Universidade de Aveiro
2022

André Ribeiro Almeida

Compressão de imagem médica para arquivos de alto desempenho

Medical imaging compression for high-performance storage systems

Universidade de Aveiro
2022

André Ribeiro Almeida

Compressão de imagem médica para arquivos de alto desempenho

Medical imaging compression for high-performance storage systems

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor Carlos Manuel Azevedo Costa, Professor Associado com Agregação do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

o júri / the jury

presidente / president	Prof. Doutor Augusto Marques Ferreira da Silva Professor Associado da Universidade de Aveiro
vogais / examiners committee	Prof. Doutor Carlos Manuel de Azevedo Costa Professor Associado com Agregação da Universidade de Aveiro
	Prof. Doutor Rui Pedro Sanches de Castro Lopes Professor Coordenador do Instituto Politécnico de Bragança

agradecimentos

Agradeço aos meus orientadores, professor doutor Carlos Costa e doutor Eduardo Pinho, pelo consistente apoio pelas reuniões e e-mails durante o ano letivo. Valorizo muito o tempo dispensado para me assistir em todas as minhas questões.

Quero também te agradecer, Mariana, pelo inesgotável apoio, durante todas as minhas dúvidas e incertezas. És o meu modelo de pessoa, que me ajudas a crescer como estudante, através do teu apoio tal como da tua disciplina e desempenho como aluna, e agora, mestre cientista.

Agradeço imenso a toda a minha família, nomeadamente os meus pais e irmã, que me sempre ajudaram neste percurso académico a todos os pontos, mesmo nos mais difíceis. Sem eles não teria estas excelentes oportunidades académicas.

Finalmente, agradeço a todos os meus amigos, com quem passei bastantes momentos de lazer, durante todo o meu percurso universitário.

palavras-chave

Imagem digital, imagem médica, compressão de imagem, compressão de imagem, formato de ficheiro de imagem, métricas, DICOM, PACS, JPEG XL, AVIF, WebP, qualidade de imagem.

resumo

Os sistemas de informação e o assunto médico são dois temas difundidos que se entrelaçam para que a ajuda médica se torne mais eficiente. Essa relação deu origem ao PACS e ao padrão internacional DICOM direcionado à organização da informação médica digital. O conceito de compressão de imagem é aplicado à maioria das imagens em toda a web. Os formatos de compressão usados para imagens médicas tornaram-se desatualizados. Os novos formatos desenvolvidos nos últimos anos são candidatos a substituir os antigos nesses contextos, possivelmente potencializando o processo. Antes de serem adotados, deve ser realizada uma avaliação que valide sua admissibilidade. Esta dissertação revisa o estado da arte dos sistemas de informação de imagens médicas, nomeadamente os sistemas PACS e a norma DICOM. Além disso, são abordados alguns tópicos de compressão de imagens, como as métricas para avaliação do desempenho dos algoritmos, finalizando com um levantamento de três formatos modernos: JPEG XL, AVIF e WebP. Foram desenvolvidos dois projetos de software, onde o primeiro realiza uma análise dos formatos com base nas métricas, utilizando conjuntos de dados DICOM e produzindo resultados que podem ser utilizados para a criação de recomendações sobre o uso do formato. A segunda consiste numa aplicação capaz de codificar e descodificar imagens médicas com os formatos abordados nesta dissertação. Essa prova de conceito funciona como um arquivo de imagens médicas para armazenamento, distribuição e visualização de dados compactados.

keywords

Digital imaging, medical imaging, image compression, image compression, image file format, metrics, DICOM, PACS, JPEG XL, AVIF, WebP, image quality.

abstract

Information systems and the medical subject are two widespread topics that have interwoven so that medical help could become more efficient. This relation has bred the PACS and the international standard DICOM directed to the organization of digital medical information. The concept of image compression is applied to most images throughout the web. The compression formats used for medical imaging have become outdated. The new formats that have been developed in the past few years are candidates for replacing the old ones in such contexts, possibly enhancing the process. Before they are adopted, an evaluation should be carried out that validates their admissibility. This dissertation reviews the state of the art of medical imaging information systems, namely PACS systems and the DICOM standard. Furthermore, some topics of image compression are covered, such as the metrics for evaluating the algorithms' performance, finalizing with a survey of four modern formats: JPEG XL, AVIF, and WebP. Two software projects were developed, where the first one carries out an analysis of the formats based on the metrics, using DICOM datasets and producing results that can be used for creating recommendations on the format's use. The second consists of an application that encodes and decodes medical images with the formats covered in this dissertation. This proof-of-concept works as a medical imaging archive for the storage, distribution, and visualization of compressed data.

Contents

List of Tables

List of Figures

List of Listings

Acronyms

PACS Picture Archiving and Communications Systems

DICOM Digital Imaging Communications Standard

CT Computer Tomography

MG Mammography

SM Slide Microscopy

CS Compression Speed

DS Decompression Speed

CR Compression Ratio

SSIM Structural Similarity

PSNR Peak Signal-to-Noise Ratio

MSE Mean Squared Error

JXL JPEG XL, Joint Photographic Expert Group format, XL version

AVIF AV1 Image File Format

BPS Bits per sample

SPP Samples per pixel

1. Introduction

1.1. Overview

With the advent of information technology, the medical field has resorted to its applications. Medical Imaging is one of the products of that association. Images acquired for diagnosis are stored and communicated between medical equipment using Picture Archiving and Communication systems (PACS).

As the name suggests, PACS comprise a set of components that allow the storage and transmission of images acquired by medical devices, such as x-ray, tomography, or ultrasound scanners, and displayed in a visual form in distinct network places.

The Digital Imaging Communications in Medicine (DICOM) standard defines a structure for files containing medical images and related text data. It also defines other protocols for the communication of this medical information, including textual reports and waveforms.

Digital image compression is a very successful practice when a huge volume of images needs to be stored in a central repository or transmitted throughout the network. It consists of finding ways to reduce the size of the image while preserving the image quality. There are always two processes that are involved in this modus operandi: encoding and decoding. The encoders take raw images, or image files with a different format, as input and produce a coded binary stream that is pre-defined by a file format standard, such as jpeg and png. Those formats employ a set of what is called encoding techniques that carry out the actual compression of the image information to an encoded form, reducing its size. This, in practical terms, increases the effectiveness of storage and communication processes in PACS environments. The decoders do the reverse of their encoder counterpart, parsing the image into another format or raw form, the latter parsing being vital for the displaying of the images' contents. These tools are sometimes referred to as a single unit, called a codec, a term that combinates the words encoder and decoder.

There are two general modes by which the image can be compressed: lossless and lossy. Lossless compression does not change the original image's contents, being the decompressed data exactly equal to the original. The lossy mode takes advantage of the lack of human perception to notice small changes, by slightly changing the images' contents to improve the ratio of compression. The latter approach, despite losing quality causes degradation to the user experience in some cases and diagnosability in medical

contexts, it achieves significant improvements to the compression performance, reducing the image file size even more.

The field of medical imaging has much to gain when employing this practice because when the PACS storage capacity is increased with free codecs, costs are reduced. Another advantage is that the image transmission times are also reduced, a critical issue in real-time remote visualization of studies. Nowadays, the last aspect is even more relevant with the proliferation of web-based viewers that allows access to medical images from any location. However, today's medical imaging systems still rely on more dated compression methods, such as JPEG 2000, that provide generic compression with high computational cost or low space utilization, or no compression at all. Conversely, conventional image compression state-of-the-art has been rapidly evolving in recent years. For instance, WebP, AVIF, and JPEG XL (JXL) codecs are some of the image-encoding formats that promise unprecedented high-fidelity compression capabilities, and the potential to become ubiquitous on the Web.

Herewith, it is important to assess the improvements of the codecs' capabilities in the world of compression, therefore allowing a further reduction of costs related to image storage and transmissions.

1.2. Goals

This dissertation aims to assess the performance of recent Web compression codecs in the field of medical imaging modalities, aiming for a further reduction of costs related to image storage and transmissions in internet scenarios. The work will evaluate the recent state-of-the-art image compression formats over distinct medical imaging modalities and anatomical regions. In this process, distinct quality metrics will be used, such that a quantitative set of results can be extracted, which allows the comparison of the competing codecs and the establishment of recommendations for production.

In sequence, a medical imaging archive will be developed, using the studied codec formats. This proof-of-concept software solution will allow the bidirectional transcoding of compression formats, namely with DICOM uncompressed formats, and the provision of standard interfaces to ensure interoperability with legacy systems.

1.3. Outline

Chapter 2 carries out a literary review of PACS, DICOM, Dicoogle, and state-of-the-art image codecs. The latter topic covers basic concepts related to image

compression, followed by an overview of the metrics that can quantitatively evaluate different characteristics of the encoding process. Lastly, still, on the same topic, the recent formats are reviewed.

Chapter 3 covers the evaluation of the modern formats using some of the performance metrics reviewed in chapter 2 and python to develop the benchmarking framework that carries out the experimental evaluation.

Chapter 4 covers the software project developed as a proof of concept to the studied formats, comprising an extension to the Dicoogle open-source project.

2. State of The Art

2.1. PACS

2.1.1. History

The concept of PACS started appearing around the decade of 1980 [1]. It began as a service to the Computer Tomography (CT) modality, with software replacing film for patient reports [2]. Over time, these systems started developing networks for communications, first by serial link but then using the Ethernet technology used nowadays. In the last, sizable local area networks (LANs) have been widely used for information transmission, the so-called PACS networks [2]. More recently, the PACS-cloud solution has become very popular in shared inter-institutional processes like, for instance, regional repositories and teleradiology. Figure 1 shows a basic layout of the components of a generic PACS system.

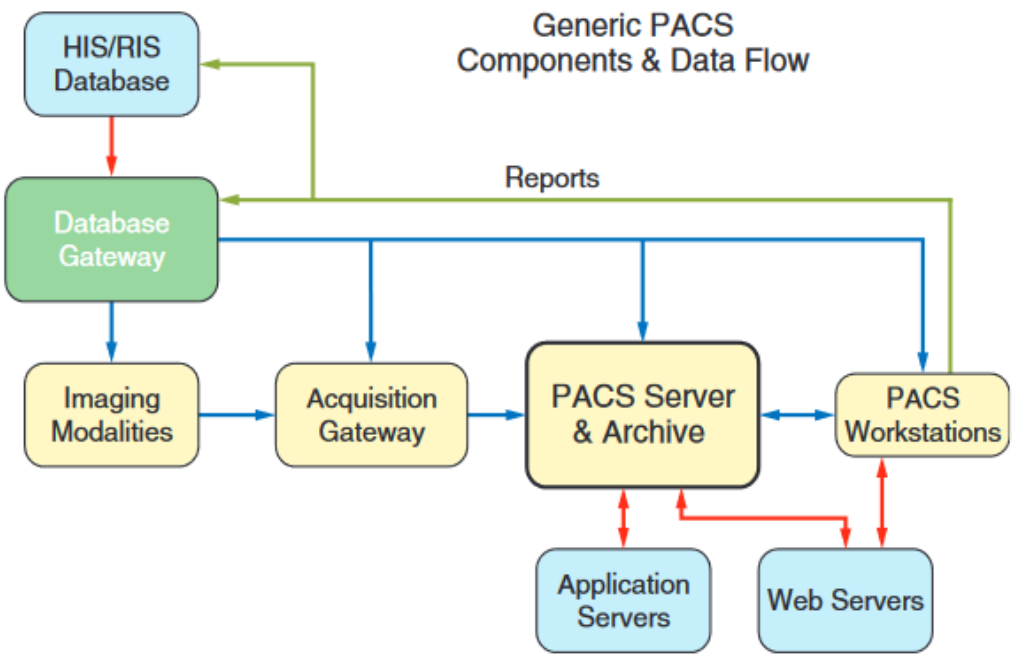


Figure 1 - Generic PACS Components and Data Flow Diagram [3]

2.1.2. Overview

PACS can be seen as both a networking and storage system [1]. The concept goes hand in hand with the DICOM standard, the latter being a series of guidelines for information formatting, storing, and communicating over a (PACS) network [4]. These networks are instrumental because they enable the sharing of the studies anywhere within its reach [2], invigorating the process of transitioning between image acquisition and diagnosis, and allowing growth in productivity in such environments. As a storage service, it is very beneficial because it attends to the problems arising from space limitations, which become more prominent in the era of big data [1].

From a more traditional perspective, PACS consists of a central server holding a database of medical images [1] and respective metadata. The data can be queried from physicians throughout the hospital network. Nowadays, they can be in the Cloud and provide services to several institutions globally.

Another essential feature is the capability to communicate with other healthcare information systems, such as radiology information systems (RIS) and hospital information systems (HIS). This way medical imaging data can be linked with the patient electronic record using standard identifiers [1].

2.1.3. Workflow

PACS workflow encompasses the steps from the acquisition stage down to the visualization or diagnosis. We will discuss these phases next, as in [5], [6].

The acquisition stage is the process of producing the images with the special equipment associated with the specific medical imaging modality. It is worth noting that it is of utmost importance that the images maintain the most information possible, allowing an optimal diagnosis process. This directly correlates to massive storage requirements.

The second step is the distribution process, where the generated images are uploaded from the acquisition equipment into the PACS storage archive. Raising the bandwidth can mitigate the problem, but there are other ways to address the problem, such as improving data compression. The latter approach can be done in a few ways, not only from the standpoint of image compression but also file compression, a process adopted in [7].

The final step of the workflow is visualization. At the workstations, the physicians retrieve the specific study, that is to be the subject of diagnosis, from the PACS archive, and carry out its analysis. This represents the main bottleneck in the PACS workflow,

where, especially at low connections, the transmission can take significant time, which harms the productivity of the physician. The images are sometimes subject to compression, with eventual loss of quality, but the effect, however, cannot be more severe than the “just noticeable difference” threshold, from the viewpoint of the professional. This is because such deterioration can cause misdiagnosis, and thus, lead to injury to patients [8]. When Computer Aided Diagnosis is introduced for image analysis, the acceptable level of quality loss can be even lower [4]. Image compression will be explained further in this chapter.

2.1.4. Architectures

The organizational structure of a PACS can follow multiple models, as proposed by H. K. Huang [3].

Firstly, there is the classic standalone type, where images are acquired and directly stored in a central repository, which is then broadcasted to all workstations in a store-and-forward approach [5]. This has the advantage of the information being rapidly served to the users but suffers from an impractical content passing problem, meaning that multiple workstations would receive all the studies instead of only the ones that need assessment (in that workstation). Another problem with this approach is the congestion of the network bandwidth since medical imaging data comprises a load of sizable information, and the workflow must not suffer unnecessary delays.

A more common approach in our days is the client-server architecture, where the studies are forwarded to the storage repository after the acquisition process, as in the stand-alone. The difference lies in the distribution phase, where only the clients that specifically request the medical data can read it. This has the immediate advantage of reducing the bandwidth of the connections since the transmission of studies to uninvolved workstations will no longer take place. An indirect perk exists as well, which is the viability of implementing a procedure of access control, enabling only authorized personnel to visualize the medical data.

Finally, an additional methodology has been taking place over the recent years, the web-based architecture being very used in modern Cloud installations. Aided by DICOM’s guidelines, it uses web technologies, such as browsers, and web services defined in the standard, such as QIDO to query, WADO to access, and STOW to store the information. The advantages of this model are the ease of access to the data from any machine with a browser and a web connection.

2.2. DICOM

2.2.1. Introduction

With the bloom of Medical Imaging Services, a problem arose: image equipment provided and handled only with a specific proprietary data format. With this, the problem consisted of incompatibility between medical departments, because each could have different service providers. This inflexibility proves as a disadvantage, for example, when a patient has relocated to a different location and starts using the services of another hospital, which can have different, incompatible, imaging service providers. A physician might need the historical files of the person to carry out a more accurate diagnosis [4].

The DICOM standard was created to solve these compliance issues. In 1983, the ACR-NEMA committee started working on some first drafts, forming the ACR-NEMA 300-1985 standard or ACR-NEMA 1.0, released in 1985, and perfected with an improved version, the ACR-NEMA 2.0, in 1988. Still, this second version was limited from a communication standpoint. Moreover, the networking technologies started evolving, further motivating one more increment to the norm – the ACR-NEMA 3.0 or DICOM 3.0. From that point on, no major upgrade such as that has taken place, and DICOM remained DICOM 3.0. Notwithstanding, it has continued steadily evolving with the publishing of documents called supplements and the addition of parts to the standard that covers new areas [4].

2.2.2. Information Model

2.2.2.1. DICOM Information Model

DICOM defines a general structure and organization for the data based on the real world. It consists of a hierarchical, patient-oriented model. Thus, the top-level division separates each patient with its data. For each patient, we have multiple studies executed upon him. In other words, a process of image acquisition for diagnosis, stored, a posteriori, in archives. In the third level of information, for each study, we have a set of series, or a collection of acquired images, each one containing several images, consisting of the last level of the information hierarchy [9].

2.2.2.2. Information Object Definition

An IOD is a data structure for defining DICOM objects of a specific modality, i.e. work like data templates. There are multiple IOD, one per modality (e.g. US, CT, or XA), and can support single or multiple real-world objects, or entities, as well as metadata on the image acquisition process. There are two subtypes of IODs – composite IODs, which represent data on several entities of the DICOM Model of the Real World. Normalized IODs, on the contrary, represent data from a single entity [9].

2.2.3. Data Encoding

If DICOM IODs are blueprints for data structure definition, DICOM object instances comprise the real-world data itself, from a practical viewpoint [9].

These objects can be managed in memory and stored in specialized binary files named DICOM files (*.dcm). These files' format is as follows: a preamble, involving 128 bytes set to 0, followed by a string of four characters 'DICM' encoded in ASCII. Lastly, a DICOM object encompasses a list of DICOM elements – the content of the file [10].

