Breast Histopathology Dataset

O *Breast Histopathology* é um *dataset* que já incide diretamente no contexto do projeto a realizar, isto é, o processamento e a classificação de imagens biomédicas.

O *Invasive Ductal Carcinoma* (IDC) representa um dos subtipos de cancro da mama, mais comuns nos pacientes[1]. Numa fase inicial, cabe aos médicos efetuar uma análise da amostra, de modo a que seja possível identificar zonas saudáveis e/ou zonas compostas por tumor invasivo e não invasivo[1].

As amostras são representadas em imagens, que resultaram de um processo de *scanning* a tecidos dispostos em *glace slides*[1].

Este *dataset,* é composto por milhares de imagens (277524), inerentes a 280 pacientes. Destas 277524 imagens, 198738 representam tecidos saudáveis, enquanto que as restantes 78786 imagens descrevem tecidos com presença de tumor.

O objetivo do *dataset* é classificar as amostras dos pacientes, de acordo com dois *targets* possíveis (problema binário), amostra saudável ou presença de tumor.

Seguidamente, irá ser efetuada uma breve análise ao *dataset*. De modo, a que seja mais fácil entender o problema.

Análise Exploratória do Problema:

Com o objetivo de aumentar o conhecimento sobre os dados, foi efetuada uma breve análise exploratória. Esta análise tem como objetivo central ajudar o autor a entender os dados, a identificar condições que são necessárias ter em consideração na modelação e processamento dos dados e análise de métricas.

Dessa forma, foi criado um *script* em *Jupyter*, permitindo dessa forma efetuar uma clara separação dos *scripts* inerentes aos modelos, e à análise exploratória. Por outro lado, a utilização do *Jupyter Notebook* na visualização e estudo inicial dos dados torna-se mais simples e intuitiva.

Previamente, à demonstração da análise realizada é necessário esclarecer e demonstrar a estrutura que fora considerada, no armazenamento das imagens inerentes ao *dataset*. Visto que, é necessário conhecer a estrutura considerada, de modo a entender as tarefas subsequentes.

A Figura 1 descreve um excerto da arquitetura do Projeto, e dos seus respetivos ficheiros. Devido ao enorme volume de dados, 277000 ficheiros, foram demonstrados apenas amostras relativas a um paciente, sendo exibidas duas amostras referentes às duas possíveis classes do problema.

```
##Description tree structure of images directory
def rootTree(rootDir):
    list_dirs = os.walk(rootDir)
    for root, dirs, files in list_dirs:
        for i, d in zip(range(2), dirs): #ONLY RUNS TWO TIMES
            print(d)
        for j, f in zip(range(2), files):
            print(f)
```

```
rootTree("../breast_histopathology/input/")
```

```
breast-histopathology-images
IDC_regular_ps50_idx5
10253
10254
0
1
10253_idx5_x1001_y1001_class0.png
10253_idx5_x1001_y1051_class0.png
10253_idx5_x501_y351_class1.png
10253_idx5_x501_y401_class1.png
```

Figura 1 –Exemplo da estrutura que agrega as amostras do dataset

A raíz do projeto é definida através da pasta *breast_histopathology*. O projeto agrega uma outra pasta que tem como principal objetivo efetuar uma separação entre os dados e o código, pasta *input*. As pastas que se seguem na *tree*, correspondem às pastas resultantes da extração das amostras do problema. Essas pastas são: *breast-histopathology-images,* que incorpora uma única pasta no seu interior: *IDC_regular_ps50_idx5*.

Esta pasta fornece acesso às amostras relativas a cada um dos pacientes analisados. Ou seja, reúne uma pasta para cada um dos pacientes analisados. Estas pastas são identificadas através do *id* do paciente, por exemplo e considerando a Figura 1, as pastas 10253 e 10254 correspondem às pastas que contêm as amostras desse paciente específico.

Estas pastas (identificador do paciente), contêm duas pastas no seu interior. Uma pasta que contêm as amostras do paciente que são saudáveis, representadas através da pasta "0", e ainda a pasta: "1", que contêm amostras classificadas com presença de tumor (*Invasive Ductal Carcinoma*).

Por último, ambas as pastas contêm amostras do paciente, sendo estas representadas através de imagens. A pasta "0" contêm apenas imagens da classe 0 – saudável, sendo estas imagens representadas da seguinte forma: *idPacient_*******_class0.png*. Da mesma forma, a pasta "1" contêm apenas amostras – com presença de tumor, onde a única diferença na identificação da imagem é o valor da *classe*: *idPacient_*******_class1.png*.

Posteriormente, foi efetuada uma análise aos dados do problema, mais concretamente ao nº de pacientes, e ainda ao nº de amostras existentes, e à sua distribuição o longo das duas classes do problema.

Através da visualização da Figura 2, contatamos que o nº de pacientes do problema é de 279. Ou seja, o nº de pacientes é reduzido, e mediante o objetivo do estudo do *dataset*, este valor pode revelar-se reduzido. Mas, como o objetivo do estudo do autor prende-se com a classificação binária das amostras do problema, é um fator mais importante o nº de amostras existentes.

```
#HOW MANY PATIENTS ARE IN DATASET'S --> ID FOLDERS LIKE 10253
numberPatients = os.listdir("../breast_histopathology/input/breast-histopathology-images/IDC_regular_ps50_
idx5/")
print(len(numberPatients))
```

```
279
```

Figura 2 - Nº de pacientes do dataset

Sendo assim, procedeu-se à averiguação do nº de amostras existentes no *dataset.* O *BarPlot* ilustrado através da Figura 3 demonstra os resultados obtidos.

