UNIVERSIDADE DE BRASÍLIA
HOSPITAL UNIVERSITÁRIO DE BRASÍLIA
PROGRAMA DE RESIDÊNCIA MÉDICA EM RADIOLOGIA E DIAGNÓSTICO POR IMAGEM

VINICIUS MARTINS VILELA

INTRODUÇÃO AO PROCESSAMENTO DE IMAGENS *DICOM*, LINGUAGEM DE MÁQUINA E REDES NEURAIS: UMA ABORDAGEM PRÁTICA.

BRASÍLIA
2022

VINICIUS MARTINS VILELA

INTRODUÇÃO AO PROCESSAMENTO DE IMAGENS *DICOM*, LINGUAGEM DE MÁQUINA E REDES NEURAIS: UMA ABORDAGEM PRÁTICA.

Trabalho apresentado como requisito parcial para conclusão do Curso de Pós-Graduação lato sensu, modalidade Programa de Residência Médica em Radiologia e Diagnóstico por Imagem, do Hospital Universitário de Brasília.

Orientador: Dr. Wagner Diniz de Paula

BRASÍLIA

2022

Autorizo, apenas para fins acadêmicos e científicos, a reprodução total ou parcial deste Trabalho de Conclusão de Curso, desde que citada a fonte.

_____ _____

Assinatura Data

AGRADECIMENTOS

Agradeço a minha família por todo suporte e carinho. Aos meus pais, Wilma e Aderison, por todo amor incondicional. A minha esposa Dra. Laís Dutra de Freitas por todo amor, auxílio e paciência, sendo meu ponto de apoio em momentos difíceis. A Dra. Mariana Vilela por ser exemplo de dedicação e fonte de inspiração.

Agradeço ao meu orientador Dr. Wagner Diniz de Paula pela disponibilidade, ajuda e conselhos durante todo o período de residência. Ao Dr. Henrique Metzger pelo auxílio e paciência.

Agradeço a Dra. Mayra Veloso Ayrimoraes Soares, supervisora do programa de Residência Médica, por nos orientar durante toda a residência, pela sua dedicação e por ser um exemplo profissional a ser seguido.

Agradeço aos meus colegas de residência Dr. Érico Alexandro Vasconcelos de Menezes, Dr. Marcus Vinicius Bastos Sasaki e Dra. Amina Muhamed pela amizade, companheirismo e apoio durante o período de residência.

Agradeço aos meus demais colegas de Residência, *staffs* queridos e profissionais do setor que contribuíram para a minha formação e direta ou indiretamente com este trabalho.

RESUMO

As redes neurais são consideradas o estado da arte para automação e reconhecimento de imagens. O entendimento básico sobre a computação e as principais metodologias envolvidas faz-se necessário para construir as habilidades mínimas para criação e implementação de projetos na área.

Python é uma das linguagens de programação mais utilizadas no meio científico para o desenvolvimento e implementação desses algoritmos. Os seus atuais frameworks de desenvolvimento disponibilizam bibliotecas específicas para leitura e processamento de imagens médicas digitais.

A padronização dos exames através do padrão Digital Imaging and Communications in Medicine (DICOM) permite o uso da biblioteca Pydicom para leitura e manipulação de seus atributos.

Os dados da imagem podem ser manipulados com operações algébricas, porém várias bibliotecas permitem a implementação em alto nível das principais transformações, aplicações de filtros, detecção de curvas e bordas etc.

O OpenCV disponibiliza, de maneira acessível, funções para processamentos avançados de imagem, incluindo a localização de templates contidos em uma imagem de interesse com execução em bloco.

Softwares de análise de imagens científicas, como o ImageJ, podem ser utilizados para otimizar o tempo durante o processo de criação de um banco de dados de templates.

O estudo de linguagem de máquina depende da compreensão dos métodos mais comuns de aprendizado supervisionado e não supervisionado.

Algoritmos de regressões lineares e classificações constituem a base primária desses conceitos. Os modelos de agrupamentos podem ser representados pelo K-Means, o qual pode ser utilizado nas segmentações.

As redes neurais podem variar desde modelos mais simples, como o perceptron, até modelos mais complexos para identificação de imagens. Bibliotecas como Keras e TensorFlow tornam a sua implementação mais intuitiva. A avaliação dos resultados com o TensorBoard possibilita a comparação das execuções dos diferentes modelos, facilitando sua interpretação.

Conhecer as especificidades e limitações dos métodos permite ao radiologista realizar correções e adequações na aquisição das imagens e na construção de um banco de dados para treino.

ABSTRACT

Neural networks are considered the state of the art for automation and image recognition. A basic understanding of computing and the main methodologies involved are necessary to build the minimum skills for creating and implementing projects in the area.

Python is one of the most used programming languages in the scientific world for the development and implementation of these algorithms. Its current development frameworks provide specific libraries for reading and processing digital medical images.

The standardization of exams through the Digital Imaging and Communications in Medicine (DICOM) allows the use of the Pydicom library to read and manipulate its attributes.

The image data can be manipulated with algebraic operations, but several libraries allow the high-level implementation of the main transformations, application of filters, detection of curves and edges, etc.

OpenCV provides, in an accessible way, functions for advanced image processing, including finding templates contained in an image of interest with block execution.

Scientific image analysis software, such as ImageJ, can be used to optimize time during the process of creating a template database.

Studying machine language depends on understanding the most common methods of supervised and unsupervised learning.

Linear regression algorithms and classifications form the primary basis of these concepts. Clustering models can be represented by K-Means, which can be used in segmentations.

Neural networks can range from simpler models, such as the perceptron, to more complex models for image identification. Libraries like Keras and TensorFlow make their implementation more intuitive. The evaluation of the results with the TensorBoard makes it possible to compare the executions of the different models, facilitating their interpretation.

Knowing the specifics and limitations of the methods allows the radiologist to make corrections and adjustments in the acquisition of images and in the construction of a database for training.

LISTA DE ABREVIATURAS

API	A*pplication Programming Interface* (Interface de Programação de Aplicação).
DICOM	*Digital Imaging and Communications in Medicine* (Comunicação de Imagens Digitais em Medicina).
HIPAA	Health Insurance Portability and Accountability Act (Lei de Portabilidade e Responsabilidade do Seguro Saúde).
IDE	I*ntegrated Development Environment* (ambiente de desenvolvimento integrado).
NFC	*Near Field Communication* (tecnologia de campo de comunicação próximo).
PACS	P*icture Archiving and Communication Systems* (sistema de comunicação e armazenamento de imagem).
PHI	Protected Health Information (informação de saúde protegida e/ou sensível).

DEFINIÇÕES

Open source	Código fonte disponibilizado e licenciado com uma licença de código aberto no qual o direito autoral fornece o direito de estudar, modificar e distribuir o software para qualquer um e para qualquer finalidade.
Framework	Abstração ou plataforma que une subconjunto de funções, bibliotecas ou códigos para utilização em finalidades genéricas ou específicas, dependendo de sua construção.
LGPD	Lei Geral de Proteção de Dados Pessoais.
Pixel	Menor elemento ou componente de uma imagem digital.
Kernel	Interface central entre o hardware e os processos executados por um computador.
Software	Programa ou conjunto de instruções executáveis.
UH	Unidade de Hounsfield.

ÍNDICE

1. INTRODUÇÃO **11**

2. OBJETIVOS **12**

2.1. Objetivo geral *12*

2.2. Objetivos específicos *12*

3. METODOLOGIA **13**

4. DESENVOLVIMENTO **14**

4.1. Linguagem de programação / Python *14*

4.2. Ambiente de desenvolvimento *15*

4.3. Bibliotecas comumente utilizadas *21*

4.4. Interfaces gráficas para segmentação e visualização de imagens DICOM *22*

4.5. Python para manipulação e abertura de imagens DICOM *23*

4.5.1. Manipulação de dados e atributos 23
4.5.2. Anonimização 27
4.5.3. Visualizar imagens com Python 28
4.5.4. Salvando imagens com formato diferente 29
4.5.5. Gerando histograma 30
4.5.6. Transformando para escala de Hounsfield 30
4.5.7. Controle de janela, nível e filtro 31
4.5.8. *Colormap* e Inversão dos valores da imagem 35
4.5.9. Transformações 37
4.5.10. Trabalhando com imagens sequenciais 38
4.5.11. Plotando a imagem em diferentes planos. 40
4.5.12. Redimensionamento 41
4.5.13. Plotando 3D 42
4.5.14. Segmentação 44
4.5.15. Aplicando máscara em uma imagem 46

4.6. Linguagem de máquina – Machine learning *47*

4.7. Introdução aos modelos supervisionados *48*

4.7.1. Regressão linear 48
4.7.2. Regressões logísticas/não lineares e sistemas de classificação 54
4.7.3. Avaliação da adequação do modelo aos dados 56
4.7.4. Classificação Vs Agrupamento 57
4.7.5. Agrupamento: Clusterização (K-Means) 57
4.7.6. Tipos de agrupamento: Plano e Hierárquico 59
4.7.7. Localização de imagem Utilizando OpenCV 59

4.8. Redes Neurais *69*

4.8.1. Perceptron e o modelo neuronal 69
4.8.2. Redes neurais convolucionais (CNN) e reconhecimento de imagem 72
4.8.3. Preparando o banco de dados/*dataset* 72
4.8.4. Classificação de Pfirrmann 73
4.8.5. Preparando as imagens dos intervalos discais para criação do Dataset 74

4.9. Limitações *81*

5. CONCLUSÃO **82**

6. REFERÊNCIAS **83**

APÊNDICE A **85**

APÊNDICE B **86**

APÊNDICE C **89**

APÊNDICE D **90**

1. INTRODUÇÃO

Saber programar ou conhecer uma linguagem de programação não é um pré-requisito para o desenvolvimento de habilidades nos campos da ciência de dados, automação e linguagem de máquina. Existem papeis e designações diferentes que podem ser atribuídas ao cientista de dados, as quais, não necessariamente, estão associadas a programação.

O conhecimento prévio sobre construção de algoritmos e programação básica facilitam o seu estudo e aceleram a aprendizagem. É desejável que o profissional tenha entendimento básico dos principais conceitos, porém o seu desconhecimento não é fator impeditivo.

Os exemplos disponibilizados permitem realizar uma abordagem prática com enfoque nos principais temas relacionados ao processamento de imagens médicas. A complexidade do conteúdo é incrementada gradativamente até o tema de linguagem de máquina e redes neurais.

2. OBJETIVOS

2.1. Objetivo geral

Abordar o aspecto computacional básico das metodologias de processamento de imagens médicas digitalizadas, compreendendo desde as etapas de pré-processamento até os conceitos de linguagem de máquina e a implementação de uma rede neural simples.

Enfoque na programação e conceitos gerais relacionados ao tema.

2.2. Objetivos específicos

1. Demonstrar diferentes métodos de processamento de imagem.
2. Definir e exemplificar diferentes metodologias utilizadas em linguagem de máquina.
3. Utilizar algoritmos de execução em bloco para segmentação automática dos intervalos intervertebrais.
4. Implementar um projeto piloto de uma rede neural para classificação de discos intervertebrais no exame de coluna lombar com a sequência sagital ponderada em T2.

3. METODOLOGIA

Este trabalho consiste de uma revisão sistemática da literatura com execução de exemplos práticos que envolvem a programação relacionada com o processamento, construção de algoritmos de linguagem de máquina e redes neurais para a avaliação de imagens médicas digitais.

A abordagem realizada evolui de maneira gradativa com linguagem de programação em Python e códigos executados com exemplos práticos dentro do contexto da radiologia.

Foi realizado em estudo pelas documentações disponíveis para as principais bibliotecas utilizadas, assim como páginas de referências em programação e linguagem de máquina, incluindo o *GitHub, Python Package Index, Kaggle* etc.

Os principais bancos de dados foram pesquisados com as palavras chaves: *Pydicom, OpenCV*, processamento de imagens médicas, linguagem de máquina, redes neurais.

4. DESENVOLVIMENTO

4.1. Linguagem de programação / Python

O Python é uma linguagem de programação de alto nível, cuja sintaxe é intuitiva, oferecendo maior facilidade de entendimento, legibilidade aos usuários menos experientes e, atualmente, é uma das linguagens de programação mais utilizadas em todo o mundo. No entanto, embora funcione bem como linguagem para iniciantes, também é adequada para tarefas mais avançadas e complexas.[1]

Em comparação com linguagens como C/C++/C# *e Java*, o Python oferece suites como *Jupiter Notebook* as quais permitem mesclar código com anotações/explicações, sendo possível executar ou reexecutar blocos e segmentos individualmente (tabela 1).

Existem algumas pequenas desvantagens em relação a linguagens mais estruturadas, como as múltiplas versões disponíveis e em constante atualização, formatação como parte da sintaxe, velocidade de compilação e execução e algumas bibliotecas dependem de módulos e compilados de C/C++ para serem executadas.[2]

-Python é uma linguagem moderna e de alto nível com tipagem dinâmica e sintaxe e semântica simples e consistentes.
-Python é multiplataforma, altamente modular e adequado tanto para desenvolvimento rápido quanto para programação em larga escala.
-É razoavelmente rápido e pode ser facilmente estendido com módulos C ou C++ para velocidades mais altas.
-O Python possui recursos avançados integrados, como armazenamento de objetos persistentes, tabelas de hash avançadas, sintaxe de classe expansível e funções de comparação universais.
-O Python inclui uma ampla variedade de bibliotecas, como processamento numérico, manipulação de imagens, interfaces de usuário e scripts da web.
-É suportado por uma comunidade Python.

Tabela 1. Algumas das vantagens do Python[2]

4.2. Ambiente de desenvolvimento

As plataformas de distribuição de pacotes Python e R constituem uma das maneiras de facilitar a implementação e configuração do ambiente para programação.

O Anaconda é uma das mais populares plataformas para distribuição de pacotes e suites de desenvolvimento (IDES) relacionados com ciência de dados, automação e linguagem de máquina. O mesmo possui uma edição para uso individual, multiplataforma, código aberto e com suporte pela comunidade científica, a qual contém múltiplas opções de ambientes de desenvolvimento integrados.

Existem, contudo, outras edições da plataforma que possuem custo e licenças específicas melhor descritas e abordadas em *https://docs.anaconda.com/anacondaorg/*.

A instalação é dependente de sistema operacional, contudo instaladores individuais estão disponíveis em livre acesso no próprio site da documentação.

Uma vez instalado, a interface de usuário não difere muito entre os diferentes sistemas operacionais.

A página inicial ou ***anaconda.navigator*** permite acesso intuitivo às diferentes opções de configuração. Na página de ***Home*** é possível escolher instaladores para os diferentes IDES e ferramentas de manipulação (figura 1).

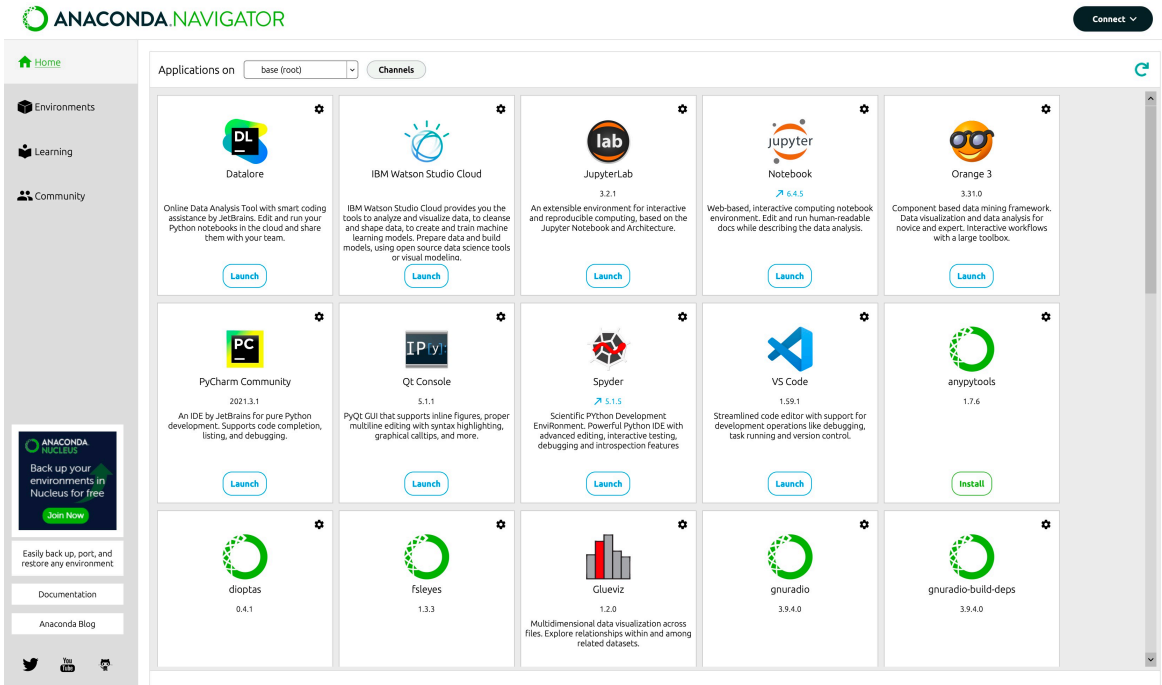


Figura 1. Pagina inicial da interface de usuário ou **anaconda.navigator**.

É importante observar que o ambiente raiz (*root*) está pré-selecionado após o processo de instalação padrão, e o recomendável é criar ambientes virtuais para projetos com diferentes propósitos (figura 2).

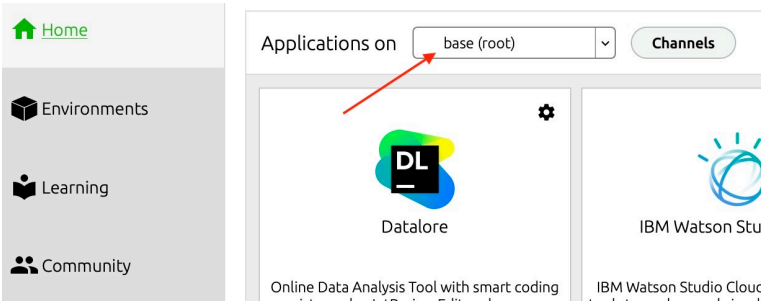


Figura 2. Ambiente raiz (root) pré-selecionado (seta). Geralmente utilizado apenas como base para criação dos diferentes ambientes virtuais secundários a novos projetos ou propósitos.

Existe a aba que permite criar novos ambientes virtuais a partir de um pré-existente. A funcionalidade de listar os pacotes instalados e buscar por novos também está disponível nessa mesma janela (figura 3).

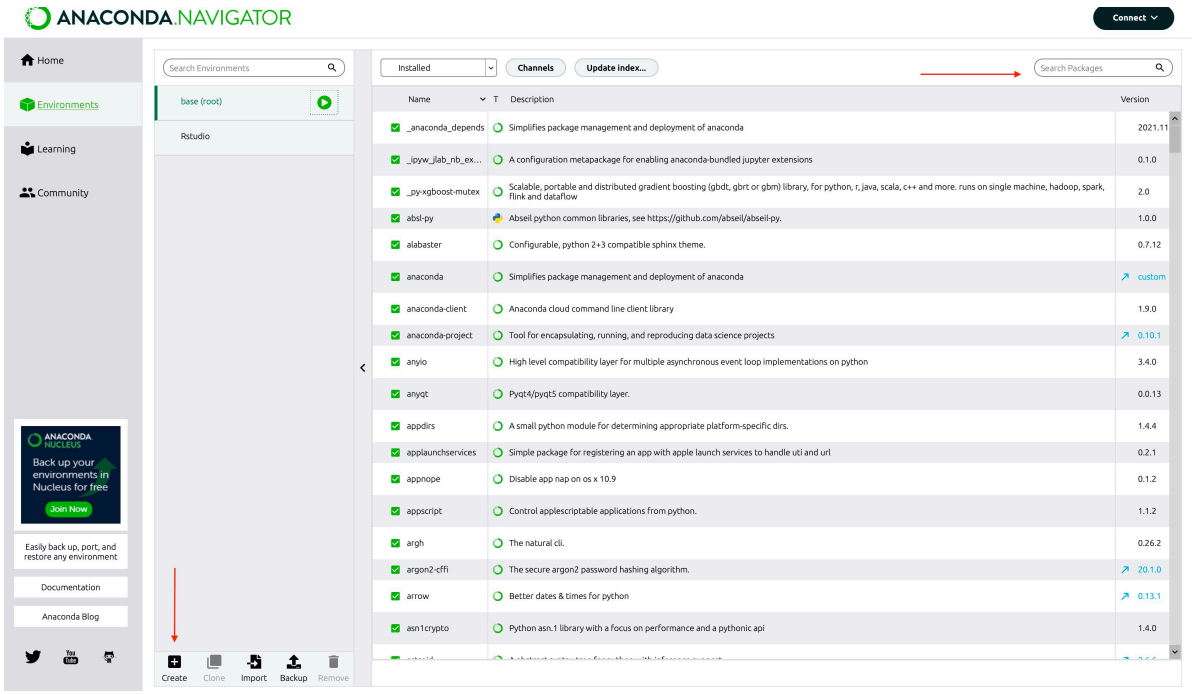


Figura 3. Aba onde é possível criar um novo ambiente e listar os pacotes instalados para o ambiente atual selecionado (setas).

Também é possível abrir o terminal de comando com as variáveis de ambiente e compilador designado para o ambiente atual (figura 4).

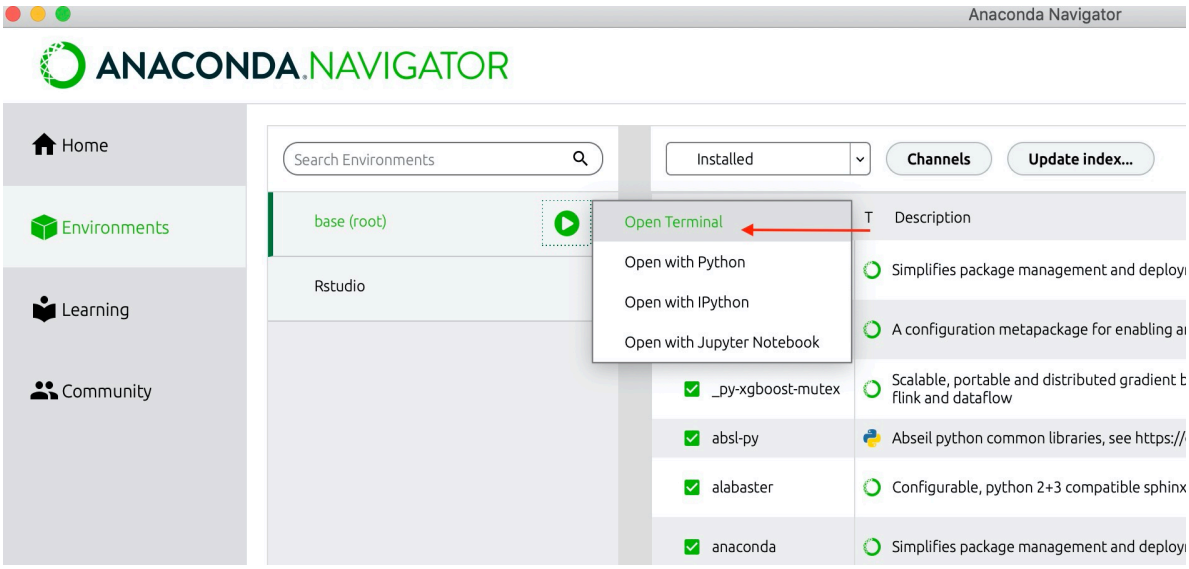


Figura 4. Funcionalidade de abrir o terminal para o ambiente atual selecionado (seta).

A instalação dos pacotes de desenvolvimento pode ser realizada pela própria interface gráfica ou pelo terminal de comando. A interface permite selecionar de maneira mais intuitiva os pacotes não instalados e realizar a tentativa de instalação dos mesmos - que na verdade executam em plano de fundo os comandos no terminal.

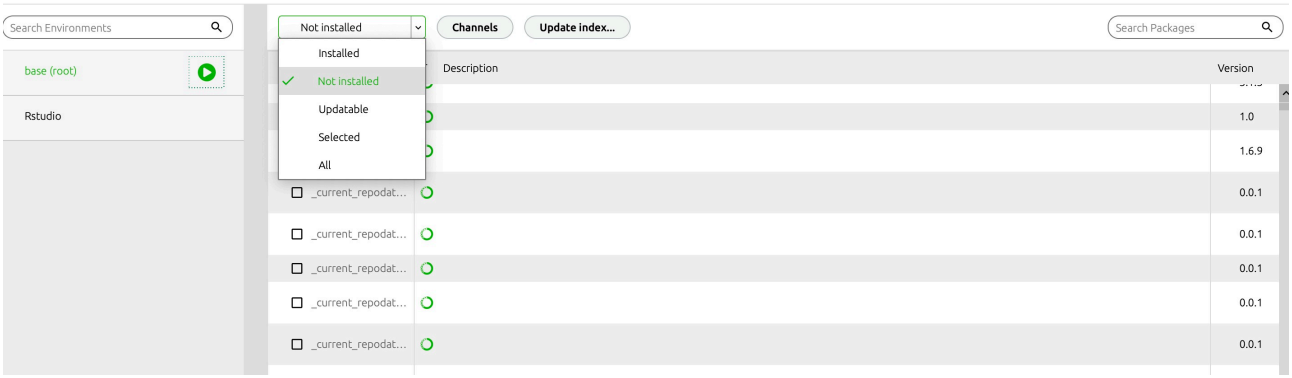


Figura 5. Instalação de pacotes pela interface gráfica.

No console/terminal há dois binários executáveis responsáveis pelo gerenciamento de pacotes do Python, o ***Package Installer for Python (pip)*** – que está linkado ao próprio compilador e suite Python – e o ***Conda Installer (conda)*** que está ligado à suite do *Anaconda*.

Ambas as aplicações apresentam sintaxe específica para listar pacotes, instalar, desinstalar, atualizar etc (figura 6).