A DICOM object is divided into data elements, each one organized in a TLV structure – tag, length, and value. The *Tag* section contains a unique identifier, which links the element to a real-world object property, defined in the DICOM dictionary. The *Length* part contains the size, in bytes, of the *Value* of the element. This *Value* contains the actual data regarding the property of the real-world entity. A DICOM element can hold data, for example, for a patient's name, e.g.: The *Tag* value is (0010,0010), *Length* is 14 if the *Value* is "Marcelo Rebelo". Optionally, the element can contain an extra attribute addressed as VR, meaning *Value Representation*, which informs about the type of data in the *Value* section. This attribute is present when the DICOM file is marked as "Explicit VR" in the *Transfer Syntax* element [11]. If defined as "Implicit VR", the DICOM data dictionary must be consulted for every element presented in the TLV structure.

The content section of the DICOM file can be generally divided into 2 main segments – the files' metadata and the data set. This metadata is the information that describes the second segment. The data set is the content instant described by the respective SOP Class, which union the IOD with what is called a DICOM Service Element, present in the metadata [10]. Also worth mentioning is a DICOM element that exists in this segment, which is the "Transfer Syntax" element with the tag (0002,0010). This element defines how the remainder of the object is encoded – byte order (little/big endian); whether it is explicit/implicit VR or information on compression. This element is useful for communication between DICOM nodes, which will be addressed later in this chapter [11].

2.2.4. Communications

In a PACS network, DICOM defined each node as an AE (Application Entity), identified by an AETitle, the node's addressing string with a maximum length of 16 bytes [12].

The communication paradigm is of a client-server relationship type, where the term for the client is Service Class User (SCU) and for the server is Service Class Provider (SCP) [12]. After an association is established (described in 2.2.4.3) the SCU requests services from an SCP in the same manner that a workstation requests retrieval of data from a storage archive – in this example, they are SCU and SCP, respectively [12].

2.2.4.1. DIMSE commands

The standard defines a set of commands that are essential for the correct functionality of a PACS network. They define various communications operations [13]. Of those, there are two types, each for a specific type of IOD – normalized and composite. Hence, there are the (DIMSE) normalized and composite services [12], which are listed in tables 1 and 2.

Composite services	Operations with composite IODs
C-Get	Retrieve a specific object by its UID
C-Store	Push an object to a remote archive
C-Move	Transfer an object from one AE to another
C-Find	Query an archive by the object's attributes
C-Echo	Verification service

Table 1 - DIMSE Composite Services List [13]

Normalized services	Operations and notifications with normalized IODs
N-Get	Retrieve information
N-Set	Modify information

N-Action	Perform an action
N-Create	Create an instance of an SOP Class
N-Delete	Delete an instance of an SOP Class
N-Event-Report*	Report an event to another AE (DIMSE Service User)

Table 2 - DIMSE Normalised services [13]

2.2.4.2. SOP Classes

DICOM has defined a merged definition of information, called the Service-Operation Pair (SOP) Class. This represents an association between an IOD and a DIMSE service [14], defining data in a certain context of the PACS workflow. One such example is “CT Scan Storage” where CT Scan is the IOD and storage is the DIMSE service [15].

Every DICOM file is associated with one SOP Class. Each DICOM object is also uniquely identified by an SOP instance, defined by a unique ID (UID) [15].

2.2.4.3. Associations

If two nodes in a DICOM-compliant PACS network (DIMSE service users) wish to communicate, they establish what is called a DIMSE association, open with the A-Associate service, and closed with one of the following: A-Release or A-Abort [16].

Those associations are divided into two phases. The first is the association establishment and negotiation, where a presentation context is sent from the requester to the requestee and the receiver chooses the supported transfer syntax (as addressed in 2.2.3), such as byte-order or the existence of compression (and which method, if existent) [13]. The second phase is the actual data transfer.

2.3. Dicoogle

Dicoogle software is an open-source PACS archive [17], developed at the University of Aveiro. This software runs under a browser and thus follows the web-based PACS architecture. As an archiving software, it evolves from the traditional database model with the capability of providing extensible indexing and retrieval functionalities [18], among others, which will be referred further ahead.

The architecture paradigm of this project is modular, providing a gateway through which new functionalities can be implemented [19], independently of the core platform. Such a trait is possible thanks to the existence of plugins, thanks to the Dicoogle Software Development Kit (SDK) [17], with the Dicoogle Learning Pack providing a starting ground for plugin development [17]. The SDK also enables this development, defining an interface contract that the developers must abide by when creating their plugins [17]. There are multiple types of these extensions, which are the storage, index, query, web service, and, web UI [17]. In a Dicoogle instance, the core interconnects and manages all existing plugins.

From those plugins, we will explore a bit further the storage and web-service plugins.

The storage plugins deal with storing data on the platform the instance is running on. It defines where and how the DICOM files are written or read. The DICOM files can be stored in distinct file-system places or even cloud-storage services. They provide an abstraction layer for such operations from the core service and can also be augmented with features such as encryption, anonymization, and compression. These plugin functionalities are triggered when a C-Store is issued to the archive [17].

The web-services plugins allow a vast scope of functionalities to be extended upon Dicoogle. There are two subtypes of this plugin: one that is developed through Jetty Servlets; another that is REST based. Where the former is more powerful in terms of versatility, the latter can be more limited, although easier to learn, in contrast to the alternative [17].

Dicoogle was chosen because it is one of the best choices among its competitors for the best PACS archiving open-source software, as reviewed by Lebre R. et al. [17].

2.4. Image Compression Overview

Images, when concerning computer science, are generally regarded as matrices, or two-dimensional arrays (or three, in the case of volumes stored in multi-frame images), in which each element is called a pixel. Each pixel is usually composed of several scalar properties, or channels, depending on the color space of the image [20]. If the colorspace is RGB (Red Green Blue), there are 3 channels, one for each color to store their intensity on the pixel. There are others used for image representation, such as HSI, CYM, etc [20].

In their original, uncompressed form, they can easily reach ample dimensions, especially in the medical imaging field, which can reach several megabytes or gigabytes

in each acquisition [4], [8]. Additionally, in that context, they are frequently acquired, altogether being costly in matters of disk space used [8], which can create bottlenecks in the storage and transmission processes in the PACS workflow. Thus, the slowness in the distribution phase can cause delays in the hospital workflow, as well as create a heavier load, space-wise, upon the archiving equipment [8].

There are two types of methods to compress images: lossless, where there is no loss of quality of the decompressed image compared to the original one [4]; lossy methods degrade the image quality, trading it for a significantly better CR (compression ratio) [4]. The latter takes advantage of the traits of the HVS to identify which fragments of the image can be safely removed with the least possible toll on appeal or fidelity [20].

Images in the medical sphere have a certain number of details that are crucial for a correct diagnosis, hence, there is little room for fidelity trade-off for compression ratio gain. In other words, if lossy compression is applied to these images, the level of compression must be near-lossless, because of the low margin of tolerance for loss of image fidelity [4]. This allows us to reduce the image size further than lossless capabilities, while mitigating the loss of any kind of crucial information that would hurt the validity of the associated acquisition, or, in the worst case, lead to misdiagnosis [8].

Traditional compression formats used for medical imaging are dated. Currently used methods consist of JPEG2000, JPEG, JPEG-LS, and some others lossy or lossless, or none at all [7], [8]. So, it is essential to evaluate state-of-the-art formats, such as JPEG XL, AVIF, HEIC, and WebP for their viability in this environment. In particular, because the modern codecs start to be natively supported by web-browser clients that are used by DICOM Web viewers.

Followingly, I will broach the topic of compression performance metrics and then the modern compression formats. Those formats that will be introduced do not include proprietary formats such as HEIC [21], which would hinder the inter-compatibility if introduced to the medical informatics environment. This is because it is a proprietary format, which makes it less accessible than an open format [4].

2.4.1. Performance Metrics

Following is a table resuming the main quantitative metrics that measure the performance of compression and quality lost in codecs applying the lossy strategy.

Metric - Description	Formula
Compression ratio (CR): It refers to how much smaller the compressed image is than the original one.	"Compression Ratio = Original Size/Compressed Size (applicable when both the original image and compressed image are of the same size)."
Time (of compression or decompression): Meaning the time taken for the image to be decoded or encoded. Can be expressed as a unit of time (seconds, milliseconds, etc.), or as a relational unit providing a ratio between the image size and the time of (de)compression, or (de)compression speed (DS/CS, respectively), for example, 4MP/s.	NA
"**Mean Square Error (MSE):** The mean square represents the total squared error value between the compressed and the original image."	$$\mathrm{MSE} = \frac{1}{\mathrm{MN}} \sum_{i=1}^{M} \sum_{j=1}^{N} [X(i,j) - X'(i,j)]^2$$ where $X(i,j)$ denotes the original image, $X'(i,j)$ denotes the decompressed image, and M, N represents the size of the image
"**Peak Signal-to-Noise Ratio (PSNR)**: The PSNR is an index of quality degradation measurement based on pixel-wise differences (as the equation shows, it is based on the mean squared error). The measurement unit is the decibel (dB), where the values range from '[0, +inf[' (from zero to infinity).	$$\mathrm{PSNR} = 20 * \log_{10}(255/\mathrm{sqrt}(\mathrm{MSE}))$$
"**Structural Similarity Index (SSIM)**: The structural similarity index is a metric used to	$$\mathrm{SSIM}(X, X') = \frac{(2\mu_X \mu_{X'} + C_1) + (\sigma_{XX'} + C_2)}{(\mu_X^2 + \mu_{X'}^2 + C_1) + (\sigma_X^2 + \sigma_{X'}^2 + C_2)},$$

find the quality loss of the image based on the perceived loss of fidelity to the image. It is, thus, based on the HVS, with its approach being the comparison of the structural differences between the two images. That means the chroma differences rather than luma. [22]	where **X**, **X'** refers to the original and decompressed image, μ_X, $\mu_{X'}$ denotes the mean, and σ_X^2, $\sigma_{X'}^2$ denotes the variance of the images.

Table 3 - Quality and fidelity loss measurement formulas for lossy compression [22]

2.4.2. JPEG XL

JPEG XL is a compression format developed by the Joint Photographics Expert Group (JPEG) [23]. The project started in 2017 and the format is, as of the first of October, approved (ISO/IEC 18181-X) [24].

It is a generalized compression format, targeting a large scope of types of images, be it photographic images, synthetic (computer generated), hybrid (between the aforementioned two), or animated (multi-frame) [23].

Also, it permits parallel, progressive, and partial decoding [25] which is useful for defining the ROI of any portion of the image.

A relevant feature of this codec is transcoding losslessly a JPEG image into JPEG XL codestream, where the varDCT mode (explained below) can be used [26]. This has two advantages. Firstly, it reduces the size of the codestream by 16% to 22% [26]. Secondly, it eases the migration of older images that are already stored under the older JPEG format.

The algorithm provides two encoding modes, depending on the input image [26].

The first is called modular mode. This is designed for artificially generated images and allows lossless as well as lossy compression of such [23].

The second and main option for encoding is generally directed at photographic images, which takes advantage of the HVS for compression [23]. It uses the JPEG DCT but allows greater adaptability, with the difference in the blocks' size and shape being more flexible (varDCT): it can take the form of a square as well as a rectangle, and the size of such blocks is minimum 2 and maximum 256, regarding both the width and height [26].

The encoding options are as follows: `--distance=maxError` directs the encoding efforts towards a target quality loss, from mathematical to visually lossless (0.0 to 1.0, respectively); `target_size` or `target_bpp` determines the intended target output file size

in bytes or BPP, respectively; `--effort` determines, in a scale from 1 to 9, the encoding effort, from fastest / least effort (1) to slowest and more effort (9). On the last note, reminding the fact that JPEG XL takes advantage of the parallel computing power present in today's multi-core computers, the `--num_threads` option is also present, offering a choice on how many worker threads should be used in the compression operation [27].

In general terms, the parameters described above offer compression with a target quality, size, and/or encoding time. Tuning these parameters will offer a choice for optimizing the more important parameters based on the custom requirements. In the case of medical imaging, for example, one would prioritize quality as the foremost concern [4].

2.4.3. AVIF

The AVIF compression format is a royalty-free [21], open-source and free-to-use codec which was developed by the Alliance for Open Media (AOM). It is derived from the AV1 video codec and uses the same core compression technique for images [21]. The compression can be carried out as lossless as well as lossy [21].

As an open, royalty-free format, AVIF can uphold the interoperability principle present in medical imaging environments, alongside the DICOM standard in such terms.

Along with its parent format, AV1 [21], it started being used along the grounds of the Netflix service [24], although spreading its support throughout the web, as depicted in fig. 1.

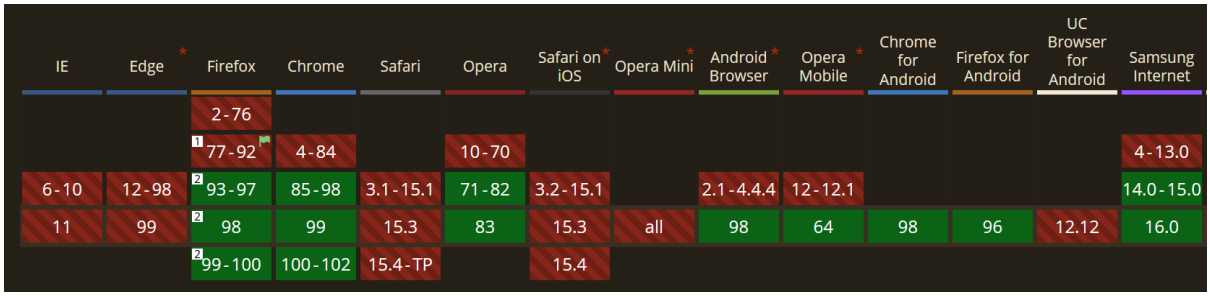


Figure 2 - AVIF Support among a set of the most popular browsers. Source: https://caniuse.com/avif. Data extracted on 12 March 2022.

The features that AVIF provides are as follows: support for High Dynamic Range (HDR); support for lossless and lossy decomposition; alpha channel; 8, 10, and 12 bits depth per sample [28]; 4:4:4, 4:2:2, 4:2:0 and more chroma subsampling [29]; and a maximum resolution of 65536x65536 [28].

There are available multiple AVIF encoders, such as cavif-rs [30], rav1e, libaom, and libavif [31]. For the cavif-rs encoder, the following lossy compression parameters can

be tuned: `--quality`, with values from 1 to 100; and `--speed`, from 1 to 10. The possibility of tuning these values enables the trade-off of quality or encoding time, respectively, thus customization can be made based on the specific requirements of the compression process [30].

2.4.4. WebP

The WebP image compression format was created by Google in 2010 based on the video compression format VP8 [32]. It provides both lossy and lossless compression, supported by most popular browsers, such as Google Chrome, Safari, Firefox, Edge, the Opera browser, and by many other tools and software libraries [32], thus being the most widely available among the referenced formats and hence the easiest to include in browser web-services for the global compatibility. The reference implementation of WebP is libwebp [33] for the library form, and cwebp and dwebp for the command-line codec toolset [34], [35]. It is a royalty-free codec [36].

Regarding the lossless compression mode, it uses the LZ77 and Huffman [37] techniques, alongside spatial and color predictors, with entropy coding, carried out next [38]. This mode uses already “seen” image fragments to reconstruct the image. The lossless output bitstream is overall 26% smaller than the PNG format [32].

One specification of this format is the maximum image dimensions of (16383 x 16383) [39].

The lossy mode uses predictive coding, based on the frame block prediction method of VP8, meaning the values in the neighboring pixels are used to predict the value of the target pixel, encoding the difference between the predicted and real values. As an evaluation, this mode produces 25 to 34% smaller images than the JPEG format for the same quality rating of the SSIM metrics index [32].

One drawback of the WebP format is the lack of support for a progressive decoding process [40], which would allow an image to be gradually made visible to the user.

Regarding the cwebp encoder [34], I will briefly overview the lossy parameters that allow setting tradeoffs for the compression process. The -size option allows choosing a target output file size, specified in bytes, that the encoder can aim for. Another option is the -psnr, allowing the user to specify the PSNR fidelity measurement, in decibels (dBs), that the encoder can aim for, for the target output. For both options mentioned previously, the tool will carry out multiple encoding iterations and try to approximate the parameters as much to the intended value as possible. The number of iterations to be done can be

chosen with the -passes option, with possible values from 1 to 10. The usage of a large value for this parameter results in a more time-consuming encoding operation, with the benefit of a more precise setting. With these basic options, the user can balance the image fidelity, size and encoding time to best fit his requirements.

More, and simpler, encoding options for the user include the `-m` option which is associated with the method of encoding, and the `-quality` parameter, which provides the user with a scale (from 1 to 100) that allows stipulating how much quality loss is acceptable for the compression. The method option allows setting the speed of encoding as faster or slower [34].

3. Benchmarking Modern Compression Formats

3.1. Introduction

Before considering the medical use of the recent formats, an assessment of their performance is in order. The codecs shall be weighed based on four metrics, mentioned in section 2.4.1: encoding and decoding speed (DS), compression ratio, and similarity index. For medical imaging, image feature preservation is paramount, so the similarity should be the highest possible.

For this experiment, python will be used to develop the benchmarking tool. A framework was planned and developed to carry out the quantitative assessment of the codecs and to assist with displaying the results in graph form.

3.2. Pipeline Architecture

As mentioned, this project was written in python, interpreter version `3.10.6`. The encoders and decoders used: `cjxl` and `djxl` provided by `libjxl` version `0.7.0` commit `ae95f45`; `cwebp` and `dwebp` version 1.2.1; `cavif-rs` v1.3.4 and `avif_decode` 0.2.2. The benchmarking framework's results, as will be shown later in this chapter, will pertain to the snapshot associated with the commit with hash ` be07cb4b1a7067fe963b8153b58ea1e57ae8d2ac`.

This pipeline is divided into 3 stages, each one being a python module. A diagram presented in figure 3 visually summarizes the pipeline.