O nº total de amostras do problema é muito elevado, existindo 277524 imagens. Contudo, recorrendo à observação da Figura 3 é possível identificar uma limitação do problema, isto é, o nº de amostras das duas classes não são balanceadas. O nº de amostras saudáveis (classe 0) é bem superior ao nº de amostras inerentes à classe 1. Sendo o nº de amostras da classe 0 igual a 198738, enquanto que o nº de amostras da classe 1 fica-se pelas 78786 imagens.

Posteriormente, é necessário identificar uma abordagem que permita efetuar um correto treino, validação e teste dos modelos, tendo em conta a diferença que existe entre o nº de amostras, de ambas as classes. Visto que, é crucial garantir que os modelos aprendem e generalizam os dados da melhor forma, considerando o maior nº de situações possíveis, evitando assim que os modelos apresentem resultados tendenciosos.

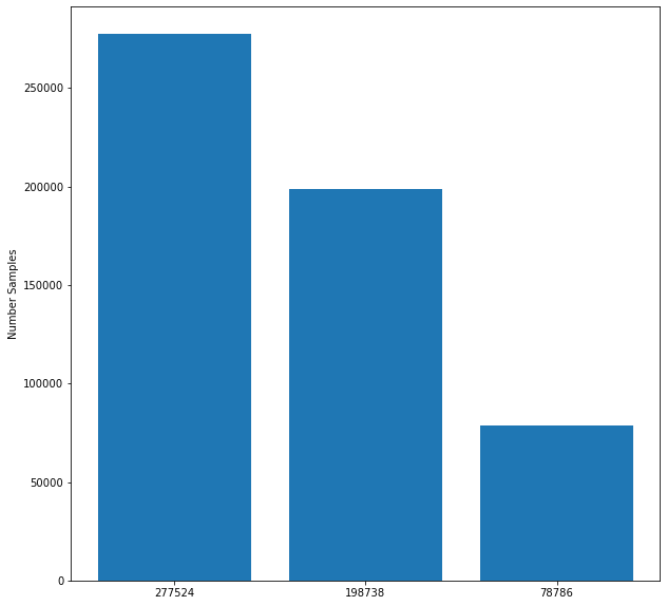

Figura 3 – Bar Plot – distribuição das amostras por classe: 1º Plot: nº total de amostras, 2ºPlot: amostras relativas à classe 0 (saudável), 3ºPlot: amostras classe 1 - Invasive Ductal Carcinoma

De modo a realizar uma análise mais incisiva, foi criado um objeto *DataFrame* com o objetivo de permitir a exploração dos dados do *dataset,* num contexto mais simples e rápido. A sua criação também irá ser importante, na transformação dos dados e respetivas classes em *numpy arrays,* para que depois se possa dar início ao processamento e tratamento dos dados, e posteriormente à criação de modelos.

O *DataFrame* criado contém todas as amostras existentes no *dataset*, ou seja, tal como já fora ilustrado através da Figura 3, 277524 imagens. Na sua identificação (identificação das imagens) foram considerados três atributos: *id* do paciente, caminho da imagem e ainda a classe à qual pertence. A Figura 4 descreve o *shape* do *DataFrame.* Este é constituído por 277524 linhas, isto é, o nº de imagens existentes, e cada linha é composta por três colunas: *id* do paciente, caminho da imagem e ainda o *target*.

```
#CHECK SHAPE OF DATA
data.shape
```

```
(277524, 3)
```

Figura 4 - Shape do DataFrame

Já a Figura 5 ilustra as cinco primeiras linhas do *DataFrame*.

	id	image_path	target
0	10253	../breast_histopathology/input/breast-histopat...	0
1	10253	../breast_histopathology/input/breast-histopat...	0
2	10253	../breast_histopathology/input/breast-histopat...	0
3	10253	../breast_histopathology/input/breast-histopat...	0
4	10253	../breast_histopathology/input/breast-histopat...	0

Figura 5 - Exemplo do DataFrame obtido (5 linhas)

Antes de continuar a análise ao *dataset,* foi efetuada uma verificação ao *DataFrame*, isto é, o autor verificou se existiam dados em falta, que podiam ter ocorrido no ato de criação do *DataFrame*. A Figura 6 ilustra a informação genérica sobre o *DataFrame*. É possível constatar que não existem valores inválidos, para cada uma das suas colunas.

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 277524 entries, 0 to 277523
Data columns (total 3 columns):
 #   Column      Non-Null Count   Dtype
---  ------      --------------   -----
 0   id          277524 non-null  object
 1   image_path  277524 non-null  object
 2   target      277524 non-null  object
dtypes: object(3)
memory usage: 8.5+ MB
```

Figura 6 - Informação sobre o DataFrame

Depois de criado o *DataFrame,* torna-se mais simples a análise e manipulação dos dados. E dessa forma, procedeu-se à análise e compreensão dos dados do problema.

Da análise realizada, foi possível identificar que o nº de amostras de cada paciente, são muito dispares entre si. Ou seja, existem pacientes que contêm um nº de amostras na ordem dos milhares, já outros pacientes possuem números bem mais simbólicos, como poucas centenas de imagens. A Figura 7 descreve o nº de *patches* referentes a cada um dos 279 pacientes do problema. A 1ª coluna representa o *id* do paciente, já a 2ª coluna ilustra o seu respetivo nº de amostras.

