




FACULTAT D'INFORMÀTICA DE BARCELONA (FIB)
FACULTAT DE MATEMÀTIQUES (UB)
ESCOLA TÈCNICA SUPERIOR D'ENGINYERIA (URV)

# Fracture detection from X-Rays with Deep Learning

*Master Thesis*

Author:
**Josep de Cid Rodríguez**

Supervisor:
Dario Garcia Gasulla, Barcelona Supercomputing Center

Barcelona, 20 January 2021

# Abstract

Bone fractures are a widespread medical condition that involves many people and money in the treatment and tracking of this disease. Spain invests lots of money every year with X-Ray scanners, radiologists, and medical staff to improve patients' treatment. However, the detection of the fracture after taking the X-Ray scan is performed manually by the doctor or radiologist.

Nowadays, modern scanners produce digital radiography images with high-quality resolution and detail. Still, we are not taking advantage of these massive amounts of data to improve the current way of diagnosing, when we could apply some Computer Vision techniques to automatize or speed up the process.

The advent and constant improvement process of Convolutional Neural Networks and Deep Learning in general during the last years has totally transformed the current state of the art in Computer Vision tasks, for a great variety of problems, presenting new approaches in terms of model architectures, training techniques and also with the development and research frameworks.

However, even if the models and the overall process is excellent, Deep Learning techniques rely on the quality and quantity of the data. There are not many public datasets out there for radiological imaging that are large enough to obtain satisfactory results for real-world data, always considering the prediction error risk when we are dealing with a medical domain.

As a member of the High-Performance Artificial Intelligence research group from Barcelona Supercomputing Center, we present this project in cooperation with the mutual insurance company Asepeyo, which shared a data set of wrist radiography with us, to let us do some research and analysis about the viability of implementing a Decision Support System to help doctors to detect bone fractures from X-Ray images.

We present a pre-processing pipeline system that allows us to create a usable data set from large raw radiography data sets efficiently. Finally, we discuss the performance of several Deep Learning models on the task of analyzing the presence of fractures in an X-Ray image.

# Acknowledgements

I would like to start expressing my gratitude towards my supervisor Dario Garcia Gasulla. His guidance and insights through this project helped me build this project as well as for the writing of this document.

I must also thank all my co-workers that are co-authors of this project, mainly my companions Víctor Giménez Ábalos and Sergio Álvarez-Napagao. Víctor has contributed actively since the beginning of the development and discussion of the project. Sergio also did that but adding the complexity of managing and setting up most of the DevOps support processes to handle the experimentation and analysis pipeline. I should not forget about Anna Arias Duart and Erika Paloma Sanchez Femat. They have been part of the research and development project team for a few months.

Additionally, I also have to thank Barcelona Supercomputing Center and Professor Ulises Cortés, head of the High-Performance Artificial Intelligence research group, for giving me the opportunity to work with this group and granting me access to the HPC clusters to experiment with our system.

We have to thank Asepeyo for providing us with the data to train our models and bring us in contact with some of their radiologists for any medical consult.

Finally, I would like to thank my family and friends to encourage and support me on my path during these last years. Especially this last year, with the unforeseen consequences caused by the COVID-19 situation, which forced me to stay in Barcelona away from home for a long time.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

A bone fracture is a medical condition where the bone is broken. A great percentage of bone fractures occur because of high force impact or stress, but can also be related to the bones' weakening due to some diseases such as osteoporosis, cancers, or osteogenesis imperfecta, commonly known as bone disease. These fractures caused by medical conditions are known as pathological fractures.

Considering only the fragility fractures caused by osteoporosis in Spain in 2017, according to some public statistics[1], there were detected around 330,000 fractures with an associated medical cost of 4,200M€ that, following the predictions will increase up to a 30% in 2030, reaching 5,500M€. The numbers are big enough for just one type of fracture during a single year, which shows us that bone fractures are a medical condition that is much more common than we can imagine, including a non-negligible cost of treatment.

While there are cases where a fracture can be easily identified from an X-Ray image, in most cases, the process is not that easy if we are not experts in radiology. It is also difficult for doctors that are not experts in radiology, and there are some cases where even an expert can have several doubts about a fracture's presence. Furthermore, it is well-known that stress and repetitive tasks are prone to produce human errors due to fatigue and, because of that, most radiologists, even including the top experts in the field, will lose precision after some time.

There is a wide range of fracture types, making detecting fractures even more difficult, as there is no unified way to detect the fractures. Some of these fracture types are avulsion, where a muscle or ligament pulls on the bone; comminuted if the bone is shattered into many pieces; compression, which

[1] https://www.osteoporosis.foundation/facts-statistics

generally occurs in the spongy bone in the spine; hairline, where the bone is fractured partially, which produces one of the most challenging fractures to detect with x-rays; or pathological due to an underlying weakened bone due to some disease, among many other fracture types. All of these types show different visual features, which complicates the task of bone fracture detection.

Asepeyo is a mutual insurance company of work accidents present throughout the national territory. They have around 150 medical centers, most of them treating and diagnosing fractures, but their main problem is the lack of expert radiologists in all the centers, which slows the process if the diagnoses are not trivial or the patient requires a second opinion or further evaluation.

I am a Research Student in the High-Performance Artificial Intelligence research group (HPAI) from Barcelona Supercomputing Center (BSC-CNS). Asepeyo contacted our group to create a collaborative project to improve fracture detections' current medical process. Therefore, we must build some support system to help them achieve higher performance accuracy and time waste. For the first phase of the project, both parts came to terms to focus only on one pathology of a single anatomic part that, in case of achieving competent results, it could be updated in future phases for having to deal with general radiographic images. Therefore, they provided us with a data set of radiography images of the selected anatomic part, wrists, that we can use to train models to solve the fracture detection problem.

This master's thesis coexists with the work we are doing in the HPAI research group, so I must specify my own contributions and the ones co-authored by some of my co-workers. Most of the work developed in the data set preprocessing chapter, specially the parallelization of the pipeline in Section 4.2.1; Wrist Filtering, Color Inversion and Rotation Alignment Steps from Sections 4.2.2.1, 4.2.2.2 and 4.2.2.4 respectively; and the Aggregator model from Section 5.3 are of my exclusive authorship. The rest of the sections constitute a work of joint authorship with other group members, where my contribution accounts for approximately 50% of the work done.

After analyzing all the difficulties and complexities of the fracture detection problem, we present the solution offered by the project that we have developed during this master's thesis. We highlight that this system's goal is to serve as a clinical decision support system and not as a substitute for a radiologist or any other medical expert. The system aims to detect bone fractures from radiography images to make the process less tiresome. Furthermore, we want to identify the presence of fractures at study level, where a study can consist of a set of different radiography images taken in a single appointment to be able to visualize a wrist from multiple points of view, each one in a different scan. We explain this information model in more detail in Chapter 3.

Hereunder we list the hypothesis that we aim to verify in this thesis:

- We can create a data set from the raw data obtained from Asepeyo.
- We can train a classifier model with the data set that can achieve satisfactory performance in the detection of fractures from radiography images.
- We can combine information of the multiple views of a single study to achieve higher performance at study level compared to the accuracy at instance level.

# Chapter 2

# Background

In this chapter, we present the necessary background required to follow the remaining of this thesis document. We start by briefly introducing some literature concepts, from the basics of Machine Learning in the Computer Vision field, passing through Deep Learning until reaching the state of the art of the techniques related to this thesis's work.

## 2.1 Classical Computer Vision

To understand and talk about this thesis's background, we have to start from Artificial Intelligence (AI) basics. AI is the ability of a computer system to do tasks that are usually done by humans because they require human intelligence and discernment. Among the multiple fields grouped under the Artificial Intelligence discipline, we have Machine Learning (ML), the subset of AI algorithms that are able to perform complicated tasks learning from the data instead of using pre-defined rules or heuristics.

This work focuses on methods that take images as their input instead of raw data, which we can classify into the ML sub-category of Computer Vision (CV). We can describe CV as the field of AI that focuses on extracting features from images to identify, discriminate, detect, or classify different objects from images. The most crucial step in a classical CV approach is the feature extraction process, probably more or at least as necessary as the model itself, as the quality of the data and the features that we can extract from it will significantly influence the resulting performance. Feature extraction consists of deciding which characteristics of an object are essential to perform the desired task, and we must implement and use the algorithms that allow extracting these. It can be trivial for a few tasks, but for more advanced problems such

as ours, this would require a large team of people with some domain experts to identify these essential features.

We can use several common and well-known methods to extract features, always depending on the type of image, domain, and task that we aim to perform. Some of them are based on edge detection by convolving kernel operators such as the Sobel or Canny operators [1], corner detection with Harris Corners [2], and blob detectors with derivatives with respect to the position (Laplacian or Difference of Gaussians [3]). There are also feature description methods that usually combine the extraction and description of the features, such as SIFT [4] or Histogram of Gradients [5].

The main problem with the classical approaches is the lack of generality. All these methods have many hyper-parameters to tune that may depend on the domain of the images. Furthermore, there is no way to detect all kinds of features, as all the methods have their strengths and weaknesses, each of them being suitable for detecting one kind of feature but unable to detect others, also having the problem of not always being invariant to most of the small variations in the input, such as rotation or scale.

All the troubles detailed above are common problems of Machine Learning methods as it is challenging to extract generic and understandable features that are abstract enough to fit the whole domain of the data, and even more without an expert in the domain. In this context, Neural Networks and Deep Learning can contribute to overcoming this problem, as we will detail in the following sections.

## 2.2 Deep Learning

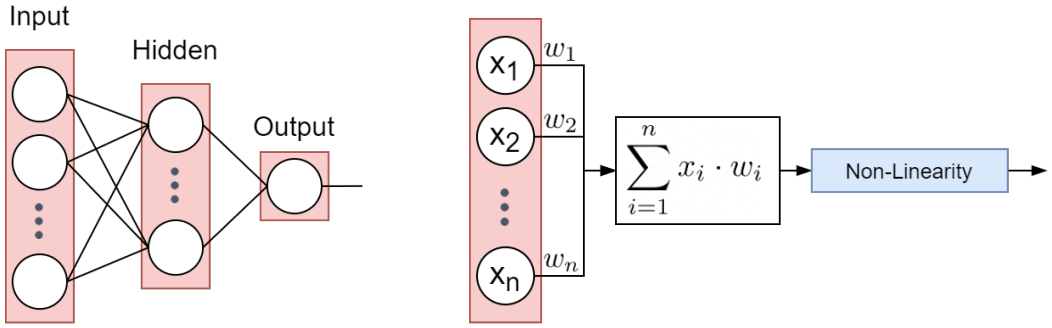
Deep Learning (DL) is a sub-discipline of Machine Learning. The main idea that characterizes DL inside ML is the Neural Network, a model that is able to detect and extract the best features for the target task automatically, without the necessity to involve a previous step to the model training of feature extraction.

In Deep Learning, the feature extraction process is merged with the learning process. Therefore, the selection is trainable, and the model itself is able to decide which features to use without having to choose which attributes should be extracted. The only drawback of this is that it requires much larger training data sets, a comprehensive data pre-processing pipeline, and an empirical hyper-parameter tuning process.

### 2.2.1 Artificial Neural Networks

Artificial Neural Networks, commonly referred simply as Neural Networks, are computing systems roughly inspired by the biological neural circuits that constitute the animal brains. These artificial neural networks consist of several layers of neurons, where the neurons from a layer are connected in a determined way with the neurons from the following layer. Neural Networks are representation learning techniques that can be used for different learning purposes, such as classification or prediction of any kind of data, and even for sample creation.

In Figure 2.1a we see a representation of a Neural Network for a binary classification problem. The first layer corresponds to the input, which can be the attributes of a data row in tabular data, pixels of an image, encoding of a word, among many others, depending on the problem and the codification of the data. The intermediate layers are the hidden layers with internal representations that are mostly humanly unintelligible, as those contain abstract features that are useful for the model to produce the final prediction. There is a single hidden layer in the figure, but in theory, there can be as many layers as necessary. The last one, the output layer, contains the value of the prediction, which in our case, as we have a binary classification problem to detect the presence or not of a fracture, consists of a single neuron which value after the application of a sigmoid function is between 0 and 1, usually considering that the predicted class is True if the value is greater than 0.5 and False otherwise.




(a) (b)

Figure 2.1: Architecture of an Artificial Neural Network.

The main improvement of Neural Networks over other predictive models, such as linear regression, is the ability to add non-linearity into the network, allowing to stack several layers to build models that are more complex and flexible than a linear classifier. The calculation for the output neuron's value

from its inputs is shown in Figure 2.1b. This process is repeated for every neuron in each layer with the adjacent neurons.

### 2.2.2 Convolutional Neural Networks

A Convolutional Neural Network [6] is a class of neural network specialized in dealing with grid-like topological data with close spatial relations, such as images or sequences. The principal difference between a vanilla ANN and a CNN is the limited connectivity and shared weights among the filters of the CNNs, as usually, only the last part of a CNN is a fully connected network. The main drawback that causes the vanilla ANN not to be suitable for image tasks is that the network size leads to an easy over-fitting, while CNNs do not, as the neurons are not connected to all other neurons in the previous layers but connected only to a small region of neurons, which results in less trainable parameters, and therefore, less risk of over-fitting.

CNNs have three basic types of blocks [7]. The main one, which has the trainable parameters are the convolutional layers, consisting of several small filters (e.g., 3x3 or 5x5), which are convolved with the input of the layer, learning features from the given spatial location. Each filter produces a feature map when sliding through the whole network, combining them channel-wise to create the next layer's input.

Another important type is the Pooling layer, which helps to downsample and compress the given data, reducing the previous layer's feature maps, which helps to reduce over-fitting, reduces the number of parameters and hence, the training time, and makes the model more robust to variations in the position of the features.

A CNN usually consists of a stack of convolutional layers interspersing pooling layers, always applying a non-linear activation function after the convolutional layer. The convolutional part's result is flattened into a one-dimensional vector and passed through a classifier, which is usually a fully-connected network, an ANN. We can see a schematic architecture of a traditional CNN in Figure 2.2. However, fewer and fewer modern model architectures use fully-connected layers, being fully-convolutional models or using other approaches.

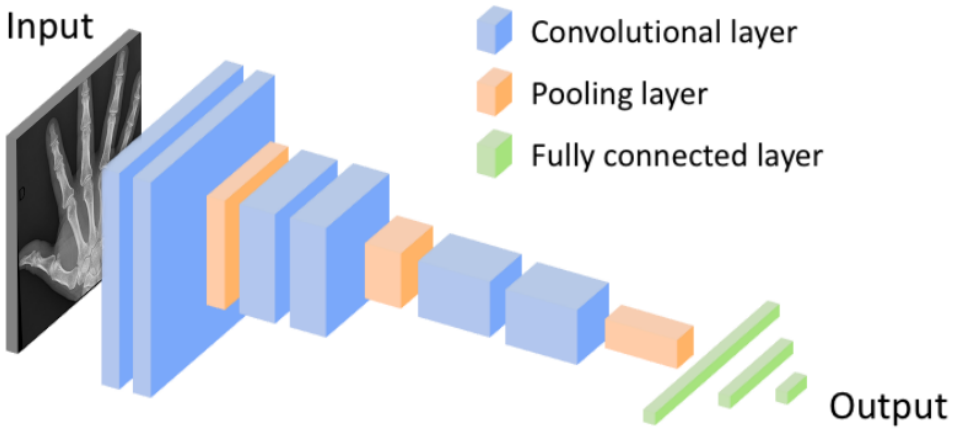



Figure 2.2: Typical Convolutional Neural Network Archtiecture.

## 2.3 Radiologic Approaches

There are many applications of Machine Learning techniques in medical imaging [8]. In this section, we specifically analyze the state of the art of Deep Learning methods applied in radiology.

### 2.3.1 Tasks

Despite that there are multiple related tasks such as segmentation, image generation, enhancement, or retrieval, we describe the tasks involved in the problem that we aim to solve in this thesis, which are classification and detection, and only the ones related to imaging, without going into detail with other kinds of data such as text models or 3-dimensional imaging.

#### 2.3.1.1 Classification

The goal of a classification task is to predict a label or class from the given object. In radiology, many problems consist of a classification task, such as distinguish abnormalities as benign or malignant, classify cancerous lesions from the genomic features, prognostication, or, as in our problem, detect bone fractures, among many other possibilities.

The vast majority of works use a CNN as the classifier model given that, as we explained in Section 2.2.2, these Deep Learning models allow an abstraction from the feature extraction process.

One of the main drawbacks of Deep Learning models is the large amount of data required to train the models. This drawback is a problem for radiology tasks because the availability of data is somewhat limited compared with other domains that have experienced a significant boost in terms of performance with the evolution of these techniques in the last years. To deal with this issue, some works work with off-the-shelf models and features [9, 10], combining hand-crafted features with off-the-shelf features extracted from a pre-trained VGG network to classify breast lesions.

Other works overcome the issue by applying transfer learning techniques [11, 12], which takes a pre-trained model on a different dataset (or even a different domain) and fine-tune the network on the dataset for the target problem, used in the classification of prostate cancers from magnetic resonance images (MRI) [13], identification of pulmonary tuberculosis from computed tomographies [14], or identification of hip osteoarthritis from radiography images [15].

Most of the studies use transfer learning to freeze and retrain the deepest layers of the network while replacing the top ones with new layers or fine-tune the pre-trained model on a new dataset to get better task-specific results. For example, works such as [16] applied transfer learning over a pre-trained GoogLeNet [17] doing fine-tuning for the specific task of classifying thyroid nodule from ultrasound images. Some works show the comparison between using networks trained from scratch and fine-tuned models [18, 19]. In both cases, the model trained from scratch performed better than the fine-tuned network. However, both face the issue of having a small training dataset.

If there is enough data available, it is also possible to train the network from scratch, with random initialization. Most of the network models are based on some of the most common and basic architectures, such as AlexNet [7], or VGG [20] with some modifications that tend to reduce the number of layers and weights, to solve problems such as detecting Alzheimer from brain MRI [21, 22], and grading or sating for glioma tumors from MRI [23] or chest prognosis from computed tomographies [24].

Recent convolutional network architectures from the last years, allow us to train better models with less parameters and more efficiently. Some of these architectures are the Inception architectures [17, 25, 26], Densely Connected Convolutional Networks [27] or the Residual Networks [28], used in from-scratch trainings [29] or used as fine-tuned networks for classification [30] or as feature extractors [31], which outperformed the previously used models.

Other approaches use more exotic architectures for classification purposes, such as Auto-Encoders [32] or stacked Auto-Encoders [33, 34], training those from scratch in an unsupervised way, for example, to determine the presence

of Alzheimer using stacked denoising Auto-Encoders [35], classification of lung nodules [36], and the identification of multiple sclerosis lesions from MRI [37]. These models are used to extract feature representations from the hidden layers for further classification.

#### 2.3.1.2 Detection

The goal of a detection task is to locate and mark an object in an image. In radiology, it is crucial to detect and extract a region of interest for further classification or segmentation [38, 39, 40].