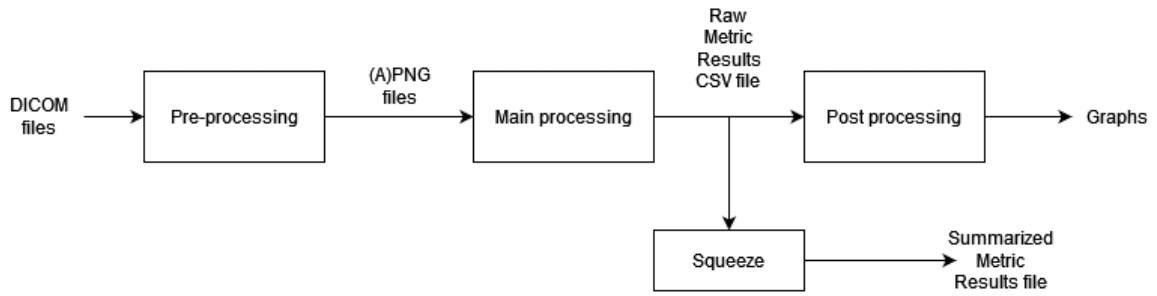


Figure 3 - Schematic of the main flow structure of the pipeline.

The first stage is referred to as the pre-processing one. This module takes a set of DICOM files as input, outputting, for each input file, a png file with the contents of the DICOM file's image in lossless form. The main purpose of the sub-program is this parsing operation, although another feature is contained within the same. The module saves the

png files with their names containing some crucial information that will be needed in the next stage of the pipeline of execution.

The second stage is the main processing step of the experiment. The program takes the output from the pre-processing as input. The key objective in this part is to execute the codecs and extract the evaluating metrics, which are the speed of operations, the ratio of compression, and the image distortion level. The results are saved as a CSV file in raw form.

The third and last stage is where the raw data outputted from the main processing module is processed. The results consist of graph representations of the CSV data. This stage is also addressed as post-processing.

3.2.1. Pre-processing

This stage takes a set of DICOM files as input and saves the respective images as 16-bit png or apng. A brief diagram of the structure of this stage is defined in figure 4.

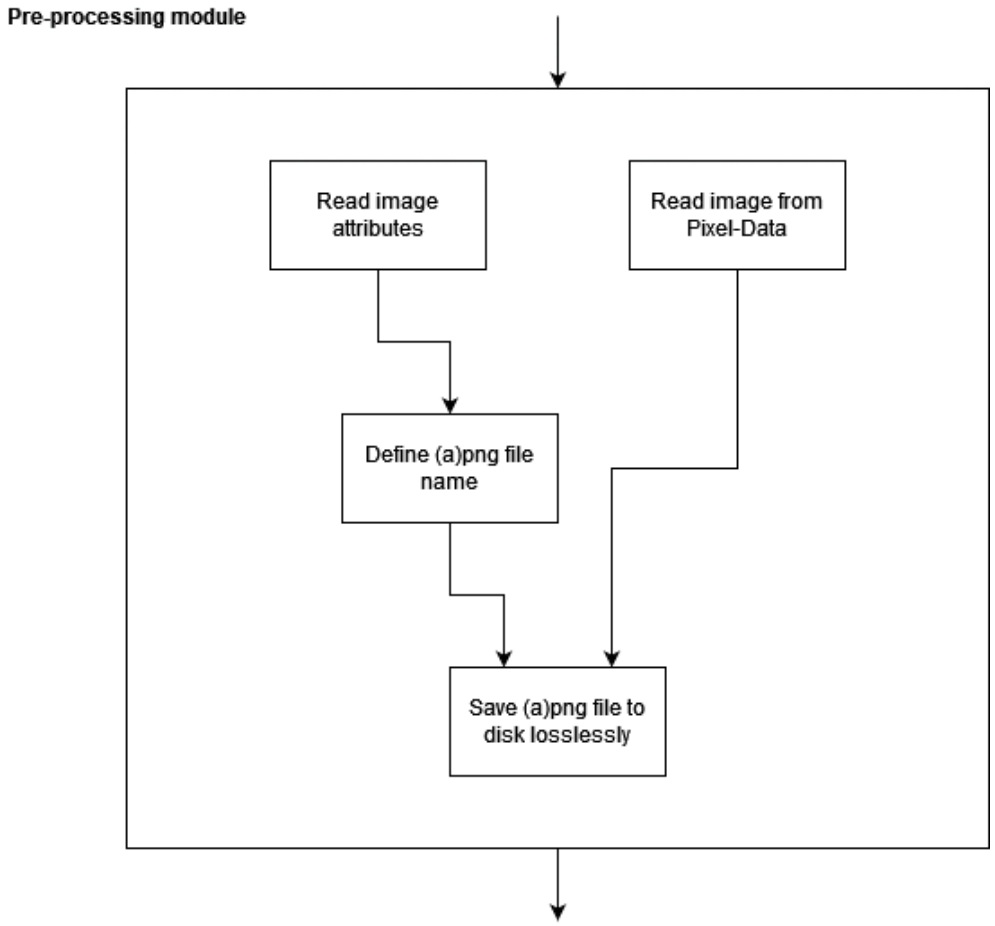


Figure 4 - Underlying operations of the pre-processing stage.

The output files are saved in a known path, to be later accessed by the next stage's script. Also, the names of the files include various attributes of the respective DICOM file:

1. The codeword (CS) of the modality, as set in the DICOM standard (0008,0060).
2. The body part being studied (0018,0015) - or NA if the attribute is null.
3. The color space [or photometric interpretation (0028,0004)] of the image.
4. The number of frames (0028,0008) of the images.
5. The bits per sample, bit-depth, or the number of bits stored (0028,0101) of the images' pixels.
6. The samples per pixel (0028,0002), or the number of color channels, of the image.
7. An ID to avoid repeated names with the same attributes above
8. A lossless format extension (.png when single frame, .apng for multi-frame)

All images have 16 bits per sample to normalize all images to a common bit depth. Also, for multiple channel images, if the color space is YBR_FULL_422, it is translated to RGB before being encoded and written to disk, as output (a)png files.

3.2.2. Main processing stage

This is the core stage of the pipeline, where the output from the previous one, the (a)png images, are encoded and decoded. Simultaneously, the metrics are computed to construct an analysis over each (de)compression process. Such metrics can be divided into three types: ratio, speed, and similarity.

The ratio of compression can be represented in two ways: by how many times the encoded image gets smaller over the original one; or by a measurement called average bits per pixel (bpp), where $compressed.bpp = original.bpp * \frac{compressed.size}{original.size}$. In this case, the former is used.

The speed metric evaluates the speed of compression as well as decompression, which can be represented by how much information is encoded per second. In this experiment, the units are megapixels per second (MP/s).

Lastly, the similarity type is unique in terms of certainty and objectivity. It is easy to correctly measure the loss of quality, but it is hard to automate the evaluation of the significance of each loss. Because of that difficulty, multiple metrics are chosen, among a pool of available ones. The MSE, PSNR, and SSIM metrics were chosen, the first two

being better for evaluating the average difference pixel-wise, whereas the latter evaluates in a more structural perspective.

Each metric relates to a characteristic, which is usually a point of trade-off that the format can optimize in deterioration of another.

Figures 5 and 6 present the schematics of the main processing stage flow for png and apng input types, respectively. The main idea for the module that takes png, depicted in figure 5, is that the images are encoded with all codecs. For each of those encoding processes, the speed of compression is recorded. After the compression, the size of the encoded image is compared to the original version of the same image. The software proceeds afterwards to decode the image, where the speed of that process is also recorded. Lastly, the decoded image is compared to the original one, in order to compute the similarity level metric.

When the input file is apng, as portrayed in figure 6, the approach is slightly different, because of there being multiple frames. The file is still treated as a single unit. The JXL codec automatically encodes and decodes this file as if it were a single-frame image. For AVIF and WebP, however, the bundling needs to be manually done by the benchmarking software, and the compression is performed frame-by-frame.

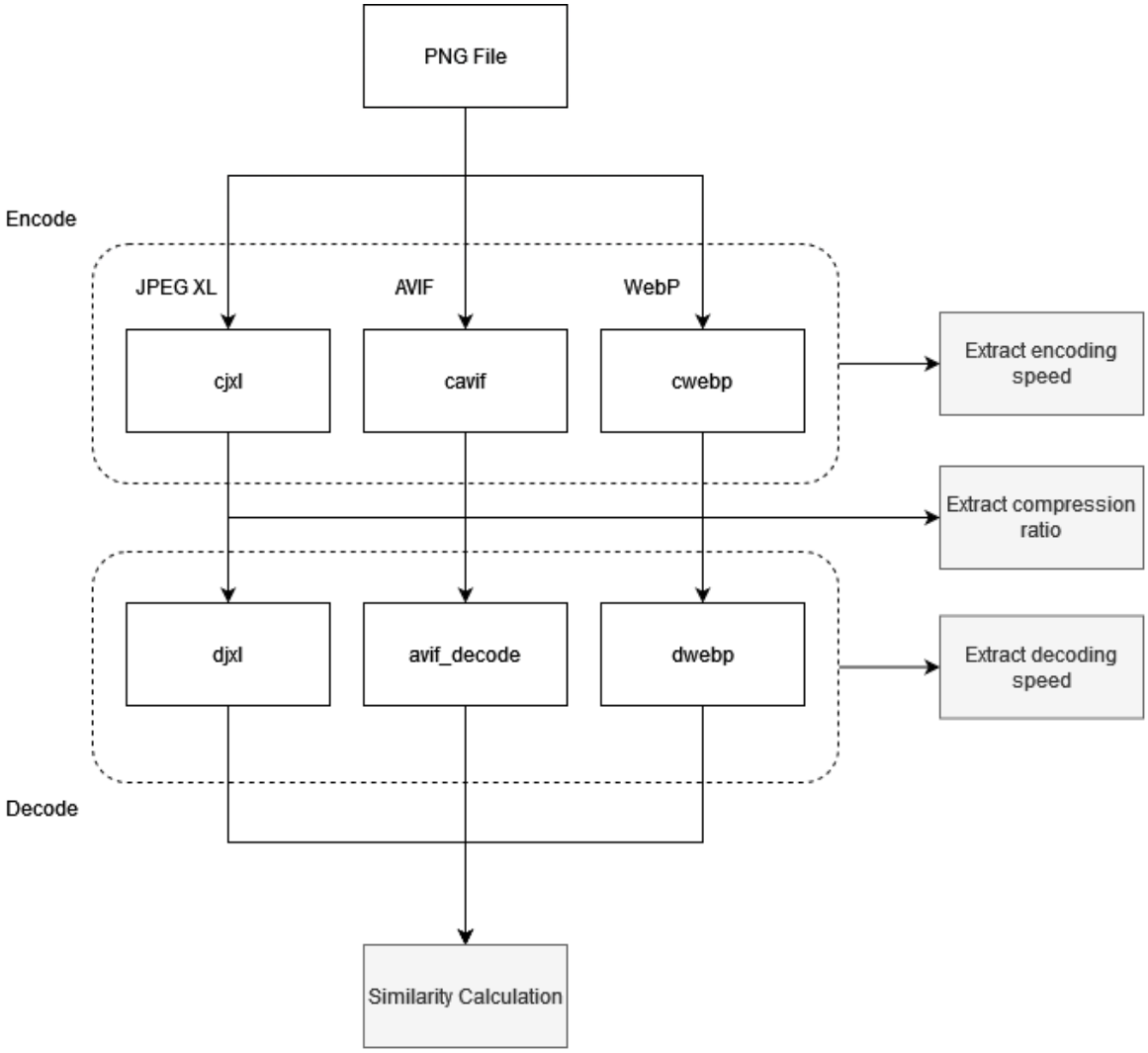


Figure 5 - Main processing stage flow of operations when the input file is PNG.

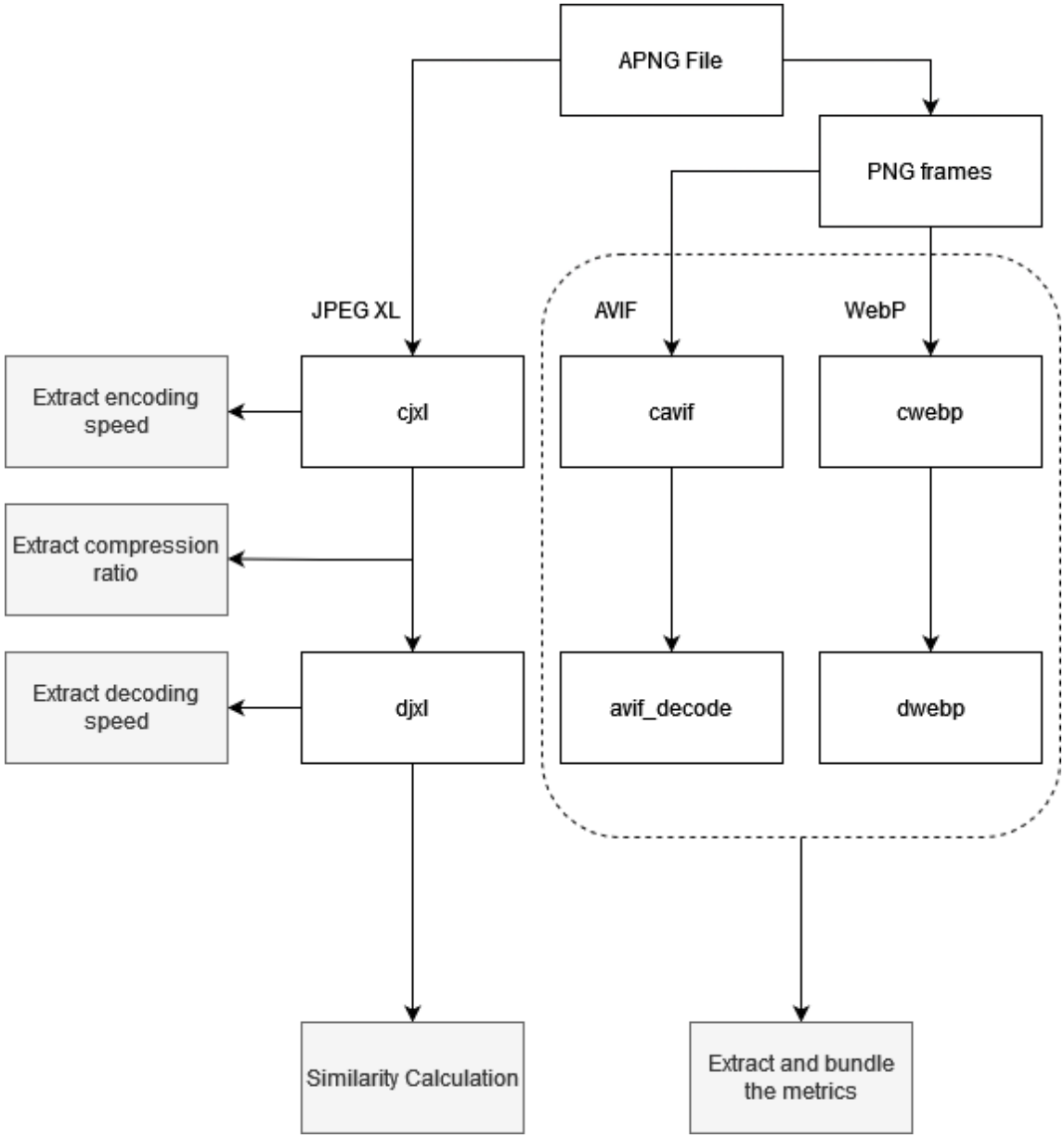


Figure 6 - Schematics of the main processing stage flow of operations when the input file is APNG

3.2.2.1. Result files

Thus, the first of two files being written (outputs) by this script, is a CSV file, containing the header shown at listing 1, whereas a sample of the body rows is displayed at listing 2.

```
filename,cs,ds,cr,mse,psnr,ssim
```

Listing 1 - CSV Results file contents' header

```
CT_HEAD_MONOCHROME2_1_12_1_155_q0.0-
e7.jxl,2.2281010663854084,5.790200226736841,2.2717179510651375,0.0,inf,1
.0
CT_HEAD_MONOCHROME2_1_12_1_155_q0.25-
e7.jxl,5.211229056500375,11.806986567920115,124.97553956834533,66.480138
91763244,54.020279776520006,0.9999281898876653
CT_HEAD_MONOCHROME2_1_12_1_155_q0.5-
e7.jxl,5.179205092605015,9.895181057597535,174.23871614844535,355.174652
7059189,46.74277931460587,0.9993824730954134
```

Listing 2 - CSV Results contents

Starting from the right column and moving to the left ones, we can see, firstly, that the three rightmost columns refer to the similarity metrics group.

The 'cr' field is associated with the compression ratio, providing a value of how many times the encoded image file is smaller than the uncompressed version of the image, each one measured in bytes.

The ‘cs’ and ‘ds’ fields regard the speed of compression and decompression, respectively, in MP/s, meaning Megapixel per second of encoding.

The last field to the left regards the instance of compression, which again comprises the file used as input [base name without the file extension] and contains information about which quality-effort options’ values were given in the encoding instance, as well as the format of the encoding [latter half (qXXeY.fmt) of the string]. This column is unique in each row.

The second file is in JSON format, which groups compression instances by each DICOM / compression attribute in a hierarchical structure: the first group by quality, effort, and format combinations, then by modality, body part, depth, samples per pixel, and lastly, by bits per sample.

Each of the entries (json[qef][modality]...) displays a dictionary, whose keys are metric IDs, and the respective values are also a dictionary. For each metric, the respective dictionary contains statistic values, and always follows the format shown in listing 3.

```
"metric": {"min": 1.0, "max": 2.0, "avg": 1.5, "std": 0.2}
```

Listing 3 - Example of a line of statistical results for a given metric, regarding the json results file.

There is also another entry at the same level of the “metric”, with the key name "size" and value of the total count of the compression instances that belong to the parent attributes grouping.