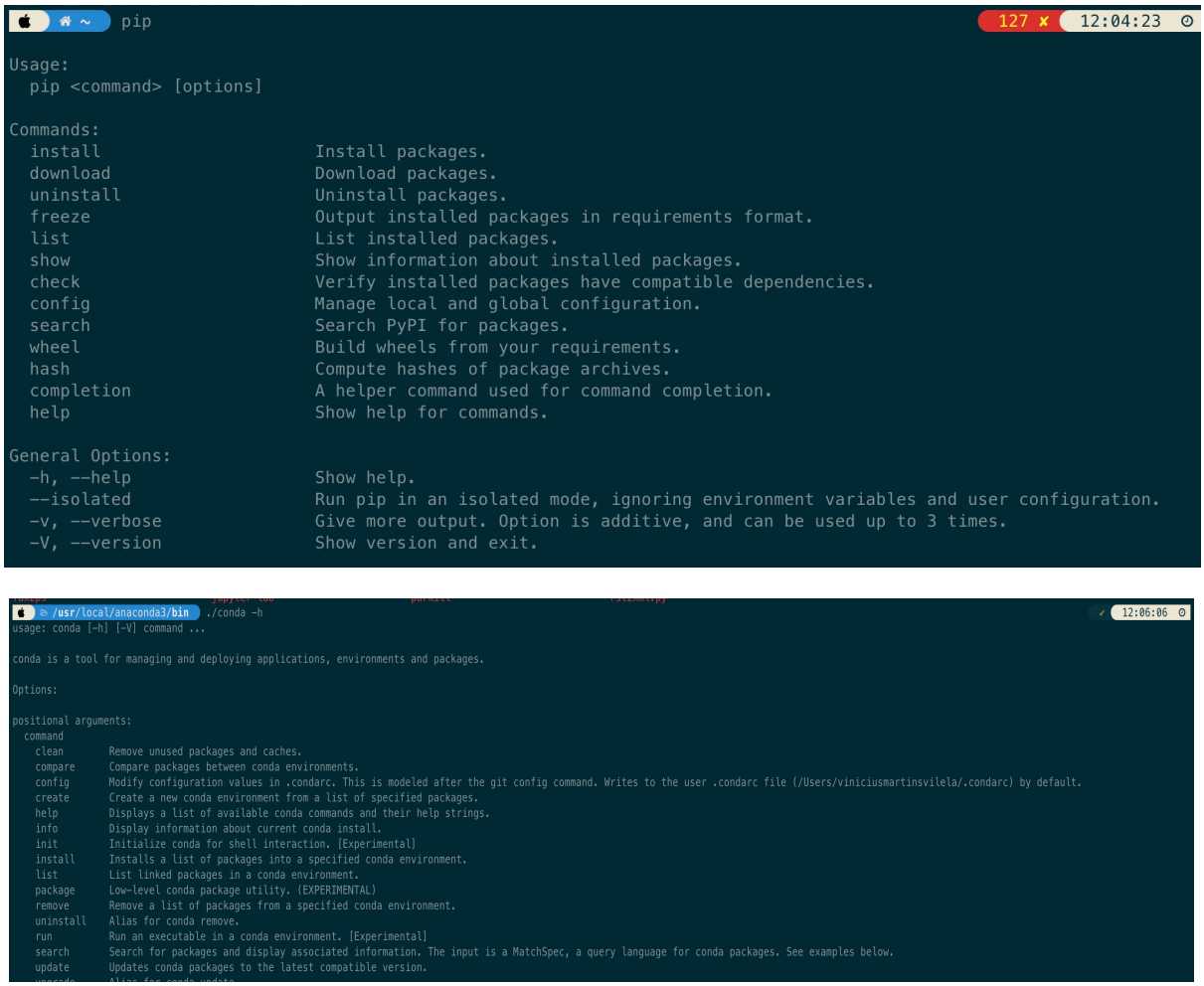
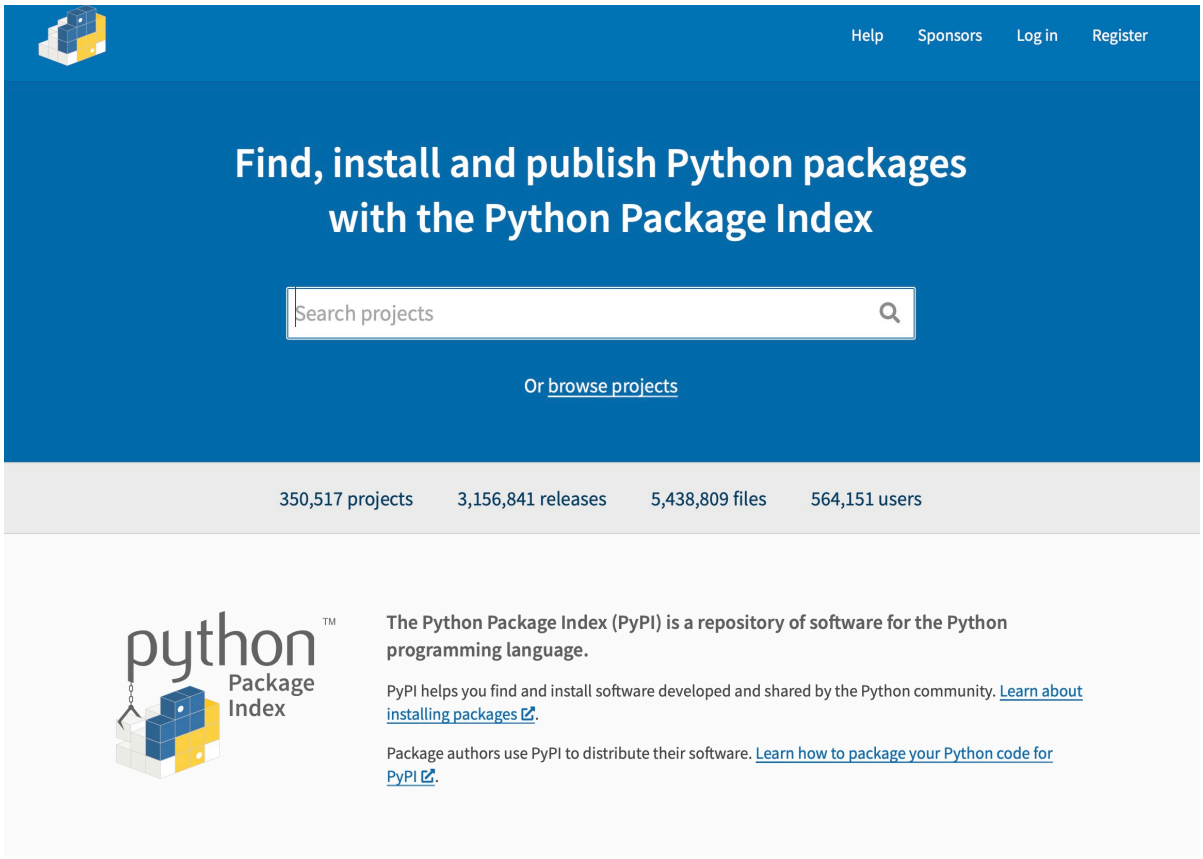


Figura 6. Gerenciadores de pacote ***pip*** e ***conda*** com suas principais sub-rotinas.

Um dos exemplos de sintaxe para instalação dos diferentes pacotes pelo terminal de comandos é o ***"pip install pydicom"*** ou ***"conda install -c conda-forge pydicom"***.

Suas principais sub-rotinas procuram os instaladores e duas dependências nos repositórios de distribuição cadastrados. Dessa maneira é possível pesquisar por pacotes e bibliotecas diretamente nesses repositórios (figura 7).[3]



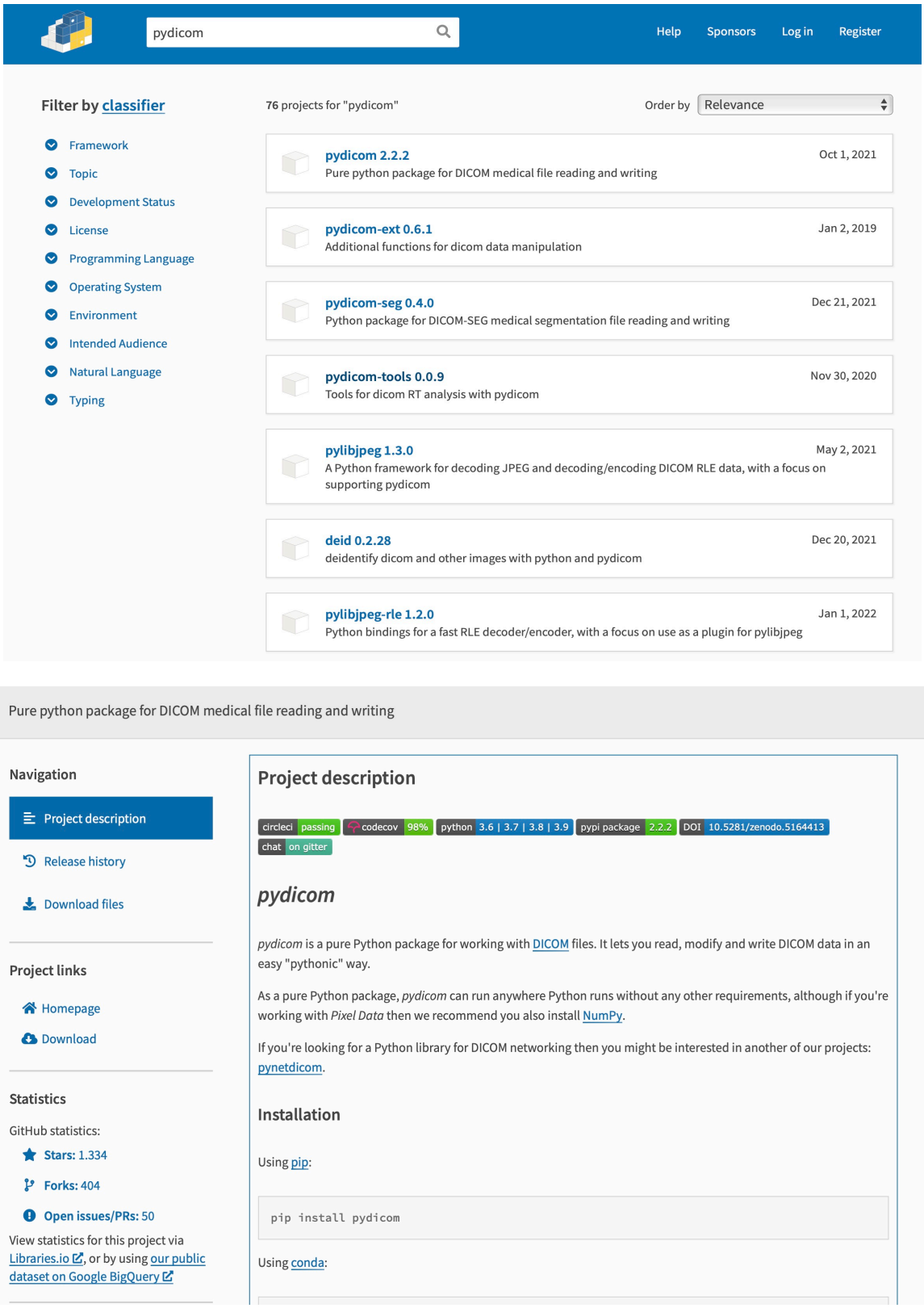


Figura 7. Repositório e base de dados Python Package Índex (https://pypi.org), onde conseguimos buscar e acessar a documentação (incluindo processo de instalação para diferentes pacotes).

4.3. Bibliotecas comumente utilizadas

O vasto conjunto de bibliotecas disponíveis para estatística, ciência de dados, processamento e visualização de imagens dificulta a escolha de um ponto de início para o aprendizado.[6] As bibliotecas e pacotes descritos servem como guia para facilitar o entendimento da maioria dos scripts de código aberto e tutoriais disponíveis:

- OS: acesso a funções básicas do sistema operacional;
- Glob: manipulação e *parser* de caminhos dos arquivos no sistema operacional;
- Datetime: manipulação de dados em formato de tempo e período (date);
- Plotly: biblioteca para visualização iterativa;
- PyQt5: Qt é um conjunto de bibliotecas C++ multiplataforma que implementam APIs de alto nível para acessar muitos aspectos de sistemas móveis e desktop modernos. Isso inclui serviços de localização e posicionamento, multimídia, conectividade NFC e Bluetooth, um navegador da Web baseado em Chromium, bem como desenvolvimento de interface do usuário tradicional;
- Numpy: rotinas matemáticas com capacidade de integrar com C/C++ e Fortran;
- Theano: biblioteca e compilador de otimização para manipular e avaliar expressões matemáticas, especialmente com vetores;
- Pandas: manipulação de dados em matrizes, arquivos CSV, etc.
- Matplotlib: biblioteca para plotagem de dados;
- Scikit-Image: biblioteca para processamento de imagem;
- Scikit-Learn: biblioteca voltada para linguagem de máquina e ciência de dados;
- OpenCV: biblioteca especializada em visão computacional;
- Tensorflow: biblioteca para desenvolvimento e para realização de aprendizado de máquina com código aberto baseada na biblioteca Torch, usada para aplicativos como visão computacional e processamento de linguagem natural;
- Keras: *Framework ou API* – que podem rodar scripts e chamar funções de outras bibliotecas, entre elas o TensorFlow
- PyTorch: *Framework ou API* – para desenvolvimento e realização de aprendizado de máquina com código aberto baseada na biblioteca Torch, usada para aplicativos como visão computacional e processamento de linguagem natural;

- Pydicom: pacote para trabalhar com arquivos DICOM, permitindo leitura, escrita, modificação etc;
- Pydicom-tools: pacote de funções variadas para auxílio com manipulação de arquivos DICOM;
- Pydicom-seg: pacote para leitura e escrita de DICOM segmentados permitindo conversão para formato aceito pelo ITK;
- Pydicom-net: pacote para utilização de protocolos de comunicação;
- Deid: pacote para anonimização das imagens DICOM;
- NiBabel: pacote fornece acesso de leitura e gravação a alguns formatos comuns de arquivos médicos e de neuroimagem, incluindo: ANALYZE (simples, SPM99, SPM2 e posterior), GIFTI, NIfTI1, NIfTI2, CIFTI-2, MINC1, MINC2, AFNI BRIK/HEAD, MGH e ECAT, bem como Philips PAR/REC. Permite acesso de leitura e escrita a arquivos de geometria, anotação e morfometria do FreeSurfer. Há algum suporte muito limitado para DICOM.
- Tkinter: biblioteca para construção de interface gráfica básica;

Contudo, nem todas as bibliotecas foram utilizadas na realização deste trabalho. O propósito dessa lista é fazer com que o leitor esteja familiarizado com os nomes das principais bibliotecas disponíveis.

Durante a execução do código, as bibliotecas podem ser importadas com as seguintes instruções, dentre as várias maneiras:

> import biblioteca as pseudônimo

> from biblioteca import função

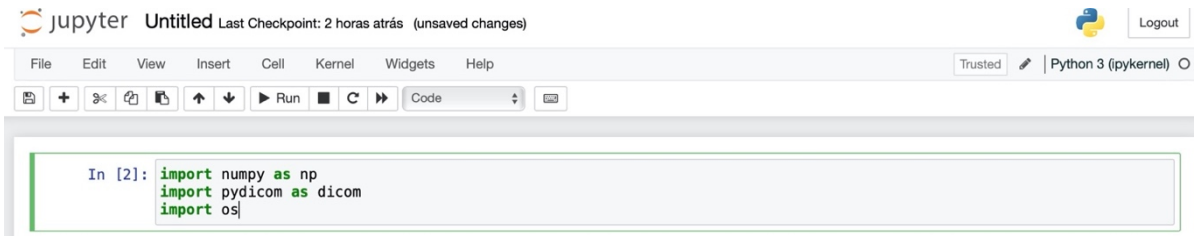


Figura 8. Importando algumas bibliotecas para uso no código.

4.4. Interfaces gráficas para segmentação e visualização de imagens DICOM

Existem múltiplos softwares com interfaces gráficas apresentando diferentes licenças para utilização e conjuntos de funcionalidades.

Essas ferramentas podem facilitar o processamento para treino e visualização de imagens, entretanto, podem dificultar o processo de uniformização/padronização das segmentações, uma vez que a interface gráfica pode ser dependente de avaliação visual subjetiva e/ou conhecimento e familiarização específica com determinado programa.

Exemplos de ferramentas de código aberto e distribuídas gratuitamente para uso são: ***ImajeJ, ITK-SNAP e 3DSlicer.***

4.5. Python para manipulação e abertura de imagens DICOM

4.5.1. Manipulação de dados e atributos

Uma das funções que torna o ***Jupyter notebook*** tão atrativo é a possibilidade de adicionar comentários e explicações juntamente com o código executável.

A biblioteca ***Pydicom*** disponibiliza funções básicas que permitem ler os arquivos ***.dcm*** e importar os dados em um objeto do tipo ***DICOM*** (figuras 9 e 10)[7].

```
Importando as bibliotecas para manipulação de imagens DICOM
```

```
In [1]: #Sistema operacional e manipulação de caminhos
        import os, glob
        #Biblioteca para matemática
        import numpy as np
        #Biblioteca com funções para leitura do DICOM
        import pydicom as dicom
        #Biblioteca para plotagem
        import matplotlib.pyplot as plt
        import matplotlib.image as mpimage
        #Biblioteca scientifica para manipulação de imagem - alias do scikit-ima
        from skimage import exposure
        import skimage.morphology as morp
        from skimage.filters import rank
        #Biblioteca para manipular as variáveis do tipo data
        from datetime import datetime
```

Figura 9. Importando bibliotecas para manipulação do código.

```
Carregar e ler imagens DICOM

In [2]: #Carrega a imagem na variável imagemDicom
        imagemDicom = dicom.read_file("/Users/viniciusmartinsvilela/PycharmProje

        Imprimir o metadado do arquivo DICOM

In [3]: print(imagemDicom)

        Dataset.file_meta -------------------------------
        (0002, 0000) File Meta Information Group Length  UL: 196
        (0002, 0001) File Meta Information Version       OB: b'\x00\x01'
        (0002, 0002) Media Storage SOP Class UID         UI: CT Image Storage
        (0002, 0003) Media Storage SOP Instance UID      UI: 1.3.12.2.1107.5.1.
        4.24623.4.0.5731957618322094
        (0002, 0010) Transfer Syntax UID                 UI: Explicit VR Little
        Endian
        (0002, 0012) Implementation Class UID            UI: 1.3.6.1.4.1.25403.
        1.1.1
        (0002, 0013) Implementation Version Name         SH: 'Dicom 0.1'
        (0002, 0016) Source Application Entity Title     AE: 'SERVERAE'
        -------------------------------------------------
        (0008, 0005) Specific Character Set              CS: 'ISO_IR 100'
        (0008, 0008) Image Type                          CS: ['ORIGINAL', 'PRIM
        ARY', 'AXIAL', 'CT_SOM5 SPI']
        (0008, 0016) SOP Class UID                       UI: CT Image Storage
        (0008, 0018) SOP Instance UID                    UI: 1.3.12.2.1107.5.1.
        4.24623.4.0.5731957618322094
```

Figura 10. Importando bibliotecas para manipulação do código.

Ao utilizar o comando ***print*** com a variável do tipo ***DICOM***, o kernel irá listar os atributos/variáveis do metadado. O metadado do arquivo ***DICOM*** nada mais é que um dicionário ou tupla que consiste de um vetor de atributos contendo chaves como índices (figura 11).

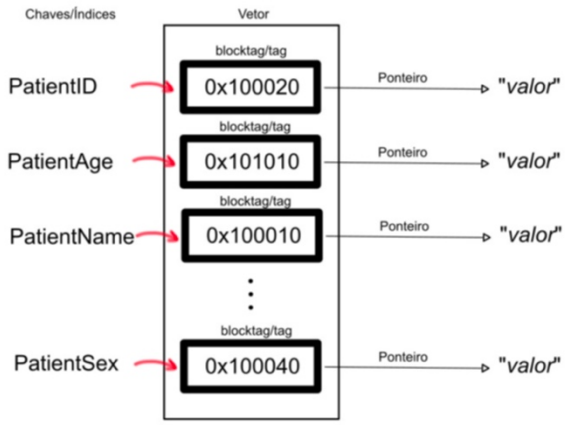


Figura 11. Desenho esquemático de um vetor indexado.

Existem vários atributos, estruturas (dicionário), funções e procedimentos definidos dentro da classe da instância ***dicom*** criada. É possível listar essas variáveis através do manual

da biblioteca - disponível em https://pypi.org -, ou utilizar as funções de edição e expansão disponíveis em algumas IDEs como exemplo o ***PyCharm*** (figura 12)[3].

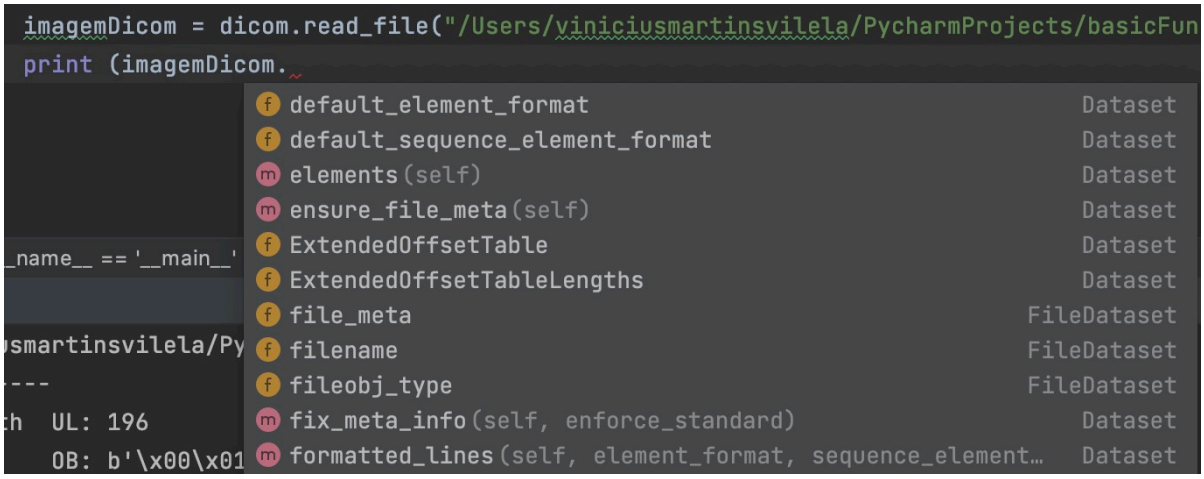


Figura 12. Ferramentas de expansão para verificar atributos e funções de uma classe ou instância.

Dessa maneira, funções ou atributos específicos podem ser invocados da biblioteca.

Após criar uma instância e ler um arquivo ***.dcm*** é possível acessar as chaves específicas, uma vez que a biblioteca ***pydicom*** consegue realizar buscas dentro do dicionário.

Os atributos correspondem a uma ***tag*** (grupo, elemento) que retorna o valor correspondente para a mesma, caso ela exista no conjunto de dados.[6]

O código abaixo exemplifica como podemos listar os índices/chaves, imprimir um atributo específico e, caso não tenha chave atribuída a uma *tag*, podemos selecionar uma por sua posição (em octetos), no vetor, apesar de não recomendado (figura 13).

```
In [4]: # Função que lista os seus índices/chaves
        print (imagemDicom.dir())
        # Imprimindo atributo específico
        print (imagemDicom.ProtocolName)
        # Imprimindo atributo com a tag
        print(imagemDicom[0x080060].value)

['AccessionNumber', 'AcquisitionDate', 'AcquisitionNumber', 'Acquisitio
nTime', 'BitsAllocated', 'BitsStored', 'BodyPartExamined', 'Columns', '
ContentDate', 'ContentTime', 'ConvolutionKernel', 'DateOfLastCalibratio
n', 'DistanceSourceToDetector', 'DistanceSourceToPatient', 'Exposure',
'ExposureTime', 'FilterType', 'FocalSpots', 'FrameOfReferenceUID', 'Gan
tryDetectorTilt', 'GeneratorPower', 'HighBit', 'ImageComments', 'ImageO
rientationPatient', 'ImagePositionPatient', 'ImageType', 'InstanceNumbe
r', 'InstitutionAddress', 'InstitutionName', 'KVP', 'Manufacturer', 'Ma
nufacturerModelName', 'Modality', 'PatientAge', 'PatientBirthDate', 'Pa
tientID', 'PatientName', 'PatientPosition', 'PatientSex', 'PhotometricI
nterpretation', 'PixelData', 'PixelRepresentation', 'PixelSpacing', 'Po
sitionReferenceIndicator', 'ProtocolName', 'ReconstructionDiameter', 'R
eferencedImageSequence', 'ReferringPhysicianName', 'RequestedProcedureD
escription', 'RescaleIntercept', 'RescaleSlope', 'RotationDirection', '
Rows', 'SOPClassUID', 'SOPInstanceUID', 'SamplesPerPixel', 'SeriesDate'
, 'SeriesDescription', 'SeriesInstanceUID', 'SeriesNumber', 'SeriesTime
', 'SliceLocation', 'SliceThickness', 'SoftwareVersions', 'SourceImageS
equence', 'SpecificCharacterSet', 'StationName', 'StudyDate', 'StudyDes
cription', 'StudyID', 'StudyInstanceUID', 'StudyTime', 'TableHeight', '
TimeOfLastCalibration', 'WindowCenter', 'WindowCenterWidthExplanation',
'WindowWidth', 'XRayTubeCurrent']
COLUNA LOMBAR ROTINA PADRAO
CT
```

Figura 13. Imprimindo atributos específicos.

A biblioteca também permite realizar modificações em um atributo privado tais como deletar, adicionar e/ou modificá-lo. Permite também anonimizar o conteúdo do metadado manualmente, apesar de existir uma biblioteca para isso – ***deid***. Isso pode ser útil para uma limpeza completa das informações, incluindo os parâmetros de aquisição e outros dados técnicos[8,9].

É possível buscar pelas chaves que se iniciam com um conjunto de caracteres ou os contenham em sua sequência (figura 14).

Existem alguns scripts prontos com código fonte aberto que permitem listar os atributos dos metadados em uma estrutura de árvore.[7]

```
        Para buscar todas os índices que se iniciam com uma string específica

In [5]: print (imagemDicom.dir("pa"))

['BodyPartExamined', 'DistanceSourceToPatient', 'ImageOrientationPatien
t', 'ImagePositionPatient', 'PatientAge', 'PatientBirthDate', 'PatientI
D', 'PatientName', 'PatientPosition', 'PatientSex', 'PixelSpacing']
```

Figura 14. Usando a função dir para listar que contenham um conjunto de caracteres

Cada atributo do arquivo DICOM pode ser alterado, em duas simples etapas:

> dicom.Atributo = 'novo valor'

> dicom.save_as('caminho para arquivo')

4.5.2. Anonimização

A política de proteção de dados é um tema amplamente discutido no meio científico. Apesar das recorrentes discussões, a LGDP - Lei nº 13.709/2018 é a legislação brasileira que regula as atividades de tratamento de dados pessoais. Essa lei altera os artigos 7º e 16º do Marco Civil da Internet.

Nos Estados Unidos a HIPPA consolidou como obrigação legal a anonimização e proteção dos dados dos usuários, estabelecendo fluxos para utilização de informações tanto no âmbito comercial ou científico. Dessa maneira, é necessário anonimizar os atributos que constituem o PHI. Diferentes protocolos e guias para os níveis de anonimização ou desidentificação podem ser encontrados no site do departamento de saúde americano (figura 15) [8].

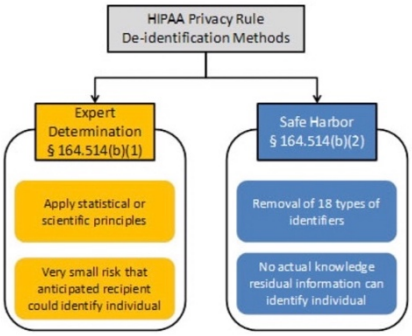


Figura 15. Diferentes métodos de deidentificação/anonimização de acordo com as regras definidas pelo HIPPA.