Standard solutions to this task combine two phases. The first one is a high sensitivity region of interest candidate detector [41]. A typical approach consists of using a regression model to predict the bounding box coordinates of the region of interest [42, 43], based on many of the standard architectures commented in the previous section, which may produce a large number of false positives. With the high sensitivity of this first phase, it produces a large number of false positives. This is where the second phase comes into play, classifying the regions extracted with the previous step and filtering those that are false positives. Like classification tasks, most of these models are often pre-trained using other datasets [44, 45].

The models from the two phases are initially trained separately but trained together during the last training stages. We can also find some directly end-to-end architectures, such as Region-based Convolutional Neural Networks (R-CNN) [46], or their improvements Fast R-CNN [47] and Faster R-CNN [48]. These models use convolutional networks to extract a feature map and a classification network that predicts each candidate's category. These are some examples of these models used for intervertebral disc detection from X-ray images [49] or colitis detection from computed tomographies [50].

Finally, some single-step alternatives eliminate the first candidate phase, using well-known approaches such as You Only Look Once (YOLO) [51] with specific variations for breast mammograms with BC-DROID [52]; Single Shot MultiBox Detector [53] for breast tumor detection from ultrasounds [54]; or RetinaNet [55]. These YOLO-based solutions present a trade-off that balances a higher speed and fewer data requirements, with a lower recall and more localization errors compared to Faster R-CNN, as well as the struggle to detect close and small objects.

In this thesis, we are going to focus on the binary classification part about the presence of a fracture in radiography, as this is the goal of the first phase of the project with Asepeyo. Even though we plan to focus on detection in future phases of the project, we should not forget about this part, given that

both tasks are connected, and we can not analyze and propose classification models without taking into account this subtask.

### 2.3.2 Ethical Concerns

When we talk about AI, there are always ethical challenges involved, as most of the systems require data from the users to have a successful performance and keep improving. Some studies in the United Kingdom from a few years ago [56] show that even if the data is anonymized, most of a 63% of the adult population do not feel comfortable with allowing the usage of their personal information to improve healthcare, and even medical students feel skeptical about thinking that AI will be able to establish a diagnosis, especially in image-related tasks [57], such as in our project.

One solution to have a better view of these systems is to build Decision Support Systems (DSS) instead of systems that entirely replace the human-in-the-loop. This is especially important for healthcare tasks, as even though medical staff can commit errors, we are able to analyze and improve them, and they have common sense, but there are no exact metric evaluations for AI systems, which may cause unforeseen consequences. However, if we take as an example the DSS in healthcare organizations from the United States, almost half of the medical staff that uses these systems think that they can produce fatal errors or that there is an over-hype on what a DSS can do or achieve [58].

There are several fronts that we must consider when we work with or develop these kinds of systems. We can group them into three blocks, related to the data, product, and its deployment. Regarding the data, we must ensure the correct anonymity of the data and the access to it, as it tends to contain sensitive data regulated by the data protection acts of each country or region.

When developing the solution, we must also consider if the data, and therefore the model is accurately representative of the whole target population as well as trying to overcome any font of bias. It is also essential for a DSS to offer some kind of explainability, which is currently a hot field in AI.

Finally, we must ensure a proper monitoring and tracking system to fix and detect any possible unexpected events when we deploy the solution. The involved people also need to be informed about the presence of the new system and its scope. This process also includes the patients, not only the doctors, to have clear transparency standards.

We do not offer our system as a deployed solution, but we should not forget about all these. Nevertheless, all the data we work with is anonymized, with all the sensible metadata fields are removed from it.

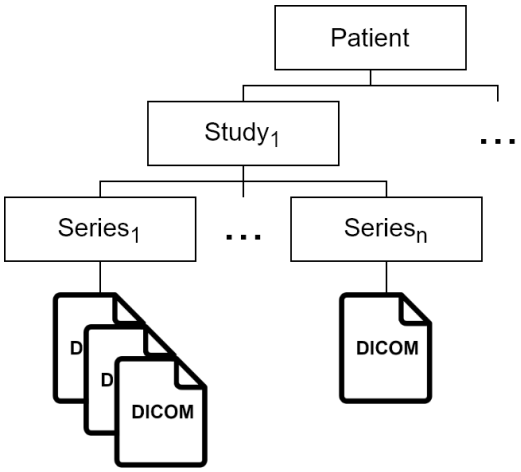
# Chapter 3

# Problem Description

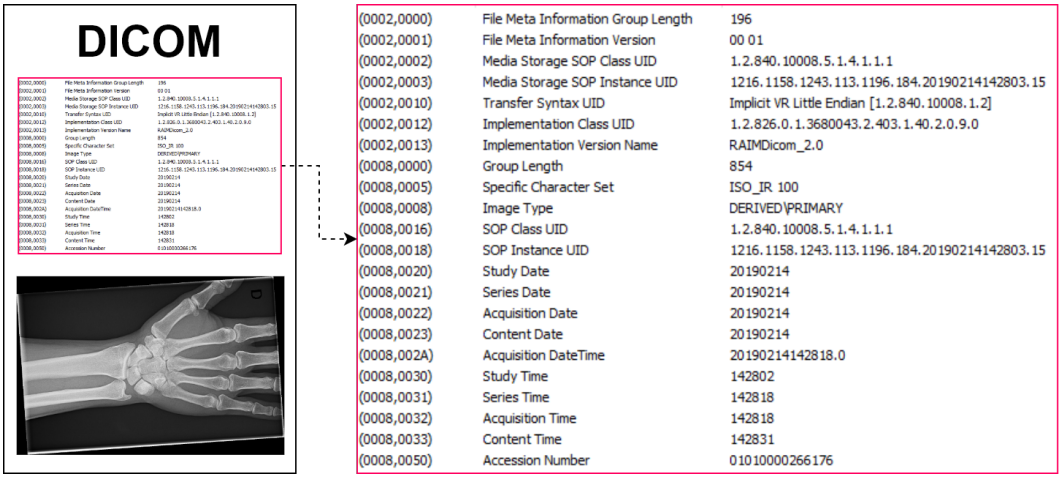
This project's primary goal is to detect the presence of bone fractures from radiographic images of wrists. As we have introduced before, we decided to focus only on wrist images for this phase of the project with Asepeyo to, later on, extend it for images of any other anatomic part. The used data format is DICOM, a specification for the creation, transmission, and storage of digital medical image and diagnosis data. It defines a data dictionary, data structures, file formats, client and server services, workflow, and compression, among other things.

The data structure that constitutes the information model is composed of a set of studies. A study represents an exam or procedure corresponding to the set of radiography images taken in a single doctor's appointment regarding a single patient. At the same time, each study consists of one or multiple series. A series generally equates to a specific type or modality of data or a patient's position on the acquisition device, for example, lateral, frontal, or axial. Each series may contain multiple DICOM object instances that commonly consist of image data but also can represent other kinds of data such as reports or waveform objects.

In Figure 3.1 we show a schematic representation of the information model. More specifically, in our data, we deal with multiple studies composed of single series, each of those with one or more DICOM instances.




(a) Information model schematized.




(b) DICOM files internal representation. DICOM format follows a dictionary-like structure where the key corresponds to an internal code associated to a type of metadata, e.g. acquisition time or accession number. It has also an special element, pixel data, which contains the radiography image.

Figure 3.1: Information model.

It is important to remark that in our data set, each sample corresponds to a radiography instance. However, the goal of our task is to classify each of the studies to generate a label for the whole set of instances that correspond to the same study, as we are diagnosing the presence of a fracture in the whole appointment, which, as we have commented, can consist of multiple scans.

## 3.1 Data Statistics

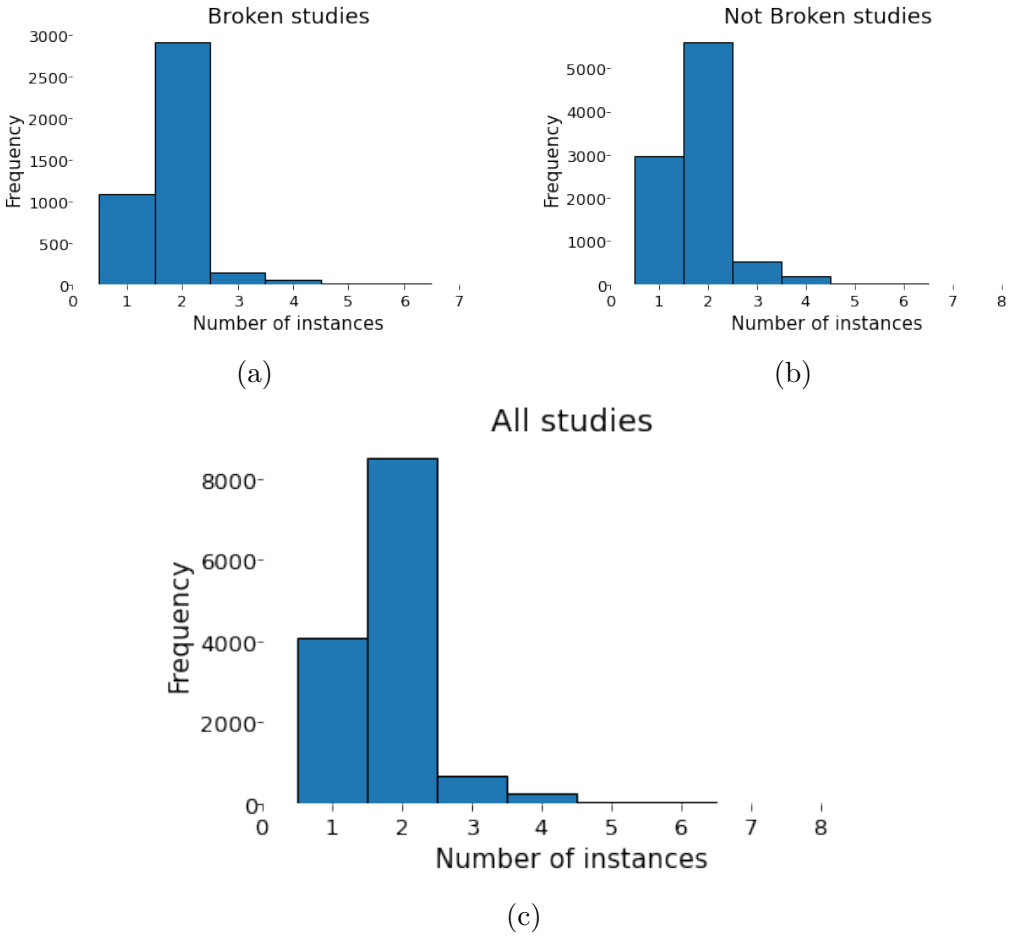
Our original data set has a total of 13,606 studies. However, we read the data and the labels from different sources, and we detected few mismatches of studies with missing labels or vice versa. After removing these cases, we have 13,538 potential studies that we can use to train and validate our model. One of the first problems that we notice when we inspect the data is the data unbalance for broken and not broken classes, having 4,214 studies tagged as broken and 9,324 as not broken, which results in a proportion of 31.13% and 68.87% respectively.

Data unbalancing is a recurrent problem in real-world data sets that may affect the training process. It may not be something problematic, depending on the application. If we consider an example where class A happens occurs 99% of the time and B only the 1%, the model will tend to classify any sample as class A, obtaining a 99% of accuracy. The model will be correct, and it is unlikely that any other model improves the performance. However, in some cases, the critical task is to see what happens with the B class, even if it only occurs once in a few times. This last situation is precisely the case that we face with our problem and data, where we are interested in detecting fractures, the minority class. A solution to this problem is to intentionally bias the data to get a model that, even though it may perform worse in terms of accuracy, can give us better results regarding what we are interested in detecting. We address this problem with some solutions in the next chapter, specifically in Section 4.3.

The number of instances for each study varies a lot. In Figure 3.2 we can observe the histogram of the number of radiography instances for each study. To generate the histograms, we first remove an outlier study that has 195 instances. This study corresponds to a set of axial radiography images, and we opt to discard it from our data set. From the histograms of broken studies (3.2a) and not broken studies (3.2b), we can see that both classes follow the same distribution of the number of instances per study, so we can simplify the analysis and consider the overall histogram of all the studies (3.2c).

We see that the vast majority of studies have one or two instances, most of them with a pair of instances that usually correspond to a frontal and lateral views. There are also a few studies with three or four views and a tiny number of studies with between five and eight different views.

Unlike many academic or toy data sets, our data do not have a typical shape either similar aspect ratio between all samples. In Figure 3.3 we can observe the great variety of sizes with a scatter plot that shows the width and height of all the images in the data. Some sizes are quite common, such as




(a) (b)

(c)

Figure 3.2: Histograms of the number of instances per study.

500x500 or sizes around 3000x2500. However, the vast majority have non-common shapes, gong from images with 250 pixels on one side to others with 4,000. This lack of standards is also a problem that we analyze and fix later in Chapter 4, by applying some transformations and pre-processing steps, as well as in the first experiment from Section 6.2.1, checking the trade-off between smaller resizes and information loss.

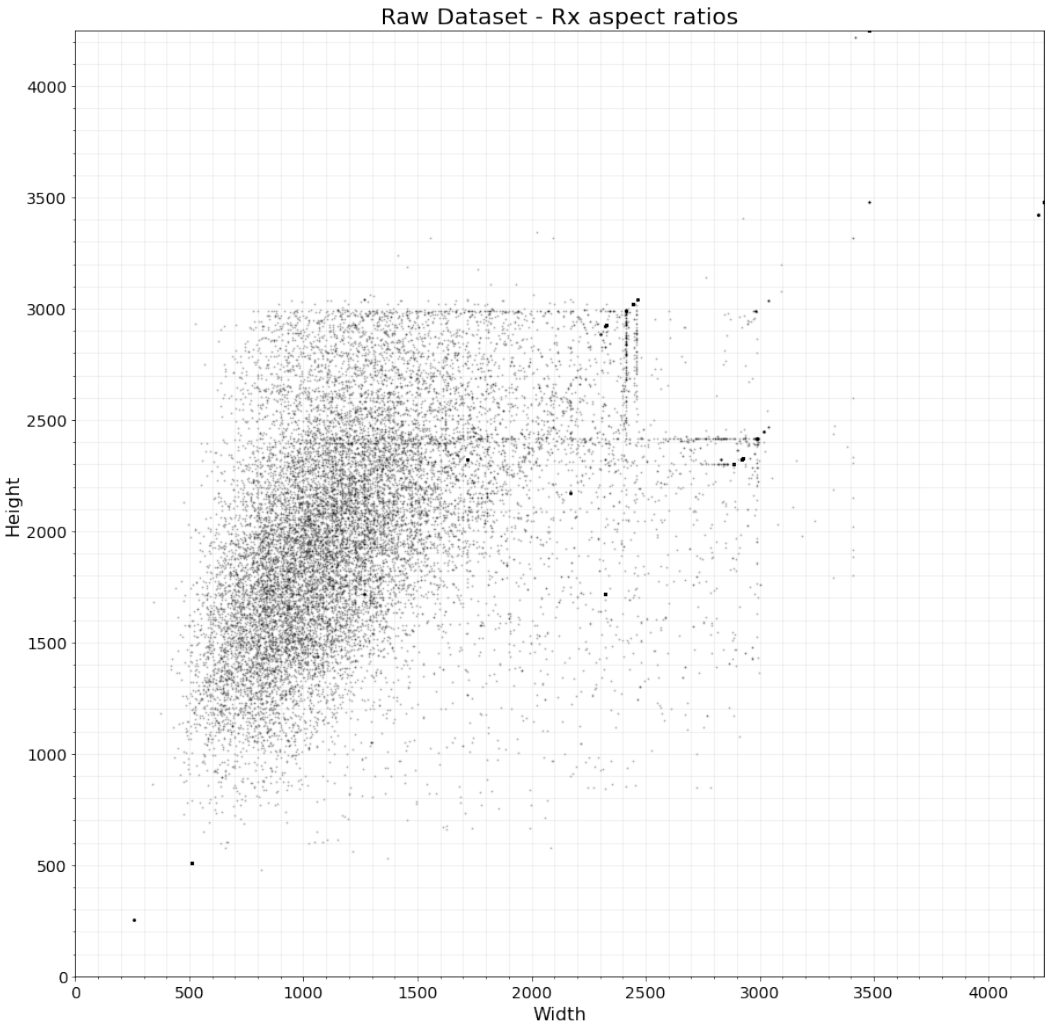



Figure 3.3: Aspect Ratios of the Raw Data plotted in terms of width and height of every sample. Thicker dots represent a larger number of samples with that specific size.

# Chapter 4

# Dataset

In this chapter, we explain the pre-processing steps and pipelines developed to clean and simplify the data efficiently, so we can use it with our models and achieve better performance.

To fully understand this chapter, recall Chapter 3, where we explain the conceptual parts that define the data, such as the distinction between a study and radiography or instance, as well as the data distributions and aspect ratios of the original raw data, without applying any kind of pre-processing.

## 4.1 Raw Data Transformation

During the first experiments on modifying the data to check the effects of different pre-processing techniques, we noticed that one of the heaviest bottlenecks for this problem is the DICOM data format. DICOM format is handy for handling medical data, especially for medical field end-users, such as radiologists. The format is able to handle all the information about the medical image centralized in a single file, combining the medical image itself with all the complementary metadata that allow the specialists to get more details than with just the image, all this, following a standardized format that unifies the way of working with medical data that has a wide assortment of metadata fields.

However, this format is not suited for computing a high number of samples efficiently, as files can be quite heavy. After performing an execution time profile, we detected that a considerable amount of the time was spent with the file reading process rather than the task itself. This is probably because the libraries that handle DICOM processing are not as extended in terms of use as other common libraries that work with standard image formats or CSV

files.

Being aware of this performance bottleneck, we decided to decouple the pre-processing pipeline into two separated processes being this the first one, named Raw Data Transformation, which is in charge of transforming the DICOM files into PNG images and CSV files, these last ones, to store the useful metadata information. The tool used to read and extract information from the DICOM files is the `Pydicom` library with `GDCM` as the pixel data back end library.

While running the Raw data transformation process, we found out that some samples contained problems, either in the image, which was corrupted, or in the metadata with corrupted or unusually codified internal fields. For these cases, we dropped the sample if the data was impossible to recover, such as with corrupted images, because the image is the keystone of the samples. We also removed the problematic metadata fields for all the samples to simplify the output data. These dropped metadata fields corresponded in their majority to the software or hardware piece's internal identification codes, so it will not cause a problem to get rid of them.