Listing 4 depicts a sample of the structure of the json results file.

```
{
      "q0.0-e7.jxl": {
        "CT": {
            "HEAD": {
                  "1": {
                  "1": {
                        "12": {
                              "cs": {
                              "min": 1.7325240573447402,
                              "max": 2.633146615574192,
                              "avg": 1.9932699641445968,
                              "std": 0.15526587182095214
                              },
                              "ds": {
                              "min": 4.635259831511387,
                              "max": 9.483243576299936,
                              "avg": 5.403815335489965,
                              "std": 0.7184953614346689
                              },
                              "cr": {
                              "min": 2.000902835724936,
                              "max": 2.4508610643532305,
                              "avg": 2.1564965596736823,
                              "std": 0.08987462871142923
                              },
                              "mse": {
                              "min": 0.0,
                              "max": 0.0,
                              "avg": 0.0,
                              "std": 0.0
                              },
                              "psnr": {
                              "min": Infinity,
                              "max": Infinity,
                              "avg": Infinity,
                              "std": 0.0
                              },
                              "ssim": {
                              "min": 1.0,
                              "max": 1.0,
                              "avg": 1.0,
                              "std": 0.0
                              },
                              "size": 193
```

```
                        }
                    }
                    }
                }
            }
}
```

Listing 4 - Sample from the json results file.

In this example, statistics are provided over all metrics from files encoded with the JPEG XL format with settings --distance of 0.0 and --effort of level 7 that have the following attributes: CT modality, head as studied the body part, single framed, 1 sample per pixel (grayscale) and 12 stored bits, or bits per sample. It is also shown that there are 193 encoded instances with these attributes.

3.2.2.2. Multi-frame Images

There are some nuances concerning multi-frame images. Since the pre-processing stage outputs `.apng` files for each image of this type, the pipeline has two options when encoding them with JPEG XL, AVIF, or WEBP, based on whether the respective encoders have the support for APNG format or not. If they have, which is the case only for JPEG XL, nothing needs to be done, besides the normal encoding/decoding procedure. Otherwise, in the case of AVIF and WEBP, a different approach must be taken.

Since cavif and cwebp do not accept `.apng` formats as input, each frame of that file is temporarily written to `.png` format files. That allows the frames to be singularly encoded and decoded with each of these algorithms. The metrics are obtained by a posterior aggregation of those of each frame, such as averaging the speeds, the similarity indexes, and, regarding CR, dividing the sum of the sizes of the uncompressed frames by the sum of compressed bitstreams' sizes of those frames.

3.2.3. Post-processing

The data provided in text files are not enough to provide an understandable set of results. To improve this process, the post-processing stage was introduced, consuming results data structures and featuring the production graphs that intuitively summarise the results more directly.

The preferred graph type to display the results' contents was the bar graph, which, by grouping by the images' attributes, as well as by the format of compression, produced each figure that will be shown in section 3.3. Then, there is one X axis and two Y axes. The X-axis is of a discrete type and represents the different quality values used to

compress the images. For each X-axis value, there are two bars - one referring to the left Y axis, the other one to the right Y axis. Each Y axis is a compression metric (one of the four), and the bars point to the value of the averages of the metric results for all respective compression instances. Each bar also has, if there is more than one compression instance, an error line representing the standard deviation of the same data.

The matplotlib library was used to render the figures.

The module overall gives the options to show all figures at once or write them to disk, the latter option with the choice of compressing them into a zip file. We can either choose to generate figures of bars or figures of point clouds. In either case, all combinations of metrics are generated (to enable easy comparison) and grouped then by other filters. For each such combination, if the write-to-disk option is enabled, there is a directory that contains all graphs that compare those two metrics. The directory will be inside the zip file if the zip option is set.

For the experiment results in section 3.3, only bar graphs are used.

3.3. Dataset and technical specs

3.3.1. Dataset Description

The dataset is a set of DICOM files comprising the input of the first stage program of the pipeline, upon which all the results will be based on.

The set contains files from CT, mammography (MG), and slide microscopy (SM) modalities.

The CT modality files all contain the head as the associated body part, and are greyscale and single frame, with 12 of bit-depth.

The MG modality files all contain the breast as the associated body part, are all grayscale, and have single and multiple frame instances, all of them with 10 stored bits.

The SM modality files don't contain an associated body part. Also, the photometric interpretation is YBR_FULL_422, meaning the number of channels is 3. This set is comprised of multi-frame as well as single-frame images and all have 8 stored bits (per sample).

This last case is the case where a feature in the pipeline, that translates YBR_FULL_422 format into RGB, is used.

3.3.2. Hardware Specifications

The pipeline was run on a machine with the specifications in table 4.

Operating System	Ubuntu 22.04
Kernel Version	5.15.0-43-generic (64-bit)
Graphics Platform	X11
Processors	12 × 11th Gen Intel® Core™ i5-11400H @ 2.70GHz
Memory	7,4 GiB of RAM
Graphics Processor	Mesa Intel® UHD Graphics

Table 4 - Specifications to the Local Machine that ran the benchmark

3.3.3. Software Specifications

The pipeline can be run using a 3.10.6 python interpreter or above. The library dependencies are defined in the requirements.txt file, represented in listing 5.

```
matplotlib~=3.5.2
numpy~=1.22.1
pandas~=1.4.3
pyDICOM==2.3.0
opencv-python~=4.5.5.64
scikit-image~=0.19.3
Pillow~=9.0.1
apng~=0.3.4
gdcm~=1.1
pylibjpeg~=1.4.0
pylibjpeg-libjpeg~=1.3.1
```

Listing 5 - Project dependencies

3.4. Results

3.4.1. Introduction

The graphics provided by the last processing module of the pipeline (post-processing) will be used as the base for this results section, regarding the modern formats' evaluation (main experiment) as well as for the JPEG extended secondary one.

The ranges defined by the 'average +- standard deviation' error bars will be the format of the description of the results. The same section of the descriptions will be subdivided by modality and body-part attributes, and then by the frame multiplicity. For each of those subsections, the charts will be displayed with the results, and below will be the observations of them, all divided into four paragraphs – one for each metric. Each paragraph will describe, for each applicable format, the intervals where the format's results are generally fitting in, as for that metric measurement results.

The multi-frame sections include more figures, one set per number of frames. This segregation is carried out to keep individuality between results. If the results were to be aggregated among all multi-frame images, the result would, conceptually, be a study on each frame, which is the same as the single-frame study.

Also worth noting is that the WebP format is only included in the slide microscopy results because it is the only dataset with 8-bit images, which is the format's maximum supported bit-depth.

The illustrations of the results include some attributes that may be ambiguous, the "DEPTH", "SPP" and "BPS" – meaning that they refer to the number of frames, the number of samples per pixel (or the number of color channels) and the number of bits per sample (or bits per color channel), respectively. Another feature of the graphs that might also not be very clear is that some y-axes are valued with scientific notation, specified at the top-left corner of the rectangular region of the bars.

The JPEG experiments were made to provide a comparison of the new to the presently used codecs. They include the similarity and CR metrics but not a comparison of speeds of encoding/decoding because of the different methods used when compared to the other, new codecs – all the new ones take png files as input, whereas the jpeg codec, dcmcjpeg from dcmtk (DICOM toolkit), takes dicom files. The reading and writing operations are different, so comparing may not be appropriate. Lastly, the SM results are not covered because the results were not regarded as suitable for displaying.

3.4.2. Codec Quality Settings

This table summarises the quality parameter values used in the experiment for every file and each format. AVIF and WebP encoders have a quality parameter, however JXL's is slightly different. Even though the encoder contains a parameter with that name, another named 'distance' was chosen because it has finite boundaries (0.0 to 25.0, in contrast to the quality setting, that is -inf to 100). The values used in the experiments are shown in table 5.

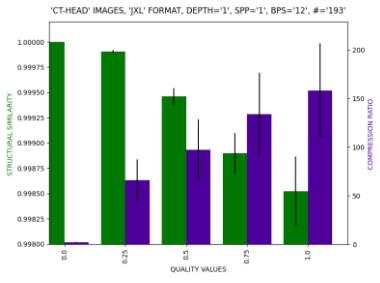
Format	Parameter name	#1	#2	#3	#4	#5
JPEG XL	Butteraugli distance	1.0	0.75	0.50	0.25	0.0
AVIF	Quality	80	85	90	95	100
WEBP	Quality	80	85	90	95	100

Table 5 - Quality settings used for the experiments (lesser quality values are to the left).

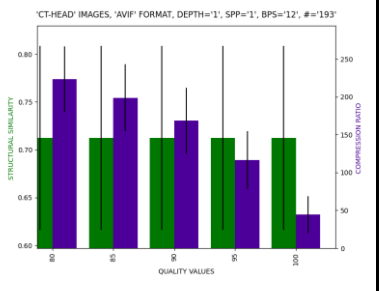
3.4.3. CT – HEAD

This section includes a head computer tomography dataset with 193 12-bit grayscale images, all of them being single-frame.

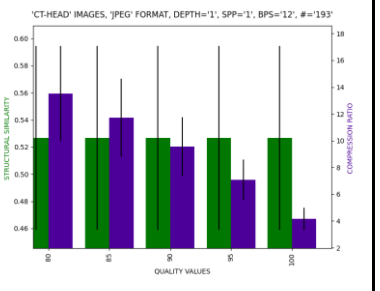
3.4.3.1. Single frame



JXL SSIM-CR



AVIF SSIM-CR



JPEG SSIM-CR

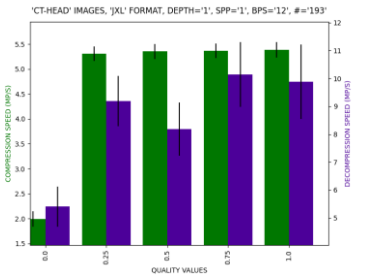
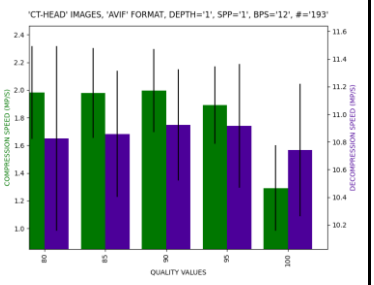
		---
JXL CS-DS	AVIF CS-DS	---

Table 6 - CT Single frame Results

Regarding SSIM, JXL compressions yield a quality loss of about 0.99850 to 0.99990, where AVIF falls between 0.6 and 0.8. Regarding the JPEG (extended) experiment, the values range from 0.46 to 0.60.

Concerning CR, JXL offers from ~50 up to ~200. AVIF ranges around 20 and 260, whereas JPEG ratios fall around 3 and 11.

Regarding DS, JXL compresses with speeds of ~7 to ~11 (except for lossless, with ~5) MP/s, while AVIF shows values of ~1.0 to ~2.3.

For CS, JXL offers about ~5.2 to ~5.6 MP/s speed for lossy settings and ~2.0 for the lossless setting. AVIF values range from about 10.2 to 11.3 MP/s.

These results show that JXL similarity levels are higher than AVIF, as well as faster encoding and decoding speeds. On the other hand, AVIF features compression with a higher ratio. All the new codecs feature higher similarity levels as well as compression than the JPEG used nowadays.

3.4.4. MG - BREAST

This section includes a dataset with 10-bit grayscale images from the mammography modality. Among these, there are 4 single frame and 3 multi-frame images.

3.4.4.1. Single frame

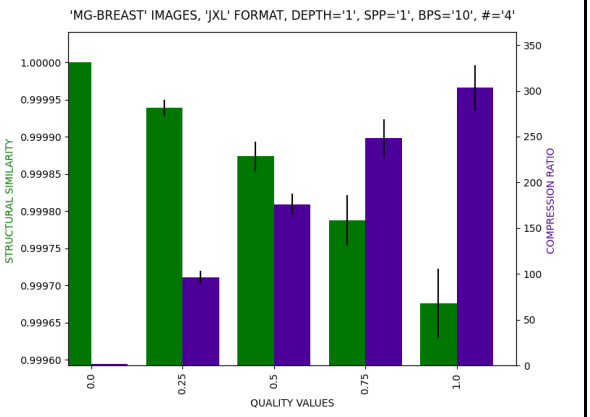
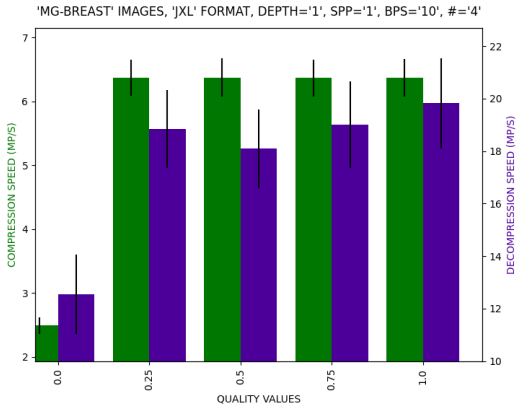
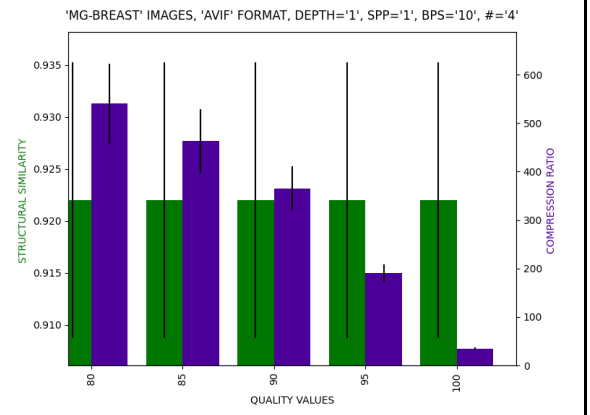
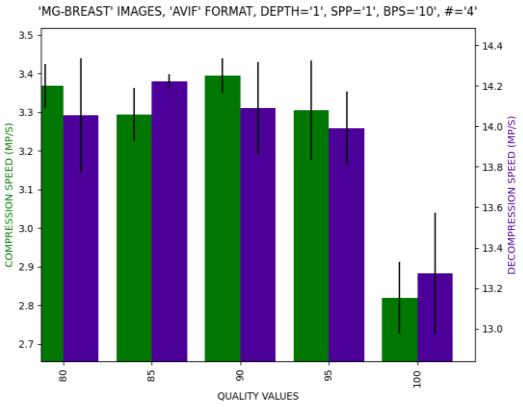
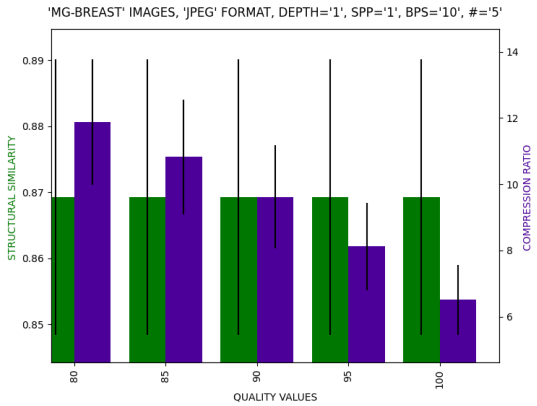
	
JXL SSIM-CR	JXL CS-DS
	
AVIF SSIM-CR	AVIF CS-DS
	-
JPEG SSIM-CR	-

Table 7 - MG Single frame Results

Regarding the quality metric SSIM, JXL offers values of 0.99962 up to 0.99995 for lossy quality settings. AVIF yields values of between ~0.91 and ~0.935 among all of the quality settings.

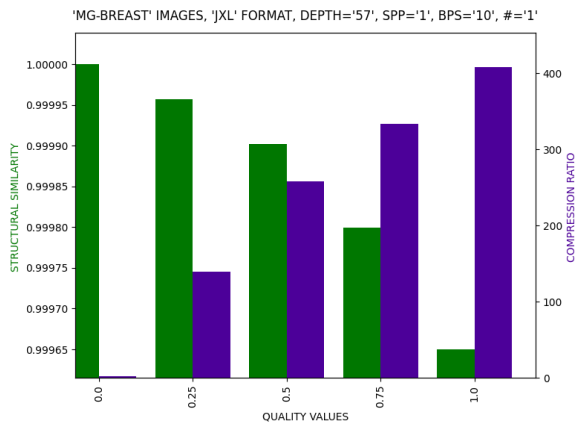
Concerning CR, JXL of ~90 to ~325 and AVIF 30 to 610.

With regards to DS, JXL has a decoding speed of ~16 to ~22 MP/s for lossy settings, and ~11 to ~14 MP/s for lossless. AVIF yields speeds of ~13.0 to ~14.3 MP/s.

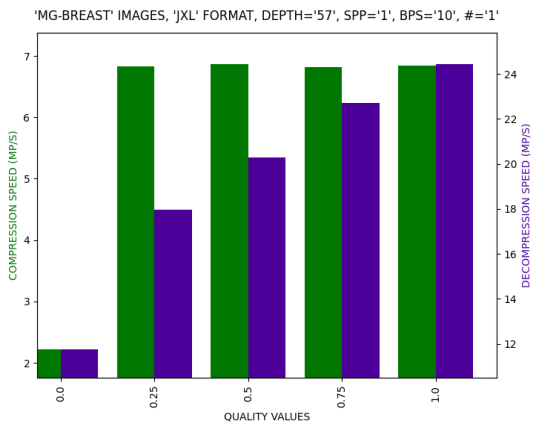
Regarding CS, JXL compressed with a speed of about 6.0 to ~6.6 MP/s, for lossy values, and ~2.5 for lossless compression. AVIF speeds display values of ~2.7 to ~3.45 MP/s

To sum up, JXL similarity levels are higher than AVIF, as well as faster encoding and decoding speeds. On the other hand, AVIF features stronger compression.