Um método não muito prático consiste em deletar os atributos e salvar sobre o arquivo original (figura 16). Também é possível salvar as variáreis em arquivos encriptados para eventual necessidade de reidentificação.

```
Deletando elementos DICOM individualmente
```

```
In [ ]: del imagemDicom.PatientName
        del imagemDicom.AccessionNumber
        del imagemDicom.PatientID
        del imagemDicom.PatientBirthDate
        imagemDicom.save_as("/Users/viniciusmartinsvilela/PycharmProjects/basicF
```

Figura 16. Deleção de atributos de arquivo individual.

Seguindo o mesmo princípio, é possível criar uma iteração deletando os itens de cada arquivo dentro de uma pasta ou subpasta, recursivamente (figura 17)[9].

```
Deletando atributos de maneira recursiva dentro de uma pasta específica
```

```
In [ ]: # Armazena todos os caminhos de arquivos dcm para uso na iteração
for arquivos in glob.iglob('/Users/viniciusmartinsvilela/PycharmProjects
                           recursive=True):
    # Carrega a imagem na variável imagemDicom
    imagemDicom = dicom.read_file(arquivos, force=True)

    # Armazena o nome dos folder que as imagens estão
    nomeDoFolder = os.path.basename(os.path.dirname(os.path.dirname(arqu

    # Deleta atributos
    del imagemDicom.PatientName
    del imagemDicom.AccessionNumber
    del imagemDicom.PatientID
    del imagemDicom.PatientBirthDate

    # overwrite the original file
    dicom.write_file(arquivos, imagemDicom)

    # output progress to the screen
    print(arquivos)
```

Figura 17. Deleção de atributos dos arquivos de uma pasta.

Existe a preocupação com imagens digitalizadas que podem conter elementos escritos dentro do próprio vetor de imagem. Dessa forma elementos de texto podem ser digitalizados junto com os conteúdos da imagem, não sendo acessíveis com chaves ou índices.

Existem scripts específicos para identificação de escrita em imagens, alguns deles por ***OpenCV***, os quais podem ser úteis em imagens CR e em imagens digitalizadas de ultrassonografia.

Deid exige configuração e fluxo próprio – a documentação pode ser encontrada na sua página do ***GitHub***[9].

4.5.3. Visualizar imagens com Python

A biblioteca ***matplotlib*** oferece uma série de maneiras de visualizar as imagens dentro do arquivo ***.dcm***. Os dados da imagem podem ser acessados pelo vetor de ponteiros em ***pydicom.pixel_array*** (figuras 18 e 19)[10].

```
In [9]: print (imagemDicom.pixel_array)

[[ 79  69  65 ...  55  61  64]
 [ 82  68  56 ...  26  35  41]
 [ 69  58  50 ...   6  15  21]
 ...
 [120 127 133 ... 131 133 136]
 [124 128 131 ... 128 132 138]
 [124 126 128 ... 123 130 137]]
```

Figura 18. Vetor com os valores de Pixel.

```
In [ ]: Plotar imagens arquivos

In [8]: #Pré configura o Matplotlib para plotar as imagens em um padrão de 6 x 6
        plt.figure(figsize = (6,6), dpi=100)
        #Plota a imagem usando a escala de cinza
        plt.imshow(imagemDicom.pixel_array, cmap=plt.cm.gray)

Out[8]: <matplotlib.image.AxesImage at 0x11d618130>
```

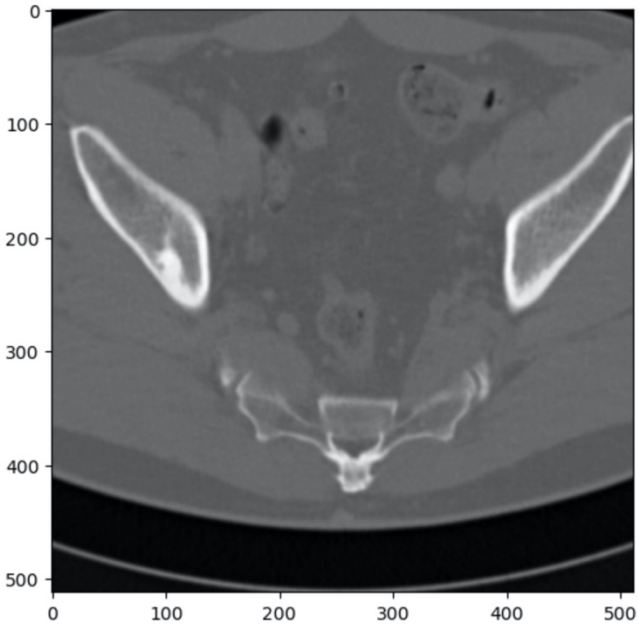


Figura 19. Plotando imagem com matplotlib.

4.5.4. Salvando imagens com formato diferente

Existe conversão habilitada para formatos ***.JPG***, ***.PNG*** ou ***.TIFF***. Ao enviar uma imagem para publicação, os periódicos geralmente solicitam que o arquivo de imagem seja enviado no formato ***.TIFF*** (figura 20).

Salvando em outros formatos

```
In [ ]: imagemDicom.imsave('exemplo.jpg', imagemDicom.pixel_array, cmap=plt.cm.g
        imagemDicom.imsave('exemplo.png', imagemDicom.pixel_array, cmap=plt.cm.g
        imagemDicom.imsave('exemplo.tif', imagemDicom.pixel_array, cmap=plt.cm.g
```

Figura 20. Salvando em outros formatos.

4.5.5. Gerando histograma

Gerar o histograma costuma ser parte de pré processamento e pode ser usado em regressões ou outros algoritmos (Figura 21).[10,11,12]

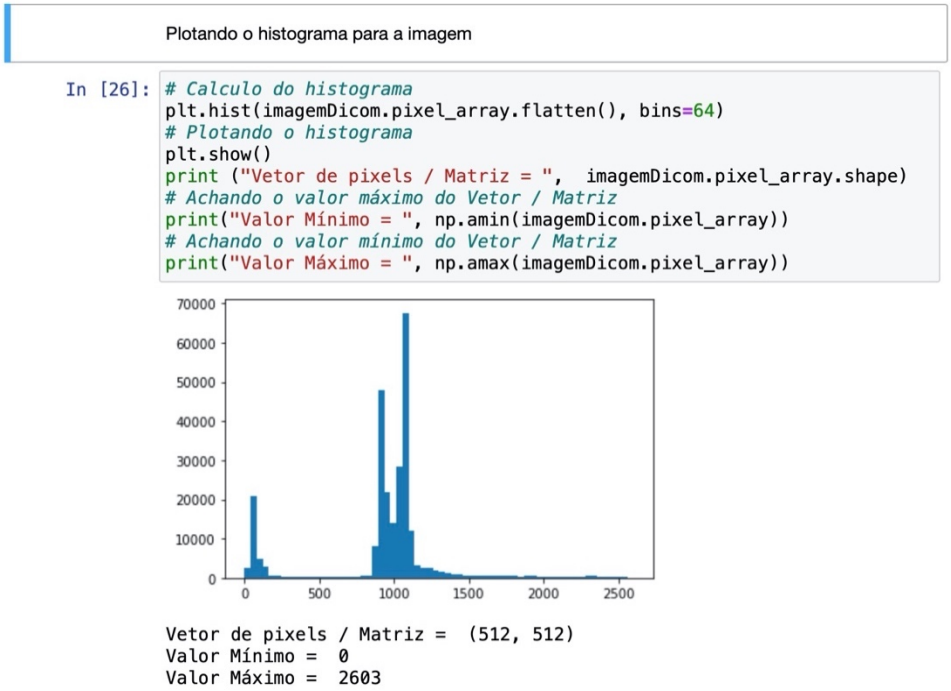


Figura 21. Plotando histograma.

4.5.6. Transformando para escala de Hounsfield

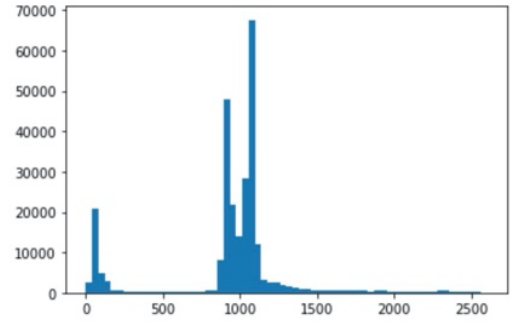
A escala da unidade de Housfield (UH) é uma transformação de um coeficiente de atenuação linear para uma escala determinada adimensional. O coeficiente atenuação linear (μ) da água é utilizado como padrão – 0 UH. Os demais valores de atenuação são distribuídos em uma escala que geralmente possui valores máximos de 3500 e mínimos de -1000.

A atenuação de um dado material é expressa por:

$$UH_{material} = 1000 \times \frac{\mu_{material} - \mu_{água}}{\mu_{água}}$$

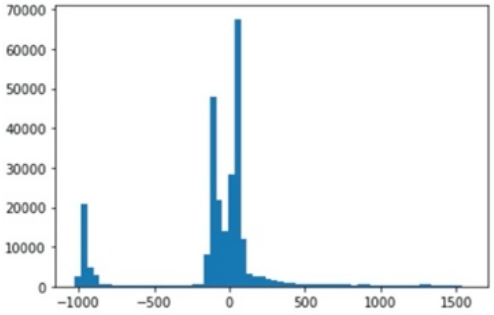
Dessa maneira conseguimos utilizar os atributos ***RescaleIntercept*** e ***RescaleSlope*** para obter a o vetor de imagem dentro da escala de Hounsfiled (figura 22) [11].

```
In [48]: # Calculo do histograma
         plt.hist(imagemDicom.pixel_array.flatten(), bins=64)
         # Plotando o histograma
         plt.show()
         print ("Vetor de pixels / Matriz = ",  imagemDicom.pixel_array.shape)
         # Achando o valor máximo do Vetor / Matriz
         print("Valor Mínimo = ", np.amin(imagemDicom.pixel_array))
         # Achando o valor mínimo do Vetor / Matriz
         print("Valor Máximo = ", np.amax(imagemDicom.pixel_array))
```




```
Vetor de pixels / Matriz =  (512, 512)
Valor Mínimo =  0
Valor Máximo =  2603
```

```
In [49]: intercept = imagemDicom.RescaleIntercept
         slope = imagemDicom.RescaleSlope
         uh_imagem = imagemDicom.pixel_array * slope + intercept

         # Calculo do histograma
         plt.hist(uh_imagem.flatten(), bins=64)
         # Plotando o histograma
         plt.show()
         print ("Vetor de pixels / Matriz = ",  uh_imagem.shape)
         # Achando o valor máximo do Vetor / Matriz
         print("Valor Mínimo = ", np.amin(uh_imagem))
         # Achando o valor mínimo do Vetor / Matriz
         print("Valor Máximo = ", np.amax(uh_imagem))
```




```
Vetor de pixels / Matriz =  (512, 512)
Valor Mínimo =  -1024.0
Valor Máximo =  1579.0
```

Figura 22. Adequando a matriz para a escala de Hounsfiled.

4.5.7. Controle de janela, nível e filtro

O controle de janela e nível é realizado nos visualizadores *DICOM*, geralmente com uma ferramenta de tração/"*drag and drop*".

Os filtros de imagem para tecidos específicos podem ser aplicados através da definição dos valores máximo e mínimo do pixel a partir do cálculo do centro e largura da janela desejados.

Sendo assim:

img_min = centroJanela - larguraJanela // 2

img_max = centroJanela + larguraJanela // 2

Para todos os valores abaixo de **img_min,** o valor mínimo é atribuído ao píxel. Assim como para todos os valores acima de **img_max,** o valor máximo é atribuído. Como exemplo temos a aplicação de uma janela óssea (Figura 23).

```
In [54]: def filtro_imagem(imagem, centroJanela, larguraJanela):
             img_min = centroJanela - larguraJanela // 2
             img_max = centroJanela + larguraJanela // 2
             imagemFiltrada= imagem.copy()
             imagemFiltrada[imagemFiltrada < img_min] = img_min
             imagemFiltrada[imagemFiltrada > img_max] = img_max

             return imagemFiltrada
         #gerando imagem com janela de osso
         plt.imshow(filtro_imagem(uh_imagem, 400, 1000), cmap=plt.cm.gray)

Out[54]: <matplotlib.image.AxesImage at 0x12179fb20>
```

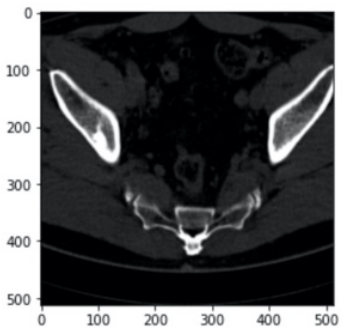
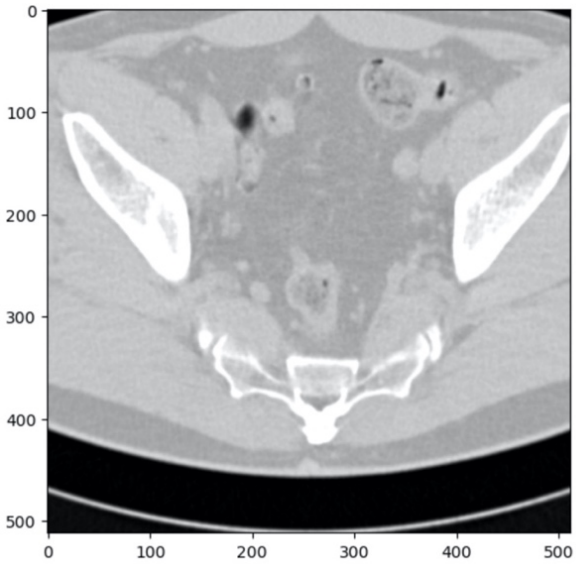


Figura 23. Filtro ósseo.

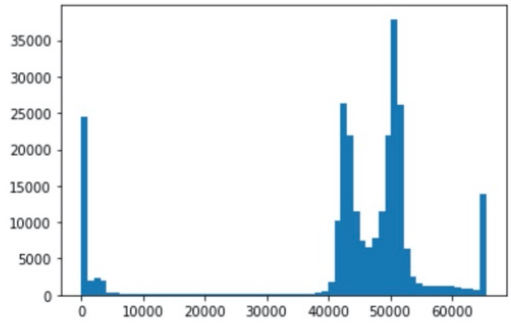
O controle da janela também pode ser realizado através do ***exposure.rescale_intensity*** disponível na biblioteca do ***scikit-image***[12]. A função permite ajustar o conjunto de valores máximos e mínimos das janelas a partir dos valores de pixels existentes. (figuras 24, 25 e 26).

Controle de janela e nível

```
In [50]: plt.figure(figsize = (6,6), dpi=100)
         p_limiteInferior, p_limiteSuperior = np.percentile(imagemDicom.pixel_arr
         img_imagemManipulada = exposure.rescale_intensity(imagemDicom.pixel_arra
         figure = plt.imshow(img_imagemManipulada, cmap=plt.cm.gray)
```




```
In [51]: plt.hist(img_imagemManipulada.flatten(), bins=64)
         plt.show()
         print("Percentil inferior = ", p_limiteInferior)
         print("Percentil superior = ", p_limiteSuperior)
```




```
Percentil inferior =  65.0
Percentil superior =  1368.0
```

Window edge	percentile	pixel value	color
lower	20.0	3.0	blackest black
higher	99.5	253.0	whitest white

Figura 24, 25 e 26. Encurtando a janela com os valores máximo e mínimo dentro dos percentis 5 e 95. Histograma da janela re-escalada. Exemplo de valores para pixel mais escuro para percentil 20 e o valor do pixel mais branco para o percentil 99,5.

O A*pêndice 2* apresenta um código baseado na *referência 5*, que permite a modificação dinâmica da janela (figura 27).

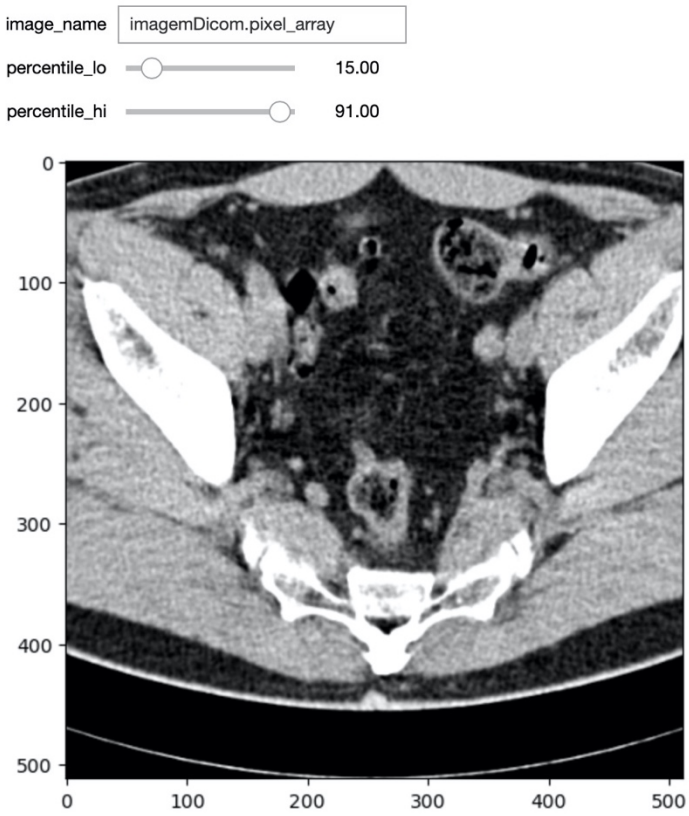
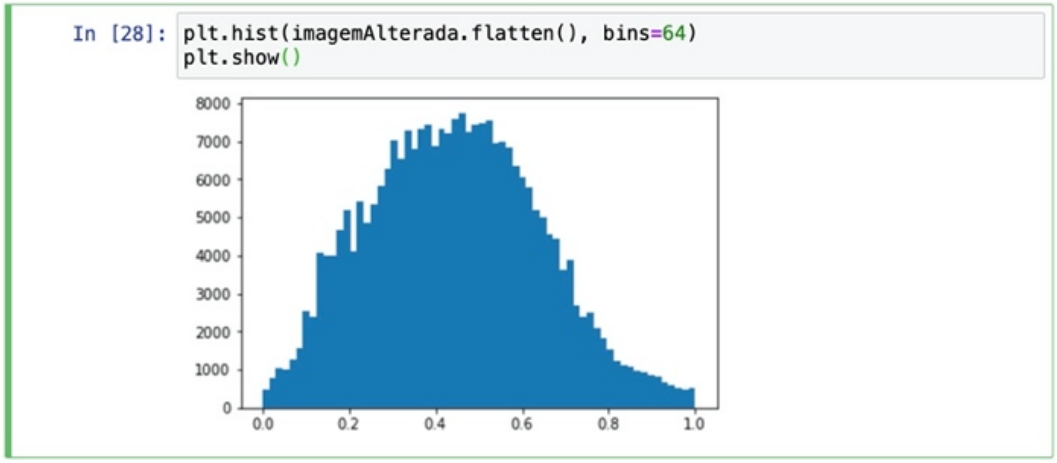


Figura 27. Controle dinâmico do contraste.

Existe também a possibilidade de equalização do histograma (figura 28 e 29).[12]



```
In [38]: plt.figure(figsize = (6,6), dpi=100)

         imagemAlterada = exposure.equalize_adapthist(imagemDicom.pixel_array, cl
         figure = plt.imshow(imagemAlterada, cmap=plt.cm.gray)
         plt.colorbar()

Out[38]: <matplotlib.colorbar.Colorbar at 0x11e6621c0>
```

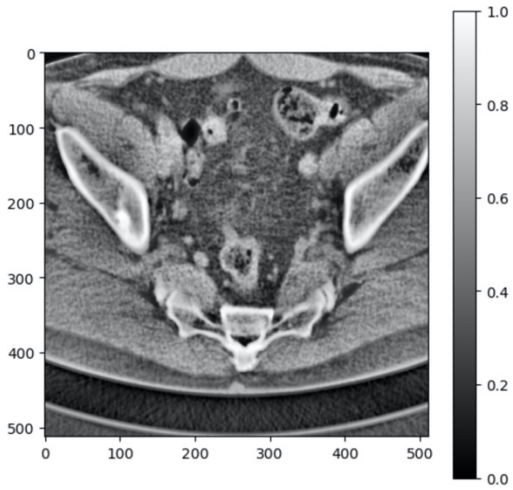


Figura 28 e 29. Equalização dos valores e seu reflexo no histograma.

Outras funções podem ser facilmente implementadas com as disponibilizadas pela biblioteca ***scikit-image*** e ***exposure***.

4.5.8. *Colormap* e Inversão dos valores da imagem

É possível selecionar diferentes espaços de cor durante a plotagem da imagem (figura 30), [Apêndice 3].

```
Plotando com diferente colormap

In [38]: plt.figure(figsize = (6,6), dpi=100)

         figure = plt.imshow(imagemDicom.pixel_array, cmap='jet')
         plt.colorbar()

Out[38]: <matplotlib.colorbar.Colorbar at 0x120d69a00>
```

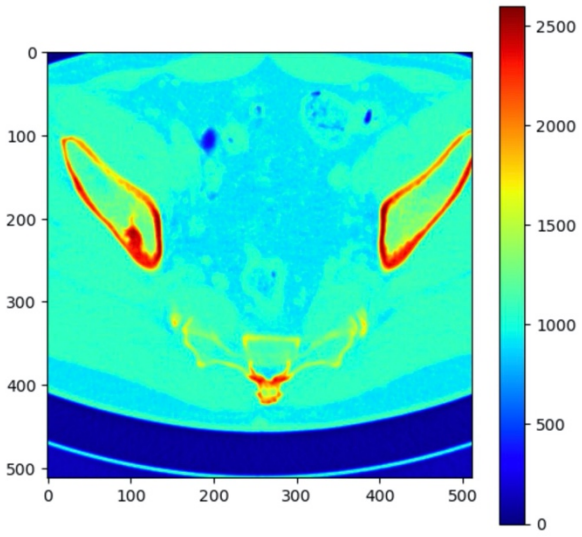


Figura 30. Plotando no *colormap jet*.

Geralmente, o índice ***PhotometricInterpretation*** do metadado é responsável por armazenar o esquema de cores que foi padronizado para a sua análise (***colorspace*** ou ***colormap***). Em casos específicos pode ser útil um esquema de cor diferente, especialmente para mamografias e radiografias.

O processo de inverter a escala ou negativar a imagem pode ser extremamente útil na identificação de nódulos ou alterações corticais[13].

Uma maneira muito simples de inverter a escala de cinza é utilizar o ***colormap=gist_yarg*** disponível na biblioteca ***Matplotlib***, a qual plota com o inverso da escala de cinza (figura 31).

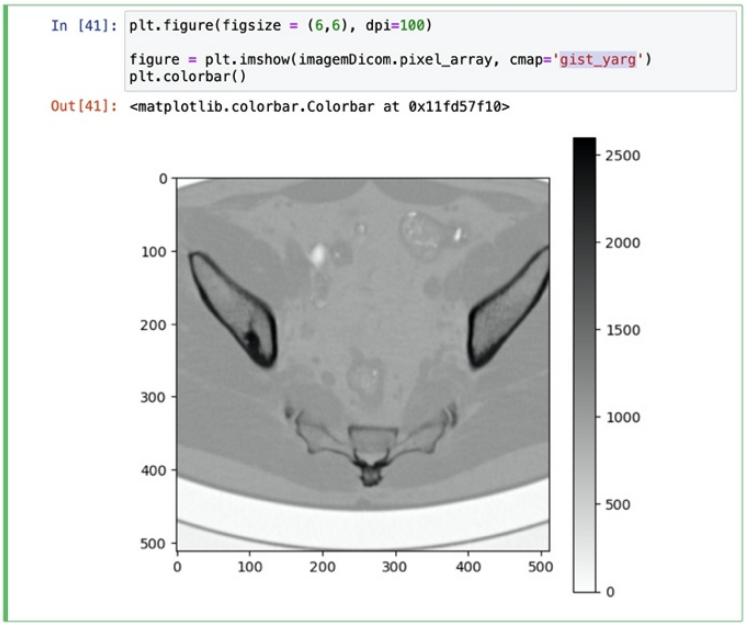


Figura 31. Fazendo inversão dos valores da escala de cinza com o ***colormap gist_yarg*.**

Porém para realizar a inversão real e armazenar no vetor de imagem, é necessário manipular os valores para que os pixels mais escuros sejam os mais claros e vice-versa.

Para tanto é necessário calcular o intervalo a partir dos valores máximo e mínimo dos pixels e subtrair o valor do pixel atual por esse: $P_{invertido} = Intervalo - Valor_{pixel}$ (figuras 32).

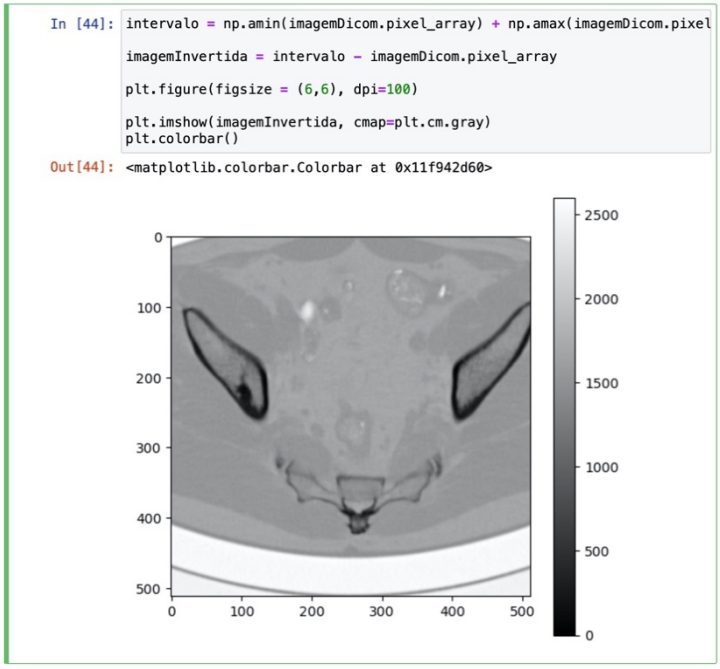


Figura 32. Inversão do valor de pixel com modificação no vetor da imagem.

4.5.9. Transformações: rotacionamento, espelhamento horizontal e vertical da imagem ...

Podemos usar a biblioteca ***Numpy*** para realizar inversões horizontal e vertical (figura 32).