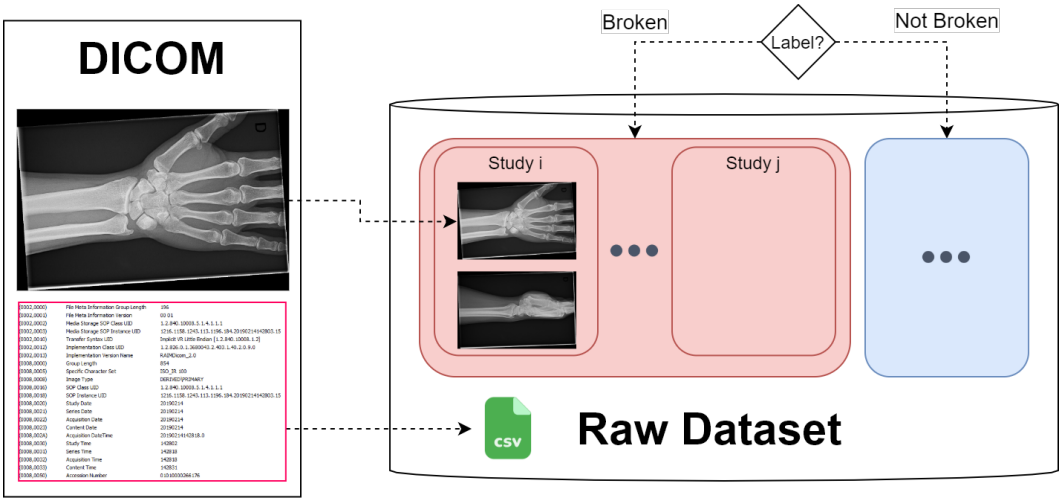



Figure 4.1: Schematic design of the raw pre-processing pipeline.

In Figure 4.1 we illustrate the whole Raw Pre-processing process. After this raw data transformation, the dataset is already split internally between Broken and Not Broken classes in separated folders to ease the process of extracting or identifying specific cases for each of the labels to predict. Inside each of the classes folder, we can find all the studies that are tagged as the corresponding label, and inside of each of them, we have the list of radiography for that study in PNG format. In the root folder of the output data set, we have a

metadata CSV file in which columns are all the available metadata fields in the DICOM files, excluding the removed ones. Each row of this file corresponds to a radiography, indexed by a radiography and study identifiers as two new fields. This gives us a CSV file with a row cardinality corresponding to the total number of radiography instances. We extrapolate the label at instance level from the original CSV data as we are given a single label for the whole study, independently if it is possible to observe a fracture in a radiography or not. In Chapter 6 we experiment and discuss the ways to aggregate the labels and the possible sources of noise that these can cause.

## 4.2 Dataset Preprocessing

Once we have all the data in PNG and CSV format, we can start with the proper pre-processing process. This section is dedicated to the pipeline that we define for the pre-processing task. The speed-up that we obtained when removing the DICOM dependencies allowed us to perform a higher number of experiments and helped us determine a better configuration of pre-processing steps and their respective parameters.

Figure 4.2 shows the pipeline's final steps used to pre-process the data set, and for each of them, in the following sub-sections, we explain in detail the motivation for using it, as well as the set of parameters that it requires to achieve a good performance.

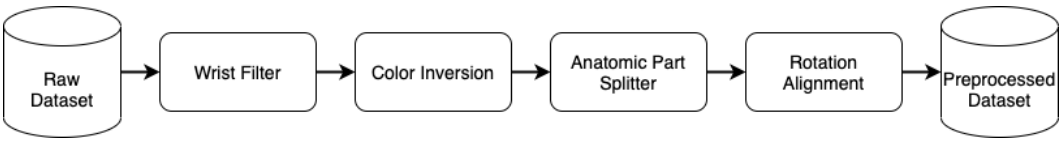



Figure 4.2: Pre-processing pipeline design.

The pipeline is a sequential single-step operations module, each of them idempotent and producing the same output format as the input. Because the output format and structure does not change with respect to the input, we maintain the PNG-CSV data duality and the same folders and files structure.

### 4.2.1 Parallelization: Improving Performance

After the first executions, we could observe that the pre-processing job took more than 30 hours to complete the pre-processing of the whole dataset,

running the process in the BSC's supercomputer cluster MareNostrum[1]. Nevertheless, we can do it better because, given that there is no inter-dependence between studies, we can pre-process each of them in parallel, taking advantage of the high number of CPUs available in the MareNostrum cluster. However, the disk's current data allocation is a bit complex to manage the bottleneck between processes. The reading process from the big metadata file is the leading cause of the bottleneck. To fix that in a simple way, we slice the file into multiple metadata files, one for each study, containing the information about the radiography associated with that study. In Figure 4.3 we define the final file scheme of the raw pre-processing pipeline.

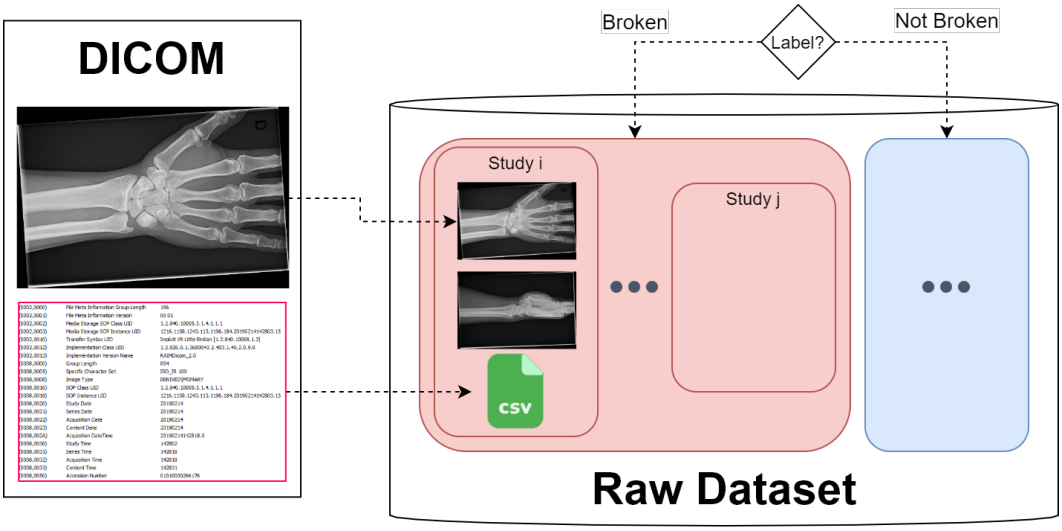



Figure 4.3: Improved schematic design of the raw pre-processing pipeline, adapted for parallelization purposes fragmenting the CSV metadata in small files for each study.

By doing so, we are able to fully parallelize the process, given that every study is self-contained and there are no dependency nodes that need to be tracked. Using 48 CPUs of the same type, we are able to run the whole job in only one hour. This performance improvement boosts, even more, our capacity of generating new data sets with different pipeline configurations, such as the hyper-parameters of some steps or the order of these, without having to wait more than one day to play with the new data.

[1]MareNostrum's CPU: Intel Xeon Platinum 8160 24C at 2.1 GHz

### 4.2.2 Preprocessing Steps

Hereunder, we explain the pre-processing steps in the same order that they are applied to each study. We have a pre-processor module that acts as a sequential step applier at study level. Therefore, we iterate over all the studies in the given data set, and for each study, we apply sequentially the steps $s_1..s_n$, each one receiving as input the output of the previous step, or the original study in the case of the first step.

#### 4.2.2.1 Wrist Filter

We recall that we are only focusing on wrist radiography data for this first phase of the project. Because our training data came from a real-world data set, its raw format is not very clean. Despite trying to train a model that explicitly detects wrist fractures, we noticed a non-negligible number of non-wrist radiography images, such as craniums, pelvises, or torsos, among many other anatomical parts. Even that there exists a metadata field that identifies the anatomic that appears in the radiography, but we found several samples wrongly tagged, e.g., a torso image tagged as an arm, and a lack of consistency between matching tags.

After checking this with radiologists, we saw that, except for the most current systems, the majority of the medical software that is used to process these DICOM files from the X-Ray scanner treat this field as a free text field (among many other critical fields for identification purposes), which gives the doctor absolute freedom to write the tag, resulting in multiple languages, inconsistent capitalization, lack or presence of accent marks, among many other incompatibilities for the same label.

For this step in which we have to deal with a free text field, we start by computing and storing a histogram of the values contained in the anatomic part metadata field. In Figure 4.4 we show this histogram. There is a total of 187 different labels, but to simplify, we just show the first 20, mainly because the occurrences approximate a Zipfian distribution, and the 21st most common label has only 68 occurrences in front of the total 21,372 analyzed labels. Most of these later elements are repetitions with some spelling alterations.

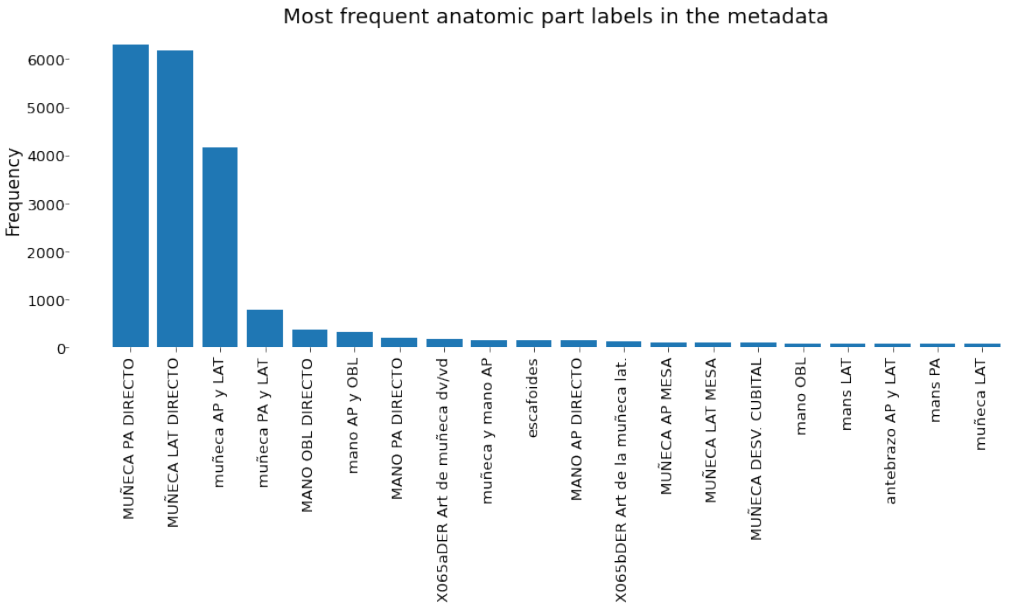



Figure 4.4: TOP-20 most frequent labels for anatomic part identification

After the histogram extraction, we analyze the information contained in the labels that are related to wrists, and we build a filter that handles if the input samples have a wrist-related label. This filter needs to be better than simple string matching, so we consider slight differences such as strange character codifications, case-sensitivity, or missing characters.

This step is essential because we aim to reduce as much as possible the noisy samples that we introduce into our model, and given that the percent of non-wrist samples is not very large but still significant, it is essential to filter these. It is also the first step to be executed in the pipeline to avoid performing useless computations of other steps on samples that will be lately filtered out and speed-up the process as much as possible. This step removes a total of 724 different instances.

#### 4.2.2.2 Color Inversion

Instead of following the common black background with white bones that we are used to, many samples followed the opposite color scheme. This can be produced by different X-Ray scanners' models or after some possible modifications of the radiologists that analyzed the samples, as their software allows them to change the contrast or color scheme of any radiography for better human visualization purposes.

This step aims to detect the images which color schemes are inverted, i.e., white background with black foreground. These images are inverted to obtain the usual color scheme, a black background with white foreground, to obtain a consistent image distribution in terms of color to indicate that lower values (black) are the foreground, while higher values correspond to the bones (white).

One solution would be to build another binary classifier model that is able to discriminate between both color schemes. However, the lack of labeled samples and the apparent simplicity of the task led us to create and follow a simple heuristic that extracts information from the corners of the image and calculate their mean, following the formula in Equation 4.1a. If the resulting value is higher than a certain threshold, we consider that the background is white, so we must invert the color scheme, as we express in Equation 4.1b. As the image is expressed in `uint8` data format, the pixels are in range 0..255, so we just need to subtract each pixel from 255, obtaining the color reversed image.

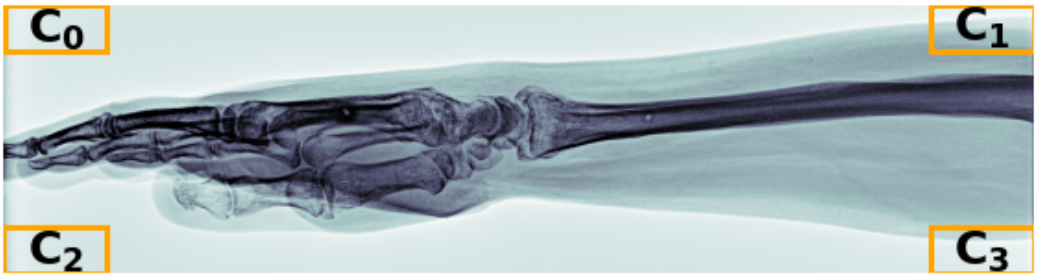
$$S_{img} = \frac{1}{n}\sum_{i=0}^{n-1}\left(\frac{1}{C_W \cdot C_H}\sum_{x=0}^{C_W-1}\sum_{y=0}^{C_H-1} pixels_{img}(C_i^{x,y})\right) \tag{4.1a}$$

$$\text{color\_inversion(img)} = \begin{cases} S_{img} \geq T & 255 - img, \\ img, & \text{otherwise.} \end{cases} \tag{4.1b}$$

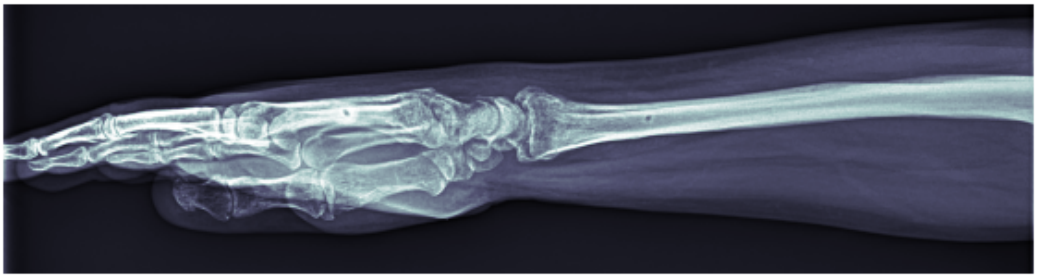
Many images do not have a clear black background, but one with gray tones, although sometimes there are also some noisy white lines in the edges. After some experiments and to avoid setting a threshold value that is too low for these cases, we defined this threshold value to 100 (images are in the range 0..255). We illustrate the process in Figure 4.5, where applying this step transforms the image in Figure 4.5a to its equivalent but with the opposite color scheme (Figure 4.5b).

This is an important step that we use to standardize the input data, simplifying the data domain, so the model always sees the black parts as the background and the regions of interest, the bones, are gray and white. It is the second one in the pipeline because we need the images to follow the proper color scheme to perform the next steps defined in Section 4.2.2.3 and 4.2.2.4.

Later on, as we comment in Section 4.4, we also experiment with the use of this pre-processing step as a data augmentation technique to overcome the possible errors caused by this step outputs due to its simplicity, as well as allowing the model to see a greater variety of data.




(a) Image with its original color scheme. The inverted color scheme must be detected and changed to fit the proper distribution. The four extracted corners $C_{0..3}$ are highlighted with orange rectangles.

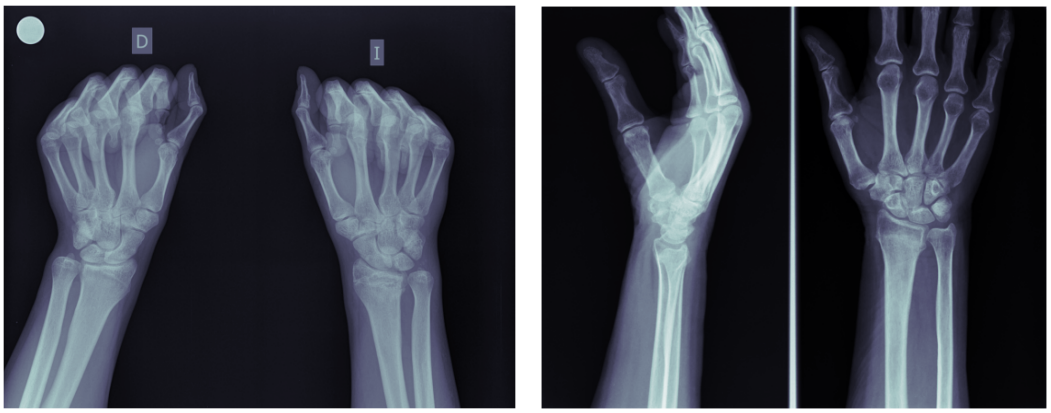


(b) Resulting image after applying the color inversion step.

Figure 4.5: Data set image with the colors inverted (4.5a) that is converted into the proper format (4.5b) after applying the color inversion step defined in Equations 4.1.

#### 4.2.2.3 Anatomic Part Splitter

Some of the images contain multiple anatomic parts in the same radiography. In Figure 4.6 we can observe some of these examples. One of the reasons that can produce these images is that both arms are placed together in the same scan (4.6a), but it can also happen that the radiologist has joint both images side by side in a single image to compare or check both of them quickly (4.6b).

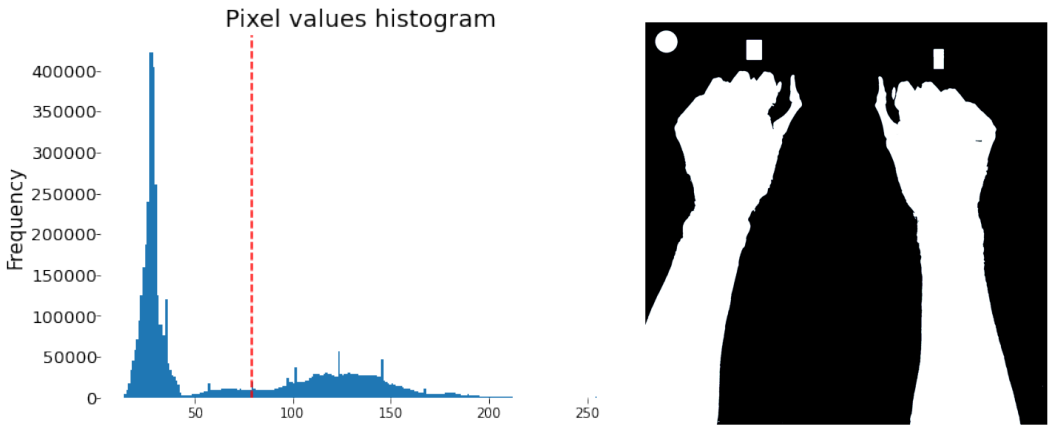



(a) Both wrists scanned side by side. (b) Wrists joined after the scan.

Figure 4.6: Radiography samples with multiple anatomic parts in the same image, for different possible reasons.