```
id
8863      979
8864     1133
8865      712
8867     1642
8913      955
          ...
16568     828
16569     337
16570     917
16895     151
16896    1127
```

Figura 7 - Nº de patches por paciente

No âmbito do problema a resolver, esta disparidade de amostras que existe entre os pacientes do problema, não se revela prejudicial.

Outro aspeto complementar que fora evidenciado, prende-se com o diferente nº de amostras, entre ambas as classes, para os vários pacientes do *dataset*. Isto é, como já fora mencionado atrás a classe 0 (“saudável”) encontra-se mais representada que a classe 1 (“presença de tumor”), contudo foi possível identificar que existem realidades muito distintas entre os vários pacientes do problema. Ou seja, existem pacientes em que mais de 70% das suas amostras são amostras pertencentes à classe 1, por outro lado existem pacientes em que apenas 20% das suas amostras representam amostras da classe 1. Sendo que, perto de 40% dos pacientes reúne um conjunto de amostras inferior a 20% da classe 1.

Este facto pode estar diretamente relacionado com o estado de saúde dos pacientes, ou seja, pacientes com tumor mais avançado reúnem um maior conjunto de amostras com *IDC*, já pacientes com menor incidência, reúnem um menor conjunto de amostras com *IDC*. Contudo,

esta reflexão revela-se apenas uma “teoria do autor”, sendo que a justificação para este facto pode não estar relacionado com isso. Este facto também não promove quaisquer implicações para o estudo a realizar.

As Figuras 8 e 9 demonstram respetivamente a percentagem de amostras referentes à classe 1, para cada um dos 279 pacientes do problema, enquanto que a Figura 9 ilustra o nº de pacientes, que contêm um nº de amostras da classe 1 inferior a 20%, ou superior a 80%.

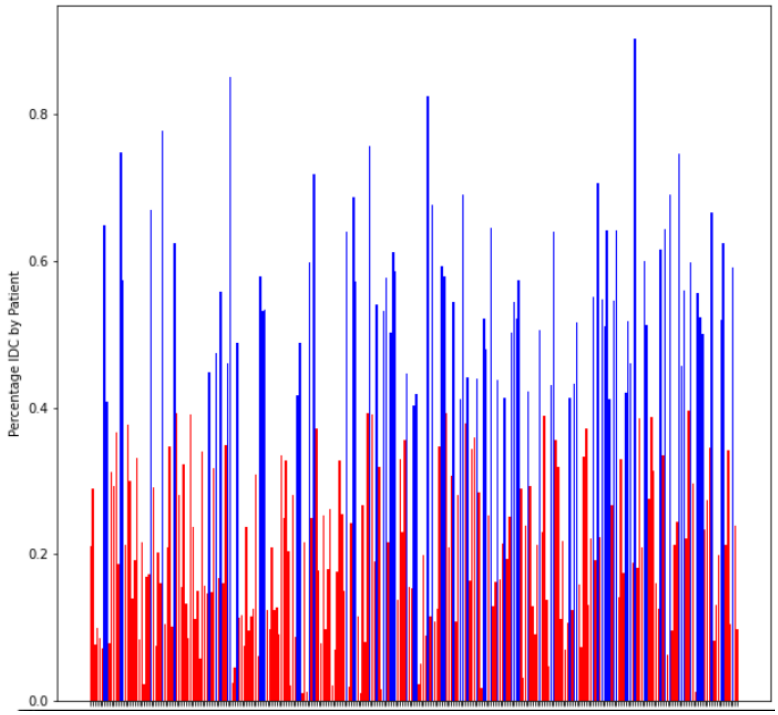

Figura 8 - Percentagem de amostras da classe 1, por paciente - Vermelho inferior a 0,4 , Azul superior a 0.4

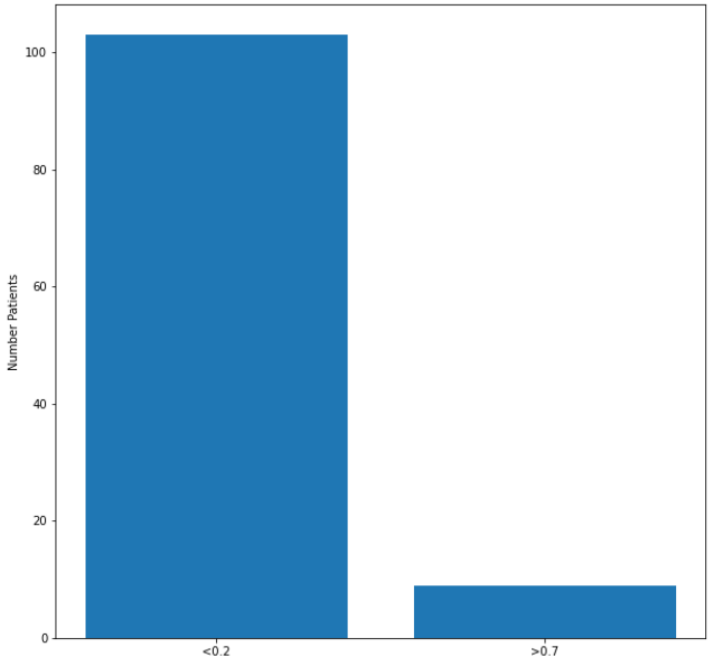

Figura 9 - Nº de pacientes com um nº de amostras da classe 1 inferior a 0.2 e superior a 0.8

Finalmente, foi efetuada uma pequena análise ao "ponto" mais importante do estudo a realizar: as imagens.

As imagens presentes no *dataset* estão representadas por uma altura e largura de 50 pixeis. Para além disso são imagens RGB, ou seja, compostas por três *channels*. A Figura 10 descreve uma das imagens do *dataset*.

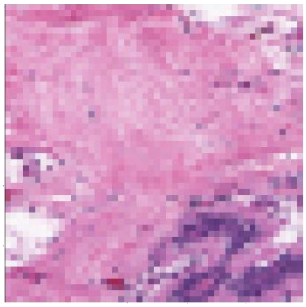

Figura 10 - Exemplo de uma imagem

O *Invasive Ductal Carcinoma (IDC)* é um tipo de cancro que começa a ser desenvolvido através dos *milk ducts*. Estes "caminhos" são responsáveis por transportar o leite que é produzido nos lóbulos, até ao *nipple*[2].

À medida que o cancro se vai desenvolvendo, o mesmo vai se "arrastando" para outras zonas da mama, mais concretamente os seus tecidos e lóbulos[2]. A Figura 11 ilustra uma representação alusiva do interior de uma mama (sendo aqui considerado apenas, as componentes essenciais ao entendimento do problema).

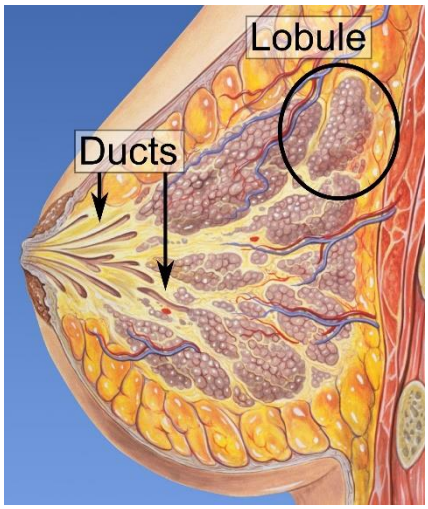

Figura 11 - Milk ducts e Lóbulos – Fonte: https://commons.wikimedia.org/wiki/File:Lobules_and_ducts_of_the_breast.jpg

Na Figura 12 podemos visualizar a diferença entre um *Normal Duct* e um *Invasive Ductal Carcinoma*, bem como a sua progressão. Através da sua visualização é possível constatar que um *Normal Duct* toda a componente de *epithelium* mantem-se inalterada. Já num IDC, a quantidade de *epithelium* vai se alterando, acabando por transpor o *Myopithelium*.

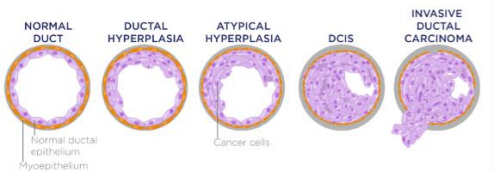

Figura 12 – Diferença entre Normal Duct e IDC

De modo, a verificar a complexidade que existe na classificação das imagens do *dataset,* foi efetuada uma análise de algumas imagens referentes à classe 0 e ainda a outras imagens referentes à classe 1. As Figuras 13 e 14 demonstram respetivamente imagens inerentes à classe 0 e ainda à classe 1.