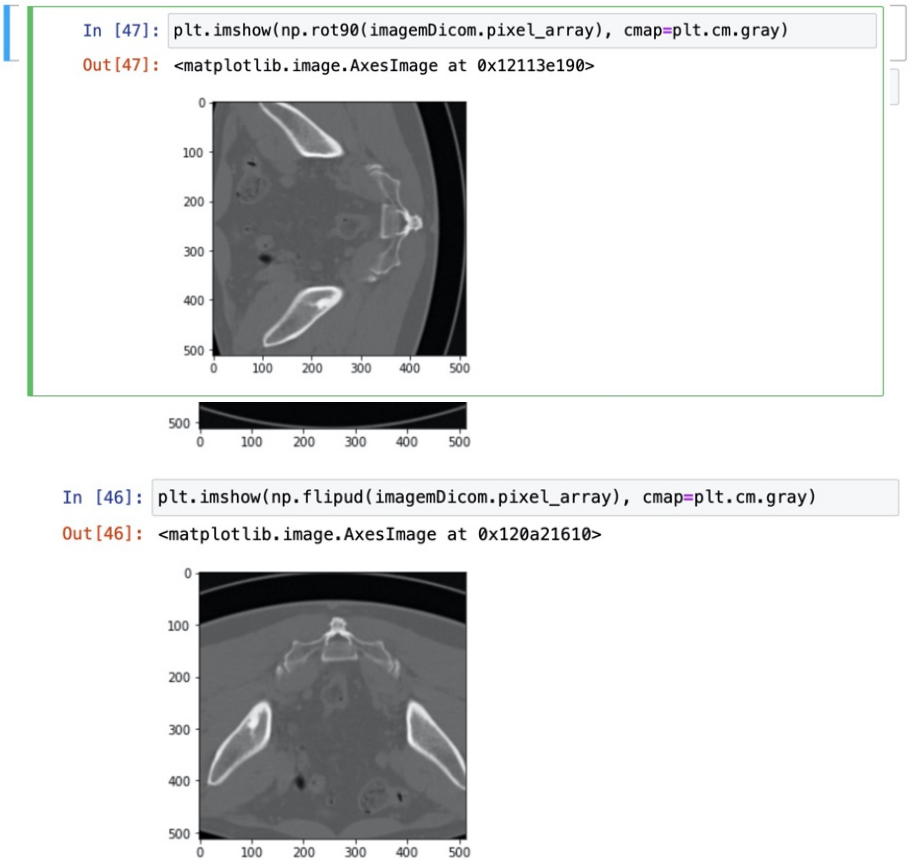


Figura 33. ***Flip*** horizontal e vertical, e rotação em 90 graus usando a biblioteca **Numpy**.

Uma série de filtros morfológicos está disponível na biblioteca de ***scikit-image,*** podendo ser utilizados para remoção óssea, retirada de ruido externo, etc.

Essas operações de rotação, translação e transformação são modelos algébricos de multiplicação de matrizes, como exemplo da figura 34:

Rotação em torno do eixo x, y

$$\mathbf{R_x}(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotação em torno de um eixo arbitrário

$$\begin{bmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & -t'_x \\ 0 & 1 & 0 & -t'_y \\ 0 & 0 & 1 & -t'_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix}$$

Escalonamento

$$\mathbf{S}(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figura 34. Alguns modelos algébricos para transformações de matrizes.

A biblioteca ***OpenCV*** possui funções otimizadas em alto nível para facilitar a implementação dessas transformações, dentre elas ***cv2.getRotationMatrix2D*** e ***cv2.flip***, usadas em conjunto com a ***cv2.warpAffine***. A biblioteca do ***OpenCV*** é extensa e foge ao escopo do trabalho, no entanto é desejável que o usuário tenha um conhecimento básico de suas funcionalidades, principalmente para entender algumas operações dentro do pré-processamento[14].

4.5.10. Trabalhando com imagens sequenciais

As operações básicas podem ser realizadas em exames sequenciais. Um processo rudimentar e básico consiste em ler os arquivos ***.dcm*** de um caminho específico, carregar os seus múltiplos vetores que contêm valores ***pixel_array*** para um vetor ***Numpy,*** e realizar a transformação para a escala de Hounsfield (figura 35)[4,10].

```python
def load_scan(path):
    slices = [dicom.read_file(path + '/' + s) for s in os.listdir(path)]
    slices.sort(key=lambda x: int(x.InstanceNumber))
    return slices
```

```python
def get_pixels_hu(scans):
    image = np.stack([s.pixel_array for s in scans])
    # Convert to int16 (from sometimes int16),
    # should be possible as values should always be low enough (<32k)
    image = image.astype(np.int16)

    # Set outside-of-scan pixels to 1
    # The intercept is usually -1024, so air is approximately 0
    image[image == -2000] = 0

    # Convert to Hounsfield units (HU)
    intercept = scans[0].RescaleIntercept
    slope = scans[0].RescaleSlope

    if slope != 1:
        image = slope * image.astype(np.float64)
        image = image.astype(np.int16)

    image += np.int16(intercept)

    return np.array(image, dtype=np.int16)
```

```python
id = 0
data_path = "/Users/viniciusmartinsvilela/PycharmProjects/basicFunctions/CT_Coluna_Lombar"
output_path = working_path = "/Users/viniciusmartinsvilela/PycharmProjects/basicFunctions/exameOutput/"
```

```python
patient = load_scan(data_path)
imgs = get_pixels_hu(patient)

np.save(output_path + "fullimages_%d.npy" % (id), imgs)

file_used=output_path+"fullimages_%d.npy" % id
imgs_to_process = np.load(file_used).astype(np.float64)
```

Figura 35. Carregando as múltiplas matrizes em vetor de matrizes em *NumPy*.

A estrutura de ***vetor*** disponível na biblioteca ***NumPy*** possui melhor desempenho e menor tamanho que o tipo ***lista*** nativo do ***Python***. A listas em ***python*** possuem atributos extras de controle e descrição, por exemplo: 1 byte de atributo tamanho, 8 bytes de ponteiros de referência, 8 bytes de tipo de objeto, 8 bytes de valores. O vetor ***NumPy*** armazena apenas os valores. Se determinarmos um vetor do tipo int32 teremos apenas 4 bytes alocados e se declararmos um vetor do tipo float64 teremos 8 bytes – o equivalente a menos de um terço da lista comum (figura 36).

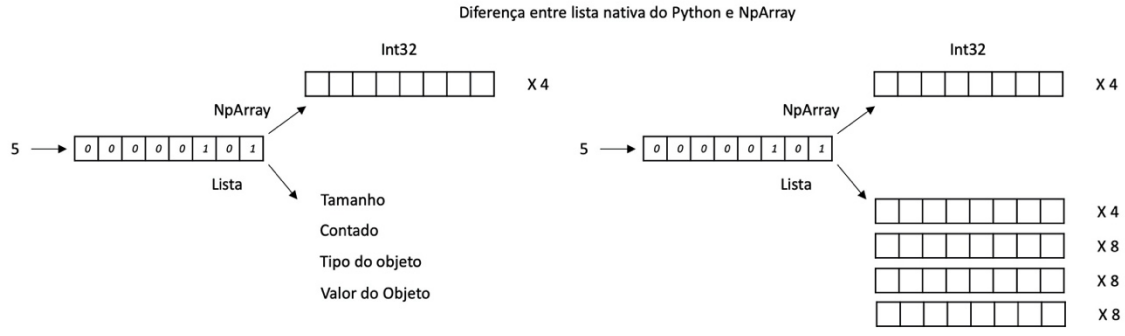


Figura 36. Diferença entre a estrutura dos vetores do ***Python*** e da biblioteca ***numpy***.

Outra vantagem da manipulação de dados com a biblioteca ***NumPy*** é a alocação contígua de memória. Os valores dos vetores são armazenados contiguamente, o que facilita o acesso ao bloco de dados, tornando-o mais rápido e aprimorando o uso do *cache* (figura 37*)*.

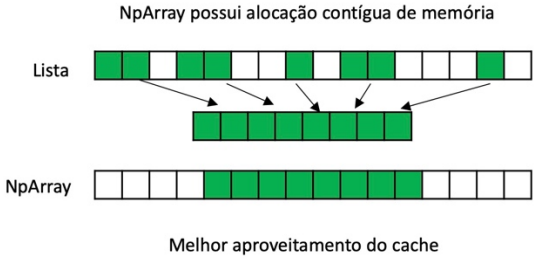


Figura 37. Diferença de alocação das memórias. Os vetores em ***NumPy*** possuem alocação contígua dos dados.

NumPy possui aplicações matemáticas, substituindo algumas bibliotecas do MATLAB, visualização e plotagem/reconstrução, recurso básico e processo interno (***backend***) para execução de outras bibliotecas como **Pandas, Connect 4, Digital Photografy**. Por fim, é amplamente utilizado na automação e aprendizado de máquinas.

4.5.11. Plotando a imagem em diferentes planos.

Após criado o vetor de matrizes, podemos representá-lo em um espaço tridimensional, no qual cada unidade cúbica representa os voxels obtidos pela aquisição sequencial. Dessa maneira, é possível plotar diferentes planos selecionando matrizes com intervalos específicos para os eixos x, y e z -altura, largura e profundidade (figura 38) [4,10].

In [21]:
```
def display_views(image):
    axial_image = image[110, :, :] # Axis 0
    sagital_image = image[:, :, 144] # Axis 2
    coronal_image = image[:, 144, :] # Axis 1

    plt.figure(figsize=(20, 10))
    plt.style.use('grayscale')

    plt.subplot(141)
    plt.imshow(axial_image)
    plt.title('Axial Plane')
    plt.axis('off')

    plt.subplot(142)
    plt.imshow(sagital_image)
    plt.title('Sagital Plane')
    plt.axis('off')

    plt.subplot(143)
    plt.imshow(coronal_image)
    plt.title('Coronal Plane')
    plt.axis('off')


display_views(imgs_to_process)
```

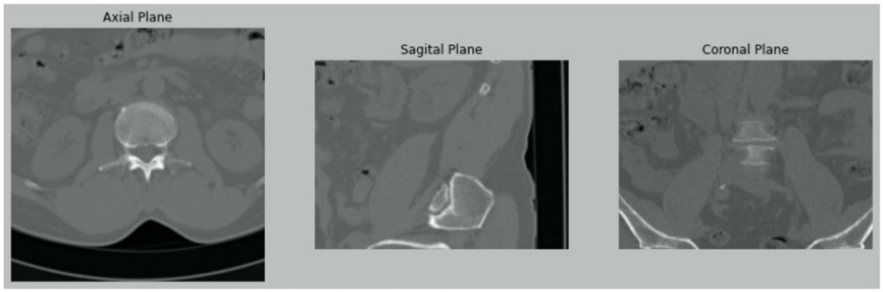


Figura 38. Selecionando diferentes planos para a nossa representação espacial.

4.5.12. Redimensionamento

Estudos com mesma matriz podem ter valores espaciais diferentes. Não é incomum exames terem diferentes protocolos e possuírem diferentes padrões de informação espacial. Esse fator é extremamente dependente do ***field of view*** (***FOV***), por exemplo: imagens com matriz de 512 x 512 podem possuir tamanho de voxel de 1 x 1 mm^2, enquanto outro estudo pode possuir tamanho de 0.6 x 0.5 mm^2.

O redimensionamento possibilita padronizar as informações espaciais para diferentes imagens. Isso pode melhorar a acurácia da rede neural, uma vez que o kernel irá receber informações dentro do mesmo espaço (figura 39)[14].

```
In [7]:
from scipy import ndimage
temp = []
temp.append(patient[0].SliceThickness)
temp.append(patient[0].PixelSpacing[0])
temp.append(patient[0].PixelSpacing[1])
print(temp)

spacing = np.array(temp, dtype=float)
diferenca = np.array(list(spacing))
print(diferenca)

new_spacing = [1, 1, 1]
resize_factor = spacing / new_spacing
print(resize_factor)

new_real_shape = imgs_to_process.shape * resize_factor
print(new_real_shape)

new_shape = np.round(new_real_shape)
print(new_shape)

real_resize_factor = new_shape / imgs_to_process.shape
print(real_resize_factor)

new_spacing = spacing / real_resize_factor
print(new_spacing)

image = ndimage.interpolation.zoom(imgs_to_process, real_resize_factor)

print(image.shape)

plt.imshow(image[1], cmap=plt.cm.gray)
```

```
['3.0', '0.45507813', '0.45507813']
[3.         0.45507813 0.45507813]
[3.         0.45507813 0.45507813]
[1140.          233.00000256  233.00000256]
[1140.  233.  233.]
[3.         0.45507812 0.45507812]
[1.         1.00000001 1.00000001]
(1140, 233, 233)
```

Figura 39. Redimensionando manualmente uma imagem com espaço de pixel de 0,45 x 0,45 mm^2 de espessura para o espaço de 1 x 1 mm^2. O vetor de matrizes com formato 512 x 512 unidades foi transformado para 233 x 233.

É necessário observar que para o redimensionamento espacial ficamos suscetíveis a distorções na imagem, tornando-a encurtada ou alongada no espaço. Uma maneira mais fácil de realizar o redimensionamento é utilizando-se das funções disponíveis no OpenCV, entre elas a ***resize.***

4.5.13. Plotando 3D

A imagem volumétrica, redimensionada e alocada em um vetor de matizes pode ser reconstruída e apresentada em formato 3D utilizando a biblioteca ***Plotly*** e ***Scikit-Image***. A função ***marching_cubes*** permite encontrar superfícies em dados volumétricos. Sendo assim, podemos utilizá-la para criar a malha (figura 40) [4,10].

```
In [12]:
def make_mesh(image, threshold=-300, step_size=1):

    print ("Transposing surface")
    p = image.transpose(2,1,0)

    print ("Calculating surface")
    verts, faces, norm, val = measure.marching_cubes(p, threshold, step_size=step_size, allow_degenerate=True)
    return verts, faces
```

```
In [13]:
make_mesh(image)
```

```
Transposing surface
Calculating surface
```

```
Out[13]: (array([[0.0000000e+00, 0.0000000e+00, 7.0189069e+02],
        [1.3836935e-01, 0.0000000e+00, 7.0200000e+02],
        [0.0000000e+00, 8.5163340e-02, 7.0200000e+02],
        ...,
        [2.3200000e+02, 2.1453778e+02, 1.1370000e+03],
        [2.3200000e+02, 2.1453362e+02, 1.1380000e+03],
        [2.3200000e+02, 2.1453229e+02, 1.1390000e+03]], dtype=float32),
 array([[      2,       1,       0],
        [      1,       2,       3],
        [      1,       3,       4],
        ...,
        [1307537, 1314301, 1307538],
        [1307538, 1314301, 1314303],
        [1307538, 1314303, 1307475]], dtype=int32))
```

Figura 40. Criando a malha a partir dos dados volumétricos.

A malha pode ser plotada utilizando a visualização 3D interativa do ***Plotly*** (figura 41).

```
In [12]:
def plotly_3d(verts, faces):
    x,y,z = zip(*verts)

    print ("Drawing")

    # Make the colormap single color since the axes are positional not intensity.
#    colormap=['rgb(255,105,180)','rgb(255,255,51)','rgb(0,191,255)']
    colormap=['rgb(236, 236, 212)','rgb(236, 236, 212)']

    fig = FF.create_trisurf(x=x,
                            y=y,
                            z=z,
                            plot_edges=False,
                            colormap=colormap,
                            simplices=faces,
                            backgroundcolor='rgb(64, 64, 64)',
                            title="Interactive Visualization")
    iplot(fig)
```

```
In [14]:
v, f = make_mesh(image, 360, 2)
plotly_3d(v, f)
```

```
Transposing surface
Calculating surface
Drawing
```

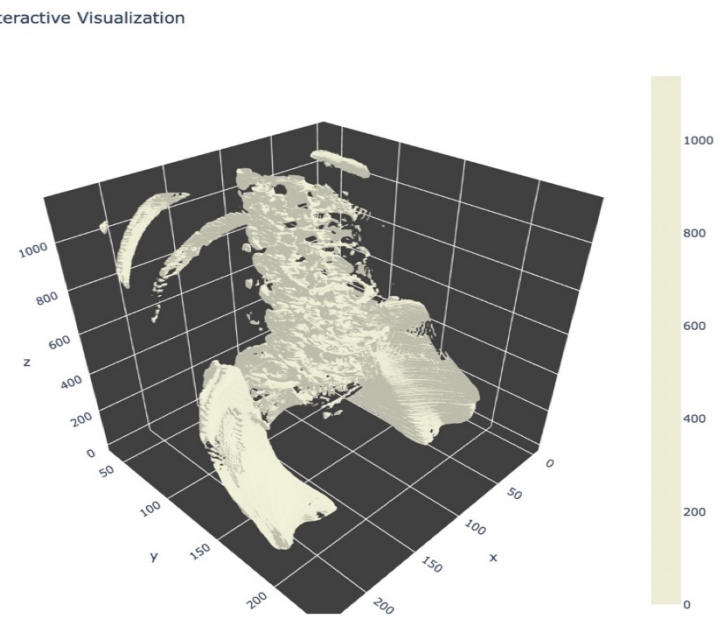


Figura 41. Criando a malha a partir dos dados volumétricos.

4.5.14. Segmentação

A segmentação pode ser realizada manualmente por meio de softwares ou utilizando-se do conjunto de bibliotecas disponíveis para ciência de dados ou linguagem de máquina. Normalizar a imagem é um passo essencial, uma vez que grande parte dos algoritmos disponibilizados trabalha melhor com valores normalizados (figura 42)[17].

```
def make_mask(img, display=False):
    row_size= img.shape[0]
#    print(row_size)
    col_size = img.shape[1]
#    print(col_size)
    mean = np.mean(img)
    print("Média:",mean)
    std = np.std(img)
    print("Desvio:",std)

    img = img-mean
    img = img/std

    #Normalizou a imagem
#    plt.imshow(img[50], cmap=plt.cm.gray)
#    print (img.shape)
    # Tenta achar a média do valor de pixel pela área desejada, no nosso caso o tecido adiposo peritoneal.
    middle = img[90:125,80:120]

#   plt.imshow(middle[50], cmap=plt.cm.gray)

    #mean = np.mean(middle)
    mean = 0
    print("Nova média:", mean)
    max = np.max(img)
    print("Máximo",max)
    min = np.min(img)
    print("Mínimo",min)
    # To improve threshold finding, I'm moving the
    # underflow and overflow on the pixel spectrum
    img[img==max]=mean
    img[img==min]=mean
```

Figura 42. Normalizando os dados e calculando a média para uma região de interesse da imagem.

Em nosso exemplo podemos utilizar um algoritmo de aprendizado não supervisionado baseado no processo de ***clusterização***, disponível na biblioteca ***Scikit-Learn***. De maneira simplificada, podemos falar que o algoritmo identifica o número k de centroides e, em seguida, aloca cada ponto de dados para o cluster mais próximo, mantendo os centroides tão pequenos quanto possível (Figura 43).

```
    #
    # Usando K-means para separar tecido adiposo de osso e músculo)
    #
    kmeans = KMeans(n_clusters=2).fit(np.reshape(middle,[np.prod(middle.shape),1]))
    centers = sorted(kmeans.cluster_centers_.flatten())
    threshold = np.mean(centers)
    thresh_img = np.where(img<threshold,1.0,0.0)  # threshold the image

    #plt.imshow(thresh_img[50], cmap=plt.cm.gray)
    eroded = morphology.erosion(thresh_img,np.ones([3,3]))
    #plt.imshow(eroded, cmap=plt.cm.gray)
    dilation = morphology.dilation(eroded,np.ones([8,8]))
    #plt.imshow(dilation, cmap=plt.cm.gray)

    labels = measure.label(dilation) # Different labels are displayed in different colors
    label_vals = np.unique(labels)
    regions = measure.regionprops(labels)
    good_labels = []

    for prop in regions:
        B = prop.bbox
        if B[2]-B[0]<row_size/10*9 and B[3]-B[1]<col_size/10*9 and B[0]>row_size/5 and B[2]<col_size/5*4:
            good_labels.append(prop.label)
    mask = np.ndarray([row_size,col_size],dtype=np.int8)
    mask[:] = 0

    #
    #  After just the lungs are left, we do another large dilation
    #  in order to fill in and out the lung mask
    #
```

Figura 43. Utilizando um algoritmo de aprendizado não supervisionado para segmentar a imagem.

Também podemos usar as funções ***erosion*** e ***dilation,*** da biblioteca ***Scikit-Image***, para controlar a melhor definição das bordas da imagem.

Erosion**:** identifica os pixels das regiões de borda por marcador, que pode ser redondo, quadrado ou customizado, e adiciona ao seu valor o mínimo valor de seus vizinhos. Resumindo, remove ilhas e pequenos objetos, afilando ou removendo as bordas.

Dilation**:** identifica os pixels das regiões de borda por marcador que pode ser redondo, quadrado ou customizado, e adiciona ao seu valor o máximo valor de seus vizinhos. Resumindo, espessa, torna objetos mais visíveis ou preenche pequenos espaços.

Uma série de filtros morfológicos estão disponíveis na biblioteca de ***scikit-image***.[14]

Após identificar os segmentos com rótulos específicos, podemos criar máscaras para aplicar na nossa imagem original, removendo ou deixando apenas a região desejada (figura 44).

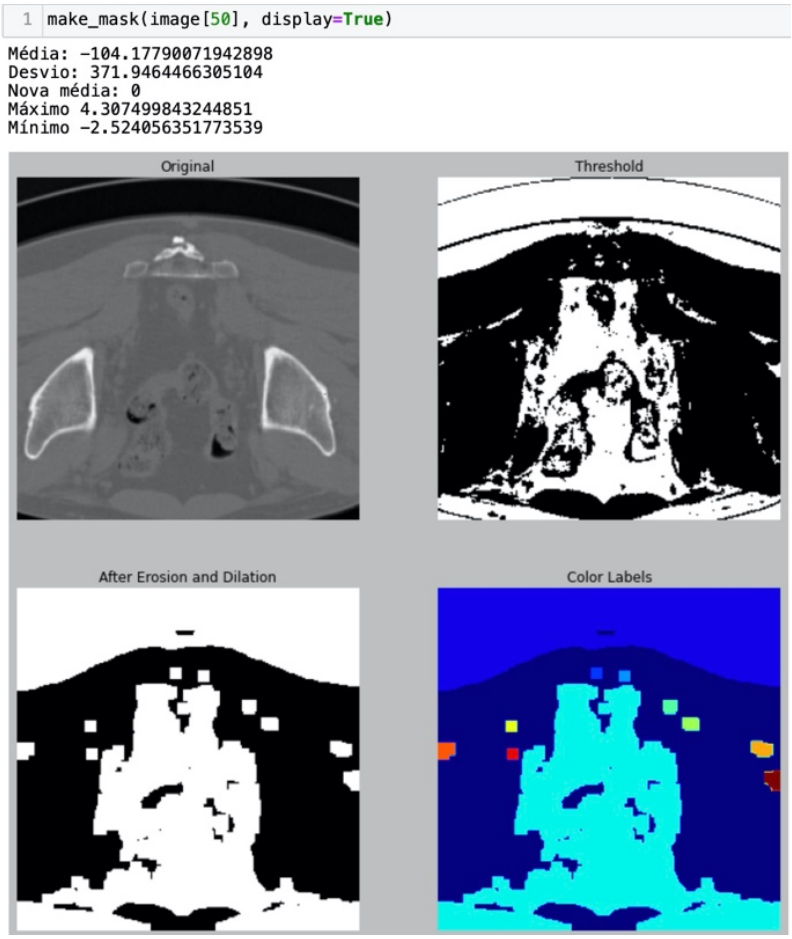


Figura 44. Separação de diferentes rótulos para aplicar as máscaras nas imagens originais, possibilitando sua segmentação.

4.5.15. Aplicando máscara em uma imagem

Uma vez obtido os rótulos, podemos criar a máscara e aplicar na matriz original. Um exemplo é a utilização dos métodos ***erosion*** e ***dilation*** para excluir ruídos externos ou retirar o artefato da mesa/maca (Figura 44 e 45). Uma vez adquirido os limites da imagem, podemos utilizar ***morfology.binary_fill_holes*** para preencher o meio da imagem com valor binário. Sendo assim temos uma matriz máscara cuja área interna terá valor 1 e a externa 0. Dessa forma basta aplicar a matriz na imagem original [4,10,13].

```python
import cv2
import os
import skimage.morphology as morph
from scipy import ndimage
display=False
imagemAbdome = filtro_imagem(uh_imagem, 40, 80)

# morphology.dilation cria uma segmentação da imagem
# Se um pixel está entre a origem e a borda do quadrado de 5x5, o pixel pertence a mesma classe

# Pode ser utilizado a função circular também morphology.disk(2)
# Nesse caso o pixel vai pertencer ao mesma classe se estiver entre o centro do círculo e o seu raio

segmentation = morph.dilation(imagemAbdome, np.ones((5, 5)))
labels, label_nb = ndimage.label(segmentation)

label_count = np.bincount(labels.ravel().astype(np.int))
# O tamanho do contagem de rótulos é o número de classes/segmentos

# O plano de fundo é a primeira classe, a qual não será usada
label_count[0] = 0

# Criamos uma máscara com a classe de maior número de pixels
# Nesse caso a pelve

mask = labels == label_count.argmax()


# Melhorar a marcara do abdome
mask = morph.dilation(mask, np.ones((5, 5)))
mask = ndimage.morphology.binary_fill_holes(mask)

mask = morph.dilation(mask, np.ones((3, 3)))

# OS pixels na máscra são compostos pos 0 e 1s
# Podemos multiplicar a imagem para pegar apenas a área de interesse
masked_image = mask * imagemAbdome

plt.figure(figsize=(15, 2.5))
plt.subplot(141)
plt.imshow(imagemAbdome,cmap=plt.cm.gray )
plt.title('Imagem Original')
plt.axis('off')

plt.subplot(142)
plt.imshow(mask, cmap=plt.cm.gray)
plt.title('Mask')
plt.axis('off')

plt.subplot(143)
plt.imshow(masked_image, cmap=plt.cm.gray)
plt.title('Final Image')
plt.axis('off')

plt.show()
```

Figura 45. Criando uma máscara e aplicando a imagem original.

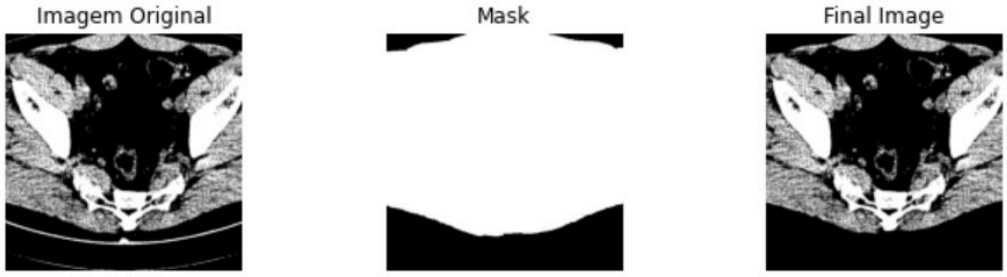


Figura 46. Criando uma máscara e aplicando a imagem original.

4.6. Linguagem de máquina – *Machine learning*

Uma definição formal para linguagem de máquina foi proposta por Tom Mitchell em 1998, na qual um programa de computador é capaz de aprender pela experiência *E* a partir de uma tarefa *T* e um parâmetro de performance *P*, se a sua performance em *T* for medida por *P* e melhorada com a experiência *E*.[1,2]

Em síntese, uma tarefa é executada, sua performance é medida e melhorada com o nível de experiência obtido. Dessa maneira, a velocidade do aprendizado está intrinsecamente ligada à velocidade e ao número de execução das tarefas.

Existe uma série de algoritmos que podem ser divididos em dois grupos principais: supervisionados e não-supervisionados. Outros mais comuns incluem aprendizado reforçado e sistemas de recomendação.

Os modelos de algoritmos supervisionados dependem da utilização de um conjunto de dados que contenham as variáveis de entrada e as saídas ou resultados “corretos” / esperados. Dentre os mais comuns temos a regressão, classificação, detecção de objetos, segmentação semântica etc. (uma abordagem mais prática permite categorizar essas metodologias apenas em problemas de "regressão" e "classificação")[1,2].