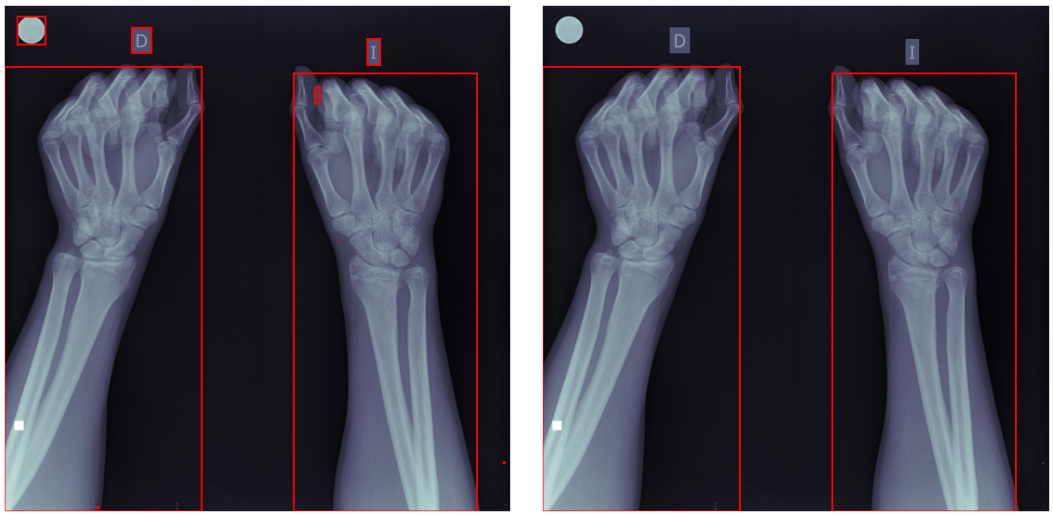
To detect and separate these sub-structures into individual anatomic part images, we use some classical Computer Vision techniques to extract and filter connected components. These methods work better with binarized images, where black represents the background and white is the object or foreground.

We start with a binarization of the image using Otsu's method [59] to calculate a binarization threshold value $\delta_{img}$ with respect to the image automatically. Otsu finds the $\delta_{img}$ value by classifying pixel values into two classes and maximizing inter-class variance. All the values greater or smaller than $\delta_{img}$ will become white or black, respectively. As $\delta_{img}$ is different for every image, we do not need to define a hyper-parameter, and it always uses a good value to perform the binarization. In Figure 4.7 we see the binarization of the previously seen radiography in Figure 4.6a.




(a) Otsu's threshold from the histogram. (b) Binarized image.

Figure 4.7: Binarization process of the image in 4.6a, calculating the Otsu's threshold with the pixel values of the image (4.7a) and setting every pixel with a value greater than $\delta_{img}$ as white and the remaining as black (4.7b).




(a) Before filtering. (b) After filtering.

Figure 4.8: Connected components extracted from the binarized image, marked with red bounding boxes. In 4.8a we see the components extracted directly from the binary image, that include a lot of small marks and noise that do not correspond to the arms. In 4.8b, we have the resulting image after filtering the small components from the binary image, obtaining the two anatomic parts that will go into separated files as new samples.

From the resulting binary image, we extract the connected components with information regarding each of them, such as their size or the centroids. We filter tiny components by size proportional to the image, removing them from the image, as they may correspond to some background noise or radiography annotations that do not correspond to the arm itself, such as the laterality. Applying the connected components extraction process again over the filtered image, we obtain a much better version of the candidate components. We show both steps of the extraction of the connected components, before and after applying the filtering, in Figure 4.8.

If there is more than one connected component, we keep the first one in the original file, and the others are copied each into new separated files with the same identifier plus a suffix. We also update the metadata files adding one row for each new file created, copying the fields from the original sample, and updating the identifier ones to point to the new files.

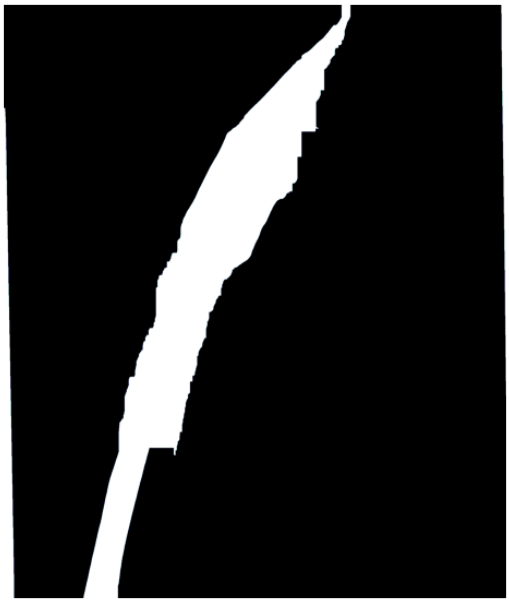
#### 4.2.2.4 Rotation Alignment

Finally, we want all the radiography images to follow the same orientation, positioned vertically with the fingers in the top and the forearm in the bottom. Considering that the arms have a lengthened shape, we perform a Principal Component Analysis [60] to compute the direction of the principal component, which corresponds to the line that goes from the fingers to the elbow (or alternatively the lowest part of the arm in the image).

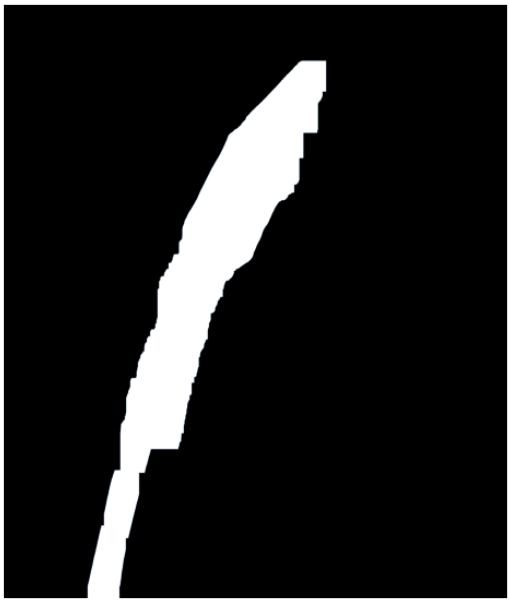
Before proceeding with the PCA first, we must clean any possible noisy pixels on the image that will act as outliers for the calculus, applying the morphological transformations open and close. To apply morphological transformations, we must work with binarized images. Therefore, we binarize the image applying the same process that we use for the anatomic part splitting step in Section 4.2.2.3. Once we have the binarized image, we can apply these morphological transformations. Open is the combination of an erosion followed by a dilation, which removes small pixels scattered around the image. On the other hand, a close operation is a reverse operation, a dilation followed by an erosion, which is useful in closing small holes inside the objects.

(a) Original image.

(b) Binarized image.



(c) Applying the Close operation.

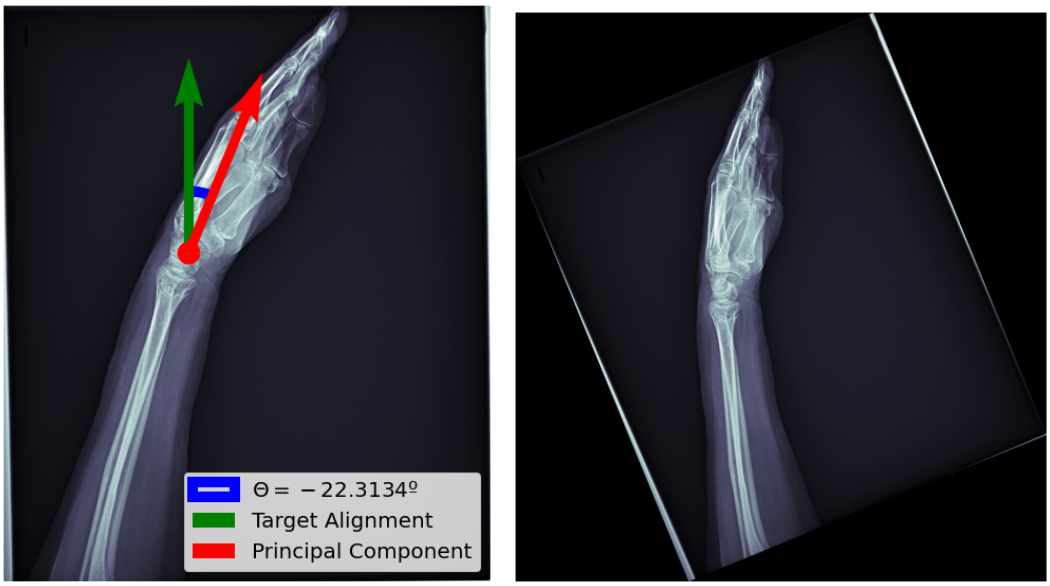


(d) Applying Close + Open.

Figure 4.9: Effect of the morphological operators in a radiography (4.9a) after binarizing it following Otsu's method (4.9b). Applying Close (4.9c), we remove "holes" in the background such as the arm's flesh, and Open (4.9d) removes noisy pixels like the white frame-like bars from the original image.

Once the image is clean, we calculate the PCA with all the pixels of the binarized image. After extracting the principal components, we can take the first one and compute the rotation angle degrees as in Equation 4.2, to finally apply the rotation transformation over the original image. We can see the calculated angles and the rotated result in Figure 4.10.

$$\theta = cos^{-1}(PC_1) \cdot 180/\pi \tag{4.2}$$




(a) Applying PCA and extracting the principal component and rotation angle.

(b) Original image transformed with the proper rotation alignment.

Figure 4.10: Considering the example image from Figure 4.9 we extract the principal components and compute the rotation angle of the first one (4.10a), applying this rotation over the original one to generate the output (4.10b).

Recalling to the aspect ratios plot in Figure 3.3 from Section 3.1 we can now observe the same type of scatter plot over the data set that we can generate after applying this last step. In Figure 4.11 we can observe the result corresponding to the pre-processed data set. Unlike in the raw data set, we see that the distribution of the image sizes follows a triangular upper-diagonal shape, which indicates that the last Rotation Alignment step works well for the vast majority of images, aligning the samples vertically and, therefore, resulting in greater height than width. We can also analyze the effect of other steps such as the Anatomic Part Splitter, as the regions with more density of

points correspond to sizes where the height is much greater than the width, as opposed to what we observed in Figure 3.3, which showed a higher density for square-like shaped images. The splitter produces this effect because single arms tend to be lengthened images, while multiple anatomic parts are more squared.

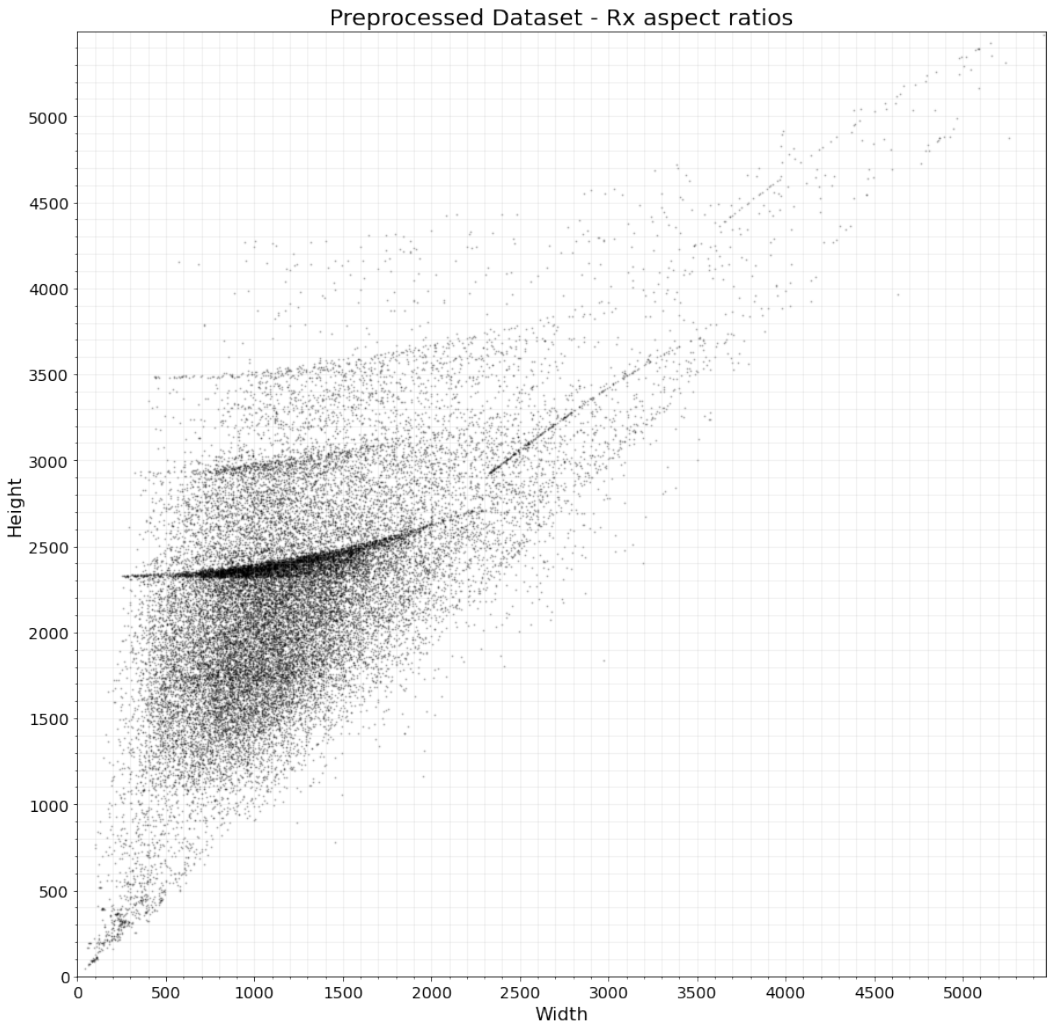



Figure 4.11: Aspect Ratios of the Pre-processed Data plotted in terms of width and height of every sample. Thicker dots represent a larger number of samples with that specific size.

## 4.3 Class Balancing and Batch Sampling

When we analyzed the data set statistics in Chapter 3, we realized that there is a notable data unbalancing problem.

There are multiple available techniques used to adjust the class distribution of a data set, such as weighting the errors to give more importance to the less represented class or oversampling the class by repeating some samples until both classes reach the same class proportion. We opt to use the opposite and roughly equivalent technique to oversampling, undersampling. The way that we implement the undersampling is strictly related to the batch sampling process.

The first thing that we do is to sample the batches randomly, so we shuffle the data set order in every epoch to get that batches between epochs do not look alike. Building and processing batches in the same order as the data set is created may lead to performance problems as if we have first all class A instances and after all the other from B, each category goes to a different batch, and in every epoch, the *ith* batch contains the same category. For performance reasons, we do not unnecessarily shuffle the list of samples after every epoch. Instead, we just assign a picking probability to each sample and build a random sampler that picks samples following a uniform probability distribution where $p = 1/n$, being $n$ the total number of samples. We also avoid repeating samples in the same epoch by leaving the already chosen elements out, setting them a pick probability of zero, and increasing the rest to $p = 1/(n - k)$, where $k$ is the number of instances that have been already sampled.

The method that we choose to perform the undersampling is to weight the sampling probability. Rather than sampling each instance following a uniform probability distribution, an instance of the less represented class has a higher probability of being chosen than one from the predominant class. By doing so, in the long term, it tends to sample an even number of samples from both classes.

Let $n_b$ and $n_{nb}$ be the number of broken and not broken samples, respectively. We assign to each sample the number of occurrence's of the opposite class, normalizing it such that the sum of probabilities of all samples add up to one, as we express in Equation 4.3, where $n_{opposite}(s)$ is the number of samples of the complementary class of the sample $s$. As for the previously explained shuffle selection, we also apply replacement to avoid choosing an instance twice in the same epoch.

$$P(s) = n_{opposite}(s)/(2 \cdot n_b \cdot n_{nb}) \tag{4.3}$$

With this process, we end up sampling $2 \cdot n$ instances in every epoch, where $n$ is the number of samples of the less represented class, having approximately $n$ broken and $n$ not broken instances. By making the selection randomly, all the samples will have a chance to end up appearing in a data batch.

## 4.4 Data Augmentation

One of the critical factors that influence a model's performance is the quality and the quantity of data. The variety of this is related to both terms, as the more varied data we can use, the better the model will be able to generalize. Data Augmentation is a common technique used to overcome the lack of large data sets or augment the data's diversity without really having new samples, only using slightly modified versions of the original samples that are generated automatically with some transformations. Some of the modifications can be the application of rotations, blurring, color changes, or projections, among many others.

There are many options on how to generate the augmented data set. Instead of the traditional method of generating new samples and appending those to the real data set, we opt for using on "on the fly" randomly selected transformations at execution time. Hence, the size of the data set at every epoch during the training process remains the same as the original data size, but the model may see altered versions of the same image at every iteration. In the long-term, this way of doing data augmentation potentially creates a much larger number of different samples, without occupying vast amounts of disk space, only at the cost of computing the transformations at execution time.

We use three data augmentation techniques for our training data set, random crops with resizing, horizontal flips, and color inversion, as we mentioned in 4.2.2.2. Some of the wrists may be at different scales, which could produce problems at evaluation time when the model can receive images at different scales than the ones it has been trained with. Randomly cropping the image between the 90% and 100% of the actual image size results in a great variety of different scales to the arms. We consider 90% to be a great value as it gives the possibility to slightly zoom the image without losing essential parts, while lower values may crop undesired parts of the image, leaving the regions where the fracture is located outside of the resulting image, which is not of our interest. After that, the images are resized to the same size to avoid inconsistencies between images cropped with different ratios. We experiment with the performance of different scaling and padding sizes in Section 6.2.1.

With the application of the rotation alignment pre-processing step from Section 4.2.2.4 we were able to unify the orientation of the arms with excellent results. However, we are not able to identify the laterality of the wrist easily, so we are not able to face all the images to the same side, e.g., with the thumbs to the left like all the samples are right hands. To avoid problems with different distributions of right-left arms and possible problems with the model validation because of a less represented laterality, we randomly apply a horizontal flip of the images with a probability of 0.5 when passing it through the model so, in practice, there will be roughly the same number of left and right wrists.

Something fundamental to be noticed after reading the pre-processing section and while reading this one is that we avoid dealing with very complex and hand-crafted pre-processing steps that tend to have difficulties in generalizing and working right with unexpected data samples, even more, notorious in our case, with bad quality of the data. Instead of spending a lot of time trying to overcome this problem, we assume that our pre-processing steps may overlook a notable amount of noisy samples, being unable to clean them, and we take advantage of that introducing a controlled noise, which mitigates the possible errors at the same time that makes the model more robust to variances in the data. As an example, let us consider the color inversion step of our pre-processing pipeline (Section 4.2.2.2). We have already commented that there are some cases where the color inversion does not work correctly due to heavy noise in the images. We can not remove the step from the pipeline as we need as many good color schemes as possible for the following steps, the anatomic part splitter, and the rotation alignment because it is necessary for the connected components and principal components extraction. However, we can use the color inversion approach to invert the color scheme with a probability of 0.5 randomly and, by doing that, we will have a comparable number of samples for both color schemes, making the model more robust to validation inputs.

Finally, we can compute the data set statistics of the pre-processed data set previously in an online process, which give us a mean of 0.34587 and a standard deviation of 0.22898 in our data (which is in range 0..1). It is a good practice to normalize the input data used to train a Neural Network model, making the training process more stable, faster and reducing the chances of getting stuck in local optima. Therefore, our last step is to normalize the images with the extracted statistics and convert them into a suitable format for the model, which are the tensors.