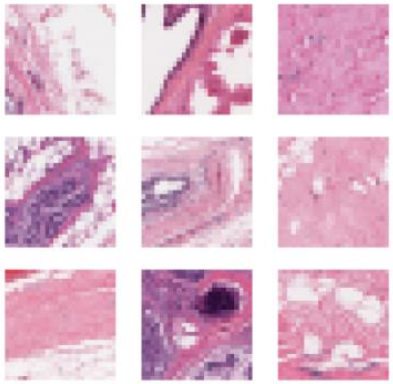

Figura 13 - Exemplo amostras Classe 0 ("saudáveis")

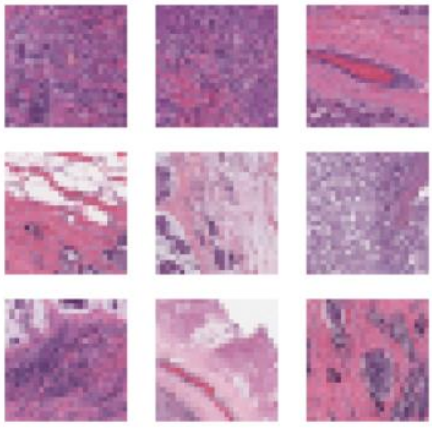

Figura 14 - Exemplo amostras Classe 1 ("IDC")

Observando ambas as Figuras, é notória a dificuldade que existe em identificar um padrão que permita classificar corretamente cada uma das imagens expressas. Essa dificuldade é tão notória que um estudo realizado [4], demonstra a efetividade que os pássaros têm na classificação correta destas imagens, superando mesmo os humanos.

O mesmo estudo [4], revela ainda que os passáros tal como os humanos revelam ainda mais dificuldades, na classificação das imagens, quando existe uma ausência de cor nas imagens, ou quando considerados ângulos distintos[4].

Apesar da dificuldade na classificação das imagens, é possível identificar uma maior presença de cores “vivas” nas imagens relativas à Classe 1, devido sobretudo ao facto do *epithelium* se arrastar ao longo do interior do *Milk Duct*.

Técnicas de pré-processamento dos dados aplicadas:

Posteriormente, e após o estudo e compreensão do *dataset*, seguiu-se a realização de um conjunto de tarefas, que têm como principal objetivo garantir que o *dataset,* reúne as melhores condições possíveis para que um modelo consiga prever corretamente, um determinado conjunto de amostras.

Antes, de dar início à fase de pré-processamento, foi necessário realizar um pequeno conjunto de tarefas, de modo a que fosse possível reunir um conjunto de dados (dados de treino, validação e de teste), que fosse "aceite", durante a execução dos modelos a serem aplicados posteriormente.

Até então, os dados estavam representados em formato *DataFrame* (objeto relativo à biblioteca *Pandas)*, mas de modo a que fosse possível treinar e prever o *ouput* de um modelo, era necessário transformar os dados, num objeto *numpy array* (caso estivesse a utilizar a biblioteca *Tensorflow,* poderia utilizar um objeto *Tensor*, mas está a ser utilizada a API *Keras*). Na realização desta tarefa, o autor contou com a ajuda da biblioteca *opencv*. Esta biblioteca ajuda a transformar uma imagem declarada num formato específico como: ".png" ou .jpeg", num objeto *numpy array.* Para além disso, permite ainda efetuar o *reshape* da imagem, num determinado *output* indicado pelo utilizador, neste caso: (*width, height, channels*).

O próximo passo a realizar prende-se com a separação do conjunto de dados, em três *datasets*: de treino, validação e ainda de teste. Neste momento, foi considerada apenas a utilização de 10000 imagens, ao invés das 277000 imagens presentes no *dataset,* de modo a reduzir o poder computacional exigido nas tarefas a serem desenvolvidas. Contudo o nº de amostras a utilizar é definido pelo utilizador, existindo assim flexibilidade total para definir o nº de amostras pretendidas pelo mesmo. No processo de separação, do nº de amostras pelos três *datasets*, foi considerada a seguinte abordagem: 60%-20%-20%, respetivamente para o *dataset* de treino, validação e teste.

Finalmente, foi ainda aplicado o conceito de *one-hot-encoding,* aos *targets,* relativos aos três conjuntos de dados criados atrás.

O conjunto de dados inicial já se encontra devidamente separado, pelos três conjuntos a utilizar nas fases de treino/validação e ainda na fase de *predict*. Os *targets* inerentes a cada *dataset* criado, também já se encontram em formato binário. Com naturalidade, segue-se a fase de pré-processamento dos dados.

A fase de pré-processamento, é normalmente dividida em "três tarefas principais": Limpeza de dados, Transformação e ainda Seleção. Seguidamente, irá ser abordado o que fora efetuado, e não efetuado tem em conta o projeto em análise.