3.4.4.2. Multi-frame



JXL SSIM-CR



JXL CS-DS

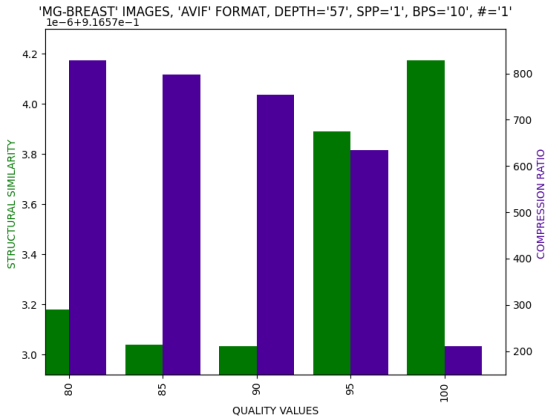
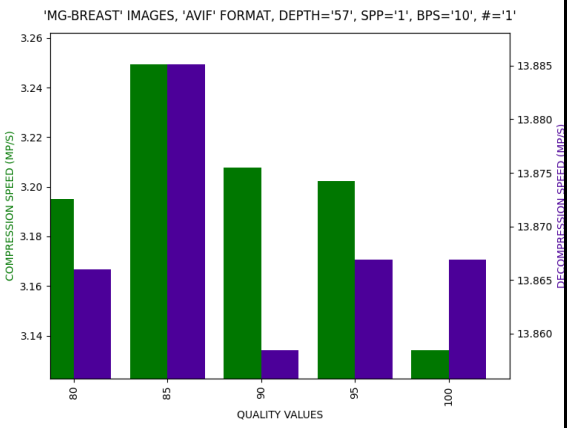
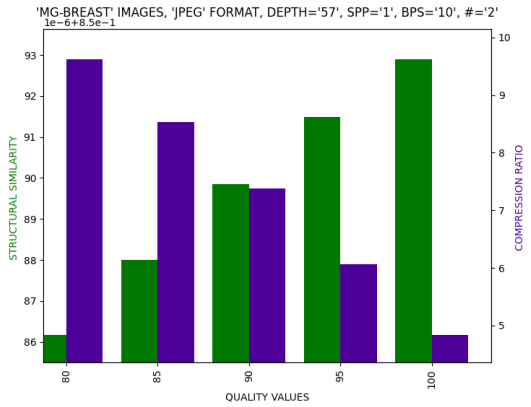
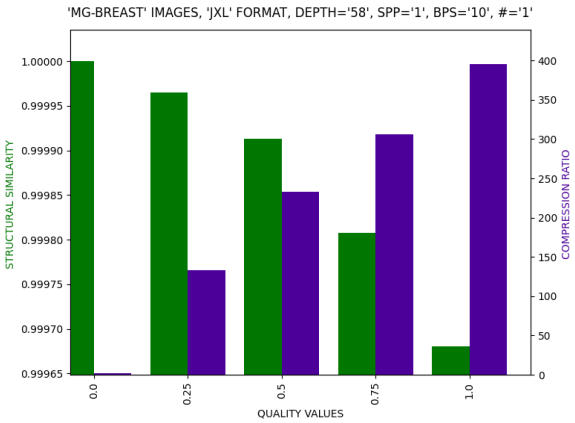
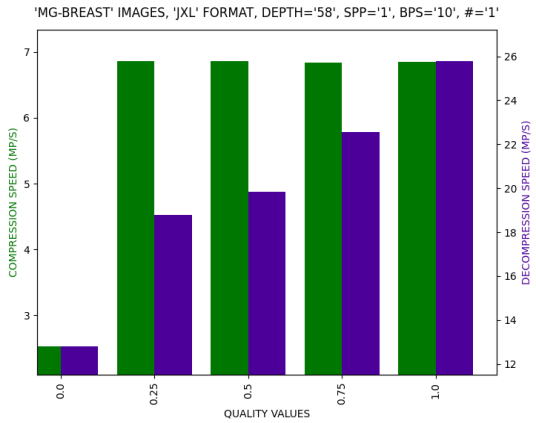
	
AVIF SSIM-CR	AVIF CS-DS
	-
JPEG SSIM-CR	-

Table 8 - MG Multi-frame (57) Results

	
JXL SSIM-CR	JXL CS-DS

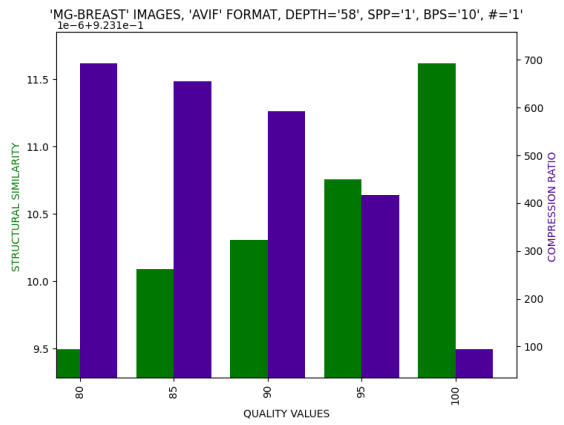
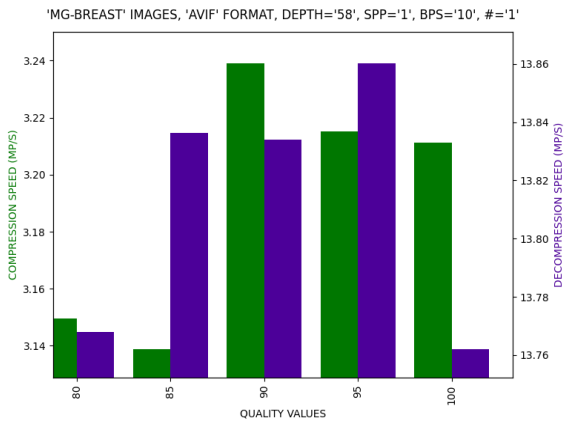
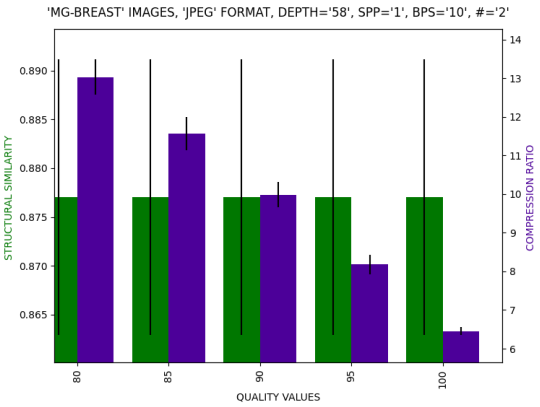
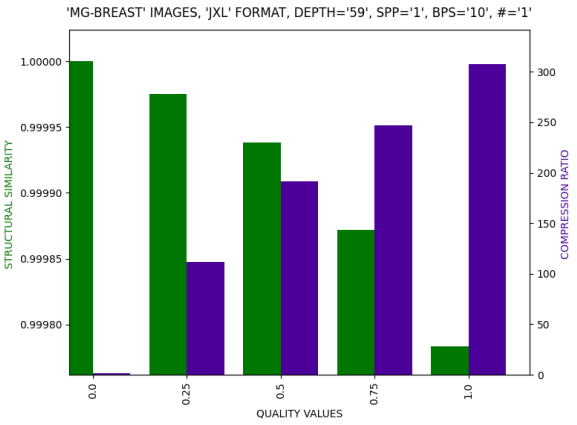
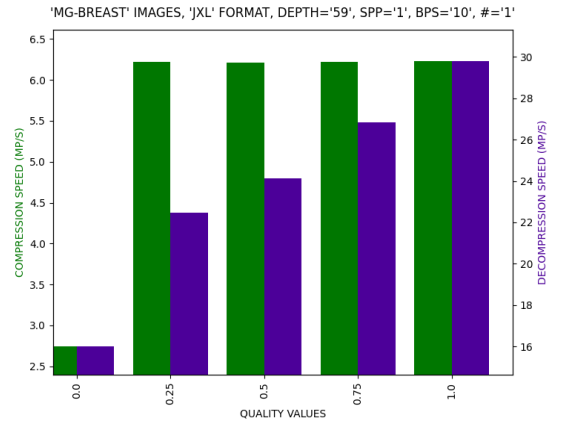
	
AVIF SSIM-CR	AVIF CS-DS
	-
JPEG SSIM-CR	-

Table 9 - MG Multi-frame (58) Results

	
JXL SSIM-CR	JXL CS-DS

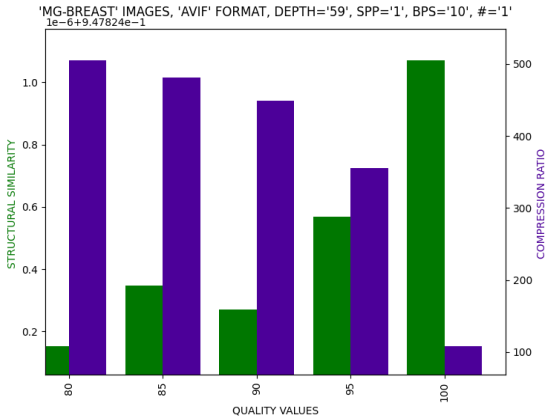
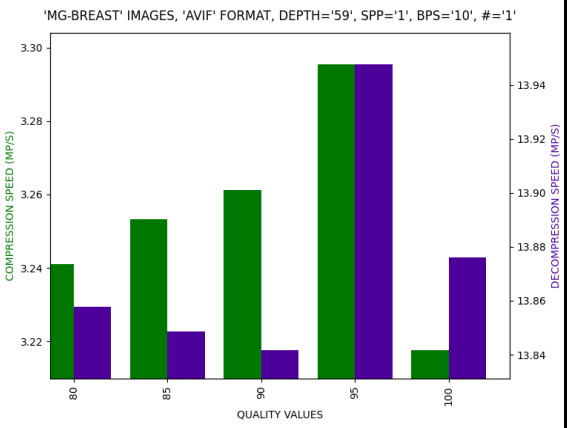
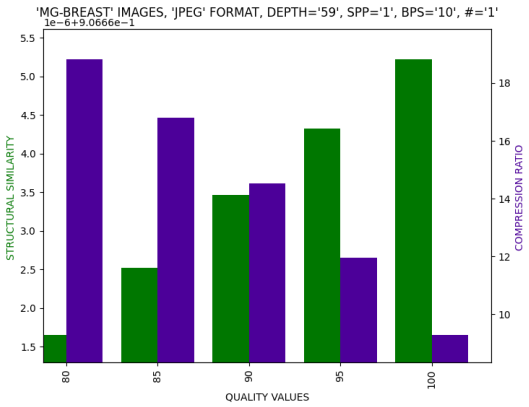
	
AVIF SSIM-CR	AVIF CS-DS
	-
JPEG SSIM-CR	-

Table 10 - MG Multi-frame (59) Results

Regarding the SSIM metric, JXL yields a quality loss ranging from ~0.99962 up to ~0.999975. AVIF, in turn, compresses to values of about 0.91 to 0.947. JPEG fidelity ranges from 0.85 to 0.907.

JXL compresses to about 100 to 400 times smaller sizes (for lossy settings). AVIF ratios range from 100 to 800. JPEG ratios fall around 4.9 and 13.3.

For the decompression speed metric, JXL decoding displays values of around 8 to 30 MP/s. AVIF yields values from ~13.76 to ~13.95.

For CS, JXL yields results of ~6.25 to ~6.9 MP/s, while AVIF compresses to speeds of ~3.13 to ~3.3.

The single frame trend continues in this section, where JXL features higher fidelity as well as the speed of encoding and decoding, except for CR, where AVIF takes a higher ratio. All the new formats surpass the JPEG levels of similarity and ratio.

3.4.5. SM - No associated body part

This section includes a dataset with 8-bit RGB images from the slide microscopy modality. Among these, there are 2 single frame and 5 multi-frame images.

3.4.5.1. Single frame

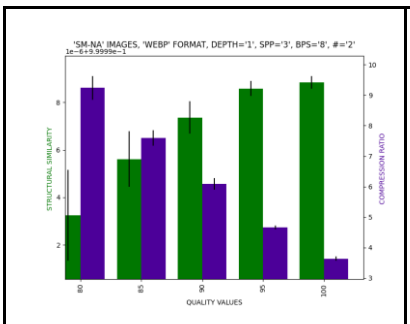
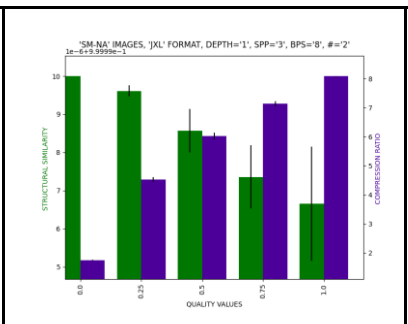
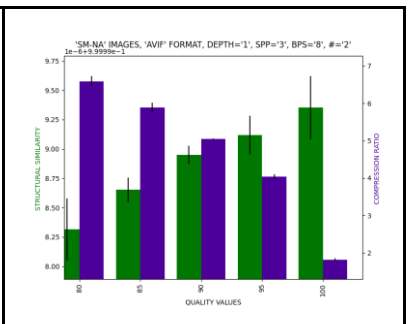
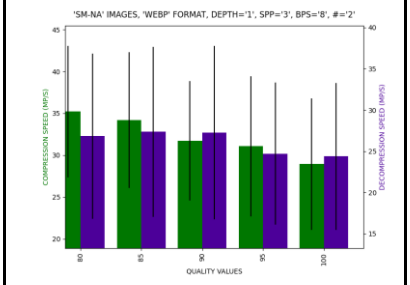
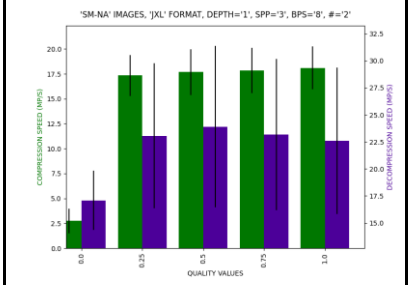
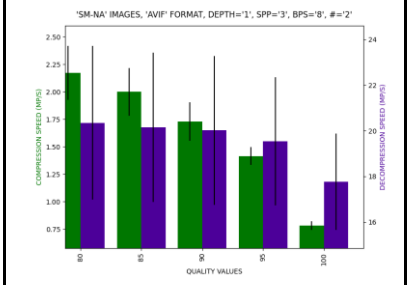
		
WebP SSIM-CR	JXL SSIM-CR	AVIF SSIM-CR
		
WebP CS-DS	JXL CS-DS	AVIF CS-DS

Table 11 - SM Single frame Results

Regarding the SSIM quality metric, the images show a similarity of ~0.99999 with all formats.

Following the rate of compression, JXL results, for the lossy settings, yield values of ~1.7 to ~8. AVIF compresses from ~1.9 to 6.7 times the size of the original image. All that while WebP yields a CR of 3.5 to 9.5.

For the DS metric, JXL decodes with speeds from ~14.5 to ~30.5 MP/s. AVIF has speeds of around 16 to 24 and WebP from ~15 to ~38 MP/s.

Regarding the speed of compression, CS, the values for JXL show to be around ~15 to ~20, while AVIF the range is ~0.75 to ~2.5, and WebP is ~20 to ~45.

These results show that the fidelity is practically the same. Regarding the other metrics, WebP displays the best CR, encoding, and decoding speeds.

JXL shows the second-best CR, followed closely by AVIF. The codecs show similar average CR. Regarding the speed of encoding, AVIF is the slowest one by a significant margin.

3.4.5.2. Multi-frame

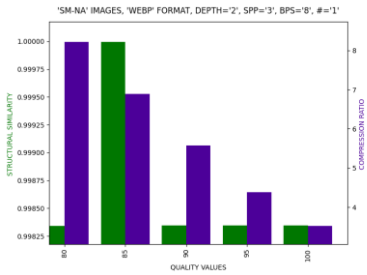
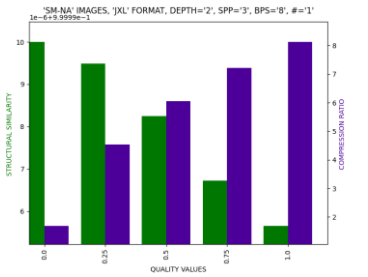
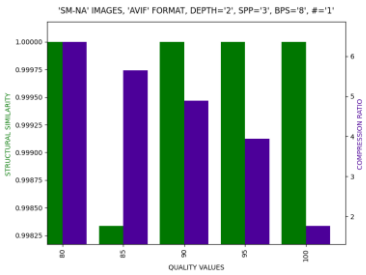
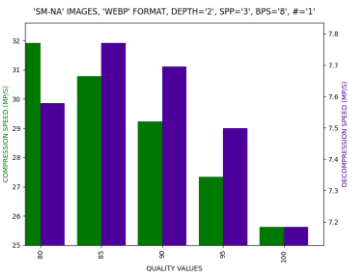
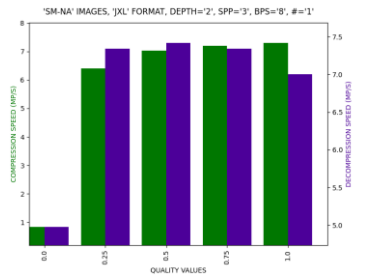
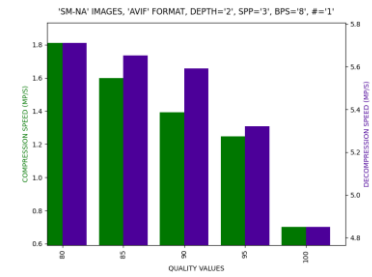
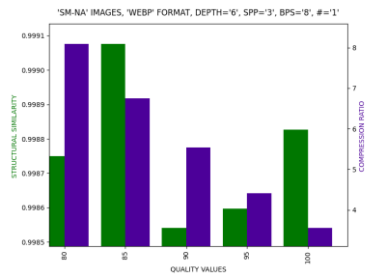
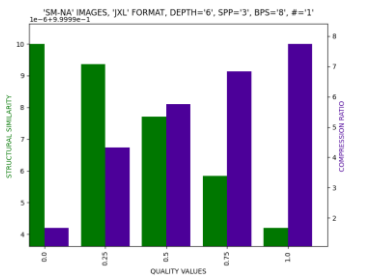
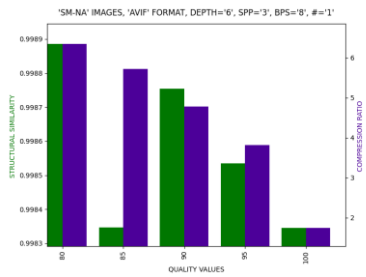
		