In Figure 4.12 we show the whole data augmnetation pipeline.

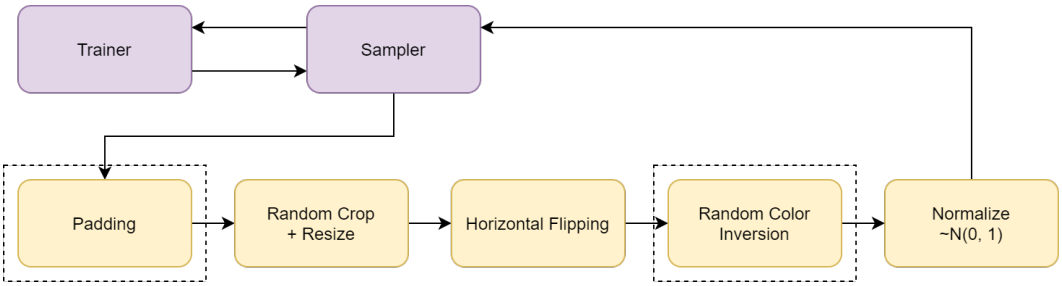



Figure 4.12: Data augmentation pipeline. Steps surrounded by dotted rectangles indicate that their usage is optional. We experiment with them in the experiments section (Chapter 6).

# Chapter 5

# Models

In this section, we present the four models that we implement and experiment with. The first three models have the same input-output values, receiving a radiography image and outputting the binary fracture prediction for that image. The last one, the Aggregator model (Section 5.3), generates a prediction for the whole study, so the input is a set of radiography that conform the same study.

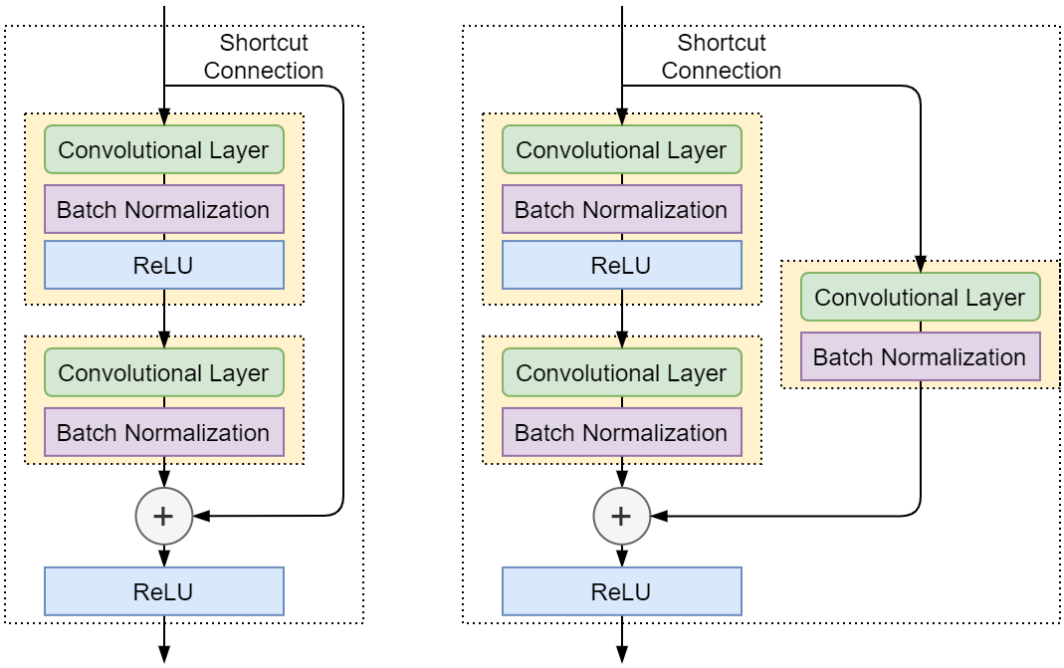
## 5.1 Residual Network

One of the first steps in the model building process is defining a baseline model that allows us to compare it with other candidate models to evaluate the performance. We decide to take a well-known model, the ResNet18 [28], which is a Residual Network with 18 layers, one of the best recent models with a good trade-off between model size, training time, and performance in standard image classification problems.

In general terms, the deeper the model, the more capable it is to achieve higher performance, as it is more flexible to adapt to any space because it has a larger parameter space to explore. However, with the deepness also emerges the vanishing gradient problem, which occurs when the loss function gradients are very close to zero, making the network hard to train. If we have many layers, this happens because these derivatives are multiplied together when using the chain rule, making the gradient decrease exponentially as we back-propagate it. Residual Network models solve this problem as they add skip connections, which allow the gradients to flow directly backward from later layers to the first ones.

ResNet18 is a network that stacks several residual blocks 5.1 we show the

architecture of these residual blocks with 18 layers of deepness. We can check the architecture of a residual block in Figure 5.1.




(a) Same dimension. (b) Different dimension.

Figure 5.1: Residual block architecture. If the dimension of the data does not change between block it applies the first type from 5.1a. Otherwise it needs to apply some extra operations to change the dimension of the data for the next layer (5.1b).

## 5.2 Rx Model

After defining a standard baseline, we create a custom model to improve the Residual Network's performance. This model is also based on Convolutional Neural Networks, but removing the main improvements that the Residual Networks adds, residual connections between layers. Nevertheless, why do we use some custom and probably less powerful models when we have well-known models that have the top performance in most classification task domains? As a reminder, the goal of this project is not just to build a classifier to detect if there is a fracture or not in an image; we aim to build a Clinical Decision

Support System that helps the doctor to find some extra information from radiography images. It must predict the task that it has to fulfill with the highest possible accuracy, but we also have to offer some explainability for the model predictions. Even that this thesis's scope is to obtain predictions about the presence of a fracture, we must not look away at the project's objectives in the long term.

When we add residual layers to a Convolutional Neural Network, we make it much more challenging to extract information that serves us to generate some explanations about the predictions. Some methods, such as Layer-wise Relevance Propagation (LRP) [61, 62] that allows extracting which pixels contribute to the final result within a scale. With that, we can not obtain a textual explanation about the result, but some are pinpointing of the region or regions that helped the most to produce that prediction, probably highlighting where the fracture is located, if the prediction and the model are correct. While it is possible to apply these explainability methods to residual networks, it becomes much more complicated.

We have done some proof of concepts to visualize what LRP is able to show us. In Figure 5.2 we can see a radiography image with a colored overlay. Red regions indicate evidence in favor of the presence of a fracture, while blue regions indicate evidence against it. We address this explainability task in more detail in Chapter 8, with the future work.

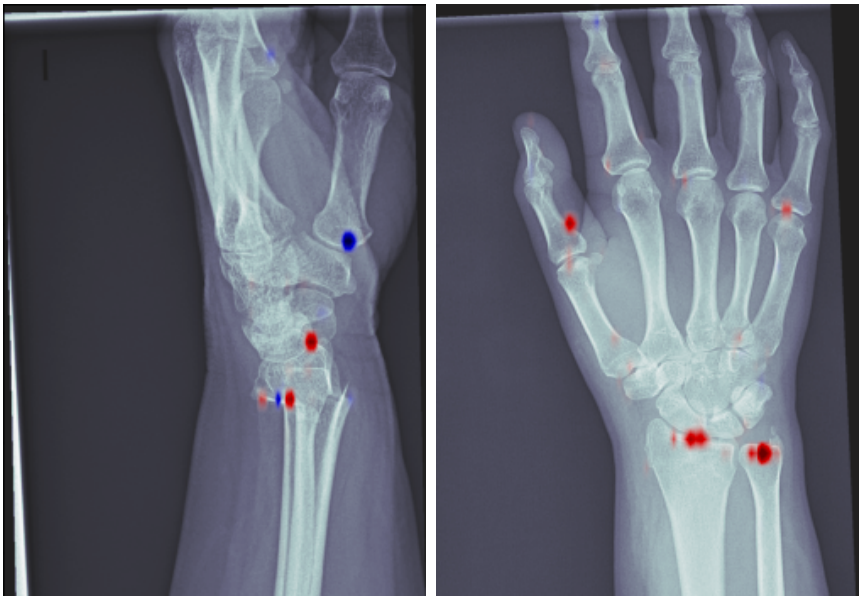
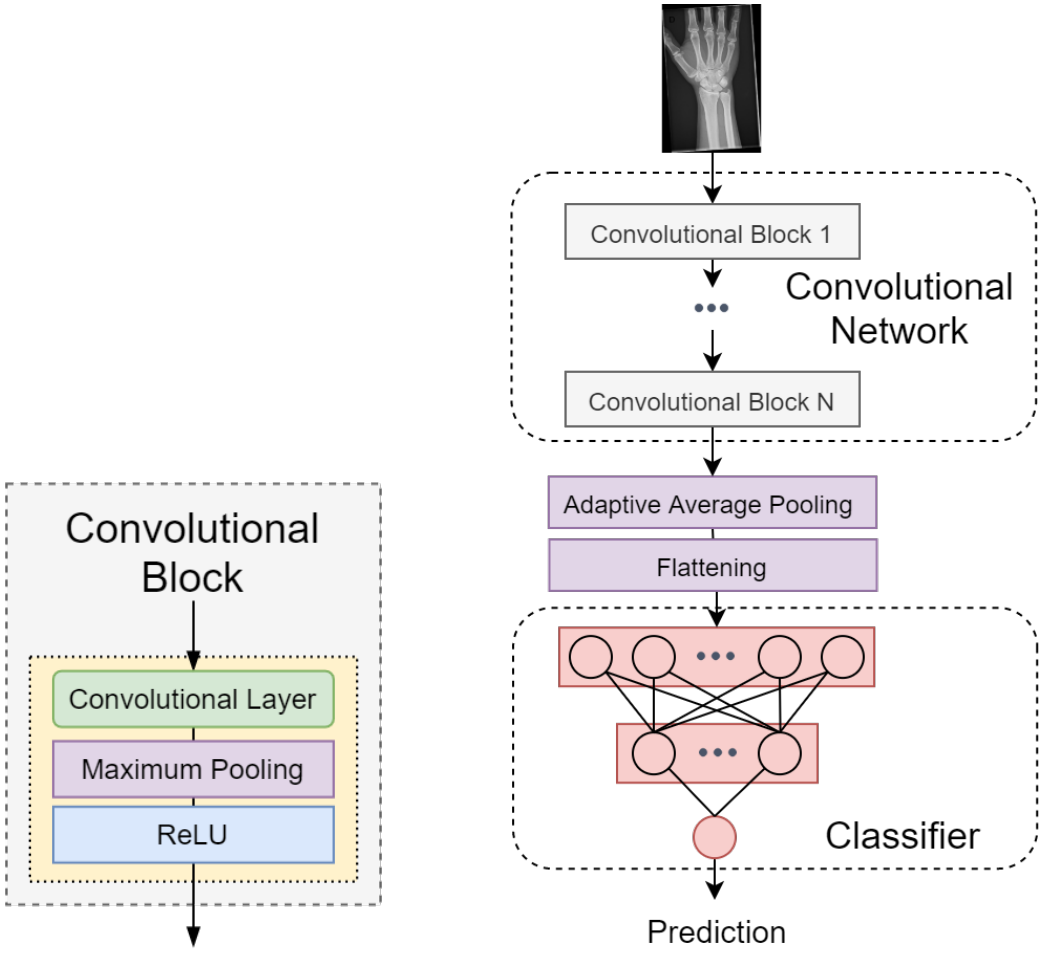


Figure 5.2: Explainability images generated with LRP.

We define a model that is built using convolutional blocks (Figure 5.3a) that consist of a convolutional layer with 3x3 kernels without padding and unitary stride. To the result, it is applied a Maximum Pooling operation of 2x2 with a stride of 2, which helps to reduce the over-fitting, providing an abstracted representation, provide transition invariance to the internal representation, and reduce the computational cost due to a smaller number of trainable parameters. Finally, we apply a Rectified Linear Unit (ReLU) function to the pooled output. The whole model comprises several convolutional blocks that are stacked sequentially. After the convolutional part, we apply an adaptive average pooling to fit a particular shape, and we flatten it into a 1-dimensional vector, which we pass to the classifier, a set of three fully-connected layers with decreasing number of neurons that end with a single neuron that is used to extract the binary prediction. We can observe the overall architecture in Figure 5.3.




(a) Convolutional block.

(b) Architecture of Rx models.

Figure 5.3: Architecture of a Model Rx (5.3b), using $N$ blocks (5.3a).

## 5.3 Aggregator Model

Finally, we consider a model that works at study level. The reason behind the creation of this model is that we consider the hypothesis, which holds that, instead of doing the aggregation of instance predictions manually, a model could be able to learn these relations and offer a higher study-level aggregated performance.

The main drawback of many multi-input model architectures is that they rely on a fixed number of input images for each object, where each image is always of the same kind of view for all the objects. For example, having three images to the multi-view projections, the first one is always for the plan view, the second for the elevation, and the third for the section. This requirement presents a problem for our data, as we can not ensure that we always have the two main views of a wrist (lateral and frontal), as we could observe in Figure 3.2.

We base our architecture on the successful work in other 3D image classification tasks with approaches such as the Multi-View CNN (MVCNN) [63]. For this model, we reuse part of the architecture used in Model Rx (Section 5.2). Instead of using the binary predictions of each image, we extract the intermediate feature maps obtained after the convolutional part, stripping the classifier away. These features are put together with a view-wise pooling operation, the result of which is passed through a new classifier that also performs a binary prediction. The pooling step allows us to work with an arbitrary number of views.

The model, represented in Figure 5.4, takes as the base the Model Rx architecture from Figure 5.3b, but stripping the classifier part. We refer to the remaining convolutional network part as the *view model*. Considering a study with M views, each view is passed through the view model without doing the back-propagation step, as if we had M copies of the network with tied or shared weights. We apply a view pooling operation at channel level, which is nothing more than a maximum pooling operation along the view dimension, resulting in a tensor of equal shape independently of the number of views. Similar to what we do in Model Rx, the result is passed through a classifier that shares the same structure of a network of fully connected layers with a single neuron in the last layer, producing a single prediction for all the images that correspond to different views of the same study.

The fact that we are taking parts of the previous models also allows us to apply Transfer Learning [64, 65], the process of reusing a model trained on one task to train it for another task. By doing so, we can take advantage of the already learned low-level features that the Model Rx has learned in previous

and independent training processes. To do so, we extract the weights from a pre-trained checkpoint of the Model Rx, taking only the weights associated with the convolutional network. In the experiments associated with this model, we analyze the difference in terms of performance of applying or not Transfer learning and the different effects when freezing the layers to train only the classifier part.

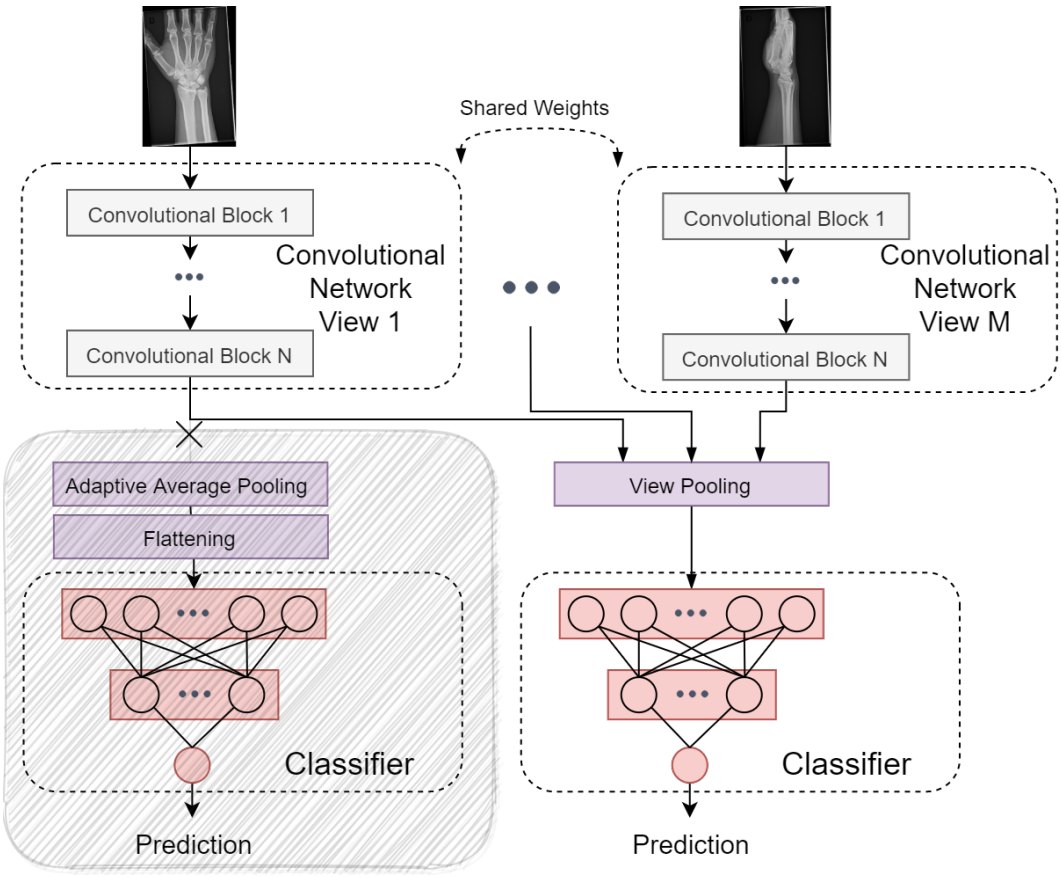



Figure 5.4: Aggregator Model Architecture.

# Chapter 6

# Experiments and Results

In this chapter, we expose and analyze the results obtained with all the model and data experiments. We start by defining the reproducible methodology that we follow, as well as the system specifications that can serve as a reference of performance in terms of time and hardware resources. Then, we examine the results of the multiple models defined in Chapter 5.

## 6.1 Methodology

One of the key assumptions of scientific research is the reproducibility of experiments, allowing any other scientist to replicate the results following the same methodology. To ensure this principle, we have designed a reproducible and auditable workflow to demonstrate the legitimacy of the results and avoid common problems such as not being able to get which code base was used to generate some specific results.

The experimentation workflow is linked with the Git version control system, more specifically using GitLab, a Git-repository manager. We submit all the jobs through Git commits, using the GitLab Continuous Integration tool, GitLab CI/CD. This allows us to, at any moment, use the commit tag to roll back the code to the one used to execute an experiment. The experiment is also linked to the used execution parameters, including references to previous intermediate experiments, such as training continuations.

We wrap the jobs executed in MareNostrum into a Singularity container, which defines the execution environment. However, for the training jobs in CTE-Power9, due to some technical limitations, we are not able to use this technology. For this case, at least, we can ensure the exact versions of the external packages used.