A tarefa de Limpeza de Dados é normalmente dividida em duas tarefas, sendo estas: tratamento de *missing values* e ainda identificação de possíveis *outliers*. Considerando, o problema em análise e ainda a Figura 6 é possível constatar que o problema, não reúne dados em falta. Não existindo assim a necessidade de realizar qualquer tipo de operação, para responder a este problema.

Já, ao nível da identificação de *outliers,* não foi considerada a aplicação de qualquer técnica para dar resposta a este problema, visto que o autor não reúne conhecimentos na área, que lhe permita identificar com certeza, se eventualmente existem *patches*, que possam ser ruidosos. Para além disso, e perante o tipo de problema em causa (problema inserido no âmbito

hospitalar), é expectável que o *dataset* represente um nível de qualidade máximo. Ainda assim, o autor recorreu à utilização de *box plots*.

Seguidamente, procedeu-se à análise e implementação da técnica de transformação dos dados do problema. A aplicação de técnicas de transformação dos dados revela-se vital na maioria dos problemas.

Revela-se fulcral a utilização deste tipo de técnicas, de modo a que seja possível reduzir as diferenças na escala, que existem entre os valores presentes no *dataset*. Este tipo de abordagem ajuda a minimizar a importância que uma determinada *feature,* teria em relação à outra, dado que é considerada a mesma escala, para cada *feature*.

Considerando, este tipo de problema em específico (que apesar de apresentar escalas iguais para cada um dos *channels,* entre 0 e 255, pode eventualmente existir uma variação distinta entre si), e ainda o tipo de modelo a utilizar: rede CNN (neste tipo de redes existe uma constante partilha de parâmetros, sendo importante que exista uma escala uniforme para os dados), é notória a importância que a transformação dos dados apresenta.

Existem duas técnicas, que são consideradas *standard,* no que toca à transformação dos dados: a standardização e ainda a normalização. Ambas as técnicas visam tornar o processo de convergência mais rápido, no treino de um determinado modelo. Contudo, existem algumas diferenças entre si, nomeadamente no seu processo de cálculo. A normalização – técnica *min-max*, revela-se mais simples, estabelecendo uma escala de [0-1], para todos os pontos inerentes ao conjunto de dados. Sendo considerada a diferença entre um ponto e o valor mínimo no *dataset,* sendo depois este valor dividido pela diferença entre o valor máximo e mínimo no conjunto de dados. A Fórmula 1 exibe esta técnica.

$$X_{new} = \frac{X - X_{min}}{X_{max} - X_{min}} \quad (1)$$

Por outro lado, a standardização, técnica *z-score*, considera a transformação dos dados, para que estes sigam uma distribuição normal. Ou seja, o objetivo assenta na transformação dos dados, de modo a que estes sigam uma distribuição normal, ou seja, tenham uma $\mu = 0$ e um $\sigma = 1$. A Fórmula 2 ilustra o cálculo considerado para o efeito.

$$X_{new} = \frac{X - \mu}{\sigma} \quad (2)$$

Para este problema, foi considerada a aplicação da técnica *z-score*. A computação da média e do desvio padrão foi aplicada ao longo dos três *channels* separadamente, no conjunto de treino. Sendo que, a média e desvio padrão obtidos no conjunto de treino, foram consideradas na computação do *dataset* de validação e ainda de treino, de modo a evitar o problema de *data leakage*.

A última técnica de pré-processamento a executar é a técnica de Redução da dimensionalidade do *dataset,* ou também conhecida por *Feature Selection*. Quando considerando um típico problema de classificação de imagens, uma das abordagens mais consensuais na redução das dimensões do problema, assenta na técnica de "redimensionamento" das imagens. O utilizador diminui ou aumenta a resolução de um determinado conjunto de imagens. Geralmente, recorre-se a técnicas de *interpolation* como por exemplo: *Nearest-Neighbor Interpolation, Linear Interpolation, Fourier Interpolation,* ou ainda *Cubic Interpolation*. Estas técnicas permitem calcular valores entre pontos das amostras, por exemplo, quando se pretende aumentar o