Em um problema de regressão, tentamos prever resultados em uma saída contínua, o que significa a tentativa de mapear variáveis de entrada para alguma função contínua.

Em um problema de classificação, por sua vez, tentamos prever resultados em uma saída discreta. Em outras palavras, o processo consiste em mapear variáveis de entrada em categorias discretas.

Exemplos:

- Regressão: Dada uma imagem de uma pessoa, é realizada a tentativa de prever sua idade.
- Classificação: Dado um paciente com tumor, é realizada a tentativa de prever se o tumor é maligno ou benigno.

Os modelos de algoritmos não-supervisionados utilizam-se apenas do conjunto de dados que contenham as variáveis de entrada, identificando as relações entre os dados e os classifica por agrupamentos (clusterização). Os mais comuns são *clustering* (k-means), detecção de anomalias (SVM), classificação, detecção de objetos, segmentação semântica e redução de dimensão pela análise de componente principal (Principal Componente Analysis).

4.7. Introdução aos modelos supervisionados

4.7.1. Regressão linear

Dado um conjunto de dados (***training set***) com ***m*** exemplos, cada um contendo uma variável **x** correspondente a entrada (atributo ou ***feature***), temos uma variável **y** de saída (alvo ou ***target***). O objetivo do modelo é prever as variáveis de saída dada uma variável de entrada nova. (figura 47)

Conjunto de dados para treino (*trainning set*) - altura e largura de espaços intervertebrais normais.

Largura do disco	Altura do disco
30	10
33	12
31	10
32	11
30	9
28	10
	...

m: número exemplos para treinos.
x: variáveis de entrada (atributos/*features*)
y: variáveis de saída (alvo/*target)*.

Figura 47. Criando uma máscara e aplicando a imagem original.

Dessa forma, os pares $(x^{(i)},y^{(i)})$ representam um exemplo de treino e os conjunto de ***i = 1, ... , m*** é chamado de conjunto de dados para treino (***training set***). O modelo então consiste de uma função que tenta se aproximar da dispersão/padrão dos dados (figura 48)

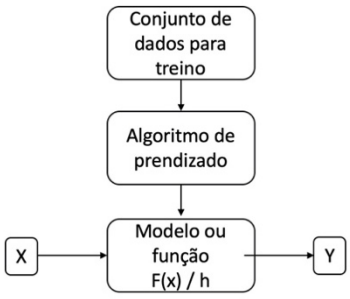


Figura 48. O modelo consiste em uma função que melhor estima a variável de saída, em detrimento dos exemplos disponibilizados no conjunto de dados de treino. O termo **h** é utilizado como hipótese.

A função linear representa, de maneira simples, o modelo dos dados que é facilmente implementada (figura 49).

$$f(x) = ax + b$$

ou

$$h\theta = \theta_0 + \theta_1 * x$$

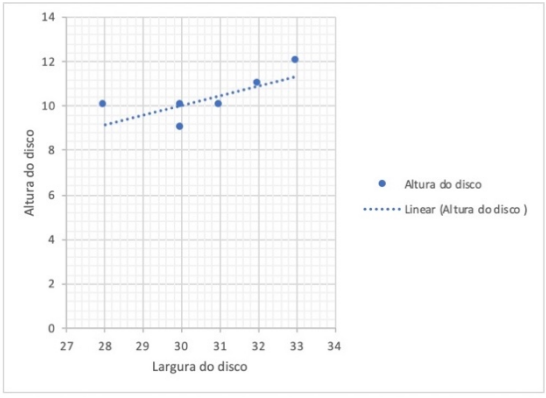


Figura 49. Representação do modelo linear (pontilhado) para representar um conjunto de dados de exemplo de largura por altura de discos normais.

Para conjuntos de dados de treino mais complexos, podemos utilizar equações não lineares (quadráticas, logarítmicas, polinomiais...).

Contudo, como adquirir os valores dos parâmetros a e b ou θ0 e θ1? O intuito é selecionar valores de modo que a diferença entre as saídas obtidas com o uso do modelo seja a menor possível dos valores das saídas disponibilizadas no conjunto de treino.

Para determinar a função linear que melhor se adequa aos dados podemos usar uma função de custo (*J*).

Como temos múltiplas entradas, a intenção é minimizar a soma do quadrado das diferenças entre todas as saídas obtidas com as desejadas, dividida pelo dobro da média. Essa é a descrição do método dos mínimos quadrados (figura 50).

$$J\theta_0, \theta_1 = 1/2m \sum_{i=1}^{m} (h\theta(x^{(i)}) - y^i))^2$$

$$\downarrow$$

$$h\theta(x^{(i)}) = \theta_0 + \theta_1 * x$$

$$Min_{\theta_0,\theta_1} : J\theta_0, \theta_1$$

Figura 50. Implementação do método dos mínimos quadrados.

A função de custo oferece um meio de comparar as diferentes escolhas para os valores iniciais de θ0 e θ1. Se plotarmos os valores de função J vamos obter uma rede em malha com formato hiperbólico em que os seus menores valores se aproximam do plano (0,0,0), ou seja, menor custo (figura 51).

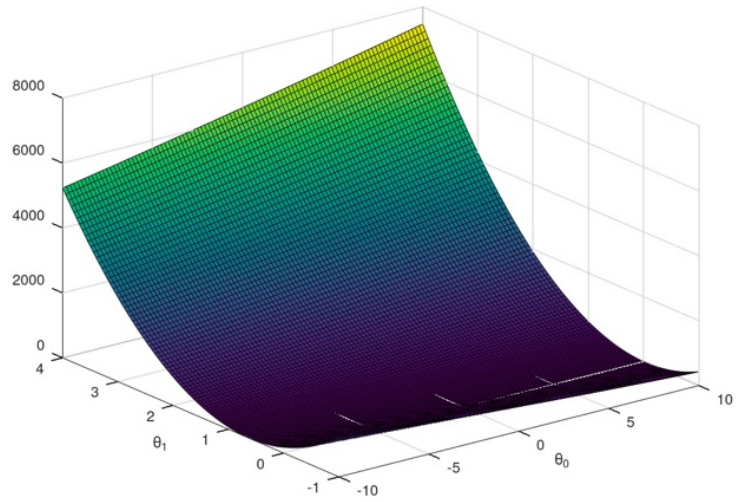


Figura 51. Representação 3D dos valores possíveis para a função de custo utilizando do método dos mínimos quadrados para o exemplo da altura e largura de discos.

O algoritmo de gradiente descendente é uma das metodologias para minimizar a função de custo, a qual tenta chegar no menor ponto de *J* a partir de valores arbitrariamente escolhidos.

A sua definição é dada por:

Repetir até a convergência:

$$\theta_j := \theta_j - \alpha * (\delta/\delta\theta_j) * J(\theta_0, \theta_1)$$

para J = 0 e j = 1

Corrigindo simultaneamente

$$temp0 := \theta_0 - \alpha * (\delta/\delta\theta_0) * J(\theta_0, \theta_1)$$

$$temp1 := \theta_1 - \alpha * (\delta/\delta\theta_1) * J(\theta_0, \theta_1)$$

$$\theta_0 := temp0$$

$$\theta_1 := temp1$$

Alpha representa nossa taxa de aprendizado ou tamanho de intervalo que vamos percorrer. Dessa maneira os valores de θ serão atualizados e reduzidos em cada iteração, até que o valor mais baixo da curva seja alcançado (Figura 52).

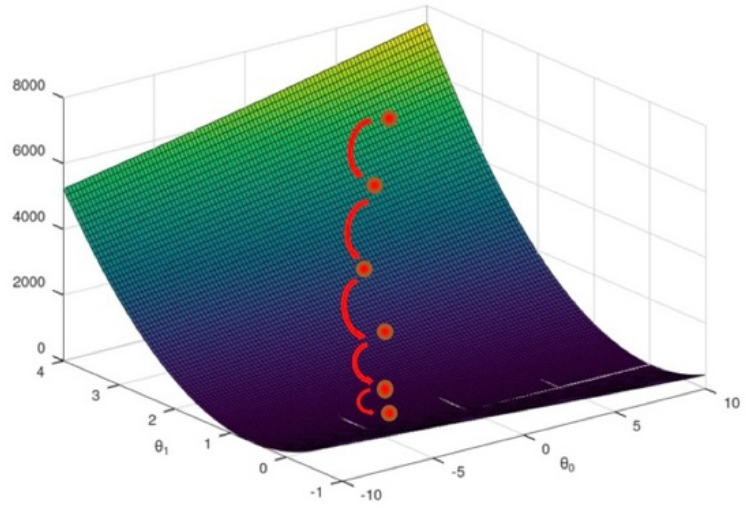


Figura 52. Atualização dos valores com gradiente descendente.

A implementação de funções mais complexas e com múltiplas variáveis, geralmente funções polinomiais, ocorre a partir do aumento do número de parâmetros na nossa hipótese, os quais serão propagados gradativamente para as outras funções:

$$h_\theta\left(x\right) = \theta_0 + \theta_1 * x_1 + ...\theta_n * x_n$$

Outros métodos podem ser utilizados para ajuste de parâmetros, tais quais: método dos mínimos quadrados generalizados, estimativa de máxima verossemelhança, regressão Bayesiana, regressão de Kernel, regressão gaussiana, etc.

Apesar das definições parecerem complexas, as bibliotecas disponíveis em ***Python*** facilitam a implementação da maioria dos modelos. Para tanto podemos utilizar o ***statsmodels*** e seu modelo de regressão linear simples (OLS). Exemplo do nosso conjunto de dados na figura 53.

Iportar bibbliotecas

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
```

Load the data

In [2]:
```
# Load the data from a .csv in the same folder
data = pd.read_csv('1.01. Simple linear regression.csv')
```

In [3]:
```
# Let's check what's inside this data frame
data
```

Out[3]:

	LAR	ALT
0	30	10
1	33	12
2	31	10
3	32	11
4	30	9
5	28	10
6	28	5
7	32	4
8	33	11
9	24	3

Regression itself

In [7]:
```
# Add a constant. Essentially, we are adding a new column (equal in lenght to x), which consists only of 1
x = sm.add_constant(x1)
# Fit the model, according to the OLS (ordinary least squares) method with a dependent variable y and an i
results = sm.OLS(y,x).fit()
# Print a nice summary of the regression. That's one of the strong points of statsmodels -> the summaries
results.summary()
```

Out[7]:

OLS Regression Results

Dep. Variable:	ALT	**R-squared:**	0.411
Model:	OLS	**Adj. R-squared:**	0.338
Method:	Least Squares	**F-statistic:**	5.591
Date:	Wed, 09 Feb 2022	**Prob (F-statistic):**	0.0456
Time:	01:09:57	**Log-Likelihood:**	-22.770
No. Observations:	10	**AIC:**	49.54
Df Residuals:	8	**BIC:**	50.14
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-13.7884	9.463	-1.457	0.183	-35.610	8.033
LAR	0.7405	0.313	2.365	0.046	0.018	1.463

Omnibus:	9.017	**Durbin-Watson:**	1.591
Prob(Omnibus):	0.011	**Jarque-Bera (JB):**	3.755
Skew:	-1.352	**Prob(JB):**	0.153
Kurtosis:	4.303	**Cond. No.**	343.

In [8]:
```
# Create a scatter plot
plt.scatter(x1,y)
# Define the regression equation, so we can plot it later
yhat = 0.7405*x1 -13.7884
# Plot the regression line against the independent variable (SAT)
fig = plt.plot(x1,yhat, lw=4, c='orange', label ='regression line')
# Label the axes
plt.xlabel('LAR', fontsize = 20)
plt.ylabel('ALT', fontsize = 20)
plt.show()
```

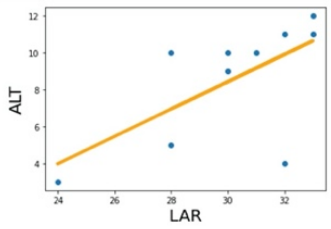
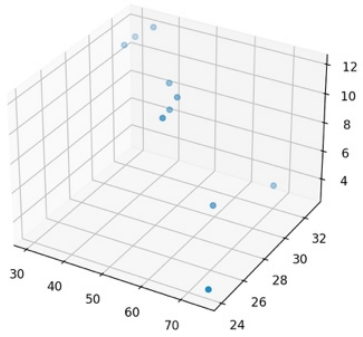


Figura 53. Modelo de implementação simples de uma regressão linear com ***statsmodel***.

A mesma aplicação pode ser usada para modelo de multivariáveis. Porém a representação gráfica da reposta pode ser limitada pela restrição da visualização de dados em mais de três dimensões, por exemplo, ao adicionar idade na tabela do exemplo de largura e altura de disco intervertebral (figura 54).

Idade	Largura do disco	Altura do disco
50	30	10
35	33	12
45	31	10
30	32	11
48	30	9
52	28	10
65	28	5
70	32	4
30	33	11
75	24	3




```
In [11]:
1 y = data['ALT']
2 x1 = data[['IDADE','LAR']]
```

Regression

```
In [12]:
1 x = sm.add_constant(x1)
2 results = sm.OLS(y,x).fit()
3 results.summary()
```

Out[12]:

OLS Regression Results

Dep. Variable:	ALT	**R-squared:**	0.883
Model:	OLS	**Adj. R-squared:**	0.850
Method:	Least Squares	**F-statistic:**	26.42
Date:	Wed, 09 Feb 2022	**Prob (F-statistic):**	0.000547
Time:	06:40:38	**Log-Likelihood:**	-14.691
No. Observations:	10	**AIC:**	35.38
Df Residuals:	7	**BIC:**	36.29
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	21.5799	8.041	2.684	0.031	2.566	40.594
IDADE	-0.2023	0.038	-5.313	0.001	-0.292	-0.112
LAR	-0.0985	0.217	-0.453	0.664	-0.612	0.415

Omnibus:	1.440	**Durbin-Watson:**	1.432
Prob(Omnibus):	0.487	**Jarque-Bera (JB):**	0.784
Skew:	0.245	**Prob(JB):**	0.676
Kurtosis:	1.719	**Cond. No.**	1.21e+03

Figura 54. Modelo de implementação de uma regressão linear multivariável com ***statsmodel***.

Dessa forma podemos plotar em 3D os resultados do modelo em um plano, sabendo que o mesmo possui os seguintes valores dos parâmetros θ (figura 55):

$$h_\theta(x) = +21.57 + -0.09 * x_0 + -0.20 * x_1$$

```
fig = plt.figure()
ax = fig.gca(projection='3d')

X = np.arange(40,80,0.05)
Y = np.arange(20,40,0.05)


X, Y=np.meshgrid(x,y)
Z = -0.20*X -0.09*Y + 21.57

ax.plot_surface(X,Y,Z)
plt.show()
```

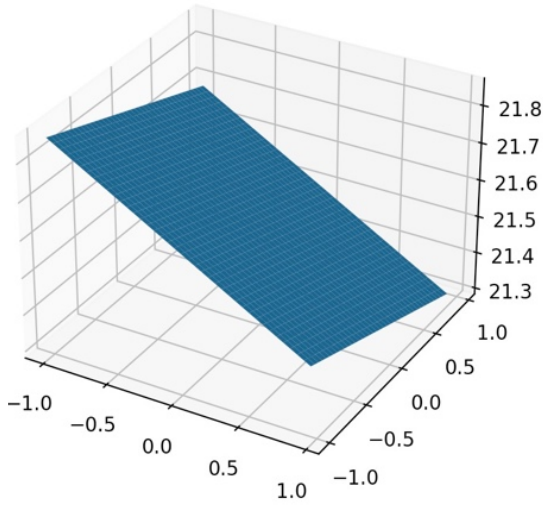


Figura 55. Plotando o modelo para a regressão linear multivariável. Nesse caso do exemplo, por se tratar de uma função com duas variáveis, somos capazes de representar como um plano.

4.7.2. Regressões logísticas/não-lineares e sistemas de classificação

A regressão logística nos permite alterar a hipótese a fim de classificar dados complexos em subgrupos de classes positivas ou negativas, basicamente as reduzindo em respostas binárias.

Diferente do modelo de regressão linear, ela se utiliza de um modelo não-linear, em que as saídas são reduzidas a pequenos valores que representam um subgrupo positivo e outro negativo, por exemplo 0 e 1 ou -1 e 1, denotando verdadeiro ou falso para uma hipótese.

Dessa maneira, a hipótese será representando por uma função não-linear, nesse caso logística ou sigmoide.

Sendo assim, para: $0 \leq h\theta(x) < 1$

$$h_\theta\left(x\right) = \theta_0 + -\theta_1 x$$

$$g(h_\theta\left(x\right)) = 1/1 + e^{-(h_\theta(x))}$$

A figura 56 compara o gráfico da função sigmoide com outras funções não-lineares frequentemente utilizadas, não somente nas regressões como também em redes neurais.

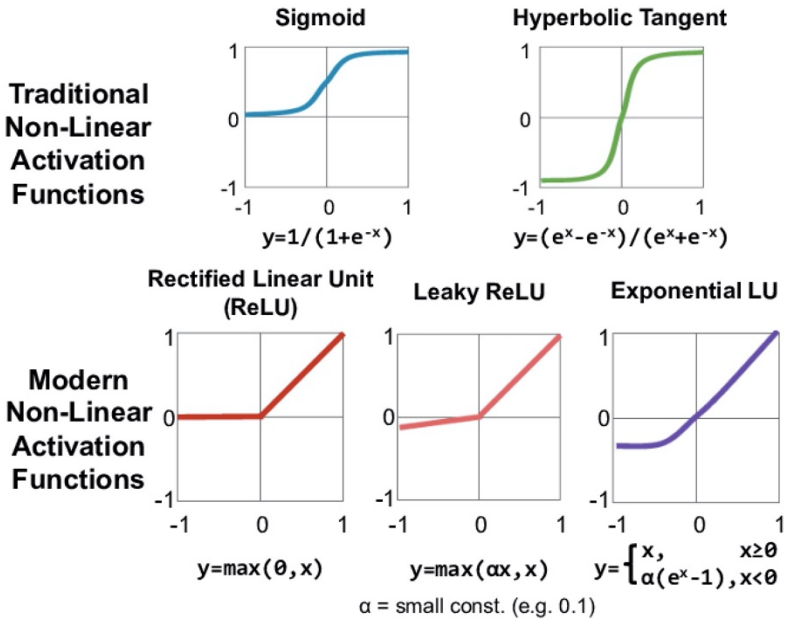


Figura 56. Comparação das diferentes funções não lineares comumente utilizadas, principalmente como funções de ativação.[18]

Da mesma maneira que fizemos para a regressão linear, vamos utilizar de uma função de custo e aplicar o gradiente descendente para ajustar os valores dos parâmetros. Podemos utilizar a regressão logística do ***statsmodel*** (figura 57).

Altura	Disco Normal
10	Yes
12	Yes
10	Yes
11	Yes
9	Yes
10	Yes
5	No
4	No
11	Yes
3	No
7	Yes
8	Yes
6	No
6	Yes
5	No
4	No
9	Yes
8	No

Plot a logistic regression curve

```
reg_log = sm.Logit(y,x)
results_log = reg_log.fit()

def f(x,b0,b1):
    return np.array(np.exp(b0+x*b1) / (1 + np.exp(b0+x*b1)))

f_sorted = np.sort(f(x1,results_log.params[0],results_log.params[1]))
x_sorted = np.sort(np.array(x1))

plt.scatter(x1,y,color='C0')
plt.xlabel('ALT', fontsize = 20)
plt.ylabel('ND', fontsize = 20)
plt.plot(x_sorted,f_sorted,color='C8')
plt.show()
```

```
Optimization terminated successfully.
         Current function value: 0.252800
         Iterations 8
```

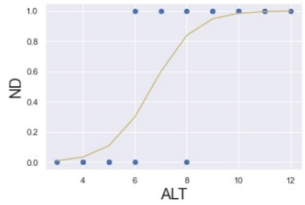


Figura 57. Cálculo do modelo de regressão logística usando o ***statsmodel***. A curva de regressão logística demostra que quanto menor os valores de altura do disco, maior a chance dele não ser normal.

As funções ***summary***, ***predict*** e ***pred_table*** listam os dados estatísticos do modelo, os resultados de previsão e, por fim, a matriz de confusão para cálculo da acurácia, respectivamente (figura 58).

```
In [9]: 1 results_log.summary()
```

Out[9]: Logit Regression Results

Dep. Variable:	ND	**No. Observations:**	18
Model:	Logit	**Df Residuals:**	16
Method:	MLE	**Df Model:**	1
Date:	Wed, 09 Feb 2022	**Pseudo R-squ.:**	0.6217
Time:	10:22:13	**Log-Likelihood:**	-4.5504
converged:	True	**LL-Null:**	-12.028
Covariance Type:	nonrobust	**LLR p-value:**	0.0001100

	coef	std err	z	P>|z|	[0.025	0.975]
const	-8.3324	3.904	-2.134	0.033	-15.984	-0.681
ALT	1.2463	0.567	2.197	0.028	0.135	2.358

```
In [10]: 1 results_log.predict()
Out[10]: array([0.98417593, 0.99867208, 0.98417593, 0.99539749, 0.94704997,
                0.98417593, 0.1089926 , 0.03398232, 0.99539749, 0.01001495,
                0.59663537, 0.8372264 , 0.29842613, 0.29842613, 0.1089926 ,
                0.03398232, 0.94704997, 0.8372264 ])

In [11]: 1 np.array(data['ND'])
Out[11]: array([1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0])