All the model training jobs and experiments are run in MareNostrum's CTE-Power9 cluster using a single execution node. To have analyzable and comparable metrics about the training time and performance with possible posterior replications in other hardware systems, we specify the system specifications in Table 6.1.

| | |
|---|---|
| CPU | 2 x IBM Power9 8335-GTH 3.0GHz 20 cores |
| GPU | 4 x GPU NVIDIA V100 volta |
| Cores | 40 (2 x 20) |
| GFLOPs | 16 GFLOPs/core + 7.8 TFLOPs/GPU (31.8 TFLOPs) |
| OS | Red Hat Enterprise Linux Server 7.4 |

Table 6.1: MareNostrum's CTE-Power9 specifications for a single node.

## 6.2 Model Results

Hereunder, we present the results obtained from several experiments with the models presented previously. As we have already commented, we consider the ResNet model as our baseline, so we use it to experiment with some non-trivial architecture or training decisions that we have raised during this document. We present the answers to these questions in the subsections of Section 6.2.1. We decide to do these experiments only with the ResNet to avoid unnecessary computational waste, as these are not toy experiments and need several hours to be completed. A grid search of all the model and decision combinations would take a massive amount of time to be computed and will probably result in some conclusions that do not make the difference. The reason behind choosing the ResNet as the model to experiment with different decisions is the same reason behind considering it as the baseline model: It is a well-known model that performs competitively in generic domains.

### 6.2.1 ResNet

With the baseline model, we can evaluate the performance of different decisions that involve the use of different data pre-processing steps or configurations. In the course of the document, we presented some questions about architecture and training decisions that require further experimentation to get response. Among other experiments that we want to try, these are the effect of padding or resize when feeding the images to the network, the trade-off between quality, image information and training speed when resizing the

images to smaller shapes, and the effect of applying the color inversion preprocessing step as a data augmentation step, to make the model more robust to some errors produced by the step itself.

#### 6.2.1.1 Scale Resizes

One of the aspects that we must take into account with Deep Learning, specially for image processing is the size of the data. Theoretically there is no limit on the size of images that we can pass through a Convolutional Neural Network. However, it increases memory requirements and computing time, especially with large batches, as more layers are required to sample down the image.

As we can verify with our data, medical images tend to have a large size with a great resolution, because those are used in tasks that require the detection or classification of small regions of interests over the image. This opposes most typical computer vision tasks such as the classification of multiple items. For these tasks, we can use very low resolution images to train the models, as it is enough detecting the overall shape or colors. However, in our case, even considering a high resolution image of 3000x3000 pixels, the fracture to localize in the image may be as thin as a hair, being represented with a few pixels. Therefore, it is unsuitable to consider a heavy resize to match other common data sets such as ImageNet (256x256) [20, 66], as we would lose a large part of the information which would probably include the fracture features, due to its reduced size.

However, we still consider multiple sizes for the images, to experiment with the trade-off between information loss and training speed, to find out the best resize shape that boosts the computational speed without loosing too much information about the fractures. We experiment with three different shapes. As the images in the data set have a lengthened after being aligned vertically, we constraint the shapes for the three experiments to have an aspect ratio of 1.5, so we at least keep as similar as possible the original aspect ratio of the images, that can vary a lot. After discussing with some expert radiologists we could not agree in the best size where there is no information loss that obfuscates the fracture information. However, they recommended us to keep at least around 500 pixels of information in one of the dimensions, as some fractures can be so thin that with smaller sizes could even stop being visible due to an aliasing effect. Therefore, we try with shapes of 341x512, 512x768 and 768x1152.

In figure 6.1 we can see the training metrics of different resizes, while Table 6.2 presents the accuracy and loss metrics for each of them. We can downscale

the images to 341x512, which increases the accuracy substantially, at the same time that it reduces the required training time.

| Size | Accuracy (%) | | Training Time |
|---|---|---|---|
| | Train | Validation | |
| 341x512 | **75.80** | **69.83** | **08:40:46** |
| 512x768 | 71.05 | 63.41 | 09:00:42 |
| 768x1152 | 68.68 | 62.15 | 10:10:43 |

Table 6.2: Accuracy of the model after 10 epochs which each of the models with different resize shapes.

### 6.2.2 Study Level Aggregation

As we highlighted while presenting the problem, the goal is to achieve study-level predictions, while currently we are stating the instance level results. To do so, we start by analyzing statistics of the predictions of the data set based on the percent of instances with correct predictions for each study. We do it with the results of the model with the chosen size in the previous experiment. We can observe the plot in Figure 6.2.

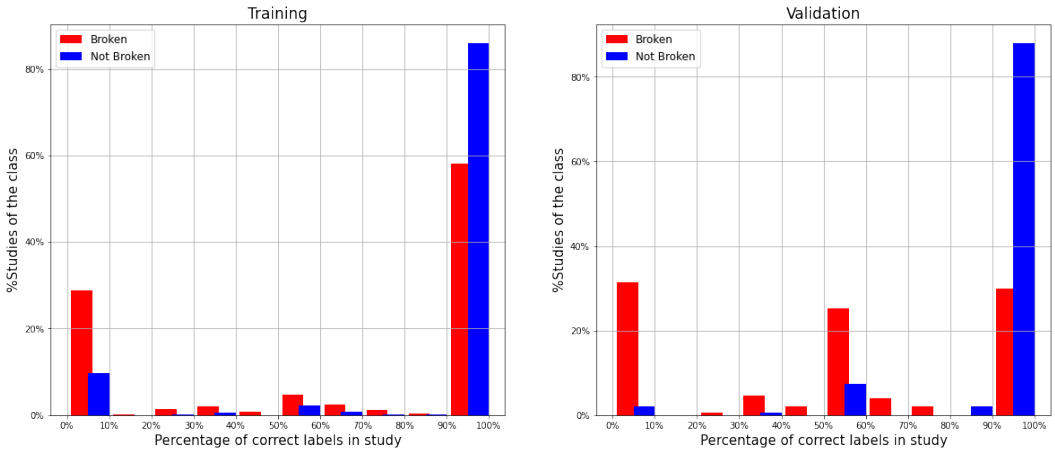
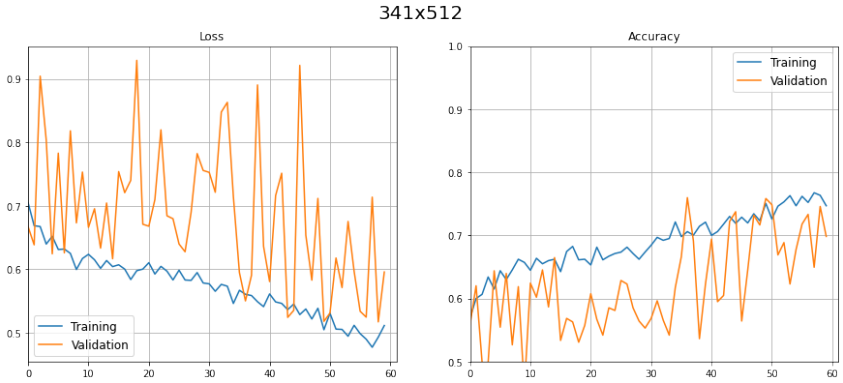
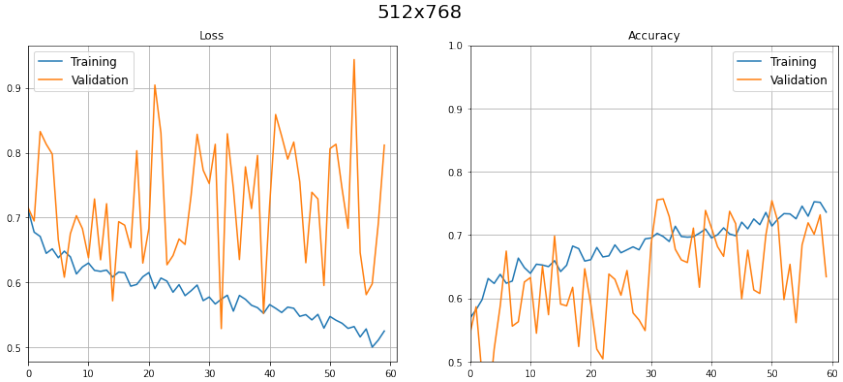



Figure 6.2: Distribution of the percentage of correct instance labels in each study, separating Broken and Not Broken classes. The bars on the left side (0%) mean that none of the instances were classified as the real label of the study, while the rightmost ones (100%) represent the opposite.

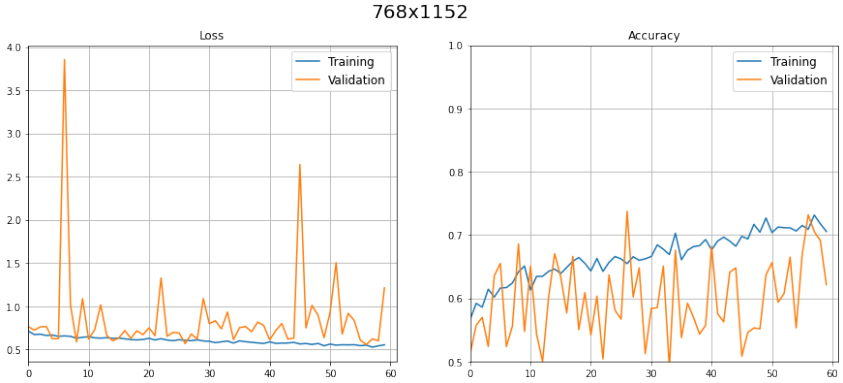
By taking a look to the data, we found two possible ways of aggregating the results of the independently generated predictions for each instance of the




(a)




(b)




(c)

Figure 6.1: Training metrics for ResNet18 models for 10 epochs with different resizing shapes: 341x512 (6.1a), 512x769 (6.1b), 768x1152 (6.1c).

same study. The first one consider that a study as broken if at least one instance of it is predicted as broken. We refer to this method as ALO (At Least One). The other is to perform a majority voting, where the study is classified as the majority class predictions of the instances.

Already from the data we can observe than the ALO aggregation will perform better. We can say that by observing the distribution shape of the central columns (excluding 0% and 100%) we can already say which method performs better. ALO sends all the central broken columns to the right (as there is at least one instance predicted as True) and vice-versa for the central not broken ones. On the other hand, majority votting will split the bars at 50% and send them to their corresponding sides (left for a percentage that is less than 50% and right otherwise). The plots show us that ALO is the best strategy as the vast majority of cases that fail to classify some instances correspond to the Broken class as a fracture is not always visible in all the instances.

In Table 6.3 we update the previous table to compare the results with the previous instance-level metric and the two methods that we want to test here. We can observe that both methods outperform the accuracy obtained at instance level, and the ALO method gives slightly better results than the majority voting for all the cases. Hence, from now on, in the tables of experiments results we are going to provide the instance accuracy (the one that we show in the training metrics line charts) along with the aggregation accuracy with ALO.

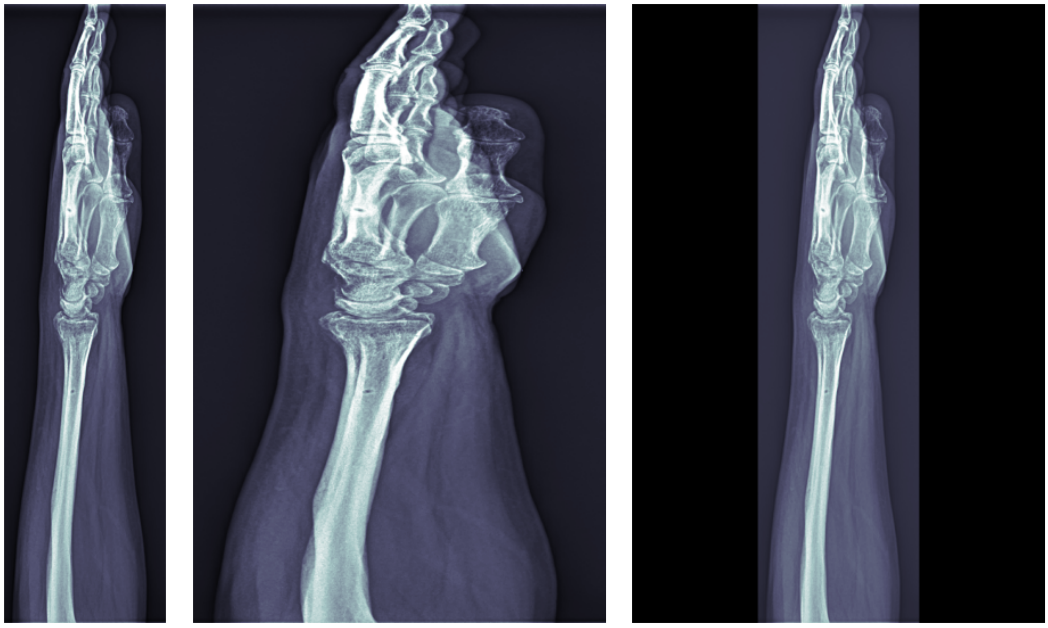
| Size | Accuracy (%) | | | | | |
|---|---|---|---|---|---|---|
| | Train | | | Val | | |
| | Instance | Majority | ALO | Instance | Majority | ALO |
| 341x512 | 75.80 | 79.16 | **80.17** | 69.83 | 76.00 | **77.67** |
| 512x768 | 71.05 | 77.65 | **78.78** | 63.41 | 66.33 | **70.67** |
| 768x1152 | 68.68 | 74.06 | **75.42** | 62.15 | 67.67 | **71.00** |

Table 6.3: Study level accuracy comparative between Majority and ALO.

#### 6.2.2.1 Padding vs. Resizing

We are concerned about the possible effects that resizing the image may produce. Recalling the diversity of aspect ratios from Section 3.1 (see Figure 3.3), we suspect that resizing images without further considerations may have a negative impact as those lose the original impact and therefore appear deformed and distorted. Our hypothesis is that we can combine a padding strategy with the resizing of the images to coincide with the size of the largest

image of the data set, so the final size will remain the same, but we can keep the aspect ratio by padding the image with a black background. In Figure 6.3 we exemplify the different resulting images obtained with both approaches.



(a) Original. (b) Scaled image. (c) Padded image.

Figure 6.3: Comparative of resize and padding effects.

We train our ResNet model for 10 epochs with the same architecture and data set, using the same seed to ensure that the weight initialization of the network and the sampling process is the same. In Figure 6.4 we compare the metrics that we can extract from both training processes.

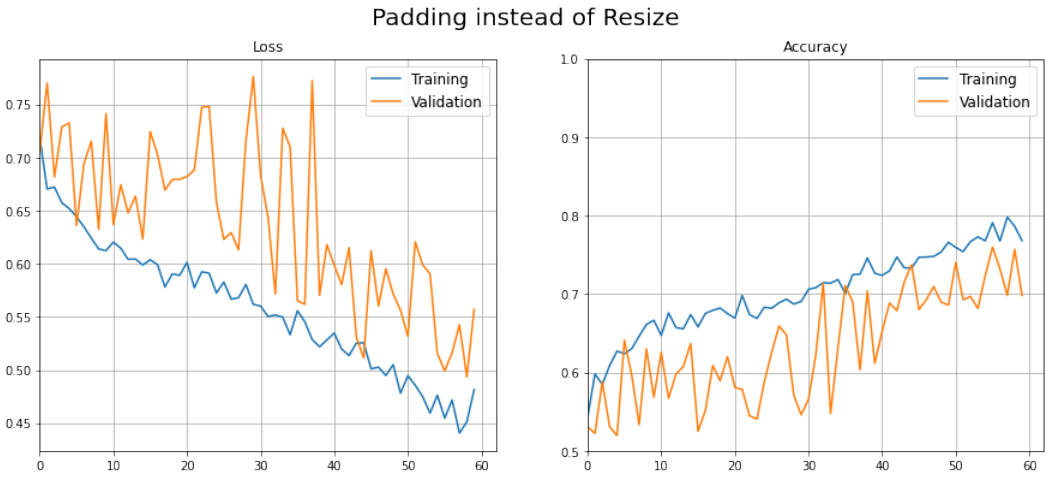



Figure 6.4: Training metrics of ResNet with 10 epochs and padded images.

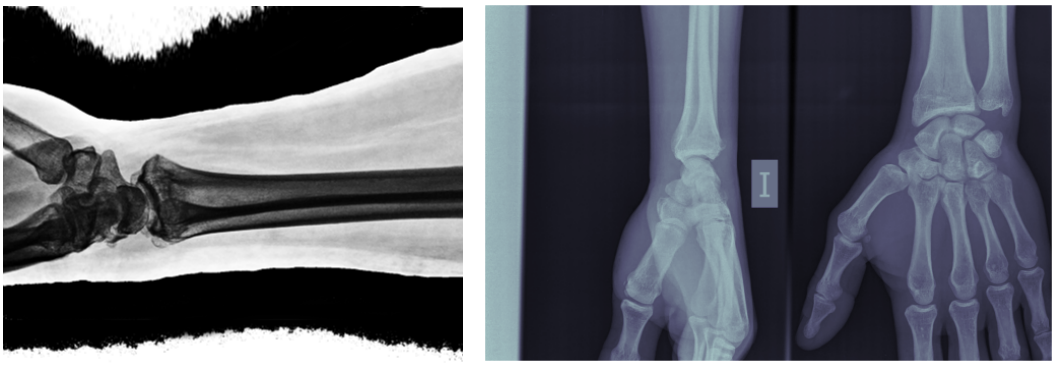
We can observe that the results obtained with the application of padding improve slightly the results than the version with only scale offers. Therefore, we add this pre-processing step to the on-the-fly transformations that are applied in the sampling process with the Data Augmentation pipeline before feeding the network. Table 6.4 shows the comparative between the usage of padding and resizing.

| Model | Accuracy (%) | | | |
|---|---|---|---|---|
| | Train | | Val | |
| | Instance | Study | Instance | Study |
| Resize | 75.80 | 80.17 | 69.83 | 77.66 |
| Padding | 77.80 | **83.33** | 69.83 | **78.33** |

Table 6.4: Accuracy of ResNet18 using Resize or Padding.

#### 6.2.2.2 Color Inversion as Data Augmentation

As we have already commented in Section 4.2.2.2, the simplicity of the color inversion pre-processing step lead to errors in the detection of the background color with some noisy images that have components that we would not expect to have. In Figure 6.5 we can observe a pair of examples where the pre-processing step fails.



(a) Image with a noisy background. (b) Framed image.

Figure 6.5: Images that will fail with the color inversion transformation from the pre-processing step. In 6.5a we detect that there is a a very strange and unusual background that will not change the image color despite of the bone being in black tones. in 6.5b the opposite case, where the image appears framed and it will be inverted because three out of four corners are whitish (two from the frame and one from the right-most finger), when it should keep the original color scheme.

Instead of investing time and effort to build a more robust supervised classifier which would require of some manual labeling process, or experiment with more sophisticated unsupervised methods, we explore the possibility of creating a new data augmentation step that inverts the colors of an image with a random probability, which will disguise the errors and could work to make the model robust enough to handle both possible color schemes, and a wider range of contrasts.

We experiment the difference between the presence or not of this pre-processing step. Furthermore, we modify the normalization source data distribution to fit the new data distribution. The standard deviation will remain the same but, as we are going to invert the colors randomly with a probability of 0.5, the new mean of the data will be 0.5 instead of 0.34587.