tamanho de uma imagem, estas técnicas calculam a "nova informação" a inserir nos novos pixéis adicionados à imagem.

Antes de proceder ao redireccionamento das imagens, é necessário ter em consideração que a sua aplicação, reduz a qualidade das imagens, o que por sua vez pode ter impacto no processo de previsão dos modelos.

Dessa forma, e considerando o tamanho das imagens do *dataset* (50*50), não foi aplicado qualquer tipo de redireccionamento às imagens, isto é, aumento do seu tamanho ou decréscimo, visto que, este valor ainda é comportável. Contudo, e caso os resultados obtidos sejam satisfatórios considerando o tamanho atual das imagens, posteriormente pode ser aplicada uma técnica de redução do tamanho das imagens, com o intuito de reduzir os custos computacionais exigidos, e obter da mesma forma resultados satisfatórios.

Outra técnica usual para reduzir a dimensionalidade, passa pelo encapsulamento dos três canais RGB, num único canal *grayscale*. Tal, como as restantes técnicas, ajuda a reduzir o poder computacional, mas reduz a qualidade das imagens, e dificulta ainda mais o processo de aprendizagem.

Arquitetura:

Depois de aplicadas as várias técnicas de pré-processamento, segue-se a criação de modelos, que visam a classificação das imagens do problema.

De modo a aumentar a flexibilidade, escalabilidade e usabilidade do código, foi criado um diagrama de classes que fornece uma visa geral do comportamento do "sistema" criado, para dar resposta à resolução do problema em estudo. Para além disso, o diagrama de classes ajuda a definir e a perceber, com maior clareza, quais os padrões "de arquitetura", que melhor se adequam, e que devem ser considerados na definição da arquitetura do sistema.

Foi despendido algum tempo na criação e implementação desta arquitetura, com o objetivo de criar uma arquitetura que permitisse, a criação, treino, previsão e otimização de qualquer *tipo* de *image dataset*. Para tal, foi considerada a utilização de vários padrões arquiteturais, tais como: Fábricas de Objectos, Estratégia, Template ou Abstração.

O desenho e a modelação das interações estabelecidas entre os vários objetos criados, estabelece uma visão mais clara do comportamento do sistema, e ajuda ainda a otimizar gradualmente a arquitetura criada.

A Figura 16 ilustra o Diagrama de Classes criado. **De realçar, que a ilustração dos métodos e atributos das classes, por vezes contêm pequenas abreviações e/ou tipos de dados ou retornos "corretos", dado que a ferramenta utilizada contêm algumas limitações nesse âmbito. Ou seja, a análise do diagrama, deve ser complementada com a visualização do código**.

Foi considerada a criação de um objeto, que têm como objetivo a agregação dos vários conjuntos de dados, que irão ser utilizados, ao longo da execução do *lifecycle* de um modelo, como: treino e previsão.

Desta forma, e observando a classe *Dataset* é possível evidenciar isso mesmo, isto é, este objeto irá agregar os vários conjuntos de dados a utilizar, durante o processo de treino e previsão de um modelo. Isto é, os *datasets* obtidos na fase de pré-processamento: dados de treino, validação e ainda de teste. Esta abordagem aumenta essencialmente a escalabilidade e a

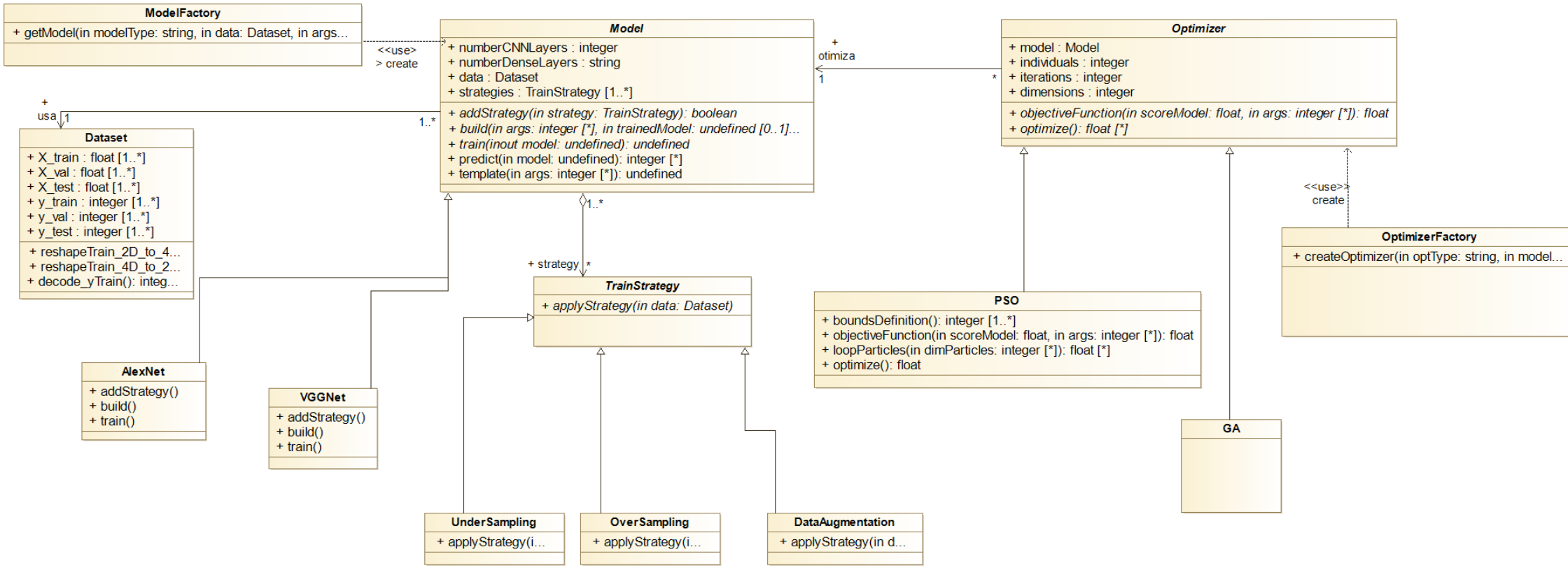

Figura 15 – Diagrama de Classes do "Sistema"

reutilização do código, ajudando o utilizador a não repetir código e a aumentar a legibilidade do código (por exemplo: evitar a criação de *templates* de uma função extensos). Para além disso, esta classe armazena funções, que aplicam funcionalidades que estão diretamente relacionadas com os dados armazenados, como por exemplo: *reshape* de um determinado conjunto de dados.