WebP SSIM-CR	JXL SSIM-CR	AVIF SSIM-CR
		
WebP CS-DS	JXL CS-DS	AVIF CS-DS

Table 12 - SM Multi-frame (2) Results

		
WebP SSIM-CR	JXL SSIM-CR	AVIF SSIM-CR

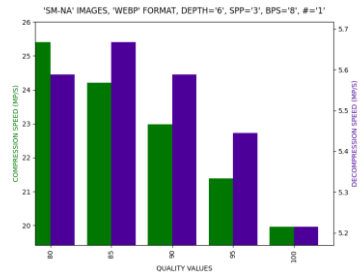
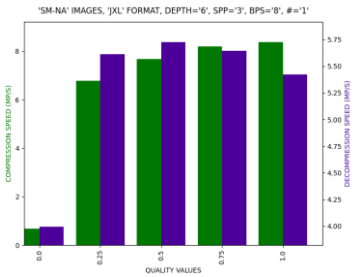
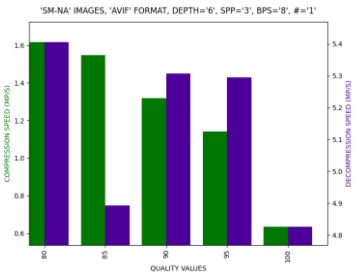
		
WebP CS-DS	JXL CS-DS	AVIF CS-DS

Table 13 - SM Multi-frame (6) Results

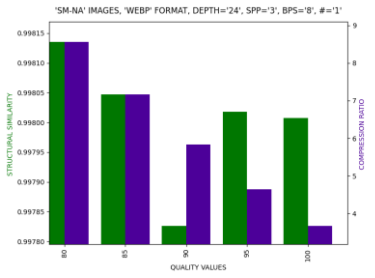
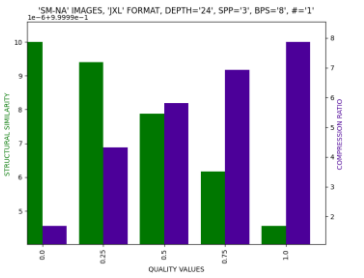
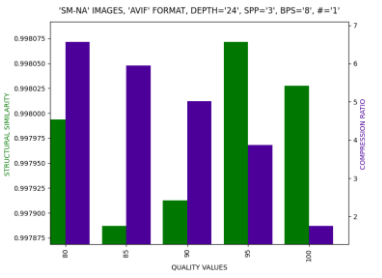
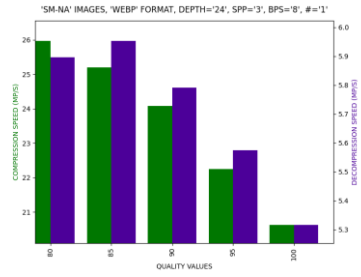
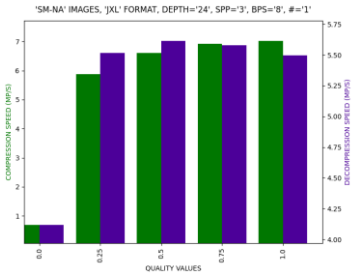
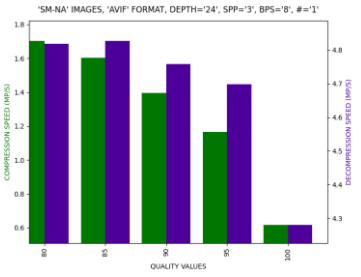
		
WebP SSIM-CR	JXL SSIM-CR	AVIF SSIM-CR
		
WebP CS-DS	JXL CS-DS	AVIF CS-DS

Table 14 - SM Multi-frame (24) Results

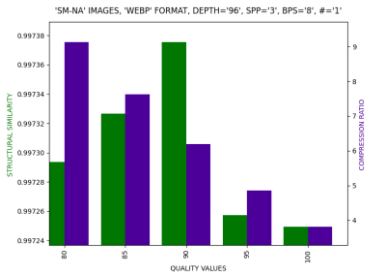
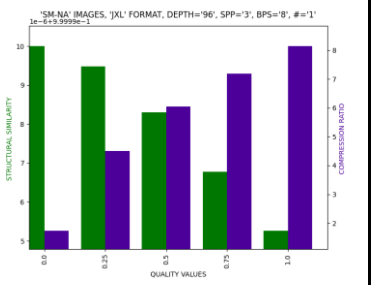
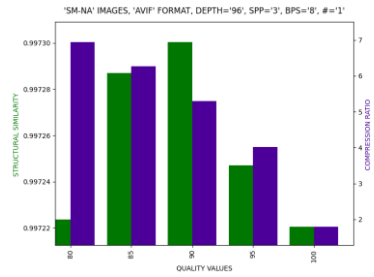
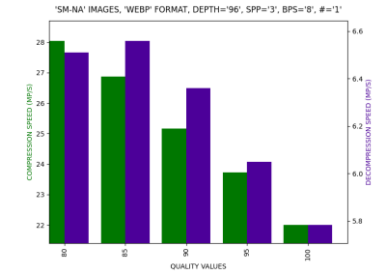
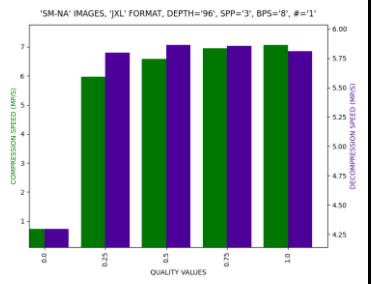
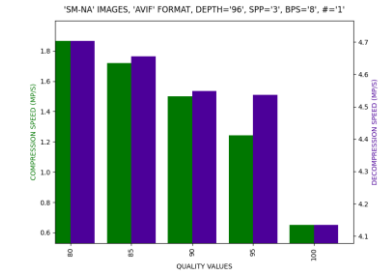
		
WebP SSIM-CR	JXL SSIM-CR	AVIF SSIM-CR
		
WebP CS-DS	JXL CS-DS	AVIF CS-DS

Table 15 - SM Multi-frame (96) Results

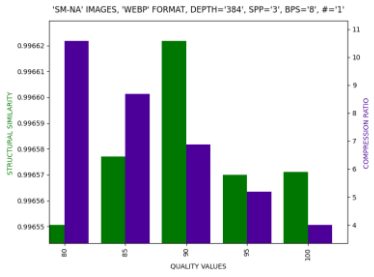
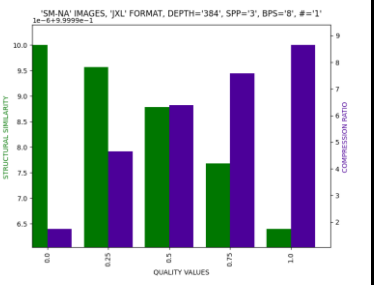
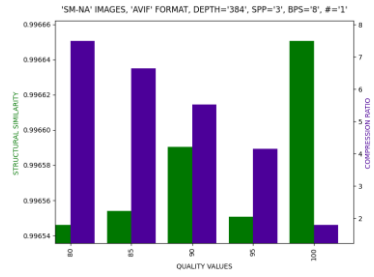
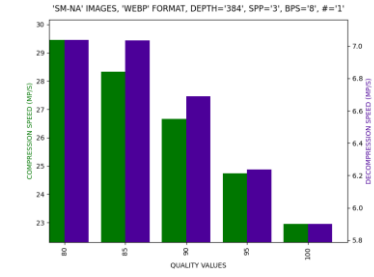
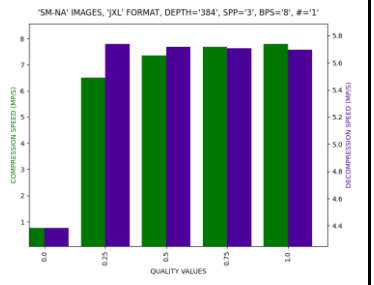
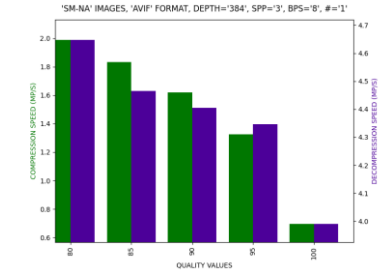
		
WebP SSIM-CR	JXL SSIM-CR	AVIF SSIM-CR
		
WebP CS-DS	JXL CS-DS	AVIF CS-DS

Table 16 - SM Multi-frame (384) Results

Regarding SSIM measurements, we can see that JXL yields again values of above 0.99999, whereas AVIF and WebP follow behind with values of ~0.996 to ~0.998.

The CR results are mainly ranging from 4.1 to 8.6 for JXL, ~1.8 to ~7.5 for AVIF, and ~3.5 up to ~10.5 for WebP compressions.

DS values fall mainly within the following intervals: 5.3 up to 7.4 MP/s for JXL, 3.9 to 5.7 for AVIF, and 5.2, to 7.8 for WebP.

Lastly, regarding CS, the intervals of the standard deviation of the measurements are as follows: 6.2 to 8.3 or JXL, 0.6 to 2.0 for AVIF, and 19.9 up to 32 for the case of WebP compressions. These are approximate results.

Generally, the trend from the single-frame section is slightly different, with the only similarity being the speed of encoding, with WebP being the fastest one and AVIF the evident slowest decoder.

The image fidelity results after the compressions are best with the JXL codec. The speed of decoding is relatively similar between WebP and JXL, those being slightly higher than the AVIF speeds.

The CR results are lowest for AVIF. At the same time, JXL and WebP show very similar average results, with WebP having a higher ceiling (for example, the values reach ratios of 1:10.5, where JXL compresses only up to 8.6, in this case).

3.4.6. Summary

Here is displayed a summary of the results – each table for each metric, where the columns show the results for each format, and each row displays the modality as well as frame multiplicity of the respective table cells.

The results are generally shown in the following form: an approximate range that covers the result set for all lossy quality entries. In other words, the lower and upper standard deviation bounds. Some cells, however, have a single value, meaning that the respective results all revolve around that one.

Mod - Multi-frame \Format	WebP	JXL	AVIF	JPEG
CT - No	NA	0.99850 to 0.99990	0.6 to 0.8	0.46 to 0.60
MG - No	NA	0.99962 to 0.99995	0.91 to 0.935	0.85 to 0.89
MG - Yes (57)	NA	0.99965 to	0.917	0.85

		0.99995		
MG - Yes (58)	NA	0.99968 to 0.99996	0.923	0.863 to 0.891
MG - Yes (59)	NA	0.99978 to 0.999975	0.947	0.907
SM - No	0.99999	0.99999	0.99999	-
SM - Yes (2)	0.998	0.99999	0.998	-
SM - Yes (6)	0.998	0.99999	0.998	-
SM - Yes (24)	0.998	0.99999	0.998	-
SM - Yes (96)	0.997	0.99999	0.997	-
SM - Yes (384)	0.996	0.99999	0.996	-

Table 17 - SSIM metric results.

Mod - Multi-frame \Format	WebP	JXL	AVIF	JPEG
CT - No	NA	50 to 200	20 to 260	3 to 17
MG - No	NA	90 to 325	30 to 610	5.5 to 14
MG - Yes (57)	NA	130 to 400	200 to 800	4.9 to 9.5
MG - Yes (58)	NA	175 to 400	100 to 700	6.3 to 13.3
MG - Yes (59)	NA	100 to 300	100 to 500	9 to 19
SM - No	3.5 to 9.5	4.4 to 8	1.9 to 6.7	-
SM - Yes (2)	3.5 to 8	4.5 to 8	1.8 to 6	-
SM - Yes (6)	3.5 to 8	4.1 to 7.8	1.8 to 6.2	-
SM - Yes (24)	3.6 to 8.5	4.2 to 7.9	1.8 to 6.5	-
SM - Yes (96)	3.9 to 9.1	4.4 to 8.1	1.8 to 6.9	-
SM - Yes (384)	4 to 10.5	4.5 to 8.6	1.9 to 7.5	-

Table 18 - Compression Ratio Results

Mod-Multiframe\Format	WebP	JXL	AVIF
CT - No	NA	7 to 10	10.2 to 11.3
MG - No	NA	16 to 22	11 to 14
MG - Yes (57)	NA	18 to 24	13.86 to 13.88
MG - Yes (58)	NA	8 to 26	13.76 to 13.86
MG - Yes (59)	NA	22 to 30	13.84 to 13.95
SM - No	15 to 38	14.5 to 30.5	16 to 24
SM - Yes (2)	7.2 to ~7.8	7.0 to 7.4	4.8 to 5.7
SM - Yes (6)	5.2 to 5.66	5.3 to 5.75	4.83 to 5.4
SM - Yes (24)	5.31 to ~5.95	5.5 to 5.6	4.29 to 4.81
SM - Yes (96)	5.79 to 6.56	5.75 to 5.8	4.13 to 4.7
SM - Yes (384)	5.9 to 7.1	5.69 to 5.72	3.9 to 4.64

Table 19 - Decompression Speed, MP/s

Mod-Multiframe\Format	WebP	JXL	AVIF
CT - No	NA	~5.2 to ~5.6	1.0 to 2.3
MG - No	NA	6.0 to ~6.6	2.7 to 3.45
MG - Yes (57)	NA	6.8	3.13 to 3.25
MG - Yes (58)	NA	6.9	3.14 to 3.24
MG - Yes (59)	NA	6.25	3.22 to 3.3
SM - No	15 to 20	0.75 to 2.5	20 to 45
SM - Yes (2)	25.5 to 32	6.2 to 7.2	0.7 to 1.8
SM - Yes (6)	19.9 to 25.3	6.7 to 8.3	0.6 to 1.6
SM - Yes (24)	20.7 to 26	5.9 to 7	0.6 to 1.7
SM - Yes (96)	22 to 28	6 to 7	0.65 to 1.85
SM - Yes (384)	23 to 29.5	6.5 to 7.8	0.7 to 2.0

Table 20 - Compression Speed, in MP/s

3.5. Discussions and Conclusion

In this chapter, a benchmarking software project was developed using python, which evaluates the performance of multiple image compression codecs, applied to medical images. The software can be characterized as a processing pipeline, where images are extracted from DICOM files to lossless png ones. The images are then encoded and decoded, where the same operations are evaluated for their performance using each of the four image compression metrics: compression speed, decoding speed, image degradation level, and compression ratio. The results are written to a CSV file, which is parsed to human readable displaying formats.

Firstly, it is worth noting that there are some irregularities among the results, specifically from the SSIM perspective, where the quality factor was not always perfectly proportional to that metric's values. These abnormalities kept appearing after multiple runs of the benchmark. The codebase of this benchmark was reviewed multiple times on the account of this issue, and after not finding the possible bug, it was conjectured that this is a trait of some of the compression algorithms, that can sometimes not behave exactly as intended. This conjecture is based on the fact that JPEG XL showed no such anomalies, and because the deviation of those values is minimal.

The results are then displayed for a dataset of Computer Tomography, Mammography, and Slide Microscopy DICOM images fed into the pipeline.

The results show that, in an overall sense, WebP seems to optimize its compressions for speed. AVIF shows similar levels of quality loss to the former, with the difference of optimizing for compression ratio. JXL similarity levels are higher than its counterparts, without losing significantly to its counterparts in terms of the rest of the factors (CR, CS, and DS). JXL is the best overall format because of the aforementioned reasons, the priority for medical images is quality, and this format is optimized in that direction.

JPEG XL has another advantage, which is the ability to take `.apng` images as input, in contrast to the other formats, where the ability to compress multi-frame images is limited to GIF format. This is a drawback to AVIF and WebP since many medical images surpass the number of bits per sample of 8, which is the limit for that format [41]. The ability to take APNG format images as input allows for a simple bulk compression of multi-frame images, as well as allowing the encoder to take advantage of applying compression techniques between frames. This is impossible for AVIF and WebP, where the

compression of this type of image is only possible with a frame-by-frame encoding of PNG-formatted images (the frames).

Regarding each modality in specific, results can vary. For the CT ones, only JPEG XL and AVIF are present, because WebP doesn't support 12-bit images. AVIF shows significantly lower image fidelity and encoding speed, however, shows slightly higher DS and CR.

For the MG modality, JXL shows to be better at preserving similarity and faster at encoding as well as decoding. AVIF, however, shows to compress up to twice the ratio of JPEG XL for some of the encodings.

Concerning the SM subset of data that was benchmarked, the similarity values show to be close to 0 (never below 0.995 for any format), however, JXL continues to best preserve the quality of the images as well as compressing the most overall, however with WebP and AVIF trailing very close in value-wise. AVIF, on the CR front, shows to produce the (although slightly) worst results. This relational trend for CR is very similar to the DS results. The CS trend, however, is unlike its metric precedents for this modality. The difference between formats is significant, where WebP shows to significantly improve, compared to the speed of decompression, contrasting the same results for AVIF, where the speed is significantly lower, and JXL falls between them, showing similar values to its speed of encoding. JXL seems, regarding similarity, to be more faithful with the quality input in the encoding options – the similarity seems to evolve directly with the said quality parameter, where the counterpart formats' similarity (at the SM section of the results, for example), has more irregular trends (along the quality option setting).

One would also observe that all the new formats surpass the SSIM fidelity and CR values of the JPEG with the above 8-bit, greyscale, MG, and CT images.

These differences point to multiple conclusions regarding recommendations for the use of the recent image compression tools, at lossy / near-lossless compression:

- JXL is the only choice when the image precision is above 12-bit
- JXL is the best choice for when the images have 12 bits of depth because AVIF compressions imply a significant loss of quality, and WebP doesn't support
- JXL is the best choice when image fidelity is paramount when compressing, without many considerations to the other metrics.
- For 8-bit RGB images (24 bits total per pixel), the best choice depends upon the requirements. The differences in performance worth mentioning are still the JXL's fidelity to the original images; however, WebP shows very good performance for

encoding speed, which can be useful when, for example, images are frequently created and stored in the PACS.