In Figure 6.6 we can observe the metrics extracted from the training process when applying a random color inversion with a probability of 0.5 to the data augmentation pipeline. We compare it with the results obtained from the previous experiment which metrics were shown in Figure 6.1a.

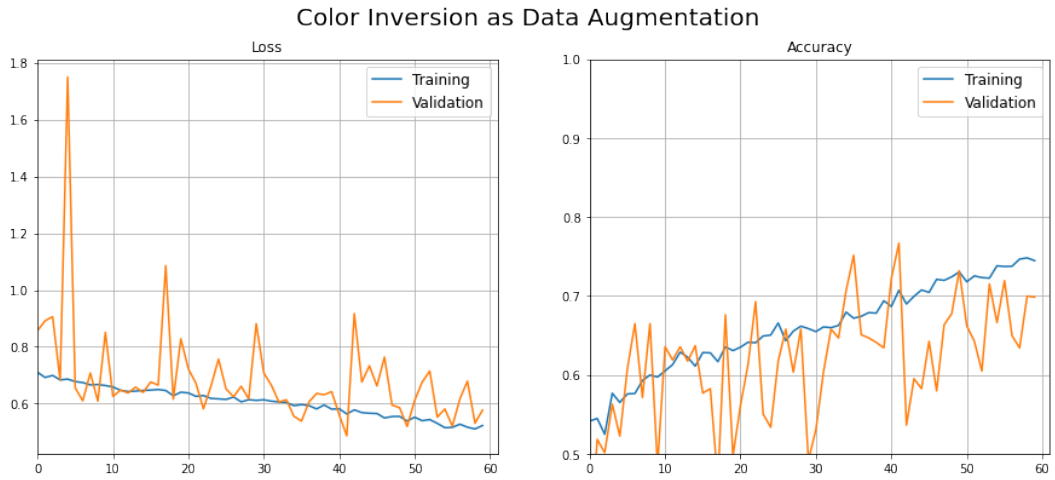



Figure 6.6: Training metrics when using Color Inversion pre-processing step as a Data Augmentation technique.

It seems that the application of the color inversion step as a Data Augmentation technique does not help to improve the model. Even worse, the loss during the training seems to be more unstable. For the lack of evidence of an improvement and the fact that it produces less stable curves, we decide to discard the hypothesis of the advantages of this method. We are obtaining these results probably because we add more considerable amounts of noise than the ones that we can find in the pre-processed data set, even if we have some incorrectly inverted images. We compare the results of this experiment with in Table 6.5.

| Model | Accuracy (%) | | | |
|---|---|---|---|---|
| | Train | | Val | |
| | Instance | Study | Instance | Study |
| Do use Color Inversion as DA | 77.79 | 83.33 | 61.73 | 69.67 |
| Do not use Color Inversion as DA | 77.80 | **83.33** | 69.83 | **78.33** |

Table 6.5: Accuracy of ResNet18 using color inversion as data augmentation.

#### 6.2.2.3 Comparative of multiple Batch Sizes and Optimizers

In this experiment we want to analyze the effect of batch size on training performance, as well as the effects of different optimizers. A problem in the machine learning community is the difficulty to make general statements about the effects of most hyper-parameters as there is no general behaviour independent of the data, task or model architecture. Several works study the effect of

small against large batch size [67, 68], that agree with the fact that small values tend to produce noisy gradients, while large ones suffer from performance degradation. Hence, there is no more choice than having to experiment with some values for this parameter.

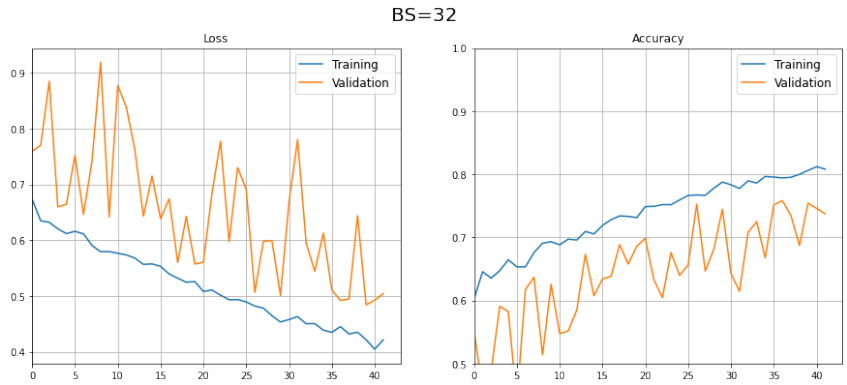
In the first experiment, where we checked the effect of different resize shapes for the data, we set the batch size at 16 so to allow us to be able to load in GPU the image batches of the largest shapes. The rest of the experiments until now continued with 16 to offer fair and comparable metrics. Now, we want to see how different batch sizes affect the performance of the model. We choose to test three more different sizes, following the typical power of two choices because of the alignment of the virtual processors onto the physical ones from the GPU, so we go with 32, 64 and 128.

In Figure 6.7 we can observe the training metrics of the same model with diverse batch sizes, which we can compare with the metrics of the previous experiments, that used 16 as the batch size. The greater the batch size, the lower the horizontal axis in the plots, because we are extracting the metrics periodically after a given number of batch updates. However, all the models in the figure are trained for the same number of epochs, so we can obtain fair results to compare.

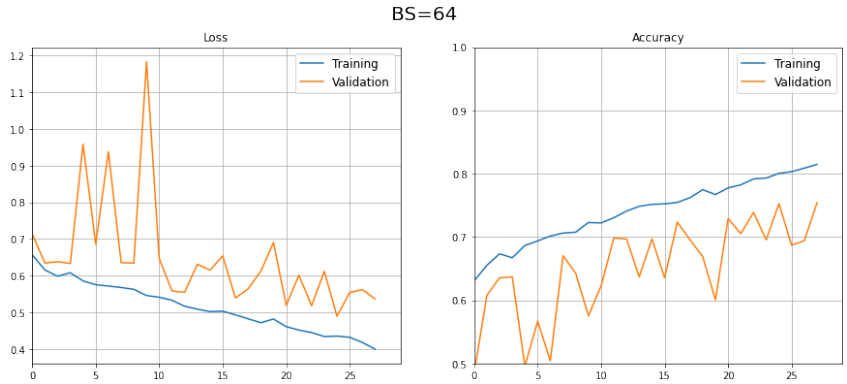
After analyzing the results we can not determine that a given batch size is significantly better than the others. Therefore, as we know that a too large batch size leads to poor generalization, we opt for choosing 32 as the batch size, as it seems that it offers a more stable performance growth curve.

As for the optimizers, we experiment with three of them: Stochastic Gradient Descent with Momentum (SGD) [69], Adaptive Moment Estimation [70] (ADAM) and a variant of this last one using an AMSGrad variation of it [71]. ADAM is one of the most commonly used optimizers, as it is a method that computes adaptive learning rates for each parameter which frees us from the search of hyperparameters to tune other heavily parametric methods such as SGD. However, ADAM may fail to converge to an optimal solution, and a SGD with momentum properly set up can outperform it. The AMSGrad modification tries to overcome this problem for ADAM, but there is no consensus about if it is able to consistently outperform Adam in practice.

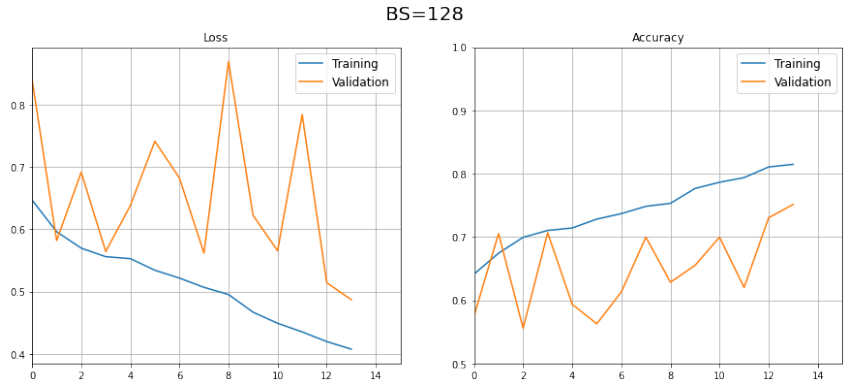
After some experiments we were unable to set-up a SGD hyper-parameter configuration that is good enough to hold up with the performance achieved with ADAM. Unfortunately, AMSGrad resulted in a slightly worse performance in practice, so we see no point on using it for the model, as it does not help us to improve the ADAM performance. Therefore, we choose ADAM as the optimizer to train the model.




(a)




(b)




(c)

Figure 6.7: Training metrics for ResNet18 models after 13 epochs with different batch sizes: 32 (6.7a), 64 (6.7b) and 128 (6.7c).

#### 6.2.2.4 Baseline Model

After exploring the multiple possibilities on the different parameters for the data augmentation transformations and multiple resizes of the data, we summarize and define the ResNet model with the best configuration found, to consider this trained model as the baseline.

From the previous sections we analyzed and extracted the improvements that we can apply to the training process to define our baseline:

- ResNet18 architecture.
- Training process with ADAM optimizer and 32 of batch size.
- All the pre-processing and data augmentation steps from Chapter 4.
- Padding for images before random crop resizing.
- Resize of the images to 341x512 pixels.

We continue the previous training from the experiments with 10 epochs to observe how far we can get with this model. In Figure 6.8 we can check the training metrics of the resulting process. We can easily observe the overfitting problem as the training loss keeps decreasing while the validation loss no longer does and instead it becomes much more irregular and starts increasing.

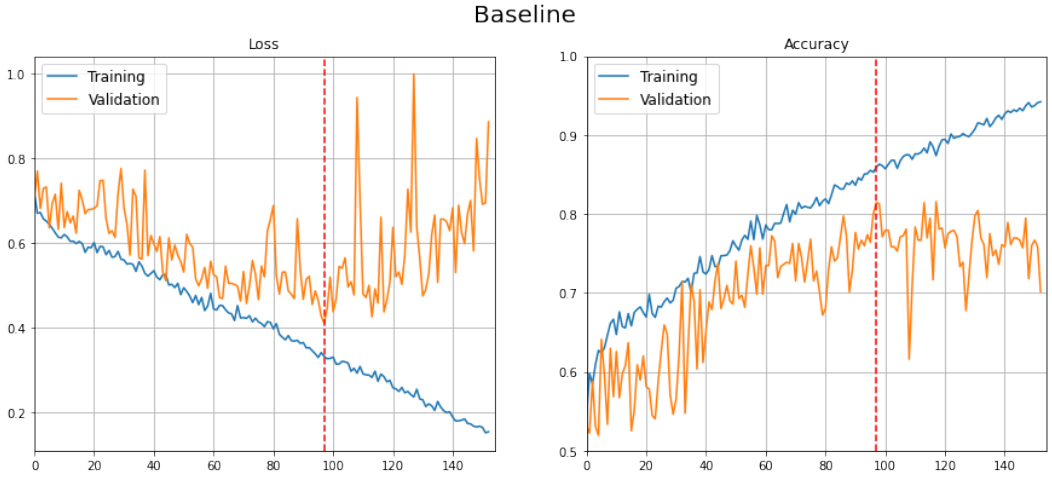



Figure 6.8: Training metrics for the baseline model.

We obtain the best checkpoint after 24 epochs, where we have a study level accuracy of **85.99%** (84.44% at instance level) with the training data,

and **85%** for the validation data set (78.35% at instance level). In Table 6.6 we show the percentage confusion matrix of the results.

| | | Predictions | |
|---|---|---|---|
| | | Not Broken | Broken |
| Labels | Not Broken | 52.15 | 9.11 |
| | Broken | 4.9 | 33.84 |

(a) Confusion Matrix over the training data set.

| | | Predictions | |
|---|---|---|---|
| | | Not Broken | Broken |
| Labels | Not Broken | 42.0 | 8.0 |
| | Broken | 7.0 | 43.0 |

(b) Confusion Matrix over the validation data set.

Table 6.6: Confusion Matrices (in percentages) for the training data (6.6a) and validation data (6.6b) with the best checkpoint of the baseline model.

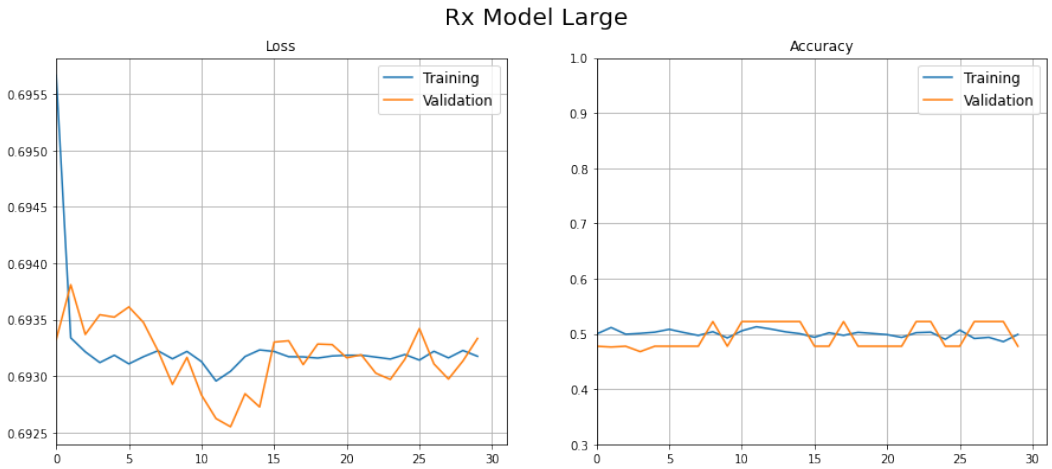
### 6.2.3 Model Rx

We define and build two models of this kind, changing the number of convolutional blocks and, therefore, the number of trainable parameters, which affects our model's flexibility to fit the given data. With the second model, the Rx lite version, we want to analyze possible undesired effects in the full Rx model, such as presenting over-fitting with large models or gradient problems derived from the network's deepness to a higher number of layers. We will refer to the first model as `Large`, with eight blocks, and to the second one as `Lite`, with half of the blocks, four.

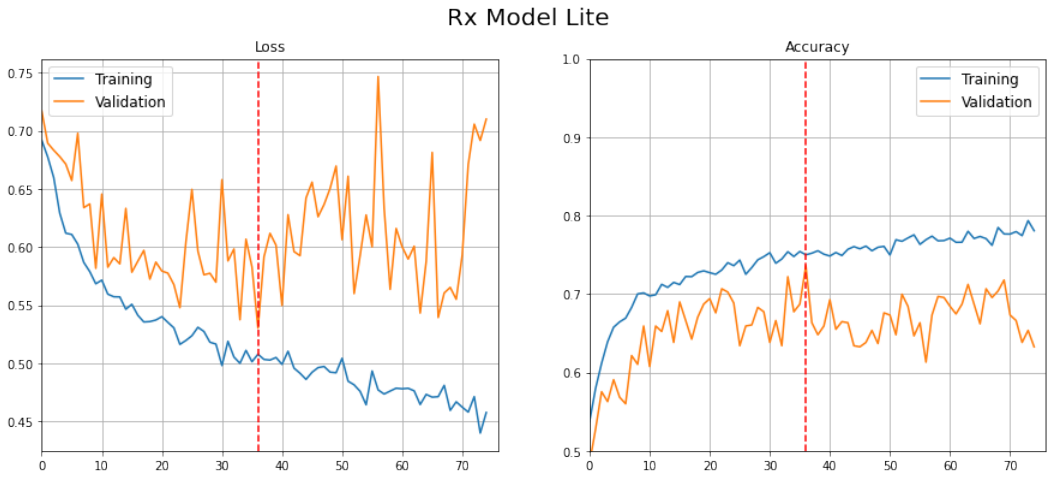
The configurations for the Data Augmentation steps, batch size, sizes, and other training properties are the same that we use to create the baseline model from Section 6.2.2.4. In Figure 6.9 we show the training curves of both models.

We can clearly observe that there is a problem with the training of the Model Rx Large in Figure 6.9a. The model seems to be unable to learn anything. We can see that the loss and the accuracy metrics are stuck since the beginning. We found out that the lack of improvement seems to be caused by the vanishing gradient problem due to the network's deepness.

While the large version is unable to learn anything, using half of the blocks in the lite version produces a result that is comparable with the performance achieved by the baseline, achieving the best checkpoint after 12 epochs (red




(a) Training metrics for Model Rx Large.




(b) Training metrics for Model Rx Lite.

Figure 6.9: Training metrics for Model Rx Large and Lite. In 6.9a we show the results for Model Rx Large (8 convolutional blocks) and in 6.9b for Model Rx Lite (4 convolutional blocks).

dotted line) in Figure 6.9b.

Finally, we discard the Large model, keeping the Model Rx Lite. As we did with the baseline, we evaluate the model with the training and validation data sets, reaching the best performance after 12 epochs, with a study level accuracy of **77.88%** (74.48% at instance level) with the training data, and **77.66%** for the validation data set (72.77% at instance level). In Table 6.7 we show the percentage confusion matrix of the results.

| | | Predictions | |
|---|---|---|---|
| | | Not Broken | Broken |
| Labels | Not Broken | 50.48 | 10.44 |
| | Broken | 11.68 | 27.40 |

(a) Confusion Matrix over the training data set.

| | | Predictions | |
|---|---|---|---|
| | | Not Broken | Broken |
| Labels | Not Broken | 42.33 | 7.67 |
| | Broken | 14.67 | 35.33 |

(b) Confusion Matrix over the validation data set.

Table 6.7: Confusion Matrices (in percentages) for the training data (6.6a) and validation data (6.6b) with the best checkpoint of the Model Rx Lite.

### 6.2.4 Aggregator

With the aggregator model, we had to rewrite many code parts from the sampling process as now, we want to group and sample together all the instances of the same study. After many changes and different tests with architecture modifications, we cannot obtain quality results, as the model seems to be stuck in training without being able to learn anything from the data. When checking the training metrics of the aggregator model, where we can observe that the training and validation accuracy are stuck from the beginning around 50%, when training the model from scratch but also when applying transfer learning from the previously trained Rx Model, which allows us to tune half of the network. This leads us to think that there can be two components wrong, the architecture of the view aggregation and the study classifier, or the data.

Comparing our data set with the data used in the works that we used as references for our model [63], we can observe some significant differences. Most 3D classification data sets are generated synthetically from 3D object

models using virtual cameras to extract several views from the same object, in most of the cases from the same perspectives. Our data is not synthetic, so the multiple images of the same study do not share common generic image attributes such as shape, coloring, or scale.

After analyzing and debugging the results, we consider that the current state of the data set makes its usage unsuitable for training this model. For now, we leave the results as they are now, but we still consider that the idea behind it has much potential, so we address some possible improvements for future work (Chapter 8), especially in terms of a more refined data pre-processing pipeline specially designed to fit with the study-level task.