As vantagens obtidas através da criação deste objeto são facilmente reconhecíveis. Um exemplo disso, remete à *classe Model*, isto é, quando necessita de aceder a um determinado conjunto de dados (ao ter associado um objeto *Dataset*), permite-lhe melhorar a flexibilidade do código e ainda a robustez.

Posteriormente, foi criada a classe abstrata *Model*. Esta classe, tem como principal objetivo descrever uma generalização, de vários modelos que podem ser aplicados neste *dataset*, tais como: *AlexNet, VGGNet, ResNet, Inception*, entre outros. Ou seja, recorrendo ao conceito de abstração desta classe existe assim uma maior flexibilidade, para o utilizador definir extratos de código específicos, a cada "variante" da classe *Model*, como por exemplo: a construção dos modelos, onde cada tipo de rede apresenta a sua própria arquitetura, ou ainda o modo de treino a aplicar em cada tipo de rede, que pode ser distinto.

Relativamente, aos atributos desta classe são essencialmente: o nº de camadas CNN, que o modelo têm, e ainda o nº de *Dense Layers*. Para além disso, têm associado a si um objeto *Data* reunindo os vários conjuntos de dados (treino, validação e teste), que serão utilizados para treinar os modelos e prever resultados. Agrega ainda um conjunto de estratégias, que são utilizadas, no processo de treino de um modelo.

Já ao nível dos seus métodos, esta classe agrega métodos abstratos e ainda métodos comuns a todas as suas generalizações. Ou seja, os métodos abstratos constituem operações, que representam variações de implementação, tendo em consideração o tipo de objeto criado. Isto é, os métodos: *addStrategy, build* e *train*, são métodos em que a sua implementação varia mediante o tipo de modelo considerado, por exemplo a construção de um modelo *AlexNet* é distinta da criação de um modelo *VGGNet*. Por sua vez, o processo de treino de diferentes tipos de modelos, também pode apresentar variações.

Já os restantes métodos constituem implementações, que são comuns a todas as generalizações. Exemplo disso, é o método de avaliação dos modelos, que é comum a todos eles. Não existindo assim necessidade de implementar diferentes abordagens, para os vários tipos de modelos considerados. Ainda assim, e caso haja necessidade de distinguir a abordagem utilizada na avaliação dos vários tipos de modelos, este método é facilmente "convertido" para abstrato, e de seguida pode-se implementar para cada classe a devida abordagem considerada (ou efetuar *pass,* da implementação abstrata).

Recorre-se ainda à implementação de um *template method,* que têm como objetivo estabelecer uma espécie de *pipeline*, que permita ao utilizador executar todo o *lifecycle* de um modelo, de uma forma contínua. Ademais, permite ainda que todas as generalizações da classe *Model*, sigam a mesma diretriz definida, para a execução correta do modelo, pois o "esqueleto" do algoritmo está definido na sua classe "pai".

Neste método, *template_method*, foi considerada a *pipeline* típica de criação, treino e previsão de um modelo. Ou seja, inicialmente é construída a arquitetura do modelo, depois segue-se o treino do modelo (considerando as várias estratégias definidas para o mesmo), e por último recorre-se à sua avaliação (*predict*).

De modo a desacoplar o código do cliente, sobre as classes concretas do objeto *Model*, foi criada uma Fábrica de Objetos, não expondo dessa forma a lógica de implementação de uma classe. Para além disso, aumenta a reutilização de código, e facilita a escalabilidade do sistema.

