida pelo utilizador e os pesos das imagens presentes na base de dados. O propósito é determinar qual das imagens na base de dados é a mais semelhante à imagem fornecida, com base nessa comparação.

Inicialmente, a função utiliza *createMeanFace* para calcular a média das imagens na base de dados, centralizando os dados e removendo as componentes comuns.

De seguida, as imagens da base de dados são centralizadas e os pesos das imagens são extraídos. A partir desses pesos, a função calcula a distância entre a imagem introduzida pelo utilizador e as imagens na base de dados. O objetivo desta comparação de distâncias é avaliar a semelhança entre a nova imagem e cada uma das imagens da base. A função termina quando conclui a comparação com todas as imagens, retornando a menor distância encontrada.

public static double[] compareEuclideanDistance(double[][] imageMatrix, double[][] newImageMatrix, int k) {  
  
 double[] MeanFace = *createMeanFace*(imageMatrix);  
 double[] newImageVector = *squareMatrixToVector*(newImageMatrix);  
 double[] centeredNewImage = *subtractVectors*(newImageVector, MeanFace);  
 double[][] centeredImages = *centralizeMatrix*(imageMatrix);  
 double[][] eigenFaces = *getEigenVectors*(imageMatrix);  
 double[][] k\_eigenFaces = new double[eigenFaces.length][k];  
  
 for (int i = 0; i < eigenFaces.length; i++) {  
 for (int j = 0; j < k; j++) {  
 k\_eigenFaces[i][j] = eigenFaces[i][j];  
 }  
 }  
 double[][] imagesWeights = *imageWeights*(centeredImages, k\_eigenFaces);  
 double[] newImageWeights = *singleImageWeights*(centeredNewImage, k\_eigenFaces);  
 double[] comparedEuclideanDistance = new double[imageMatrix[0].length];  
  
 for(int i = 0; i < imageMatrix[0].length; i++) {  
 double[] imageWeights = *getColumnVector*(imagesWeights, i);  
 double euclideanDistance = *getEuclideanDistance*(imageWeights, newImageWeights);  
 comparedEuclideanDistance[i] = euclideanDistance;  
 }  
 return comparedEuclideanDistance;  
}

1. **Resultados**

Nesta secção, serão apresentados os resultados gerais alcançados com o desenvolvimento desta aplicação ao longo dos três sprints, de forma sintética e objetiva. O foco será na discussão crítica dos resultados, com uma análise dos pontos fortes e fragilidades do projeto, assim como sugestões de otimização e melhorias a considerar. Evitar-se-á repetir explicações técnicas já abordadas previamente, priorizando a avaliação prática do desempenho da aplicação e o balanço do trabalho realizado.

## 5.1. Apresentação dos resultados

Para apresentar os resultados de forma clara, será mais fácil seguir o fluxograma de planificação, ilustrado na Figura 6. A aplicação desenvolvida apresenta dois modos de operação: o modo interativo e o modo não interativo. No modo interativo, o utilizador introduz valores ao longo da aplicação, escolhendo a funcionalidade desejada, inserindo os valores necessários e obtendo o output correspondente. No modo não interativo, o utilizador passa uma linha de comandos, fornecendo o valor da funcionalidade (1-3), os valores próprios e vetores próprios a serem utilizados, além dos caminhos necessários para a matriz de entrada, base de imagens e saída.

Como esta no fluxograma, as funcionalidades estão organizadas de forma hierárquica, com vários métodos a serem chamados dentro do método da funcionalidade correspondente. Se algum desses métodos estivesse em falta, os resultados e outputs da aplicação seriam incompletos.

5.2. Discussão dos resultados

O desenvolvimento desta aplicação resultou em cerca de 2200 linhas de código, abrangendo as três funcionalidades principais, bem como os métodos de teste para todas as funções que não geram output. Durante o processo de desenvolvimento, a equipa enfrentou algumas dificuldades, principalmente no que toca à compreensão total dos conceitos teóricos. Apesar disso, conseguimos implementar as funcionalidades, embora o tempo necessário para a implementação tenha sido superior ao esperado devido à dificuldade da temática em questão.

Reconhecemos também que a aplicação que desenvolvemos poderia ter sido programada de forma mais eficiente, com menos linhas de código repetidas, sendo que poderíamos ter tornado o que esta repetido em funções auxiliares, o que diminuiria a quantidade de código redundante e melhoraria a manutenção do programa.

Durante o desenvolvimento, foi necessário realizar testes contínuos das funções implementadas. Para tal, foi criado um método específico que permite verificar o funcionamento de todos os testes de forma centralizada, facilitando a identificação de falhas e a posterior correção das mesmas.

No entanto, uma das limitações do desenvolvimento foi a impossibilidade de implementar o modo não interativo no cmd, utilizando o ficheiro .jar. Esta falha não foi superada.

1. **Conclusão**

A conclusão tem de resumir os pontos principais explicitados no corpo central do relatório, realçar os principais resultados alcançados e a análise e discussão daí decorrentes e, por último, deve refletir sobre o trabalho desenvolvido, apontar sugestões de aplicabilidade e melhoria.

# **Referências**

David Bissi, T. (2018). *Reconhecimento Facial com os algoritmos Eigenfaces e Fisherfaces*.

Kshirsagar, V. P., Baviskar, M. R., & Gaikwad, M. E. (2011). *Face Recognition Using Eigenfaces*.

Larson, R., & Falvo, D. (2009). *Elementary Linear Algebra*. 978-0-618-78376-2

Schwaber, K., & Sutherland, J. (2020). *Manifesto for Agile Software Development*.

Trivedi, S. (2009, February 11). *Face Recognition using Eigenfaces and Distance Classifiers: A Tutorial*.

Turk, M., & Pentland, A. (1991). *eigenfacesForRecognition*.

*!*