In [12]: 1 results_log.pred_table()
Out[12]: array([[ 6.,  1.],
                [ 1., 10.]])
```

Figura 58. ***summary***, ***predict*** e ***pred_table*** e seus respectivos resultados para avaliação do modelo.

4.7.3. Avaliação da adequação do modelo aos dados

Dois conceitos estão diretamente ligados: ***underfitting*** e ***overfitting*** [16].

Overfitting: O modelo se adequa muito bem aos dados de treino, tão bem que não é capaz de expressar uma validação externa.

Underfitting: modelo não consegue realizar boas predições, pois não consegue expressar bem o conjunto de dados, possuindo baixa acurácia.

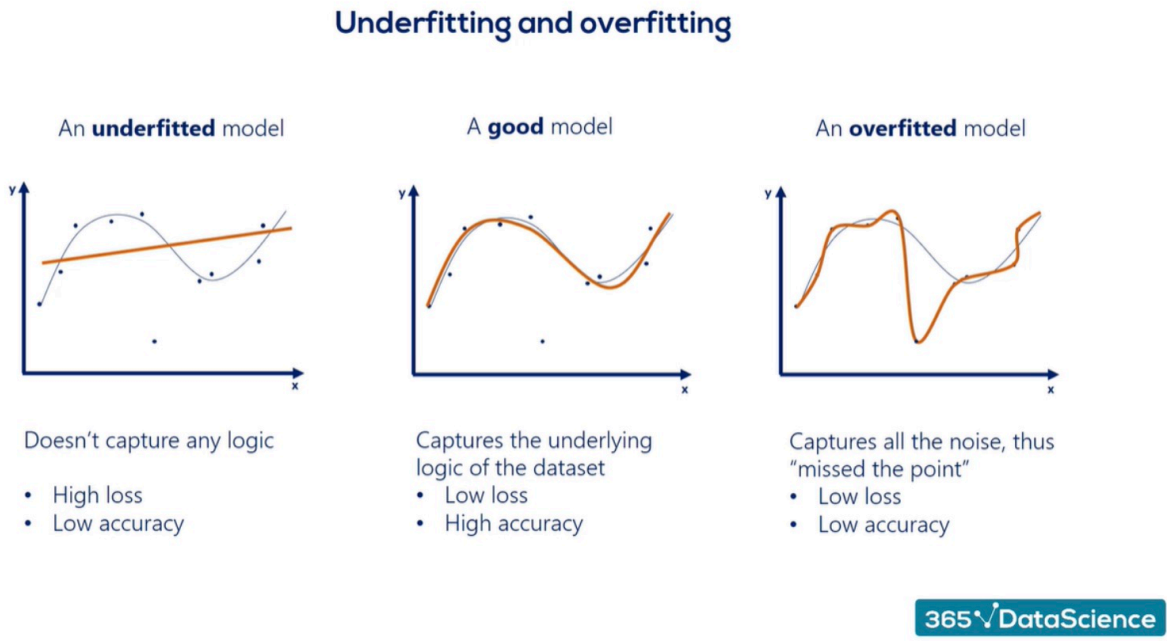


Figura 59. Exemplificando os conceitos de ***underfitting*** e ***overfitting***.[16]

4.7.4. Classificação versus Agrupamento

Os algoritmos de classificação estão agrupados dentro da classe de aprendizado supervisionado, ou seja, o conjunto de dados possui uma saída categorizada que é utilizada para treino do modelo, permitindo realizar previsões baseadas em resultados prévios. Em tese, o modelo prevê uma categoria de saída a partir dos dados de entrada.

Os algoritmos de agrupamento estão classificados na classe de aprendizado não supervisionado, ou seja, os dados não são categorizados e as saídas não são nomeadas. Em tese o modelo agrupa os pontos de dados, baseados nas similaridades entre eles e diferença entre os outros[1,2].

4.7.5. Agrupamento: Clusterização (K-Means)

Partimos do princípio de que cada agrupamento possui seu centroide, sendo definido do valor médio de distância dos pontos do agrupamento.

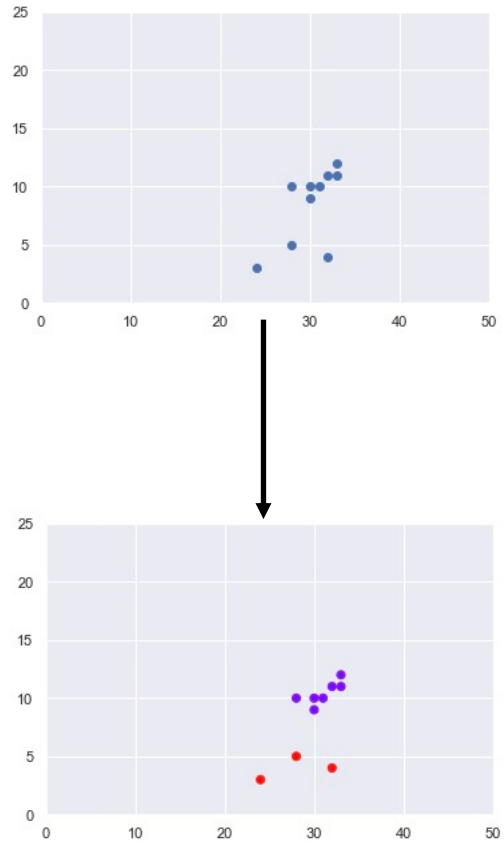
Para dado um número *k* de agrupamentos/clusters, temos um número *K* de centroides.

Dado o número de clusters K e o conjunto de dados a ser analisado, são estabelecidas posições iniciais para os centroides – os quais podem ser gerados ou selecionados aleatoriamente.

Em cada iteração são calculadas as distâncias dos centroides aos pontos mais próximos (geralmente distância euclidiana) e são reposicionadas, a partir da média entre todas as instâncias associadas àquele cluster. Para exemplificar podemos utilizar a função ***sklearn.cluster*** (figura 60).

Out[10]:

	IDADE	LAR	ALT	DN	Cluster
0	50	30	10	Yes	0
1	35	33	12	Yes	0
2	45	31	10	Yes	0
3	30	32	11	Yes	0
4	48	30	9	Yes	0
5	52	28	10	Yes	0
6	65	28	5	No	1
7	70	32	4	No	1
8	30	33	11	Yes	0
9	75	24	3	No	1



Clustering

This is the part of the sheet which deals with the actual clustering

In [7]:
```
# Create an object (which we would call kmeans)
# The number in the brackets is K, or the number of clusters we are aiming for
kmeans = KMeans(2)
```

In [8]:
```
# Fit the input data, i.e. cluster the data in X in K clusters
kmeans.fit(x)
```

Out[8]:
```
KMeans(n_clusters=2)
```

Clustering results

There are many ways to do this part, we found this to be the most illustrative one

In [9]:
```
# Create a variable which will contain the predicted clusters for each observation
identified_clusters = kmeans.fit_predict(x)
# Check the result
identified_clusters
```

Out[9]:
```
array([0, 0, 0, 0, 0, 0, 1, 1, 0, 1], dtype=int32)
```

In [10]:
```
# Create a copy of the data
data_with_clusters = data.copy()
# Create a new Series, containing the identified cluster for each observation
data_with_clusters['Cluster'] = identified_clusters
# Check the result
data_with_clusters
```

Figura 60. Utilizando o ***sklearn.cluster*** com dois centroides para separar os dois grupos de discos intervertebrais normais e discopatia.

4.7.6. Tipos de agrupamento: Plano e Hierárquico

No agrupamento plano podemos escolher o número de grupos (pela quantidade de clusters). O K-Means se encontra nessa categoria.

Outro método é o hierárquico, onde os subgrupos são definidos dentro de um modelo de hierarquia – existe o termo geral e seus subgrupos são definidos após eles. Nesses podemos encontrar os tipos aglomerativos e os tipos decisivos, onde a hierarquização ocorre de baixo pra cima e de cima para baixo, respectivamente.

As vantagens do agrupamento hierárquico incluem: mostra as possíveis ligações entre os grupamentos; legibilidade; dispensa a definição do número de agrupamentos. A grande desvantagem é a dificuldade de interpretação dos resultados caso o número de subgrupos seja muito grande. Dois métodos úteis consistem do dendrograma e mapas de calor (Figura 61).

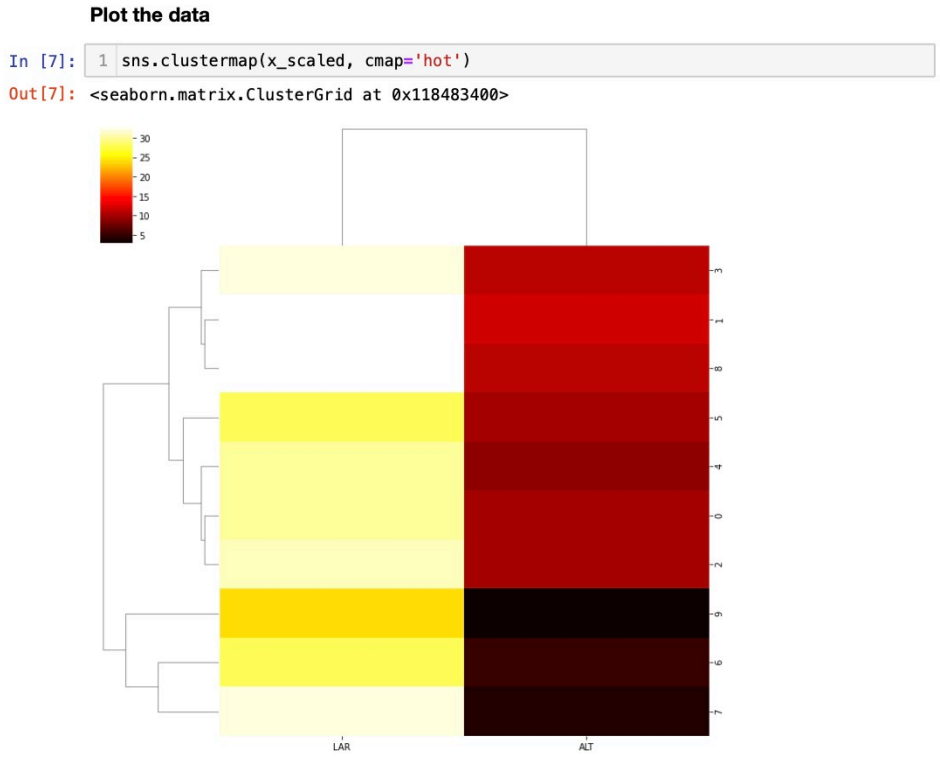


Figura 61. Exemplo de dendograma e de mapa de calor com o conjunto de dados do K-Means.

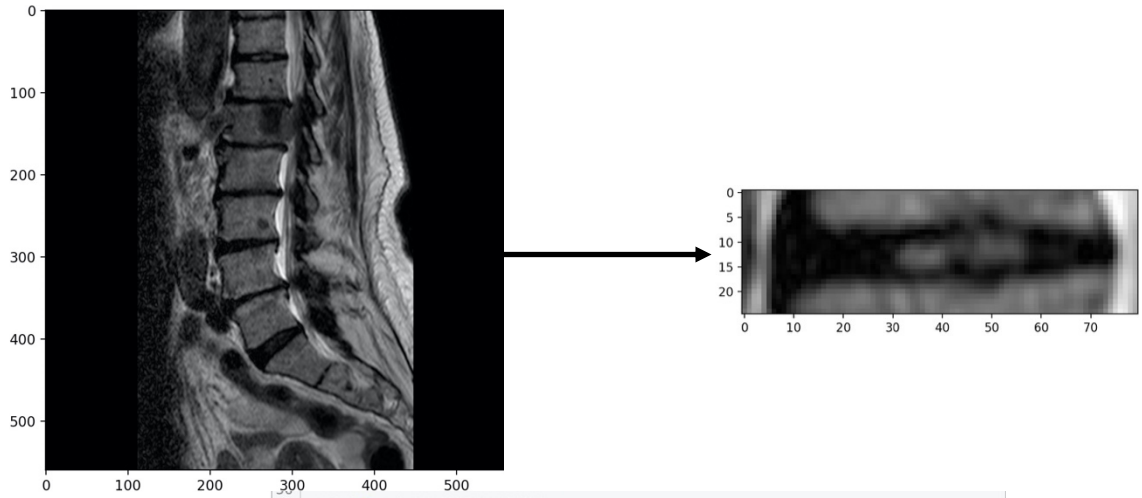
4.7.7. Localização de imagem Utilizando OpenCV

Template matching é um método para pesquisar e encontrar a localização de uma imagem de modelo em uma imagem maior. OpenCV disponibiliza função ***cv2.matchTemplate()*** para esse propósito. Durante cada iteração, o algoritmo compara o modelo com a área inicial correspondente da imagem de interesse, incrementa

gradativamente o seu posicionamento realizando novas comparações até o final da imagem. Vários métodos de comparação são implementados no OpenCV [17].

A espessura do marcador em retângulo denota o quanto a vizinhança desse pixel corresponde ao modelo.

Se a imagem de entrada for de tamanho (WxH) e a imagem de modelo for de tamanho (wxh), a imagem de saída terá um tamanho de (W-w+1, H-h+1). Depois de obter o resultado, você pode usar a função ***cv2.minMaxLoc()*** para descobrir onde está o valor máximo/mínimo. Podemos usá-lo como o canto superior esquerdo do retângulo e (w,h) corresponde à largura e altura do retângulo (figura 62).



```
method = cv2.TM_SQDIFF_NORMED
# Read the images from the file
small_image = cv2.imread('seg_sagitalT2.png')
large_image = cv2.imread('sagitalT2.png')

result = cv2.matchTemplate(small_image, large_image, method)

# We want the minimum squared difference
mn,_,mnLoc,_ = cv2.minMaxLoc(result)

# Draw the rectangle:
# Extract the coordinates of our best match
MPx,MPy = mnLoc

# Step 2: Get the size of the template. This is the same size as the match.
trows,tcols = small_image.shape[:2]

# Step 3: Draw the rectangle on large_image
cv2.rectangle(large_image, (MPx,MPy),(MPx+tcols,MPy+trows),(0,0,255),2)

# Display the original image with the rectangle around the match.
#cv2.imshow('output',large_image)
plt.imshow(large_image, cmap=plt.cm.gray)


```

```
Out[2]: <matplotlib.image.AxesImage at 0x13b297430>
```

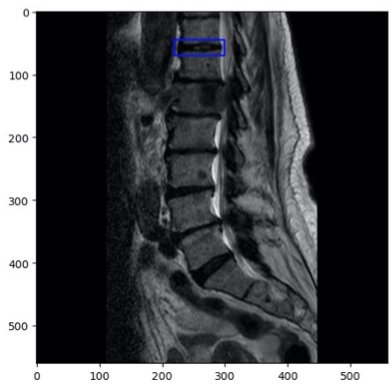


Figura 62: Ordenando o tamanho e obtendo a largura e altura do retângulo da seleção.

Porém, a função ***cv2.minMaxLoc()*** irá nos dar a localização do segmento cuja ocorrência está presente uma vez na figura.

Podemos utilizar o **cv2.*TM_CCOEFF_NORMED*** para buscar o equivalente aproximado dentro da imagem, determinando um ponto de corte como o limite inferior de coerência do modelo[17], conforme a figura 63.

```
img_rgb = cv2.imread('sagitalT2.png')
img_gray = cv2.cvtColor(img_rgb, cv2.COLOR_BGR2GRAY)
template = cv2.imread('seg_sagitalT2.png',0)
w, h = template.shape[::-1]

res = cv2.matchTemplate(img_gray,template,cv2.TM_CCOEFF_NORMED)
threshold = 0.7
loc = np.where( res >= threshold)
for pt in zip(*loc[::-1]):
    cv2.rectangle(img_rgb, pt, (pt[0] + w, pt[1] + h), (0,0,255), 1)

cv2.imwrite('res.png',img_rgb)

img_resultado = cv2.imread('res.png')
plt.imshow(img_resultado)
```

```
Out[9]: <matplotlib.figure.AxesImage at 0x13a909f70>
```

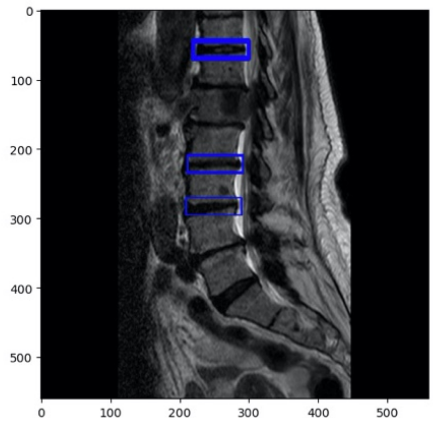


Figura 63. Utilização de ponto de corte para determinação dos limites.

A biblioteca Multi-Template-Matching (MTM) nos permite executar a pesquisa e a localização com um *template* único ou múltiplo, e sua função ***matchTemplates*** retorna o objeto "quantidade de acertos" (***Hits)***, o qual possui a lista de incidências encontradas para o limiar definido (***threshold***) e seus retângulos/locais (***bbox***). A função ***drawBoxesOnRGB*** facilita a plotagem das localizações na imagem original (figura 64)[18].

```
In [1]:
import os
import cv2
from MTM import matchTemplates, drawBoxesOnRGB
import matplotlib.pyplot as plt
from scipy import ndimage
import skimage.morphology as morph
import numpy as np


image_folder = "/Users/viniciusmartinsvilela/PycharmProjects/basicFunctions/Discos"
input_img = cv2.imread('sagitalT2.png')

listTemplate = []

small_image = cv2.imread('seg_sagitalT2.png')
large_image = cv2.imread('sagitalT2.png')


listTemplate = [('temp0', small_image)]

# Chama a função para matchTemplate com 1 imagem apenas
Hits = matchTemplates(listTemplate, large_image, N_object=4,score_threshold=0.7, me

print("Found {} hits".format( len(Hits.index) ) )
print (Hits)

Overlay = drawBoxesOnRGB(large_image, Hits, boxThickness=5)
plt.figure(figsize = (10,10))
plt.axis("off")
plt.imshow(Overlay)

Found 3 hits
  TemplateName                BBox     Score
0        temp0   (220, 45, 80, 25)  0.999871
1        temp0  (212, 210, 80, 25)  0.757553
2        temp0  (210, 271, 80, 25)  0.713099

Out[1]: <matplotlib.image.AxesImage at 0x1390e44f0>
```

Figura 64. Utilizando a biblioteca MTM com um *template* apenas.

Para utilizar as funções do MTM com múltiplos *templates* é necessário construir um banco de imagens segmentadas, e para tal finalidade podemos utilizar o ***ImageJ***.

O ***ImageJ*** é um software de código aberto de domínio público voltado para processamento e análise de imagens científicas. Consiste em uma plataforma bem elaborada que permite execução de scripts e possui várias ferramentas embutidas, tanto estatísticas quanto de processamento de imagens.

Para este estudo foram analisados 20 exames de ressonância magnética da coluna lombar, com segmentação manual dos espaços interdiscais na sequência T2 sagital, os quais foram armazenados em bloco em uma pasta de nome "***Discos***", com o intuito de compor o bloco de *templates* a ser utilizado pelo MTM (figura 65) [18].

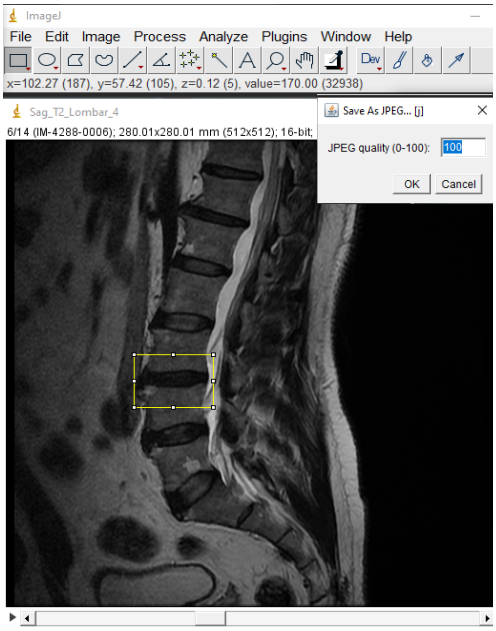


Figura 65. Utilizando o ***ImajeJ*** para construção do banco de Templates. A função ***crop*** permite separar o intervalo específico na série de imagens e o ***script*** permite salvar o conjunto no formato ***jpeg***, facilitando a execução manual da segmentação.

Figura 66. Inicialmente os 615 ***templates*** foram armazenados em pastas com seus respectivos segmentos e em um segundo momento foram agrupados livremente em uma pasta chamada "***Discos***".

Após agrupados, o caminho para os ***templates*** pode ser carregado em um vetor e a função ***matchTemplates*** irá retornar um objeto com ***tuplas*** referentes ao escore, o nome do ***template*** e a localização em um objeto ***bbox*** (figura 67) [18].

```
In [2]:
image_folder = "/Users/viniciusmartinsvilela/PycharmProjects/basicFunctions/Discos"

for filename in os.listdir(image_folder):
    template_img = cv2.imread(os.path.join(image_folder, filename))
    listTemplate.append((filename.split('.')[0], template_img))

numero_templates = len(listTemplate)
print (f"Foram encontrados {numero_templates} templates.")
print (listTemplate[1])
```

```
Foram encontrados 615 templates.
('p12_513', array([[[102, 102, 102],
        [100, 100, 100],
        [ 95,  95,  95],
        ...,
        [ 15,  15,  15],
        [ 10,  10,  10],
        [ 12,  12,  12]],

       [[ 90,  90,  90],
        [ 95,  95,  95],
        [ 93,  93,  93],
        ...,
        [ 16,  16,  16],
        [  9,   9,   9],
        [ 10,  10,  10]],

       [[ 80,  80,  80],
        [ 89,  89,  89],
        [ 87,  87,  87],
        ...,
        [ 21,  21,  21],
        [ 11,  11,  11],
        [ 12,  12,  12]],

       ...,

       [[ 96,  96,  96],
        [ 91,  91,  91],
        [ 86,  86,  86],
        ...,
        [ 50,  50,  50],
        [ 55,  55,  55],
        [ 84,  84,  84]],

       [[ 98,  98,  98],
        [ 97,  97,  97],
        [ 94,  94,  94],
        ...,
        [ 61,  61,  61],
        [ 76,  76,  76],
        [105, 105, 105]],

       [[102, 102, 102],
        [102, 102, 102],
        [102, 102, 102],
        ...,
        [ 81,  81,  81],
        [ 93,  93,  93],
        [110, 110, 110]]], dtype=uint8))
```

```
In [3]:
Hits = matchTemplates(listTemplate, large_image, score_threshold=0.7, method=cv2.TM_
```

```
In [4]:
print("Found {} hits".format( len(Hits.index) ) )
Hits
```

```
Found 19 hits
```

Out[4]:

	TemplateName	BBox	Score
0	temp0	(220, 45, 80, 25)	0.999871
35	p6_346	(435, 360, 87, 54)	0.853671
36	p6_346	(435, 305, 87, 54)	0.850584
5	p6_347	(411, 213, 87, 54)	0.847916
40	p6_346	(435, 461, 87, 54)	0.836674
48	p6_346	(417, 133, 87, 54)	0.806170
49	p6_346	(396, 28, 87, 54)	0.804881
750	p3_235	(75, 227, 91, 54)	0.798801
862	p3_233	(201, 196, 91, 54)	0.796930
327	p17_234	(80, 348, 82, 51)	0.794901
868	p2_342	(210, 257, 79, 54)	0.788756
751	p3_235	(91, 502, 91, 54)	0.788137
329	p17_234	(39, 15, 82, 51)	0.784195
335	p17_234	(70, 138, 82, 51)	0.776342
753	p3_235	(89, 437, 91, 54)	0.773349
357	p17_234	(39, 80, 82, 51)	0.754078
386	p17_234	(39, 285, 82, 51)	0.741800
127	p8_514	(243, 375, 78, 66)	0.720686
789	p9_454	(128, 291, 75, 49)	0.718925

Figura 67. Inicialmente os 615 ***templates*** foram armazenados em uma lista de *templates* e a função de ***matchTemplates*** com ***tresold*** de 70% retornou a quantidade de 19 achados.

Contudo, é possível observar que esse processo não se trata de uma rede neural, e sim de uma comparação estatística entre os pixels que é executada em bloco. A imagem de ***template*** é comparada ao longo de toda a imagem original ou de interesse. Sendo assim, quanto maior a lista de ***templates***, maior o tempo de execução.

Outra desvantagem é o reconhecimento de artefatos e outras áreas identificadas erroneamente como ***templates.*** Além disso, a função ***matchTemplates*** é extremamente sensível a rotações, translações e redimensionamentos. Dos 19 achados, apenas quatro correspondem a espaços discais e os demais referem-se a falso-positivos, em sua maior parte nas áreas externas da imagem (figura 68).

```
In [5]: 1 Overlay = drawBoxesOnRGB(large_image, Hits, boxThickness=5,showLabel=True)
        2 plt.figure(figsize = (10,10))
        3 plt.axis("off")
        4 plt.imshow(Overlay)

Out[5]: <matplotlib.image.AxesImage at 0x138179df0>
```

Figura 68. Achados na imagem de interesse com apenas quatro achados corretos e múltiplos falsos negativos na região de borda da imagem.

Existem diferentes maneiras de se pré-processar a imagem original com o intuito de minimizar a presença desses falso-positivos, algumas delas já abordadas nesse trabalho através de funções mais rudimentares.

A biblioteca do *OpenCV* possui uma extensa quantidade de funções para processamento de imagem, as quais permitem a fácil implementação de rotações, translações, redimensionamentos, filtros, detecção de contornos e bordas etc.

Essas funções nos permitem implementar de maneira mais intuitiva alguns métodos de pré-processamento já explorados no texto, dentre eles: remoção de artefato externo e redimensionamento. A extensa documentação possui múltiplos exemplos de códigos e tutoriais para serem executados.[18]

Utilizaremos as funções ***dilation*** e ***erosion*** do ***scikitlearn***, juntamente com a função de borramento, para pré-processar a imagem com o intuito de remover o ruído externo (figura 68).

```
In [6]:
#aplicando filtros
img = cv2.imread('sagitalT2.png')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

#blurred = cv2.GaussianBlur(gray, (15, 15), 0)
#blurred = cv2.medianBlur(gray,25)
blurred = cv2.bilateralFilter(gray,9,75,75)

print (type(blurred))
print (blurred.shape)


segmentation = morph.erosion(blurred, np.ones((35, 35)))
labels, label_nb = ndimage.label(segmentation)

label_count = np.bincount(labels.ravel().astype(np.int))
# O tamanho do contagem de rótulos é o número de classes/segmentos

# O plano de fundo é a primeira classe, a qual não será usada
label_count[0] = 0

# Criamos uma máscara com a classe de maior número de pixels
# Nesse caso a pelve
mask = labels == label_count.argmax()

# Melhorar a máscara do abdome
#mask = morph.dilation(mask, np.ones((5, 5)))
#mask = morph.erosion(mask, np.ones((3, 3)))
mask = ndimage.morphology.binary_fill_holes(mask)

#mask = morph.dilation(mask, np.ones((3, 3)))

# OS pixels na máscra são compostos pos 0 e 1s
# Podemos multiplicar a imagem para pegar apenas a área de interesse
masked_image = mask * gray

cv2.imwrite("masc.jpg", masked_image)
large_image = cv2.imread('masc.jpg')
```

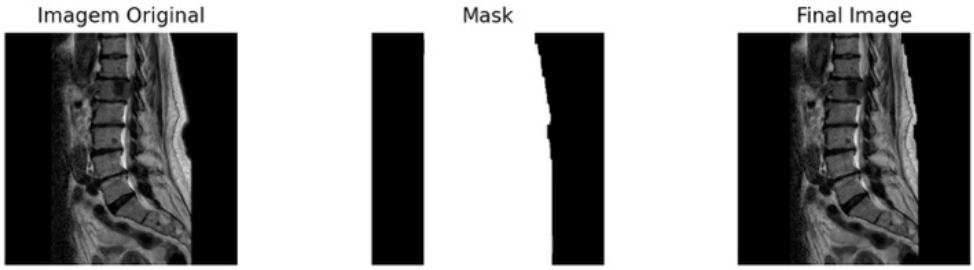


Figura 68. Pré-processamento da imagem com filtro de borramento e com formação de máscara utilizando das funções ***dilation*** e ***erosion***.

Contudo, o MTM é um algoritmo com limitações e muitas vezes somente esse processo não é suficiente. No nosso exemplo, mesmo com os dois pré-processamentos, foi encontrada uma grande quantidade de artefatos, no entanto agora localizados de forma mais homogênea na periferia da imagem, decorrentes, principalmente, dos artefatos granulados externos (figura 69).

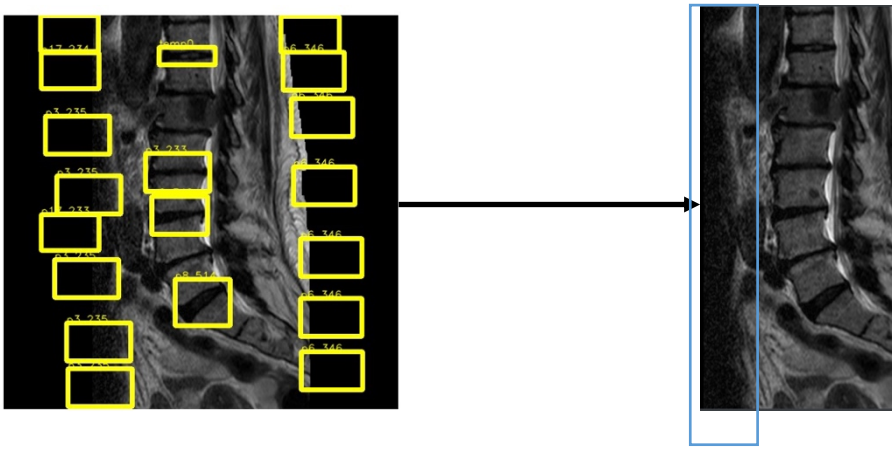


Figura 69. Mesmo após pré-processamento a quantidade de ruídos externos persiste, principalmente na região anterior da parede abdominal.

Uma maneira extremamente simples, porém muito limitada e pouco acurada, de resolver as anomalias na detecção, consiste em eliminar a região das bordas da imagem. Considerando que o exame para coluna foi localizado no centro da imagem, podemos eliminar do eixo X os primeiros e últimos 25% (figura 70).

```
#Solução não muito ideal, pegar P25 e P75

x = large_image.shape[0] // 2
y = large_image.shape[1] // 2
quart = large_image.shape[0] // 4

x, y, w, h = quart, 0, x, large_image.shape[1]
middle = img[y:y + h, x:x + w]

Hits = matchTemplates(listTemplate, middle, score_threshold=0.7, method=cv2.TM_CCOE

print("Found {} hits".format( len(Hits.index) ) )
Hits

Overlay = drawBoxesOnRGB(middle, Hits, boxThickness=5,showLabel=True)
plt.figure(figsize = (10,10))
plt.axis("off")
plt.imshow(Overlay)
plt.show()

```

Figura 70. Eliminando verticalmente as regiões periféricas da imagem.

Esse exemplo é uma forma alternativa de computar os dados, entretanto de pouca aplicabilidade no universo real, com uso restrito na prática devido a variabilidade da localização dos exames, limitando a possibilidade de seccionar a imagem verticalmente.

Uma maneira mais precisa consiste em avaliar todos os seguimentos encontrados na imagem pré-processada sem cortes e comparar seus histogramas com a imagem original.

Quando normalizado, o histograma da imagem completa apresenta um padrão mais homogêneo das distribuições dos seus intervalos de valores (figura 71).

```
hist = cv2.calcHist(large_image,[0],None,[256],[0,256])
hist = cv2.normalize(hist, hist).flatten()
plt.plot(hist,color = 'gray')
plt.xlim([0,256])
plt.show()
```

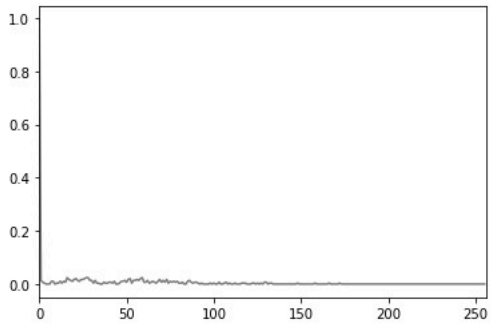
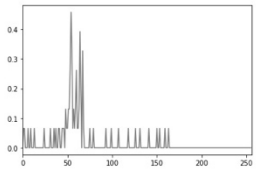


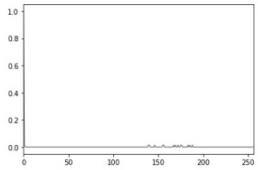
Figura 71. Histograma normalizado da imagem completa.

Os histogramas das regiões de achados falso-negativos possuem um padrão muito similar, pois sua maior parte consiste de pixels mais próximos ao valor mínimo. Por outro lado, os achados possuem um histograma normalizado mais perto da curva normal. Isso nos permite utilizar a função ***cv2.compareHist*** com diferentes modelos de funções de custo, apontando os achados com maior probabilidade de serem positivos (figura 72).