Como já fora indicado anteriormente, um *Model* agrega um conjunto de estratégias de Treino no seu interior. Estas estratégias representam diferentes abordagens, que podem ser utilizadas por um modelo, durante o seu processo de treino.

Esta abordagem foi criada recorrendo ao conceito do padrão Estratégia. Para tal, foi considerada a criação de uma *interface* que estabelece as operações (funcionalidades), que as suas implementações realizam. Este padrão revela diversas vantagens, como: o facto do comportamento de uma classe poder ser alterado em *runtime*, ou a facilidade de alterar o código de uma *concrete strategy,* sem necessidade de alterar mais nenhuma classe.

A classe *TrainStrategy* descreve a interface que é implementada, pelas várias estratégias concretas: *Undersampling, Oversampling* e *Data Augmentation*. Esta classe define várias ações, que são depois implementadas pelas classes concretas, que implementam a *interface*. Neste caso, existe uma única ação: *applyStrategy*.

Esta ação é depois implementada, em cada uma das estratégias consideradas (vários tipos de estratégias). Tal como já fora indicado previamente, estas estratégias representam abordagens, que podem ou não ser consideradas durante a fase de treino de um modelo. As estratégias consideradas: *Undersampling, Oversampling* e *Data Augmentation*, descrevem ações que reúnem abordagens de implementação distintas.

Um modelo agrega um conjunto de estratégias, sendo que o seu comportamento é variável, tendo em consideração o tipo de estratégias que utiliza. Ou seja, caso o modelo não utilize estratégia nenhuma, então o processo de treino ocorre normalmente, caso utilize por exemplo: a estratégia *UnderSampling*, então o treino do modelo vai variar, ou por exemplo caso utilize duas estratégias ainda varia mais.

Este tipo de abordagem flexibiliza o código e a interpretabilidade do mesmo. Contudo, apresenta algumas limitações: como o aumento do nº de classes, ou o nº de objetos que é necessário manter em *runtime*.

Por último, segue-se a explicação da abordagem considerada, na implementação dos algoritmos de otimização, que são utilizados no processo de otimização dos modelos criados.

Para tal, foi considerada a utilização do conceito de abstração. Sendo criada a classe abstrata *Optimizer*, que reúne um conjunto de atributos que são "partilhados", por qualquer tipo de algoritmo de otimização: nº de partículas/população, nº total de iterações e ainda o nº de dimensões do problema. Para além disso, é lhe ainda associado o modelo que irá otimizar.

Neste caso, em concreto foi considerada a utilização de apenas dois algoritmos de otimização: *Particle Swarm Optimization* (*PSO*) e *Genetic Algorithm* (*GA*). De realçar que, o algoritmo *GA* ainda não foi implementado, e dessa forma, ainda não foi especificado o "seu conteúdo" no diagrama de classes ilustrado (Figura 16).

A classe abstrata *Optimizer,* contêm dois métodos abstratos, que também são variáveis mediante o tipo de algoritmo a utilizar: a definição da função objetivo (*objectiveFunction*) e ainda a implementação da funcionalidade de otimização do modelo (*optimize*). Ou seja, o PSO descreve uma abordagem de otimização distinta do GA, e de outros algoritmos de otimização,

e para além disso cada algoritmo de otimização, pode ou não, descrever funções objetivo, distintas entre si. Dessa forma, podemos constatar observando a classe PSO, que a mesma implementa ambos os métodos abstratos descritos na classe *Opimizer*, e para além disso contém ainda outros métodos concretos, que são apenas alusivos à sua classe, isto é, são métodos necessários para complementar determinadas funcionalidades específicas do *PSO*.

Finalmente, e tal como aplicado no caso da classe *Model,* foi criada uma fábrica de objetos, que ajuda na criação dos optimizadores a utilizar. Melhorando assim, a flexibilidade e a reutilização do código.