- For 10-bit grayscale images, JXL shows better performance at similarity preservation and speed of compression and decompression. AVIF, however, shows to yield better CR overall among the two formats, which is useful for, for example, when the storage space is the bottleneck of the medical imaging workflow.

The discrepancy between the results of each modality also indicates that there can be varying results among different datasets, depending on each one's characteristics. One can also note that different codecs can show different patterns. This is one of the reasons why the software, featuring the pipeline which generated the results provided, was designed for reproducibility so that it can be fed other datasets, generating more results. The software can also be built upon by anyone, because of its open-source status, as the license is MIT.

Another advantage of the software is the possibility of the development of parsers for the CSV result files. Even though the contents are not proper for human reading, it is highly adequate for the introduction of programs that can parse this information.

4. Proof of Concept Application Proposal

4.1. Introduction

This chapter covers the proposal for a software application that employs the codecs from modern image compression formats for use in medical imaging contexts. This proposition has the goal of demonstrating the appropriateness of that environment.

The Dicoogle software is an open-source web-based PACS server, which has extensibility capabilities provided by a modular architecture and its software development kit [17] (SDK). This chapter intends to describe a plugin-set software project, an extension to Dicoogle, named Imodec. This project's goal is to provide encoding and decoding features using the state-of-the-art image compression codecs studied in chapter 3. The solution will allow us to incorporate those formats in a DICOM repository, being able to convert DICOM uncompressed formats (or lossless compressed) in the proposed DICOM compressed formats, for efficient storage. In the retrieval process, the solution will be able to revert the process, decompress the encoded pixel and return a DICOM standard file. The solution can provide also the compressed images (e.g. JXL, AVIF, WebP) directly to the client (e.g. DICOM viewer over web browser), for a performant remote visualization (i.e. reducing the latency).

Section 4.2 (Methods) starts by declaring the target environment which can run the plugin set in the first sub-section, 4.2.1. It then proceeds to review the functional requirements for this software, the implementation, and the unit-testing developed in sub-sections 4.2.2, 4.2.3, and 4.2.4. Sub-chapter 4.3, called results and discussion, summarizes the features from the user's perspective, in 4.3.1. In topic 4.3.2, the unit-testing results are shown.

Section 4.3, called results and discussion, summarizes the features from the user's perspective, in 4.3.1. In topic 4.3.2, the unit-testing results are shown.

Finally, section 4.4 concludes the chapter.

4.2. Proposal

4.2.1. Target Environment

This plugin is targeted to be run on Linux only and the validation tests were done in the Ubuntu distro. In the future, it is planned to enhance the support for other operating systems.

This is due to the flexibility provided by this environment, and since this program is a proof of concept for the modern formats, support OS-wide is not a priority.

4.2.2. Functional Requirements

As expressed, the main goal of this project is to make use of a well-known open-source PACS archive, i.e. Dicoogle, to develop a proof of concept that supports the studied modern image compression formats on medical images. It should be able to transcode the images in a DICOM file with an embedded image in pixel-data, where the provided images have not applied any lossy compression to it beforehand. The pixel-data should, therefore, be associated with a transfer syntax of *lossless encoding* or *uncompressed* for the compression to be allowed. Also, for any DICOM file that has indeed undergone this transcoding process to the modern formats, the user should be able to view the respective images.

This solution shall provide the following technical functions: definition of new transfer syntaxes, one for each image format; encoding of the images to any of the formats based on the request of the user, to be performed upon the store-scu operation; and a viewer of the newly encoded images.

4.2.3. Architecture and Implementation

In this sub-section, the two main features of a DICOM architecture plugin shall be discussed.

4.2.3.1. Store and Encoding

This plugin is responsible for the processing of incoming DICOM files, extracting pixel-data, compressing it with modern codec, creating a new DICOM file with appropriate metadata, storing it in the filesystem, and updating the repository database.

The execution of a C-Store SCP operation in the Dicoogle PACS will trigger a thread to run a store function in the Storage Plugin contract. That function contains a parameter referencing the DICOM object. The pixel-data will be encoded to a certain format, which can be chosen by the user in the solution parametrization

There is a design choice that is worth mentioning, which is the approach of using a subprocess to call the compression operation, as opposed to building java bindings to the C/C++ libraries of AVIF and JPEG XL, respectively, using the java bindings tool JNI. WebP was not mentioned in the previous statement, since it already has a java library. However, it was not used to keep the code uniform throughout the formats. In other words, the program effectively calls the command line applications to encode or decode the images.

The aforementioned design approach has the disadvantage of coupling the plugin with Linux distros OSes regarding the usage guide. For example, with Windows, the user can theoretically use Imodec. However, he must include all the codec executables in the path variable. Another disadvantage of Linux is that the user must install the dependencies, even though the steps are all specified in the usage guide.

Focusing on the encoding process of the DICOM objects, the storage plugin for Dicoogle was created with success. A schematic representation of the store and encoding operation is displayed in “figure.7”. The encoding is carried out while an object is in the process, of being stored with this plugin. The image is loaded to a BufferedImage object (from package java.awt.image) using the dcm4che2 library as the decoding provider. It is then written to a temporary directory in a lossless png format. This intermediary format serves as input to the encoding process, which is executed using the java.lang.Runtime class, using the respective encoder. Afterward, it is generated a new output file with the compressed image, such as ‘image.jxl’. The bitstream contained in that output file is then read and written over to the pixel-data attribute of the DICOM object that contains the original image.

Lastly, the following attributes are also written to the refactored DICOM object metadata: *LossyImageCompression* with the value ‘01’; *LossyImageCompressionRatio* is written with the CR value of the compression; *LossyImageCompressionMethod* is written a unique identifier of each format. Note that the DICOM standard specifies that the ISO identifier should be written in that field [9]. However, the WebP and AVIF do not have their format specification defined as ISO standards, so another unique ID is used instead.

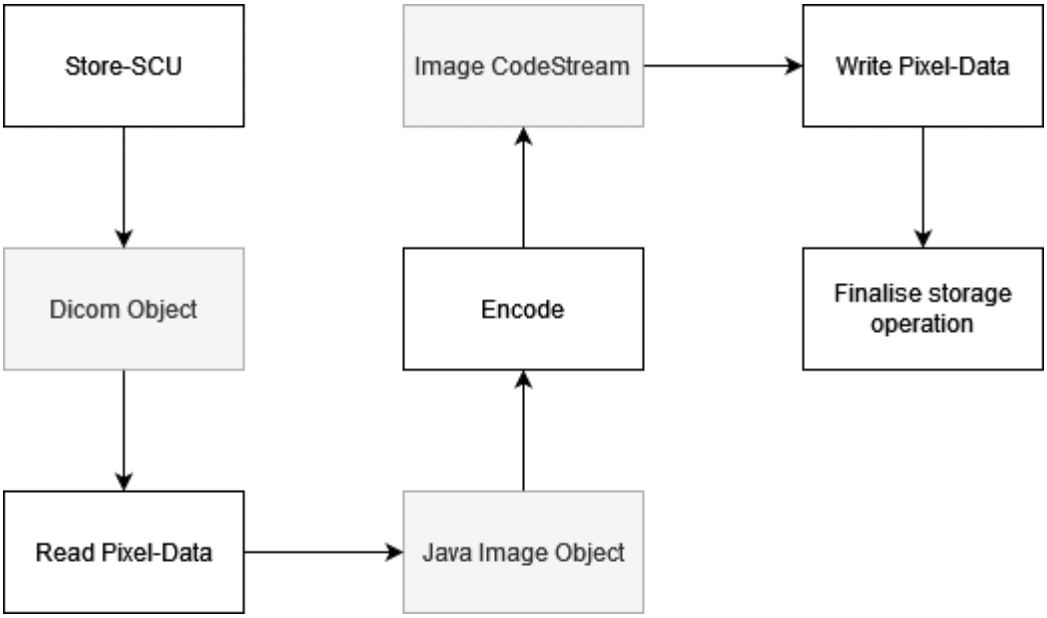


Figure 7 - Scheme for the flow of operations in the storage plugin

4.2.3.2. Decoding and viewing

This plugin is responsible for the reverse procedure, i.e. decoding de pixel-data from the new DICOM transfer syntax, and supports the visualization phase. A schematic representation of this operation is displayed in “figure.8”. This plugin is accountable for the decoding of the code-stream present in the pixel-data. Another one of its major features is the viewing module provided with the browser, where the decoded data is streamlined into the browser to be rendered and presented to the user.

To streamline the developing phase of the viewer, a minimalistic page was chosen to display the image to the user. The objective of this work was not to develop a DICOM viewer tool that, by itself, is very demanding.

For this module, a web-service, jetty servlet plugin was created. The user must provide a get request with the parameters to uniquely identify the image, such as the SOP Instance UID and the Transfer Syntax UID or format ID (with the file extension) the two latter options being required if the user wants to specify a format. If the user leaves those parameters unspecified, the native format of the image is loaded. If any of such images were not stored, an error message will appear, since naturally the images must be stored first to be viewed.

From the query parameters given by the user, the jetty plugin will try to access its storage counterpart and retrieve the respective DICOM object. The image will be loaded using the dcm4che2 java library if the transfer syntax is already present in the DICOM standard’s list. If the transfer syntax is one of the new ones covered by this project, the

decoding process is carried out entirely in the plugin-set, using the developed software and the appropriate codec tools.

The decoding process, when the format is supported by imodec, starts with the extraction of the compression bitstream present in the pixel-data attribute of the DICOM object. The bitstream is written to a file with the extension of the appropriate format, which is specified in the transfer syntax attribute. The program then also recognizes which decoder to use. The file that was created and that contains the encoding bitstream is posteriorly decoded to a png file, which is itself read and parsed, using the ImageIO package, to a BufferedImage type java object. The display phase is carried out when the buffered image is then sent to the viewing page, to be rendered and displayed by the servlet.

The former described process is specific to when the image is single frame. There is, to date, no support yet for the display of multi-frame images.

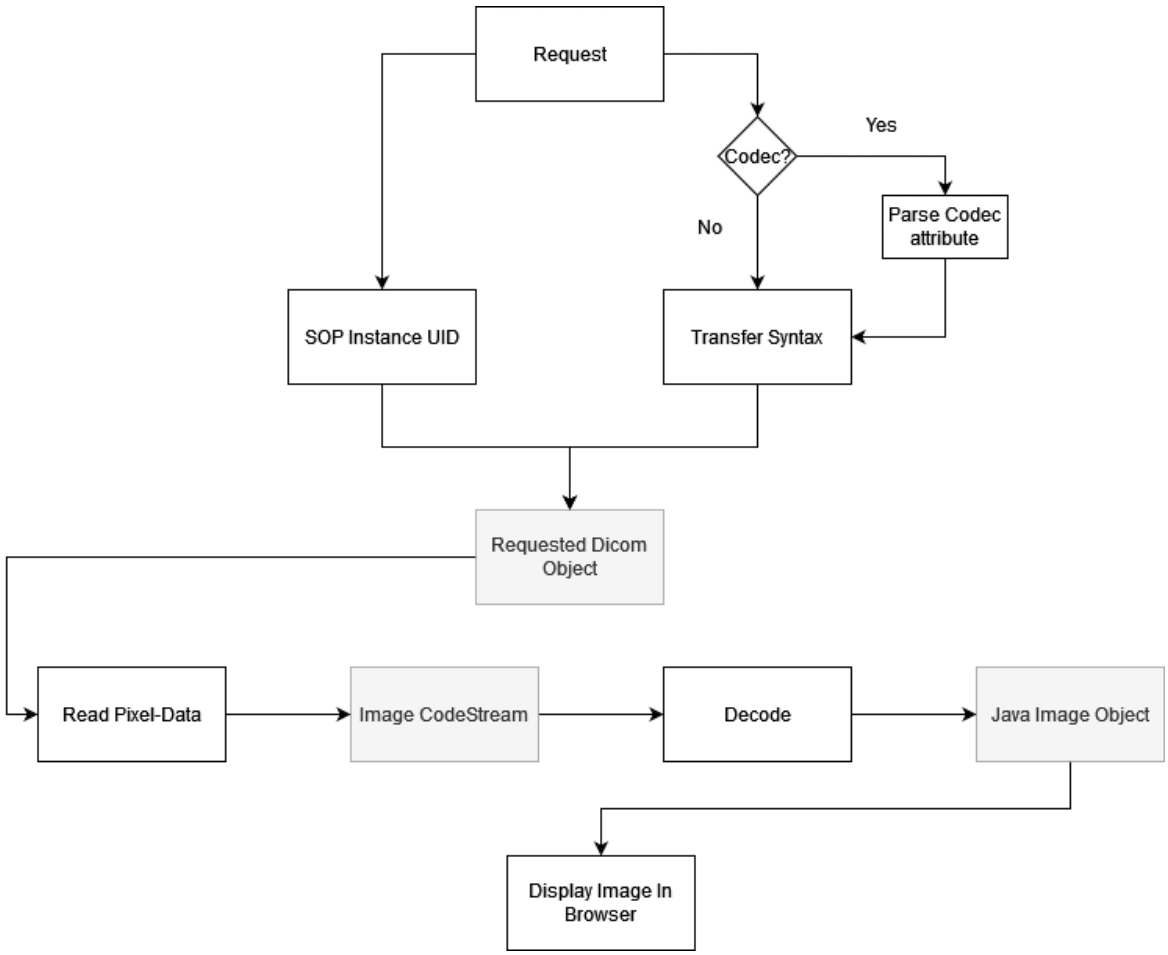


Figure 8 - Scheme for the flow of operations in the jetty servlet plugin

4.2.4. Testing

To keep quality assurance against potential bugs as well as some assurance against future accidental breaking changes, unit tests were developed. They cover utility classes that contain most of the software project's logic.

4.2.4.1. Testing environment

The specifications of the machine which ran the tests are defined in table 21.

OS	Kubuntu 22.04.1 LTS x86_64
Kernel	5.15.0-48-generic
Shell	zsh 5.8.1
CPU	11th Gen Intel i5-11400H (12) @ 4.500GHz
GPU	Intel TigerLake-H GT1 [UHD Graphics]
GPU	NVIDIA GeForce RTX 3050 Ti Mobile
RAM	7582MiB of capacity

Table 21 - Testing platform machine specifications

The dataset, on the other hand, is a collection of single and multiple-frame DICOM objects (naturally, with pixel-data). The DICOM files collection is the same one used in the benchmarking experiment carried out in chapter 3.

4.2.4.2. Testing Methodology

The library tool that was used for the unit testing was 'JUnit5' and the 'maven-surefire-plugin' was also added to the project's 'pom.xml' plugins section. This plugin allowed maven to execute the developed JUnit5 tests each time the software project is packaged or the `mvn test` command is run.

Multiple static functions were tested with these unit tests: NewFormatCodecs, MiscUtils, ImageUtils, and DICOMUtils, code that supports the transcoding capabilities provided by this plugin set. Each class's respective test class has the same name with the difference of having the *Test suffix to their name.

The NewFormatCodecs class provides functions that encode and decode images with the new formats, provided the machine contains all formats' encoder and decoder binaries. The test consisted of reading a set of DICOM files, namely the pixel-data into

`BufferedImage` objects, and carrying out a lossless encoding and posterior decoding. The decoded and original images are compared, and the test passes if they are equal.

The ImageUtils contains multiple functions used around the project, mainly loading images from DICOM objects. The tests consisted of doing the same functionality manually and doing the same with the functions, and afterwards comparing the results. This allows for maintaining backwards compatibility whenever the software is refactored.

The MiscUtils class contains logic for various miscellaneous functions. All of them are tested in MiscUtilsTest.

The last class, DICOMUtils, contains the largest functions set, and their tests verify the integrity of DICOM data objects after the operations are performed.

Some tests were only partially implemented. Such tests are disabled because they cover some unused functionality that is not currently used but can be, in a future version of imodec. Those tests are shown as skipped in the final report, and there are expected three in the instance that will be shown in the results section.

Lastly, there is a particularity among some of the tests, which are data-driven, meaning, in this case, that DICOM files are fed into them. These tests have some assumptions implemented in them that allow skipping tests when no data is found. These assumptions are not expected to fail in sub-section 4.3.2, only if another user wants to run the tests without a valid path to a DICOM dataset directory.

4.3. Results and Discussion

4.3.1. Features

4.3.1.1. New Transfer Syntaxes

The new image formats will encode the pixel-data of the DICOM objects. The transfer syntaxes define the format of the pixel-data of the DICOM objects. Thus, new transfer syntaxes are created, and shown in table 22, to define pixel-data with the bitstream of those modern codecs.

JPEG-XL	1.2.826.0.1.3680043.2.682.104.1

WebP	1.2.826.0.1.3680043.2.682.104.2
AVIF	1.2.826.0.1.3680043.2.682.104.3

Table 22 - New Transfer Syntax UID for each new image compression format

4.3.1.2. Configuring Encoding Options

This is a more advanced configuration. The user can define encoding options such as quality or speed of compression (depending on the name of the configuration parameters). A simple example of those settings is displayed in listing 6.

```
<configurations>
   <codec>all</codec>
   <jxl distance="1.0" effort="7" />
   <avif quality="90" speed="4" />
   <webp quality="90" speed="4" />
</configurations>
```

Listing 6 - Example of user-defined encoding options

The displayed settings can be found in the "*DicoogleDir/Plugins/settings/imodec-plugin-set.xml*" path, where DicoogleDir is the directory from where the user launches Dicoogle.

The *codec* tag allows the user to define which format new DICOM objects should be encoded with: *jxl* for JPEG-XL, *webp* for WebP, *avif* for AVIF, and *keep* for keeping the transfer-syntax as is. The last option is *all*, which means all the previous options at the same time, redundantly storing multiple copies - one for each aforementioned option.

The *jxl*, *avif,* and *webp* tags have empty values, holding only keyword-value attributes. Each attribute defines an option for the respective encoder, which, for the imodec version that is being described here, only supports the quality (or distance, for jxl) and speed (effort, for jxl) setting.