```
/Users/viniciusmartinsvilela/PycharmProjects/basicFunctions/result_folder/0.jpg
```



```
Comparando CORREL:0.004553495115156981
Comparando CHISQR:126.96661867678505
Comparando INTERSECT:0.3405099092051387
Comparando BHATTACHARYYA:0.808307409681335

Possivelmente válido
/Users/viniciusmartinsvilela/PycharmProjects/basicFunctions/result_folder/1.jpg
```



```
Comparando CORREL:0.9939531566274507
Comparando CHISQR:0.9454211897586927
Comparando INTERSECT:1.011133951600641
Comparando BHATTACHARYYA:0.5730286186897838

Possivelmente não válido
```

```
resultado = Hits.values
lista_bbox = []
for row in resultado:
  lista_bbox.append(row[1])
print (lista_bbox)


hist_large = cv2.calcHist(large_image,[0],None,[256],[0,256])
hist = cv2.normalize(hist_large, hist_large).flatten()
#plt.plot(hist,color = 'gray')
#plt.xlim([0,256])
#plt.show()

resultados_validos = []
result_folder = "/Users/viniciusmartinsvilela/PycharmProjects/basicFunctions/result
i = 0
for elemento in lista_bbox:
    crop_img = large_image[elemento[1]:elemento[1]+elemento[3],elemento[0]:elemento
    print (result_folder + str(i) + ".jpg")
    hist_crop = cv2.calcHist(crop_img,[0],None,[256],[0,256])
    hist_crop = cv2.normalize(hist_crop, hist_crop).flatten()
    plt.plot(hist_crop,color = 'gray')
    plt.xlim([0,256])
    plt.show()

    rCORREL = cv2.compareHist(hist_large, hist_crop, cv2.HISTCMP_CORREL)
    rCHISQR = cv2.compareHist(hist_large, hist_crop, cv2.HISTCMP_CHISQR)
    rINTERSECT = cv2.compareHist(hist_large, hist_crop, cv2.HISTCMP_INTERSECT)
    rBHATTACHARYYA = cv2.compareHist(hist_large, hist_crop, cv2.HISTCMP_BHATTACHARY

    print ("Comparando CORREL:" + str(rCORREL))
    print ("Comparando CHISQR:" + str(rCHISQR))
    print ("Comparando INTERSECT:" + str(rINTERSECT))
    print ("Comparando BHATTACHARYYA:" + str(rBHATTACHARYYA))
    print ("")


    if rCHISQR > 25.00:
        print ("Possivelmente válido")
        resultados_validos.append(i)
        path_save = result_folder + str(i) + ".jpg"
        cv2.imwrite(path_save,crop_img)
    else:
        print ("Possivelmente não válido")

    print ("")
    i = i + 1

print (resultados_validos)
```

Figura 72. Comparando os histogramas normalizado dos segmentos encontrados.

Conseguimos selecionar na imagem sem cortes os resultados mais prováveis de corresponderem a achados positivos e, dessa maneira, eliminar os falso-negativos da seleção (figura 73).

```
from PIL import Image
import matplotlib.patches as patches


im = Image.open('sagitalT2.png')
fig, ax = plt.subplots(figsize=(10, 10))
ax.imshow(im)

#crop_img = large_image[elemento[1]:elemento[1]+elemento[3],elemento[0]:elemento[0]+

#(x,y,w,l)

for elemento in resultados_validos:
    bbox = lista_bbox[elemento]
    print (bbox)
    rect = patches.Rectangle((bbox[0], bbox[1]), bbox[2], bbox[3], linewidth=1, edg
    ax.add_patch(rect)

plt.show()
```

```
(220, 45, 80, 25)
(201, 196, 91, 54)
(210, 257, 79, 54)
(243, 375, 78, 66)
```

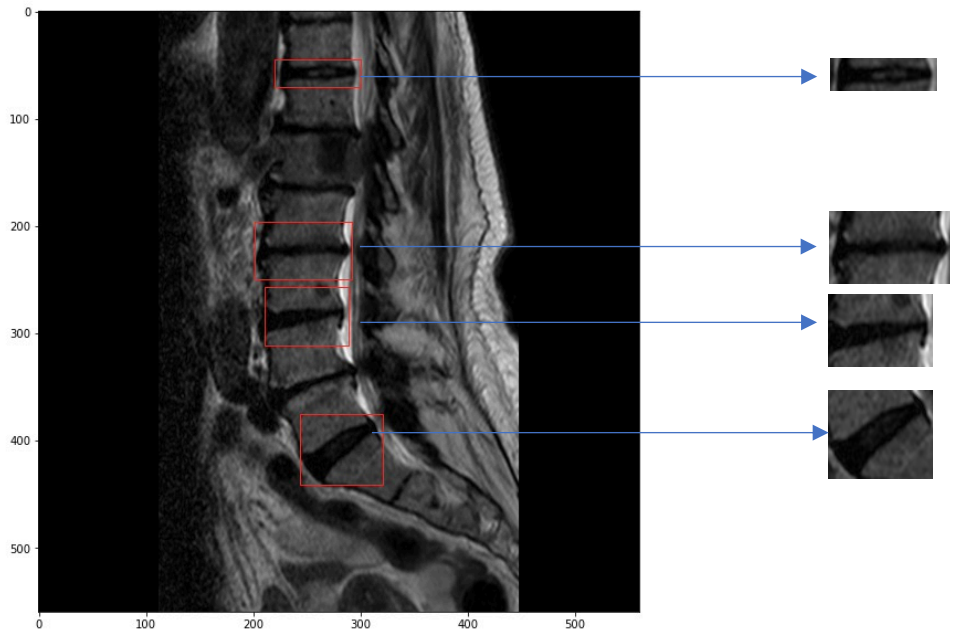


Figura 73. Armazenando somente os segmentos com maior probabilidade de serem positivos.

4.8. Redes Neurais

Possui suas origens em algoritmos que tentavam mimetizar o funcionamento do cérebro humano, simulando as respostas neuronais a eventos excitatórios e inibitórios a partir dos dados de entrada.

Foram amplamente utilizadas nas décadas de 80 e 90, e sua popularidade decresceu até o final da década de 90. Atualmente, a tecnologia ressurgiu e consiste no estado da arte para várias aplicações.

4.8.1. Perceptron e o modelo neuronal

Modelo de aprendizado supervisionado, desenvolvido nas décadas de 50 e 60 pelo cientista Frank Rosenblatt, inspirado em trabalhos anteriores de Warren McCulloch e Walter Pitts.

É um modelo matemático que recebe várias entradas, x1, x2, …, e produz uma única saída binária. São introduzidos pesos para cada entrada e a saída é dependente de uma comparação com um limiar, muito similar à função sigmoide ou uma classificação (figura 74).

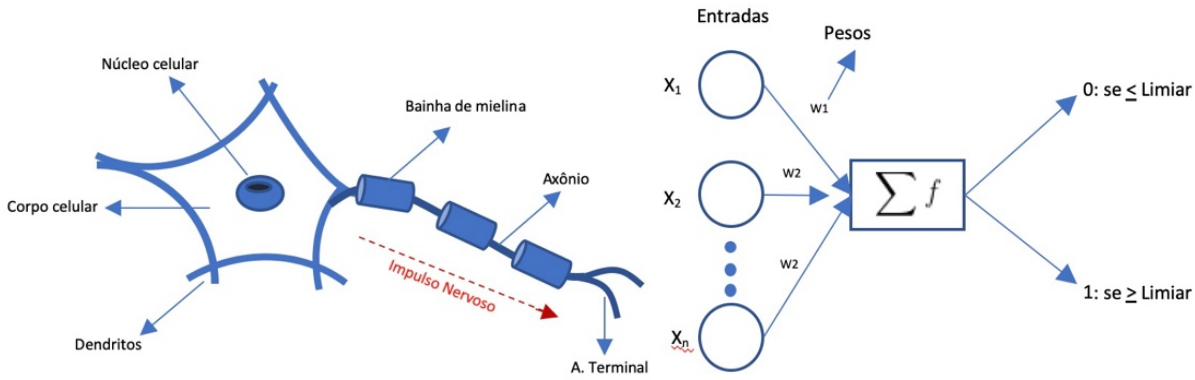


Figura 74. Neurônio e a representação de um *perceptrom* de uma camada.

Apesar do modelo matemático básico não ser capaz de representar tomadas de decisões humanas, pode ser muito útil para elucidar o funcionamento e para representar operações binárias simples.

Há, no entanto, a possibilidade do modelo agrupar múltiplas camadas, permitindo torná-lo mais complexo a fim de tomar decisões mais elaboradas.

Dessa forma, a próxima camada irá adequar pesos para as saídas das camadas anteriores, ou seja, a adição de camadas consecutivas trará a necessidade de armazenar os diferentes pesos para cada nível.

As camadas intermediárias são denominadas *camadas ocultas*, e a estrutura básica consistirá de uma camada de entrada, uma ou mais camadas ocultas e uma camada de saída (figura 75).

É possível observar que, para esse modelo, mesmo com camadas ocultas, duas transformações consecutivas sem uma função de ativação podem ser representadas apenas por uma[19].

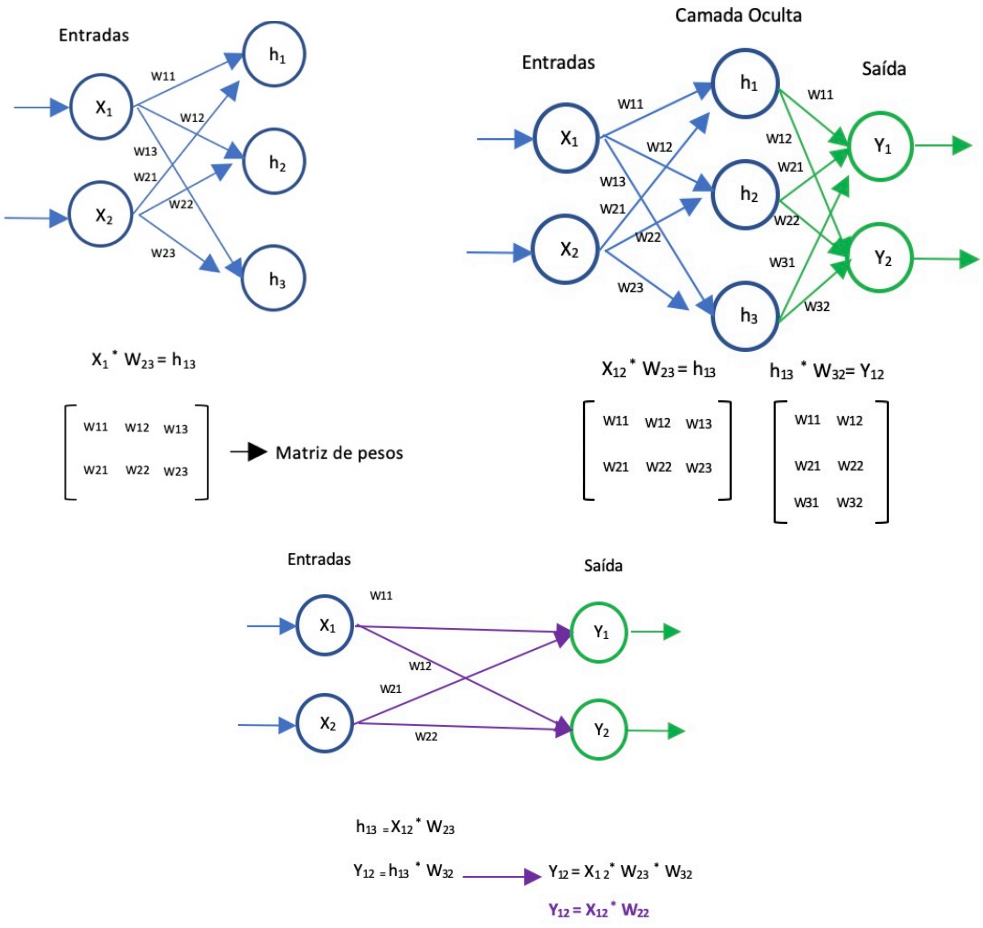


Figura 75. Representação de uma rede com uma camada oculta e a distribuição de sua matriz de pesos. Sem uma função de ativação é possível representar matematicamente apenas como uma transformação.

É chamada de propagação quando dados percorrem a rede no sentido de x para y e temos a saída em resultados, sendo que ao chegar em um resultado específico temos o final de uma época (figura 76).

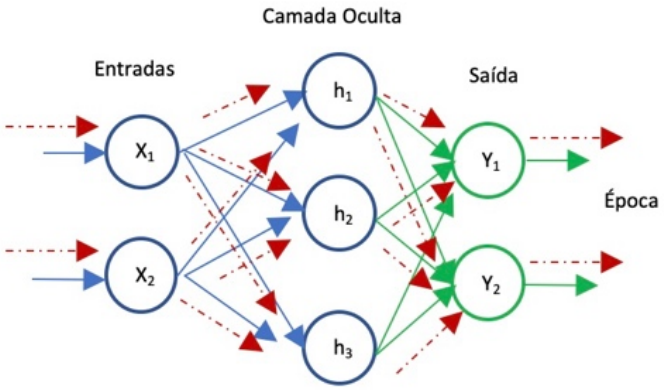


Figura 76. Representação a propagação e o final de uma época.

Porém, a nossa intenção é representar tomadas de decisão mais complexas ou humanas e, para tanto, é necessário definir funções de ativação não-linear para cada camada. A implementação com várias camadas ocultas e suas respectivas funções de ativação constituem a representação simples de uma rede neural profunda (figura 77) [19].

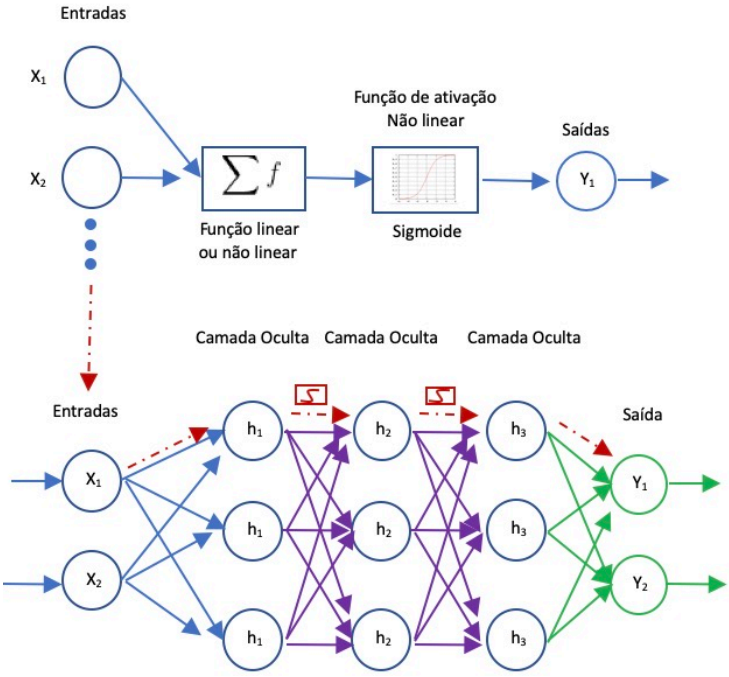


Figura 77. Representação de um modelo com funções de ativação não-lineares entre as camadas ocultas. Modelo base da rede neural convolucional (CNN) [19].

4.8.2. Redes neurais convolucionais (CNN) e reconhecimento de imagem

Redes neurais convolucionais (CNNs) são referenciadas como estado da arte para reconhecimento/detecção de imagens. Sua estrutura básica consiste de convoluções e aquisições de valores dos pixels próximos (***pooling***), conectadas a uma unidade de saída.

Cada convolução pode ser interpretada como a criação e atualização de mapas de pesos ou recursos. O ***pooling*** consiste em cálculo da média de uma região adjacente e atualização do valor em análise pela média, valor máximo, valor mínimo calculados a partir da área adjacente[19].

O algoritmo básico consiste em: transformar a imagem em uma matriz de pixels**,** selecionar o pixel de entrada, para cada iteração a região adjacente ao pixel de interesse é analisada e seus atributos são extraídos. Toda região é transformada em um valor de pixel apenas. Isso se trata da convolução.

O processo se repete movendo-se para o próximo pixel, acumulando os atributos extraídos, até chegar no final da imagem. Disso se trata o ***pooling*** (figura 78).

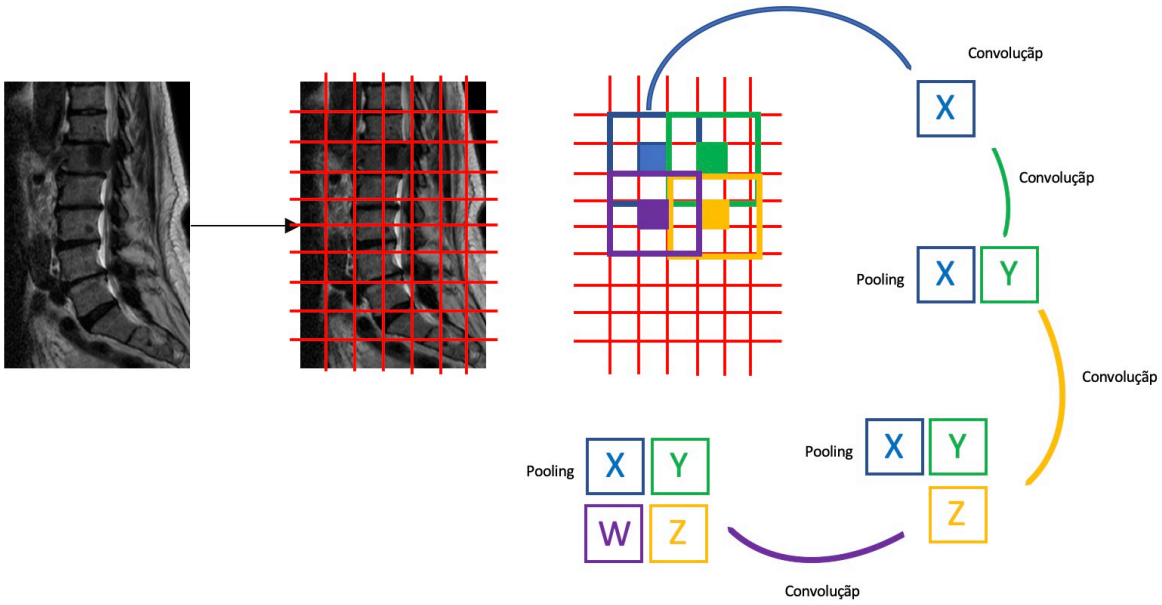


Figura 78. Interações de uma rede neural convolucional (CNN).

4.8.3. Preparando o banco de dados/*dataset*

ImageJ foi utilizado para realizar a análise de 80 exames de ressonância magnética da coluna lombar, com segmentação manual dos espaços interdiscais na sequência T2. Posteriormente as imagens foram classificadas como normal ou alterada por um radiologista

do serviço com especialização em imagem do sistema musculoesquelético (***Dr. Henrique Metzger***), de acordo com a classificação de Pfirrmann (figura 79).

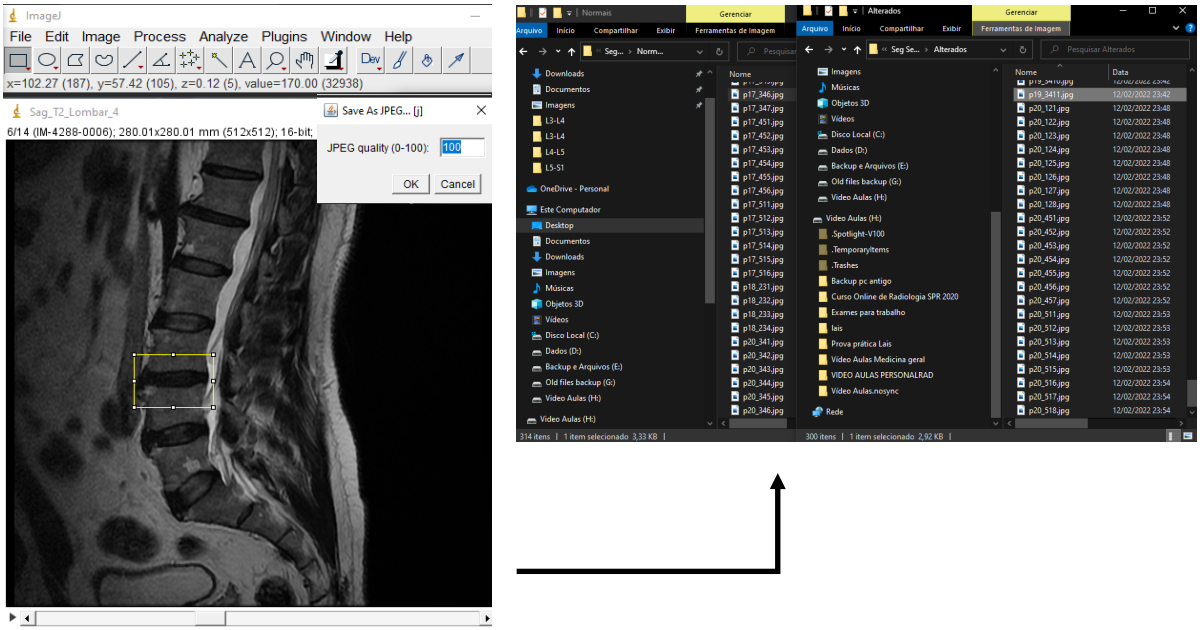


Figura 78. Segmentos classificados foram distribuídos em suas respectivas pastas de ***normais*** ou ***alterados***.

4.8.4. Classificação de Pfirrmann

O sistema de classificação de degeneração discal intervertebral de Pfirrmann é amplamente conhecido e utilizado, principalmente em decorrência de sua boa concordância intra e inter-observador.[20] A degeneração discal pode ser graduada pela sequência ponderada em T2 da ressonância magnética da coluna:

A. grau I: o disco é homogêneo com intensidade de sinal branco, hiperintenso, brilhante associado à altura normal do disco.
B. grau II: disco não é homogêneo, mas mantém o sinal branco, hiperintenso; núcleo e anel são claramente diferenciados, e uma faixa horizontal cinza pode estar presente; altura do disco é normal.
C. grau III: disco não é homogêneo com uma intensidade em queda; distinção entre núcleo e anel não é clara; a altura do disco é normal ou discretamente diminuída.
D. grau IV: disco é não homogêneo com uma intensidade de sinal cinza escuro, hipointenso; não há mais distinção entre o núcleo e o anel; a altura do disco é leve ou moderadamente diminuída.
E. grau V: disco é não homogêneo com uma intensidade de sinal preto, hipointenso; não há mais diferença entre o núcleo e o anel; o espaço do disco é reduzido.

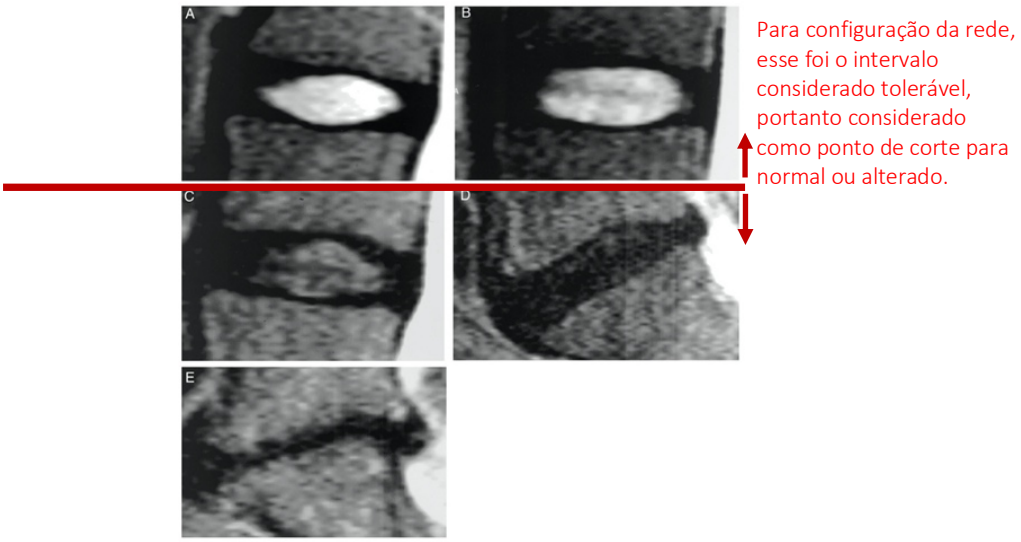


Figura 80. Classificação de acordo com a publicação original[20].

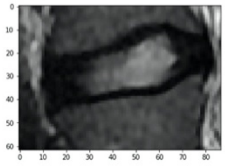
Para a implementação da rede neural, os discos classificados como normais foram aqueles com o padrão de degeneração grau um e dois. Discos alterados foram aqueles considerados com degeneração mais avançada, ou seja, graus três, quatro e cinco.

4.8.5. Preparando as imagens dos intervalos discais para criação do Dataset

Utilizando a função ***os.path.join***, podemos carregar inicialmente os arquivos em uma lista de vetores. É possível utilizar o índice dos vetores como rótulos (***labels***) e, dessa forma, os intervalos julgados classificados como "*Normais*" terão rótulo 0 e os "*Alterados*" terão rótulo 1. As imagens dos segmentos não possuem um tamanho padronizado, isso é decorrente do processo de aquisição manual com o ***ImageJ*** (figura 81).

```
import numpy as np
import matplotlib.pyplot as plt
import os
import cv2
from tqdm import tqdm

DATADIR = "/Users/viniciusmartinsvilela/PycharmProjects/basicFunctions/SegSeparados"

CATEGORIES = ["Normais", "Alterados"]

# Folder dos Normais e Alterados
for category in CATEGORIES:
    # Caminho para os dados
    path = os.path.join(DATADIR,category)
    for img in os.listdir(path):
        img_array = cv2.imread(os.path.join(path,img) ,cv2.IMREAD_GRAYSCALE)
        plt.imshow(img_array, cmap='gray')
        plt.show()

        break
    break
```

Figura 81. Criando o vetor de imagens.



Sendo assim, torna-se necessário realizar o redimensionamento para uniformizar a entrada, de modo que a rede consiga extrair e entender os atributos de maneira análoga. No nosso exemplo, o redimensionamento foi feito pela função ***resize*** do ***OpenCV***, para uma matriz de 50 x 50 pixels (figura 82).

```
In [3]:
#redimensiona
IMG_SIZE = 50
new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
plt.imshow(new_array, cmap='gray')
plt.show()
```

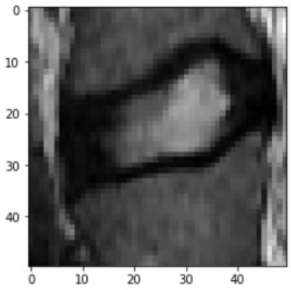


Figura 82. Redimensionamento da imagem para matriz de 50 x 50 pixels.

Em um segundo momento, os dados são agrupados entre as entradas e as saídas desejadas, sendo o primeiro índice da ***tupla*** a imagem e o segundo índice a sua classificação. Ambos são armazenados no ***training_data*** com o método ***append*** (figura 83).