4.3.1.3. Storing DICOM Files

The user can store the DICOM file on the Dicoogle server with any store-scu tool available.

4.3.1.4. Viewing The Images

To examine the stored images, the user needs to input an URL to a running browser with the SOP Instance UID, and optionally the transfer syntax UID or codec id of the respective DICOM object, following the next example:

```
http://localhost:8080/imodec/view?siuid=2.25.699061500827732051810317376
15574603347&codec=jxl
```

Listing 7 - Example for an URL that can be used to view a compressed image.

After calling the URL (assuming the DICOM object with the respective SOP Instance UID was previously stored), it is expected the result depicted in figure 9.

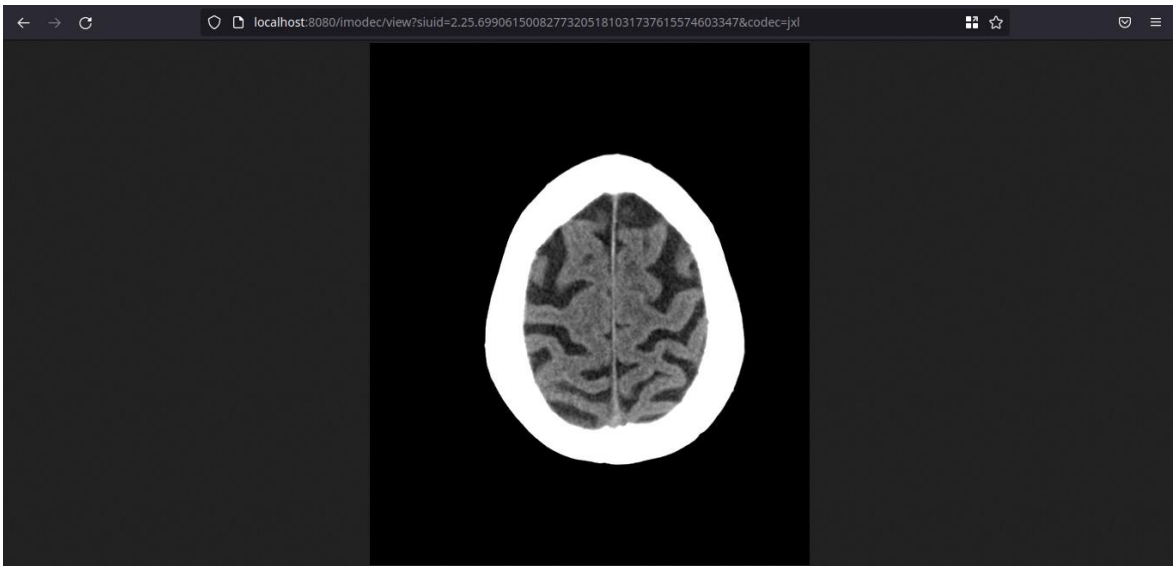


Figure 9 - Result of the GET request for viewing an image, in this case a JPEG XL compressed one.

Table 23 is a full specification of the viewer's GET request operation's query parameters, where the base URL is 'http://localhost:8080/imodec/view'.

Param Name	Requirement Status	Description
siuid	Required	SOP Instance UID of the DICOM object's image to be viewed.
tsuid	Optional	Transfer Syntax UID

		defining a version of encoding of the object with SOP Instance UID = siuid.
codec	Optional	Instead of specifying the TS UID, to see the image of a specific modern format, the user can opt to choose here which format he/her wants to see. If both parameters are used, tsuid overrides codec.

Table 23 - Get Request Parameters for the viewer.

4.3.2. Testing

As previously mentioned in the sub-section 4.2.4.2, a set of unit-tests was implemented using the JUnit5 tool, covering the majority of the project's functionality, to ensure easier debugging and better-quality code, by mitigating errors in the software logic.

All the non-disabled tests have passed, with the standard-output shown in figure 10 upon running the `mvn test` command.

```
[INFO] Scanning for projects...
[INFO]
[INFO] ---------------< pt.ua.ieeta:imodec-dicoogle-plugin-set >---------------
[INFO] Building imodec-dicoogle-plugin-set 0.1.0-b0
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ imodec-dicoogle-plugin-set ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 2 resources
[INFO]
[INFO] --- maven-compiler-plugin:2.3.2:compile (default-compile) @ imodec-dicoogle-plugin-set ---
[INFO] Compiling 1 source file to /home/archy/IdeaProjects/imodec-dicoogle-plugin/target/classes
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ imodec-dicoogle-plugin-set ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /home/archy/IdeaProjects/imodec-dicoogle-plugin/src/test/resources
[INFO]
[INFO] --- maven-compiler-plugin:2.3.2:testCompile (default-testCompile) @ imodec-dicoogle-plugin-set ---
[INFO] Compiling 1 source file to /home/archy/IdeaProjects/imodec-dicoogle-plugin/target/test-classes
[INFO]
[INFO] --- maven-surefire-plugin:3.0.0-M7:test (default-test) @ imodec-dicoogle-plugin-set ---
[INFO] Using auto detected provider org.apache.maven.surefire.junitplatform.JUnitPlatformProvider
[INFO]
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running pt.ua.imodec.util.MiscUtilsTest
[INFO] Tests run: 6, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 2.238 s - in pt.ua.imodec.util.MiscUtilsTest
[INFO] Running pt.ua.imodec.util.NewFormatsCodecsTest
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 2.127 s - in pt.ua.imodec.util.NewFormatsCodecsTest
[INFO] Running pt.ua.imodec.util.ImageUtilsTest
[INFO] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.204 s - in pt.ua.imodec.util.ImageUtilsTest
[INFO] Running pt.ua.imodec.util.GifSequenceWriterTest
[WARNING] Tests run: 1, Failures: 0, Errors: 0, Skipped: 1, Time elapsed: 0.001 s - in pt.ua.imodec.util.GifSequenceWriterTest
[INFO] Running pt.ua.imodec.util.DicomUtilsTest
[WARNING] Tests run: 10, Failures: 0, Errors: 0, Skipped: 2, Time elapsed: 196.079 s - in pt.ua.imodec.util.DicomUtilsTest
[INFO]
[INFO] Results:
[INFO]
[WARNING] Tests run: 20, Failures: 0, Errors: 0, Skipped: 2
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  03:21 min
[INFO] Finished at: 2022-09-26T18:32:11+01:00
[INFO] ------------------------------------------------------------------------
```

Figure 10 - Results of the `mvn test` command.

These tests regard the revision `6143f5dc286ff6cacaaf59b520c92230247f4f42` of the imodec line of development, which is one of the hotfixes of version 0.1.0-beta.0 of imodec.

4.4. Conclusions

This chapter covers the implementation of a set of software modules, written in Java, that extends the Dicoogle project with features of encoding any single frame DICOM files with any of the new formats, JPEG XL, AVIF, or WebP. The software also allows decoding and viewing the decoded images in the browser. The former feature is provided by a storage plugin and for the latter, a web-service (jetty-servlet) plugin was developed. These plugins are possible to implement on account of Dicoogle's software development kit [17].

The plugin set software developed provides a way for Dicoogle's users to assess the quality level of compressed medical images with the new formats, using the browser as a viewer. It is open-source with the GPLv3 license, the same as Dicoogle's, and open for more features to be built upon it.

5. Conclusions and Future Work

5.1. Conclusions

This dissertation aimed to evaluate modern compression codecs in the field of medical imaging modalities. The goal was to identify the best option according to modality, anatomic region, and the number of frames, for medical imaging storage and transmissions in internet scenarios. The work started by experimentally evaluating state-of-the-art image compression tools in the medical imaging context. For that, a literary review was carried out, reviewing the status quo of the PACS environments, and the DICOM communications and information management standard for the same environments. Afterwards, and before the survey of the recent image compression formats, another review was made for methods of evaluating the performance of image compression codecs, such as image quality degradation, compression and decompression speeds, and compression ratio. These factors are called metrics and will have a role in the following chapter, where this one closes after each of the chosen state-of-the-art formats has been reviewed.

This dissertation aimed to evaluate modern compression codecs in the field of medical imaging modalities. The goal was to identify the best option according to modality, anatomic region, and the number of frames, for medical imaging storage and transmissions in internet scenarios. The work started by experimentally evaluating state-of-the-art image compression tools in the medical imaging context. For that, a literary review was carried out, reviewing the status quo of the PACS environments, and the DICOM communications and information management standard for the same environments. Afterwards, and before the survey of the recent image compression formats, another review was made for methods of evaluating the performance of image compression codecs, such as image quality degradation, compression and decompression speeds, and compression ratio. These factors are called metrics and will have a role in the following chapter, where this one closes after each of the chosen state-of-the-art formats has been reviewed.

Lastly, a repository software solution is proposed for the application of modern formats in the medical imaging context. The proposed architecture comprises two extensions to the open-source Dicoogle project. One module is responsible for the encoding of the DICOM files upon the call of the store-scu operation for the web-server PACS. The second applicational module provides a retrieval service for consuming stored

files, compressed with the new codecs, including a light viewer of the compressed pixel-data. The latter feature of the project can be propelled by the web browsers' capabilities to decode the code-streams. The software is validated for its robustness with unit testing that covers the majority of the codebase.

5.2. Future Work

Whether viewed from a big-picture or detailing perspective, there is always room for improvement. Some ideas for future work are listed in this section.

The WebP codec contains a `near-lossless` option that encodes but takes the image's fidelity with higher priority than the default encoding mode. Experimenting with this option might help improve our understanding of the format's capabilities.

Another task could be to feed data more exhaustively upon the benchmarking tool, thus providing a better picture of the codecs' capabilities, as well as a better outlook on their performance with different modalities. For instance, we can see the JPEG XL codec's performance with 12-bit CT modality medical images, but it is not possible yet to attribute the results to the dataset being 12-bit or for being CT images. Feeding more and more diverse data onto the framework will help fix this problem.

Two statistical parsers feed from the CSV results file – one that generates graphs, other that generates a JSON file, providing a statistical overview of the results. Improving the latter or developing new parsers is also a good addition that can improve the experiments' final considerations.

The CSV results file has a column that includes all the DICOM attributes in a single string. This column should be split into multiple ones, one for each attribute. This would ease the parsing of the data table, as well as improve the readability, whenever the results need to be manually handled.

The image quality distortion metrics are an imperfect evaluation method. Introducing an easy image distortion metric extensibility feature, which would enable the user to introduce new image quality metrics, could mitigate this problem. Having more metrics can provide a better insight into the fidelity degradation of the images' contents. Also, this possible feature allows users to choose the best metrics that can be used to optimize the codecs for human as well as for computer diagnostics.

6. Annex

The associated software developed on account of this dissertation is available at the following web-repositories:

- Chapter 3 software: https://github.com/Almeida-a/ic-encoders-eval; Branch ‘main’, version ‘bf807d293fc34f8791431c230349c83e789203c1’.
- Chapter 4 software: https://github.com/Almeida-a/imodec-Dicoogle-plugin; Branch ‘main’, version ‘92bf1164323f7494a49ec618ed21e733a02ceed7, version ‘0.1.0’.

7. References

[1] D. McGeary, “PACS—An Overview,” *Biomed. Instrum. Technol.*, vol. 43, no. 2, pp. 127–130, Mar. 2009, doi: 10.2345/0899-8205-43.2.127.

[2] R. Amor, “PACS Overview: Past, Present, and Future,” *Biomed. Instrum. Technol.*, vol. 40, no. 4, pp. 281–282, Jul. 2006, doi: 10.2345/i0899-8205-40-4-281.1.

[3] H. K. Huang, *PACS and Imaging Informatics*. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2009. doi: 10.1002/9780470560525.

[4] O. S. Pianykh, *Digital Imaging and Communications in Medicine (DICOM): A Practical Introduction and Survival Guide*. Springer Berlin Heidelberg, 2008. [Online]. Available: https://books.google.pt/books?id=tFgSRghUzGsC

[5] C. Lebre, Rui; Costa, “Accounting mechanism for shared medical imaging repositories,” University of Aveiro, 2017. [Online]. Available: http://hdl.handle.net/10773/23615

[6] C. Silva, Luís; Costa, “Medical imaging services supported on cloud,” University of Aveiro, 2011. [Online]. Available: http://hdl.handle.net/10773/7245

[7] E. Pinho, “Still using gzip? Tackling data compression in modern medical imaging systems.” https://www.bmd-software.com/news/tackling-data-compression-in-modern-medical-imaging-systems/ (accessed Oct. 05, 2022).

[8] F. Liu, M. Hernandez-Cabronero, V. Sanchez, M. Marcellin, and A. Bilgin, “The Current Role of Image Compression Standards in Medical Imaging,” *Information*, vol. 8, no. 4, p. 131, Oct. 2017, doi: 10.3390/info8040131.

[9] NEMA, “DICOM PS3.3 2022c - Information Object Definitions.”

[10] NEMA, “The DICOM Standard: Media Storage and File Format for Media Interchange.” p. 48, 2016. [Online]. Available: http://DICOM.nema.org/medical/DICOM/current/output/pdf/part10.pdf

[11] NEMA, “DICOM PS3.5 2022c - Data Structures and Encoding.”

[12] NEMA, “DICOM PS3.4 2022c - Service Class Specifications.”

[13] NEMA, “DICOM PS3.7 2022c - Message Exchange.”

[14] NEMA, “DICOM PS3.1 2022c - Introduction and Overview.” pp. 1–34.

[15] NEMA, “DICOM PS3.6 2022c - Data Dictionary.”

[16] NEMA, “DICOM PS3.8 2022c - Network Communication Support for Message

Exchange."

[17] R. Lebre, E. Pinho, J. M. Silva, and C. Costa, "Dicoogle Framework for Medical Imaging Teaching and Research," in *2020 IEEE Symposium on Computers and Communications (ISCC)*, Jul. 2020, pp. 1–7. doi: 10.1109/ISCC50000.2020.9219545.

[18] F. Valente, L. A. B. Silva, T. M. Godinho, and C. Costa, "Anatomy of an Extensible Open-source PACS," *J. Digit. Imaging*, vol. 29, no. 3, pp. 284–296, Jun. 2016, doi: 10.1007/s10278-015-9834-0.

[19] R. Lebre, L. B. Silva, and C. Costa, "Decentralizing the storage of a DICOM compliant PACS," in *2021 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, Dec. 2021, pp. 2749–2756. doi: 10.1109/BIBM52615.2021.9669902.

[20] R. C. Gonzales and R. E. Woods, *Digital Image Processing Fourth Edition*, vol. 1, no. 4. 2018.

[21] M. A. Rahman, M. Hamada, and J. Shin, "The Impact of State-of-the-Art Techniques for Lossless Still Image Compression," *Electronics*, vol. 10, no. 3, p. 360, Feb. 2021, doi: 10.3390/electronics10030360.

[22] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image Quality Assessment: From Error Visibility to Structural Similarity," *IEEE Trans. Image Process.*, vol. 13, no. 4, pp. 600–612, Apr. 2004, doi: 10.1109/TIP.2003.819861.

[23] J. Alakuijala *et al.*, "Benchmarking JPEG XL image compression," in *Optics, Photonics and Digital Technologies for Imaging Applications VI*, Apr. 2020, p. 32. doi: 10.1117/12.2556264.

[24] "ISO Official Website." https://www.iso.org/home.html (accessed Oct. 05, 2022).

[25] J. Alakuijala *et al.*, "JPEG XL next-generation image compression architecture and coding tools," in *Applications of Digital Image Processing XLII*, Sep. 2019, p. 20. doi: 10.1117/12.2529237.

[26] J. Alakuijala, J. Sneyers, L. Versari, and J. Wassenberg, "JPEG White Paper : JPEG XL image coding system," no. January, 2021, [Online]. Available: ds.jpeg.org/whitepapers/jpeg-xl-whitepaper.pdf

[27] "JPEG XL Reference Implementation." https://github.com/libjxl/libjxl (accessed Oct. 05, 2022).

[28] "AVIF Related Frequently Asked Questions." https://avif.io/blog/articles/avif-faq/ (accessed Oct. 05, 2022).

[29] H. Rajora, "AVIF Image Format – The Next-Gen Compression Codec," 2020.

https://www.lambdatest.com/blog/avif-image-format/ (accessed Oct. 05, 2022).
[30] “AVIF Encoder - Rust Implementation.” https://github.com/kornelski/cavif-rs (accessed Oct. 05, 2022).
[31] “libavif - AVIF Official Library Implementation.” https://github.com/AOMediaCodec/libavif/ (accessed Oct. 05, 2022).
[32] “An image format for the Web.” https://developers.google.com/speed/webp/ (accessed Oct. 05, 2022).
[33] “WebP Implementation.” https://github.com/webmproject/libwebp (accessed Oct. 05, 2022).
[34] “WebP Encoder Documentation.” https://developers.google.com/speed/webp/docs/cwebp (accessed Oct. 03, 2022).
[35] “WebP Decoder Documentation.” https://developers.google.com/speed/webp/docs/dwebp (accessed Oct. 03, 2022).
[36] B. Gavin, “What Is a WebP File (and How Do I Open One)?,” 2019. https://www.howtogeek.com/424677/what-is-a-webp-file-and-how-do-i-open-one/ (accessed Oct. 06, 2022).
[37] D. Barina, “Comparison of Lossless Image Formats,” 2021. doi: 10.24132/CSRN.2021.3101.38.
[38] G. M. Padmaja and P. Nirupama, “Analysis of Various Image Compression Techniques,” *ARPN J. Sci. Technol.*, vol. 2, pp. 371–376, 2012.
[39] “WebP Related Frequently Asked Questions.” https://developers.google.com/speed/webp/faq (accessed Oct. 05, 2022).
[40] “FLIF By Example.” https://flif.info/example.html (accessed Oct. 05, 2022).
[41] C. Incorporated, “GIF Image File Format Specification 89a,” 1990. https://www.w3.org/Graphics/GIF/spec-gif89a.txt (accessed Oct. 07, 2022).