```
training_data = []

#criando a base para treino
def create_training_data():
    for category in CATEGORIES:
        #diretório normal primeiro - entao será o 0 - e alterado depois - será o 1
        path = os.path.join(DATADIR,category)
        class_num = CATEGORIES.index(category)

        for img in tqdm(os.listdir(path)):
            try:
                img_array = cv2.imread(os.path.join(path,img) ,cv2.IMREAD_GRAYSCALE
                new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
                #adiciona a babse de treino
                training_data.append([new_array, class_num])
            except Exception as e:
                pass
            except OSError as e:
                print("OSError: Bad img most likely", e, os.path.join(path,img))
            except Exception as e:
                print("ohhhh shit Rick", e, os.path.join(path,img))

create_training_data()

print(len(training_data))
```

```
100%|██████████| 315/315 [00:00<00:00, 2226.23it/s]
100%|██████████| 301/301 [00:00<00:00, 2224.40it/s]
614
```

Figura 83. Adicionando os vetores de imagem e classificação na base de treino.

É necessário randomizar os dados dentro da base de treino para que a amostra não tenha uma distribuição viciada em apenas casos negativos ou positivos em sequência (figura 83).

```
In [5]:
import random

random.shuffle(training_data)
```

Figura 84. Randomização dos dados de treino.

Infelizmente o ***TensorFlow*** não realiza o reconhecimento e conversão automática da lista automática de vetores em matrizes. Sendo assim, é necessário utilizar o **np.array** para converter tanto os dados das imagens em uma matriz de 50 x 50, quanto a classificação em uma matriz de 1x1 (figura 85).

```
In [7]:
X = []
y = []

for features,label in training_data:
    X.append(features)
    y.append(label)

print(X[0].reshape(-1, IMG_SIZE, IMG_SIZE, 1))

X = np.array(X).reshape(-1, IMG_SIZE, IMG_SIZE, 1)
```

```
[[[[ 27]
   [ 21]
   [ 19]
   ...
   [104]
   [116]
   [178]]
```

Figura 85. Conversão das listas em ***np.array***.

É sempre recomendado salvar externamente os dados da biblioteca de treino, para que não seja necessário reconstruí-la em toda execução/teste (figura 86).

```
In [8]:
import pickle

pickle_out = open("X.pickle","wb")
pickle.dump(X, pickle_out)
pickle_out.close()

pickle_out = open("y.pickle","wb")
pickle.dump(y, pickle_out)
pickle_out.close()

#loading

pickle_in = open("X.pickle","rb")
X = pickle.load(pickle_in)

pickle_in = open("y.pickle","rb")
y = pickle.load(pickle_in)
```

Figura 86. Armazenando os ***np.arrays*** externamente para que não seja necessário recriar em toda execução.

O próximo passo é normalizar os dados, sobretudo as imagens. Feito isso, podemos criar a entrada, camadas ocultas/densas e a camada de saída. Nesse primeiro momento

iremos executar uma rede com função análoga à rede sem camada de ativação – o que é praticamente uma classificação linear ou uma ***support vector machine*** (SVM) de ***kernel*** único.

Dessa maneira, podemos criar uma camada de entrada com 256 nós, convolução de 3x3, o formato de entrada a matriz 50x50, uma função ***ReLu*** (retorna 0 para todos valores negativos e o próprio valor para todos valores positivos) e um ***pooling*** de 2x2.

Adicionamos uma segunda camada com 256 nós, convolução de 3x3, uma função ***ReLu*** e um ***pooling*** de 2x2.

Uma camada densa é criada, contudo sem a função de ativação, e por último definimos a camada de saída com uma função sigmoide para classificação binária.

O modelo é treinado utilizando um ***batch_size*** de 32 – referente a quantidade de exames executados em bloco, três épocas e validação de 20%.

Todas as métricas são computadas e armazenadas nas variáveis de log do ***TensorBoard*** para comparação com o próximo modelo (figura 87).

```
Epoch 1/3
16/16 [==============================] - 6s 371ms/step - loss: 0.5542 - accuracy: 0.73
12 - val_loss: 0.2218 - val_accuracy: 0.9024
Epoch 2/3
16/16 [==============================] - 6s 384ms/step - loss: 0.3168 - accuracy: 0.87
98 - val_loss: 0.2364 - val_accuracy: 0.9431
Epoch 3/3
16/16 [==============================] - 6s 385ms/step - loss: 0.1955 - accuracy: 0.91
04 - val_loss: 0.2477 - val_accuracy: 0.8862
Out[3]: <keras.callbacks.History at 0x14439b910>
```

```
import numpy as np

NAME = "Disco_Normal-vs-Alterado-CNN{}".format(int(time.time()))

pickle_in = open("X.pickle","rb")
X = pickle.load(pickle_in)

pickle_in = open("y.pickle","rb")
y = pickle.load(pickle_in)
y = np.array(y)

X = X/255.0

model = Sequential()

model.add(Conv2D(256, (3, 3), input_shape=X.shape[1:]))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(256, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(64))


model.add(Dense(1))
model.add(Activation('sigmoid'))

tensorboard = TensorBoard(log_dir="logs/{}".format(NAME))

model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'],
              )

model.fit(X, y,
          batch_size=32,
          epochs=3,
          validation_split=0.2,
          callbacks=[tensorboard])

```

Figura 87. Execução e treino do modelo sem a função de ativação na camada densa de 64.

Na execução seguinte alteramos alguns parâmetros do modelo e de sua execução para realizar a comparação com o primeiro. Iremos adicionar uma função de ativação na camada densa executando o modelo com ***batch_size*** de 12, 10 épocas e validação de 20% (figura 88).

In [4]:

```
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D
# more info on callbakcs: https://keras.io/callbacks/ model saver is cool too.
from tensorflow.keras.callbacks import TensorBoard
import pickle
import time
import numpy as np

NAME = "Disco_Normal-vs-Alterado-CNN-X64{}".format(int(time.time()))

pickle_in = open("X.pickle","rb")
X = pickle.load(pickle_in)

pickle_in = open("y.pickle","rb")
y = pickle.load(pickle_in)
y = np.array(y)

X = X/255.0

model = Sequential()

model.add(Conv2D(256, (3, 3), input_shape=X.shape[1:]))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(256, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))


model.add(Dense(1))
model.add(Activation('sigmoid'))

tensorboard = TensorBoard(log_dir="logs/{}".format(NAME))

model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'],
              )

model.fit(X, y,
          batch_size=12,
          epochs=10,
          validation_split=0.2,
          callbacks=[tensorboard])
```

```
Epoch 1/10
41/41 [==============================] - 7s 150ms/step - loss: 0.4510 - accuracy: 0.78
00 - val_loss: 0.2475 - val_accuracy: 0.8699
Epoch 2/10
41/41 [==============================] - 6s 149ms/step - loss: 0.2417 - accuracy: 0.89
82 - val_loss: 0.1923 - val_accuracy: 0.9350
Epoch 3/10
41/41 [==============================] - 7s 161ms/step - loss: 0.1841 - accuracy: 0.92
46 - val_loss: 0.1799 - val_accuracy: 0.9431
Epoch 4/10
41/41 [==============================] - 7s 172ms/step - loss: 0.1592 - accuracy: 0.94
50 - val_loss: 0.2543 - val_accuracy: 0.9187
Epoch 5/10
41/41 [==============================] - 6s 153ms/step - loss: 0.1103 - accuracy: 0.95
11 - val_loss: 0.1863 - val_accuracy: 0.9431
Epoch 6/10
41/41 [==============================] - 6s 156ms/step - loss: 0.1160 - accuracy: 0.95
32 - val_loss: 0.1628 - val_accuracy: 0.9512
Epoch 7/10
41/41 [==============================] - 7s 170ms/step - loss: 0.0774 - accuracy: 0.96
95 - val_loss: 0.1332 - val_accuracy: 0.9350
Epoch 8/10
41/41 [==============================] - 7s 167ms/step - loss: 0.0446 - accuracy: 0.98
57 - val_loss: 0.1675 - val_accuracy: 0.9675
Epoch 9/10
41/41 [==============================] - 7s 172ms/step - loss: 0.0778 - accuracy: 0.96
54 - val_loss: 0.1493 - val_accuracy: 0.9350
Epoch 10/10
41/41 [==============================] - 7s 179ms/step - loss: 0.0390 - accuracy: 0.98
37 - val_loss: 0.1296 - val_accuracy: 0.9675
Out[4]: <keras.callbacks.History at 0x1456791f0>
```

Figura 88. Execução e treino do modelo com a função de ativação na camada densa, ***batch_size*** de 12 e épocas de 10.

Para utilização futura do modelo, podemos salvá-lo em uma variável no nosso ambiente (figura 89).

```
In [5]: 1 model.save('64x3-CNN.model')
```

Figura 89. Salvando o último modelo treinado no nosso ambiente.

Após o modelo executado, podemos comparar os resultados utilizando o *TensorBoard* (figura 90).

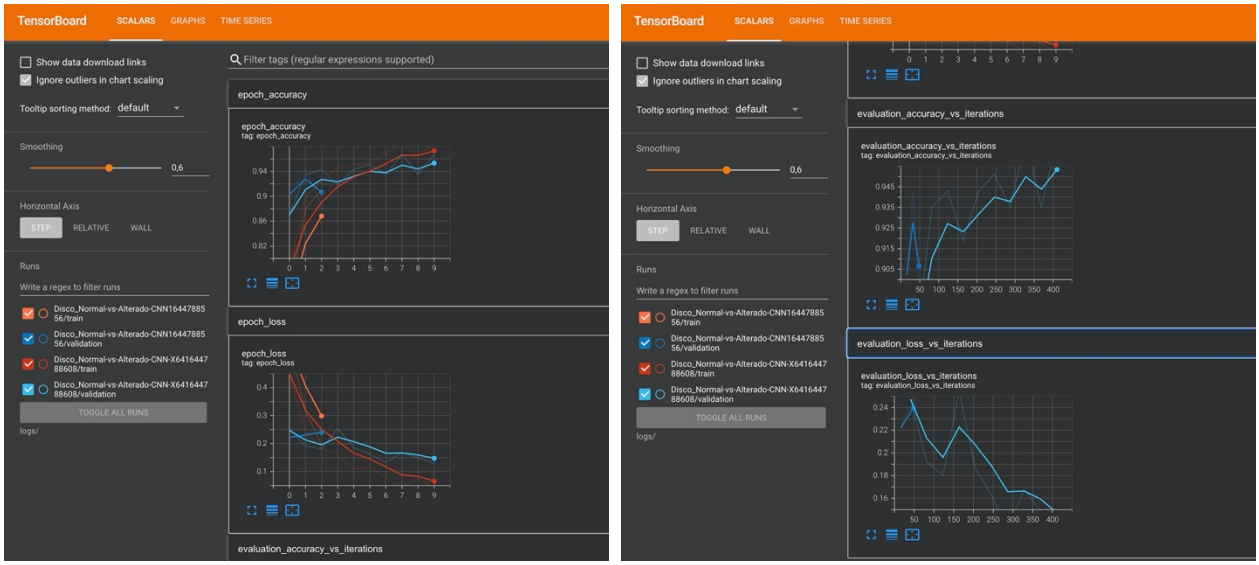


Figura 90. Análise dos parâmetros de acurácia e perda para cada execução de cada rede.

Através do *TensorBoard* podemos observar que a função de ativação apresentou um ganho em relação a acurácia e menores perdas. Para parâmetros de comparação, a primeira

rede apresentou uma perda de 0.19 com acurácia de 0,91 e acurácia para o subgrupo de validação de 0,88. A segunda rede apresentou uma perda de 0.03 com acurácia de 0,98 e acurácia para subgrupo de validação de 0,96.

Esses números refletem, todavia, apenas a validação interna, logo é necessário realizar a validação externa dos resultados. Para tanto, deixamos 20% da amostra para realizar o teste da rede, perfazendo um total de 60% para treino, 20% para validação interna e 20% para teste.

A execução do teste e a validação externa podem ser feitas em batch ou manualmente/individualmente. Basta recarregar o modelo e utilizar a função ***model.predict*** e a nova imagem como entrada (figura 91).

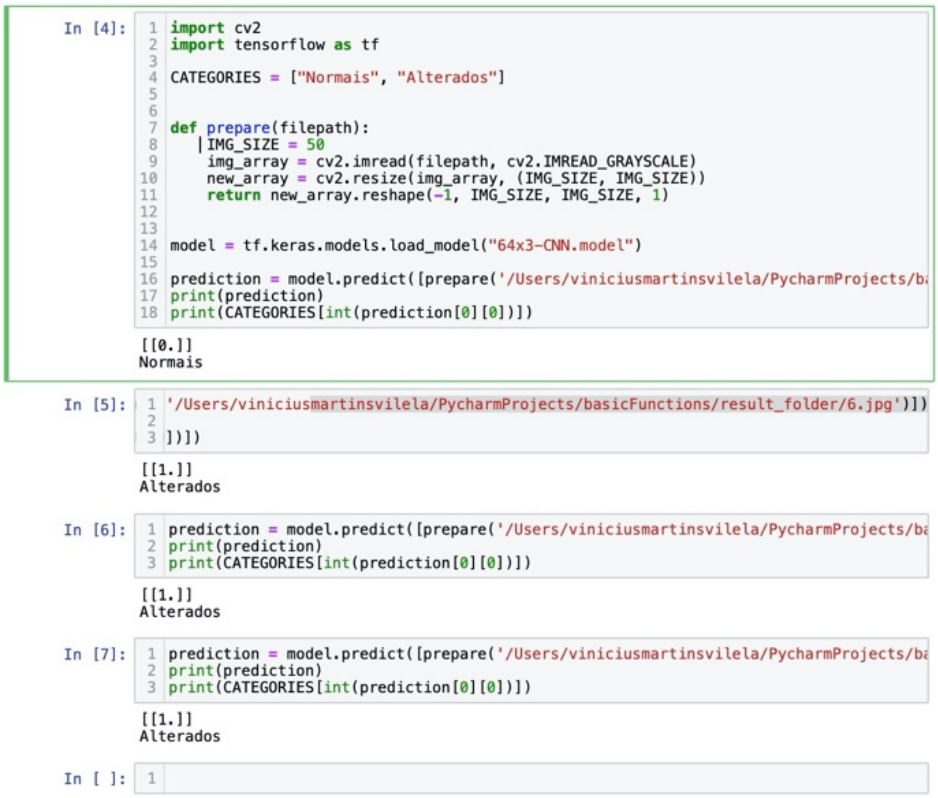


Figura 91. Execução e análise para teste da rede.

No nosso exemplo, a rede apresentou uma acurácia no teste de aproximadamente 75%. Como exemplo, o primeiro segmento detectado foi classificado como normal apesar de ser alterado, sendo necessário uma avaliação crítica das limitações da pesquisa.

Figura 92. Exemplo de disco erroneamente classificado como "***Normal***".

4.9. Limitações

A segmentação inicial dos intervalos discais utilizando-se do algoritmo de ***MTM*** é bem limitada, contudo foi utilizada apenas para exemplificar uma aquisição em bloco com métodos estatísticos fixos. A solução adequada é criar uma rede neural com imagens classificadas como "Disco" e "Não disco" para que a seleção tenha maior acurácia. Para tanto basta iterar sobre todas as imagens adquirindo intervalos médios de 50 x 50 e selecionar manualmente todas as imagens que contenham disco.

Grande parte das limitações da rede neural está relacionada à falta de padronização dos métodos de execução do exame, variantes anatômicas, artefatos e características próprias da patologia.

Um dos maiores problemas é a falta de padronização da aquisição em T2, com uma alta variabilidade nos tempos de eco e repetição em cada exame.

O algoritmo não fez distinção entre variantes anatômicas, sobretudo quanto à presença de vértebra de transição.

Artefatos de ruído e externos são extremamente complicados de serem tratados, apesar da biblioteca *OpenCV* disponibilizar funções que facilitam muito sua execução.

Uma característica importante da patologia é a possibilidade de assimetria do processo de degeneração discal. Sendo assim, um corte da imagem pode ter características normais enquanto o próximo corte do mesmo intervalo estar alterado, fazendo com que o intervalo seja erroneamente classificado como normal. Dessa maneira, essa característica afeta até mesmo a criação do banco de imagens para treino, uma vez que teremos imagens de disco que teriam classificação "Alteradas" alocadas dentro do subgrupo de imagens "Normais".

5. CONCLUSÃO

Atualmente as redes neurais são o estado da arte para o reconhecimento automatizado de imagens. Os softwares de auxílio diagnóstico e diferentes projetos relacionados a linguagem de máquina estão cada vez mais presentes na prática da radiologia.

É fundamental que o radiologista conheça os conceitos básicos utilizados no processamento e manipulação das imagens médicas digitalizadas.

O conhecimento sobre programação não é obrigatório, porém essa habilidade poderá ser um diferencial desejado.

O vínculo entre a radiologia e a ciência de dados poderá ser benéfica para ambas com alto potencial de sinergismo. A noção básica dos principais algoritmos e metodologias ajuda a estreitar as relações entre os diferentes campos de atuação.

Entender as diferentes aplicações com as padronizações necessárias para implementação e suas limitações poderá ser fator decisivo para inclusão do profissional em diferentes projetos ou, em um futuro próximo, no mercado de trabalho.

6. REFERÊNCIAS

1. Sundnes J. Getting Started with Python. Introduction to Scientific Programming with Python. 2020;1–4. https://doi.org/10.1007/978-3-030-50356-7_1.
2. Ceder NR. The quick Python book. Shelter Island, Ny: Manning Publications; 3rd ed. 2018.
3. PyPI – the Python Package Index [Internet]. PyPI. 2019. [cited 2022 Jan 10]. Available from: https://pypi.org
4. DICOM_toolbox_for_Radiologists_3 [Internet]. uwmsk.org. [cited 2022 Jan 10]. Available from: http://uwmsk.org/jupyter/Jupyter_DICOM_toolbox.html
5. General examples — pydicom 2.2.2 documentation [Internet]. pydicom.github.io. [cited 2022 Jan 10]. Available from: https://pydicom.github.io/pydicom/stable/auto_examples/index.html
6. Core elements in pydicom — pydicom 2.2.2 documentation [Internet]. pydicom.github.io. [cited 2022 Jan 10]. Available from: https://pydicom.github.io/pydicom/stable/old/base_element.html#dataset
7. Contrib-Pydicom [Internet]. GitHub. 2022 [cited 2022 Jan 10]. Available from: https://github.com/pydicom/contrib-pydicom/blob/master/plotting-visualization/dcm_qt_tree.py
8. Office for Civil Rights (OCR. Methods for De-identification of PHI [Internet]. HHS.gov. 2012. [cited 2022 Jan 10]. Available from: https://www.hhs.gov/hipaa/for-professionals/privacy/special-topics/de-identification/index.html#standard
9. Loading Data [Internet]. Deid. [cited 2022 Jan 10]. Available from: https://pydicom.github.io/deid/getting-started/dicom-loading/
10. Medical Images In python (Computed Tomography) [Internet]. Vicente Rodríguez blog. [cited 2022 Jan 10]. Available from: https://vincentblog.xyz/posts/medical-images-in-python-computed-tomography
11. Module: exposure — skimage v0.20.0.dev0 docs [Internet]. scikit-image.org. [cited 2022 Jan 10]. Available from: http://scikit-image.org/docs/dev/api/skimage.exposure.html
12. Local Histogram Equalization — skimage v0.20.0.dev0 docs [Internet]. scikit-image.org. [cited 2022 Jan 10]. Available from: https://scikit-

image.org/docs/dev/auto_examples/color_exposure/plot_local_equalize.html#sphx-glr-auto-examples-color-exposure-plot-local-equalize-py

13. Rank filters — skimage v0.20.0.dev0 docs [Internet]. scikit-image.org. [cited 2022 Jan 10]. Available from: https://scikit-image.org/docs/dev/auto_examples/applications/plot_rank_filters.html#sphx-glr-auto-examples-applications-plot-rank-filters-py
14. OpenCV: OpenCV-Python Tutorials [Internet]. docs.opencv.org. [cited 2022 Jan 10]. Available from: https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html
15. Morphological Filtering — skimage v0.20.0.dev0 docs [Internet]. scikit-image.org. [cited 2022 Jan 10]. Available from: https://scikit-image.org/docs/dev/auto_examples/applications/plot_morphology.html#sphx-glr-auto-examples-applications-plot-morphology-py
16. Overfitting vs. Underfitting: What Is the Difference? [Internet]. 365 Data Science. 2021 [cited 2022 Jan 10]. Available from: https://365datascience.com/tutorials/machine-learning-tutorials/overfitting-underfitting/
17. Template Matching — OpenCV-Python Tutorials beta documentation [Internet]. opencv24-python-tutorials.readthedocs.io. [cited 2022 Jan 10]. Available from: https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_template_matching/py_template_matching.html#template-matching-with-multiple-objects
18. Thomas LSV, Gehrig J. Multi-template matching: a versatile tool for object-localization in microscopy images. BMC Bioinformatics. 2020 Feb 5;21(1). [cited 2022 Jan 10]. doi:10.1186/s12859-020-3363-7
19. Sze V, Chen Y-H, Yang T-J, Emer JS. Efficient Processing of Deep Neural Networks: A Tutorial and Survey. Proceedings of the IEEE. 2017 Dec;105(12):2295–329. [cited 2022 Jan 10]. Doi: 10.1109/JPROC.2017.2761740.
20. Pfirrmann CWA, Metzdorf A, Zanetti M, Hodler J, Boos N. Magnetic Resonance Classification of Lumbar Intervertebral Disc Degeneration. Spine. 2001 Sep;26(17):1873–8. [cited 2022 Jan 10]. Doi: 10.1097/00007632-200109010-00011

APÊNDICE A - Softwares e suas utilizações

3DSlicer	Pacote de software gratuito e de código aberto para análise de imagens e visualização científica.
Anaconda	Plataforma de distribuição de pacotes Pythons e R.
GitHub	Plataforma de hospedagem de código-fonte e arquivos com controle de versão.
Itk e itk-SNAP	Software de visualização de imagens médicas, permitindo manipulação e segmentação, incluindo algumas ferramentas de automação.
MATLAB	*Matrix Laboratory* – software proprietário para computação numérica.
Pip e Conda	Instaladores de pacotes Python e R.
Python	Linguagem de programação.
Jupyter e Jupyter	Notebook Ambientes de desenvolvimento interativos, o segundo com interface web.
ImageJ	Processamento de imagens científicas.

APÊNDICE B

Modificado da Referência 7.

```
# dcm_qt_tree.py
# Copyright (c) 2013 Padraig Looney
# This file is released under the
# pydicom (https://github.com/pydicom/pydicom)
# license, see the file LICENSE available at
# (https://github.com/pydicom/pydicom)

import pydicom
import sys
from PyQt5 import QtGui
from PyQt5.QtWidgets import QApplication, QTreeView
import collections

class DicomTree(object):
    def __init__(self, filename):
        self.filename = filename

    def show_tree(self):
        ds = self.dicom_to_dataset(self.filename)
        dic = self.dataset_to_dic(ds)
        model = self.dic_to_model(dic)
        self.display(model)

    def array_to_model(self, array):
        model = QtGui.QStandardItemModel()
        parentItem = model.invisibleRootItem()
        for ntuple in array:
            tag = ntuple[0]
            value = ntuple[1]
            if isinstance(value, dict):
                self.recurse_dic_to_item(value, parentItem)
            else:
                item = QtGui.QStandardItem(tag + str(value))
                parentItem.appendRow(item)
        return parentItem

    def dic_to_model(self, dic):
        model = QtGui.QStandardItemModel()
        parentItem = model.invisibleRootItem()
        self.recurse_dic_to_item(dic, parentItem)
        return model

    def dataset_to_array(self, dataset):
        array = []
```

```
    for data_element in dataset:
        array.append(self.data_element_to_dic(data_element))
    return array

def recurse_dic_to_item(self, dic, parent):
    for k in dic:
        v = dic[k]
        if isinstance(v, dict):
            item = QtGui.QStandardItem(k + ':' + str(v))
            parent.appendRow(self.recurse_dic_to_item(v, item))
        else:
            item = QtGui.QStandardItem(k + ': ' + str(v))
            parent.appendRow(item)
    return parent

def dicom_to_dataset(self, filename):
    dataset = pydicom.dcmread(filename, force=True)
    return dataset

def data_element_to_dic(self, data_element):
    dic = collections.OrderedDict()
    if data_element.VR == "SQ":
        items = collections.OrderedDict()
        dic[data_element.name] = items
        i = 0
        for dataset_item in data_element:
            items['item ' + str(i)] = self.dataset_to_dic(dataset_item)
            i += 1
    elif data_element.name != 'Pixel Data':
        dic[data_element.name] = data_element.value
    return dic

def dataset_to_dic(self, dataset):
    dic = collections.OrderedDict()
    for data_element in dataset:
        dic.update(self.data_element_to_dic(data_element))
    return dic

def display(self, model):
    app = QApplication.instance()

    # create QApplication if it doesnt exist
    if not app:
        app = QApplication(sys.argv)
    tree = QTreeView()
    tree.setModel(model)
    tree.show()
```

```
        app.exec_()
        return tree


def main():
    filename = sys.argv [1]
    #uncomment next line if you want to request the input for path
    #filename = input()
    dicomTree = DicomTree(filename)
    dicomTree.show_tree()


if __name__ == "__main__":
    main()
```

APÊNDICE C

Modificado de Refeferência 5

```
#Sistema operacional e manipulação de caminhos
import os, glob
#Biblioteca para matemática
import numpy as np
#Biblioteca com funções para leitura do DICOM
import pydicom as dicom
#Biblioteca para plotagem
import matplotlib.pyplot as plt
import matplotlib.image as mpimage
#Biblioteca scientifica para manipulação de imagem - alias do scikit-image
from skimage import exposure
import skimage.morphology as morp
from skimage.filters import rank
#Biblioteca para manipular as variáveis do tipo data
from datetime import datetime
from ipywidgets import interactive, interact, widgets, Layout, Button, Box
from IPython.display import display
# turn off annoying pink warnings
import warnings
warnings.filterwarnings('ignore')

# save optimized image array to global variable so other functions can use it
global img_rescale_interactive, image_name_global

def contrast_stretch(image_name, percentile_lo, percentile_hi):
    image_name_global = image_name
    p_lo, p_hi = np.percentile(eval(image_name), (percentile_lo, percentile_hi))
    img_rescale = exposure.rescale_intensity(eval(image_name), in_range=(p_lo, p_hi))
    img_rescale_interactive = img_rescale
    plt.figure(figsize=(6, 6), dpi=100)
    plt.imshow(img_rescale, cmap=plt.cm.gray)
    plt.show()

#Carrega a imagem na variável imagemDicom
imagemDicom =
dicom.read_file("/Users/viniciusmartinsvilela/PycharmProjects/basicFunctions/CT_Coluna_Lombar/IM-0001-0172.dcm")

w = interactive(contrast_stretch, image_name="imagemDicom.pixel_array",
percentile_lo=(1,100,.5), percentile_hi=(1,100,.5))

display(w)
```

APÊNDICE D - Referencias para os mapas de cores disponíveis no Matplotlib