After this section of exploring several models, we briefly summarize which are the highlights we can extract. We have experimented with three different architectures, a ResNet (Section 6.2.1) with several experiments with it, that served us as the baseline; vanilla CNNs, with the different versions of the Model Rx (Section 6.2.3); and an aggregator model that tries to combine the information from multiple views to produce a study-level prediction (Section 6.2.4). We discard Model Rx Large due to its problem with vanishing gradients, and also the Aggregator model, as we were not able to obtain a trained model from the combination of the multiple views. In table 6.8 we summarize the accuracy obtained with the two remaining models.

| Model | Accuracy (%) | | | |
|---|---|---|---|---|
| | Train | | Val | |
| | Instance | Study | Instance | Study |
| Baseline | 84.44 | **85.99** | 78.35 | **85.00** |
| Model Rx Lite | 74.48 | 77.88 | 72.77 | 77.66 |

Table 6.8: Accuracy metrics for both selected models.

## 6.3 Noise Sources

After all the experiments with the data and the models, we are able to extract and summarize some noise sources that we found in our system that negatively affect its performance. In this section, we do not want to explain problems with the quality of the data or the pre-processing steps, such as some wrong anatomic part splittings, color inversions, or non-wrist radiography images again. Instead, we take a more in-depth look into what the data contains.

In an advanced stage of the project, with just a few remaining weeks, we obtained a subset of the data validated by a radiology expert from Asepeyo. We used this data set to focus on analyzing our models' most flagrant errors,

the false negatives. We selected 200 images belonging to 119 different studies. All the studies were tagged as a fracture in the original data. With the validated data set, 49% of the images were labeled as no fracture while the remaining 51% kept indicating a fracture.

Using the revised data set from the doctors, we can study the amount of noise that the usage of study level labels can cause in the training process, as we are unaware of the proportion of images in a study where the fracture is usually visible. From 70 studies newly labeled as fracture, in 81% of them, the fracture was visible in all the instances. The fracture was visible in half of the images in 14% of the studies, while in the remaining 4%, it was only visible in a third of the study's images. If we extrapolate these values to the whole data set, it could mean that we have approximately 10% of the training images with a wrong label.

From 200 images predicted as a fracture with a high probability, 102 were fractures while 98 were not. This shows evidence of the low trustworthiness of the labels, given that all of those are tagged as broken because the study has a fracture, but it is not possible to see a fracture in all of its samples.

We also found other unexpected noise sources in our data besides the labeling problem and the low quality of the metadata. A notable amount of images labeled as broken instances are wrists with osteosynthetic material, as they are probably from tracking appointments to check the evolution of the fracture but are still labeled as fractures. This leads us to think that it may be possible that the model is not learning to classify between fracture and no fracture with the intrinsic features of the medical condition itself, but it just detects the presence of osteosynthetic material to classify it as a fracture or as no fracture otherwise.

To evaluate the possible effect of osteosynthetic noise, we manually choose a subset of the data with 120 studies balanced between broken and not broken studies that do not contain osteosynthetic material and evaluate our two main models with this new data set. With the baseline, we obtain an accuracy of 75.41% (69.26% at instance level), and with the Model Rx 57.38% (54.42% at instance level). With these results, we can assume that the baseline is properly learning how to identify a fracture, with a low decrease in performance that is not very significant due to the small size of the new evaluation data. However, with the Model Rx, the performance decreases a lot, which suggests that the model is associating the concept of fracture with the detection of osteosynthetic material. Nevertheless, we would need more extensive data sets validated by experts to be sure that the model will work adequately with non-osteosynthetic images.

Asepeyo also notified us that they are upgrading their X-Ray scan systems.

However, our data set comprises DICOM samples from around 2016 to 2019. The new system should be able to produce results of higher quality, not only for the imaging but also on the tagging and metadata and label assignation. In the future, we could probably improve the performance of the model if we train it with newer data.

To summarize, one of the main problems that prevent us from achieving higher performance with our models is the noise in the data. The most frequent noise sources are the presence of osteosynthetic material, badly tagged instances of different anatomic parts than our target (the wrists), and old radiography images that have some defects. Furthermore, the available metadata is not reliable, neither with the anatomic part of the view, which adds more noise to the data stopping us from handling the different views in a customized way.

# Chapter 7

# Conclusions

The results of this project's experimentation are entirely satisfactory, and we can extract many conclusions and improvement ideas from it. We are able to transform our raw data into a useful data set for our Neural Network tasks, showing all the required steps to acquire the refined version. We can highlight two of the proposed and trained models, the ResNet18 and the Model Rx Lite. With the current architecture and data, we are not able to keep improving either of both models as we reach an overfitting state.

However, we are satisfied with both models' results, being able to achieve an 85% of accuracy with the ResNet18 on unseen data from the validation data set.

To end with, before addressing some extra compelling ideas and potential improvements over our work in Chapter 8 for the future, we want to summarize the main contributions that we present within this thesis.

Starting with some raw DICOM data, which we analyzed in Chapter 3, we produce a usable data set following the pre-processing pipeline described in Chapter 4. Even if the resulting data set may still have some errors, it is much better than the raw version, being able to use it for any other model that would need it, removing noisy samples, corrupted metadata fields, bad organized images, and the dependency of DICOM instead of having a traditional image and tabular data format.

The pipeline itself used for the pre-processing steps, along with the reproducible methodology linking the experiments with Git, can be considered a significant contribution. We are probably not the first research group that comes up with this or other similar methods to run and organize the experiments. However, unlike in other Computer Science branches such as Software Engineering, there is not an exact way about which are the best methodologies to proceed. We consider that after the initial overheat of setting up the system

in this way, the process of committing a job and getting all the steps of code packing, cluster connection, execution, and result retrieval and organization becomes much more straightforward and increases the efficiency, also avoiding to lose some results, the complexity of versioning and the possible lack of reproducibility for future researchers or engineers when trying to replicate our work.

From the pipelines, we want to highlight especially the raw pre-processing, as it may seem something to forget about as it only acts as an intermediate step to transform DICOM data to PNG and tabular data, and this is precisely the strong point of it. Despite being a standard format for medical data, there are few friendly ways to use DICOM in Deep Learning. With our designed pipeline, we are able not only to deal with generic data in this format, but we provide an efficient and parallelizable way that allows processing sizeable medical data sets in a scalable way.

Two of the models that we experimented with stand out from the rest, achieving high performance in this project's primary task. The first one is a CNN-based custom architecture, which gives an accuracy of 77.88% for training data and 77.66% for the validation but allowing us to use it in the near future to extract some explainability from it with techniques such as LRP, as the model does not have residual layers. On the other hand, we have a ResNet18 that, at the cost of having a more complex architecture and therefore, being more challenging to generate explainability from it, gives us a **85.99%** of accuracy for the training data, and a **85.00%** for the validation.

# Chapter 8

# Future Work

From the current state of the project, we are interested in several research lines that involve an improvement of the model performance and the extraction of some explainability from the results.

From all the analyzed models, there is at least one that we think that we can use to obtain good results in the future. Even that we discarded it for the bad results obtained, we consider that the hypothesis and motivation behind its design are not hare-brained, but we may rethink it and try different solutions. It may also help to create a specific pre-processing pipeline that includes some other CV techniques for matching images of the same object from different perspectives.

If we look at all of the models we have experimented with, we have focused all our work on convolutional-related models, from residual networks to custom convolutional neural networks for a single image and multiple view classification. However, CNNs main drawback is the lack of ability to encode relative spatial information. Hence, even if CNNs work very well for detecting certain features, these do not consider the positioning of the features with respect to each other.

Several Deep Learning fields have taken advantage of the Attention mechanism [72], mainly in Natural Language Processing (NLP) with the appearance of the Transformers [73]. The paradigmatic model is BERT [74], which produced significant improvement in terms of performance for 11 NLP tasks. Even that most applications focus on NLP, we can also use the Transformer architecture or simply attention-based models for Computer Vision tasks with improved versions. Some works prove that self-attention layers can be as expressive as a convolutional layer if there are sufficient heads, and that fully-attentional models are able to learn how to combine local behaviors that are similar to convolutions and global attention [75].

One of the main works that would be interesting to apply to our data is the Residual Attention Network [76], which combines the residual architecture with attention mechanism, stacking Attention Modules to generate attention-aware features. Other works show a specifically designed model for image treating with transformers, such as the Vision Transformer [77], which despite achieving a similar performance compared with Residual Networks of similar size, it is able to learn high-level relationships in very early training stages as it uses global attention instead of local, and it trains much faster than convolutional-based models, even using fewer hardware resources.

Furthermore, it continues to get better and better as the training data is increased. However, here is where we have to mention one of the main drawbacks that prevent researchers and engineers from training Transformer models: they demand enormous amounts of data, which, at least in the medical field, is not a very common thing to find. The requirement of having a large data set is because Transformer-based models lack some inductive biases inherent to convolutional models, such as translation equivariance and locality, being unable to generalize as well as the CNNs if they are trained with insufficient amounts of data.

For all that reasons, we consider it interesting to try to experiment and analyze the performance of Transformed-based models such as the Vision Transformer to check if we can produce better results with them.

As we mentioned before, the other front to attack to continue with this project in the near future is the models' explainability. This project aims to create a system that acts as a Decision Support System that helps the doctor analyze radiography images. Therefore, the system must provide a way to justify or indicate the reason behind the generated prediction.

Several techniques can be applied to extract visual explainability from our models, such as the already commented LRP [61, 62]. Furthermore, it could also be possible and probably obtaining outstanding results to extract explainability from attentional-based models, with Gradient-weighted Class Activation Mapping (Grad-CAM) [78], or other more specific methods especially designed for Transformer architectures such as ExBERT for BERT in NLP [79] by applying some modifications to fit our image recognition task.

Finally, after finishing this project's phase with Asepeyo, we plan to start a second phase, which could include the task of generalizing our model to analyze if it is able to learn about fractures in any other anatomic part, not only using it for wrists. They are also trying to gather a more modern data set, as modern software and scanners produce better quality results in terms of imaging but also metadata, which could reduce part of the noise caused by the lack of standardized data.

# Bibliography


[1] J. Canny, "A computational approach to edge detection," *IEEE Transactions on pattern analysis and machine intelligence*, no. 6, pp. 679–698, 1986. 6

[2] C. G. Harris, M. Stephens *et al.*, "A combined corner and edge detector." in *Alvey vision conference*, vol. 15, no. 50. Citeseer, 1988, pp. 10–5244. 6

[3] T. Lindeberg, "Detecting salient blob-like image structures and their scales with a scale-space primal sketch: A method for focus-of-attention," *International Journal of Computer Vision*, vol. 11, no. 3, pp. 283–318, 1993. 6

[4] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the seventh IEEE international conference on computer vision*, vol. 2. Ieee, 1999, pp. 1150–1157. 6

[5] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, vol. 1. IEEE, 2005, pp. 886–893. 6

[6] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. 8

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017. 8, 10

[8] B. J. Erickson, P. Korfiatis, Z. Akkus, and T. L. Kline, "Machine learning for medical imaging," *Radiographics*, vol. 37, no. 2, pp. 505–515, 2017. 9

[9] N. Antropova, B. Q. Huynh, and M. L. Giger, "A deep feature fusion methodology for breast cancer diagnosis demonstrated on three imaging modality datasets," *Medical physics*, vol. 44, no. 10, pp. 5162–5171, 2017. 10

[10] R. Paul, S. H. Hawkins, Y. Balagurunathan, M. B. Schabath, R. J. Gillies, L. O. Hall, and D. B. Goldgof, "Deep feature transfer learning in combination with traditional features predicts survival among patients with lung adenocarcinoma," *Tomography*, vol. 2, no. 4, p. 388, 2016. 10

[11] A. Sharif Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "Cnn features off-the-shelf: an astounding baseline for recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2014, pp. 806–813. 10

[12] N. Tajbakhsh, J. Y. Shin, S. R. Gurudu, R. T. Hurst, C. B. Kendall, M. B. Gotway, and J. Liang, "Convolutional neural networks for medical image analysis: Full training or fine tuning?" *IEEE transactions on medical imaging*, vol. 35, no. 5, pp. 1299–1312, 2016. 10

[13] X. Wang, W. Yang, J. Weinreb, J. Han, Q. Li, X. Kong, Y. Yan, Z. Ke, B. Luo, T. Liu *et al.*, "Searching for prostate cancer by fully automated magnetic resonance imaging classification: deep learning versus non-deep learning," *Scientific reports*, vol. 7, no. 1, pp. 1–8, 2017. 10

[14] P. Lakhani and B. Sundaram, "Deep learning at chest radiography: automated classification of pulmonary tuberculosis by using convolutional neural networks," *Radiology*, vol. 284, no. 2, pp. 574–582, 2017. 10

[15] Y. Xue, R. Zhang, Y. Deng, K. Chen, and T. Jiang, "A preliminary examination of the diagnostic value of deep learning in hip osteoarthritis," *PLoS One*, vol. 12, no. 6, p. e0178992, 2017. 10

[16] J. Chi, E. Walia, P. Babyn, J. Wang, G. Groot, and M. Eramian, "Thyroid nodule classification in ultrasound images by fine-tuning deep convolutional neural network," *Journal of digital imaging*, vol. 30, no. 4, pp. 477–486, 2017. 10

[17] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9. 10

[18] Z. Zhu, E. Albadawy, A. Saha, J. Zhang, M. R. Harowicz, and M. A. Mazurowski, "Deep learning for identifying radiogenomic associations in breast cancer," *Computers in biology and medicine*, vol. 109, pp. 85–90, 2019. 10

[19] Z. Zhu, M. Harowicz, J. Zhang, A. Saha, L. J. Grimm, E. S. Hwang, and M. A. Mazurowski, "Deep learning analysis of breast mris for prediction of occult invasive disease in ductal carcinoma in situ," *Computers in biology and medicine*, vol. 115, p. 103498, 2019. 10

[20] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014. 10, 47

[21] Z. Li, Y. Wang, J. Yu, Y. Guo, and W. Cao, "Deep learning based radiomics (dlr) and its usage in noninvasive idh1 prediction for low grade glioma," *Scientific reports*, vol. 7, no. 1, pp. 1–11, 2017. 10

[22] H.-I. Suk, S.-W. Lee, D. Shen, A. D. N. Initiative *et al.*, "Deep ensemble learning of sparse regression models for brain disease diagnosis," *Medical image analysis*, vol. 37, pp. 101–113, 2017. 10

[23] S. Khawaldeh, U. Pervaiz, A. Rafiq, and R. S. Alkhawaldeh, "Noninvasive grading of glioma tumor using magnetic resonance imaging with convolutional neural networks," *Applied Sciences*, vol. 8, no. 1, p. 27, 2018. 10

[24] G. González, S. Y. Ash, G. Vegas-Sánchez-Ferrero, J. Onieva Onieva, F. N. Rahaghi, J. C. Ross, A. Díaz, R. San José Estépar, and G. R. Washko, "Disease staging and prognosis in smokers using deep learning in chest computed tomography," *American journal of respiratory and critical care medicine*, vol. 197, no. 2, pp. 193–203, 2018. 10

[25] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826. 10

[26] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," *arXiv preprint arXiv:1602.07261*, 2016. 10

[27] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708. 10

[28] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778. 10, 37

[29] P. Korfiatis, T. L. Kline, D. H. Lachance, I. F. Parney, J. C. Buckner, and B. J. Erickson, "Residual deep convolutional neural network predicts mgmt methylation status," *Journal of digital imaging*, vol. 30, no. 5, pp. 622–628, 2017. 10

[30] P. Rajpurkar, J. Irvin, K. Zhu, B. Yang, H. Mehta, T. Duan, D. Ding, A. Bagul, C. Langlotz, K. Shpanskaya *et al.*, "Radiologist-level pneumonia detection on chest x-rays with deep learning," *arXiv preprint arXiv:1711.05225 [cs, stat]*, 2017. 10

[31] D. Kim and T. MacKinnon, "Artificial intelligence in fracture detection: transfer learning from deep convolutional neural networks," *Clinical radiology*, vol. 73, no. 5, pp. 439–445, 2018. 10

[32] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006. 10

[33] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," *Advances in neural information processing systems*, vol. 19, pp. 153–160, 2006. 10

[34] M. Ranzato, C. Poultney, S. Chopra, and Y. Cun, "Efficient learning of sparse representations with an energy-based model," *Advances in neural information processing systems*, vol. 19, pp. 1137–1144, 2006. 10

[35] A. Ortiz, J. Munilla, F. J. Martínez-Murcia, J. M. Górriz, J. Ramírez, A. D. N. Initiative *et al.*, "Learning longitudinal mri patterns by sice and deep learning: Assessing the alzheimer's disease progression," in *Annual Conference on Medical Image Understanding and Analysis*. Springer, 2017, pp. 413–424. 11

[36] D. Kumar, A. Wong, and D. A. Clausi, "Lung nodule classification using deep features in ct images," in *2015 12th Conference on Computer and Robot Vision*. IEEE, 2015, pp. 133–138. 11

[37] Y. Yoo, L. Y. Tang, T. Brosch, D. K. Li, S. Kolind, I. Vavasour, A. Rauscher, A. L. MacKay, A. Traboulsee, and R. C. Tam, "Deep learning of joint myelin and t1w mri features in normal-appearing brain tissue to distinguish between multiple sclerosis patients and healthy controls," *NeuroImage: Clinical*, vol. 17, pp. 169–178, 2018. 11

[38] Y. Al-Kofahi, W. Lassoued, W. Lee, and B. Roysam, "Improved automatic detection and segmentation of cell nuclei in histopathology images," *IEEE Transactions on Biomedical Engineering*, vol. 57, no. 4, pp. 841–852, 2009. 11

[39] A. Oliver, J. Freixenet, J. Marti, E. Perez, J. Pont, E. R. Denton, and R. Zwiggelaar, "A review of automatic mass detection and segmentation in mammographic images," *Medical image analysis*, vol. 14, no. 2, pp. 87–110, 2010. 11

[40] D. Rey, G. Subsol, H. Delingette, and N. Ayache, "Automatic detection and segmentation of evolving processes in 3d medical images: Application to multiple sclerosis," *Medical image analysis*, vol. 6, no. 2, pp. 163–179, 2002. 11

[41] H. R. Roth, L. Lu, A. Seff, K. M. Cherry, J. Hoffman, S. Wang, J. Liu, E. Turkbey, and R. M. Summers, "A new 2.5 d representation for lymph node detection using random sets of deep convolutional neural network observations," in *International conference on medical image computing and computer-assisted intervention*. Springer, 2014, pp. 520–527. 11

[42] D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov, "Scalable object detection using deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 2147–2154. 11

[43] C. Szegedy, S. Reed, D. Erhan, D. Anguelov, and S. Ioffe, "Scalable, high-quality object detection," *arXiv preprint arXiv:1412.1441*, 2014. 11

[44] H.-C. Shin, H. R. Roth, M. Gao, L. Lu, Z. Xu, I. Nogues, J. Yao, D. Mollura, and R. M. Summers, "Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning," *IEEE transactions on medical imaging*, vol. 35, no. 5, pp. 1285–1298, 2016. 11

[45] R. K. Samala, H.-P. Chan, L. Hadjiiski, M. A. Helvie, J. Wei, and K. Cha, "Mass detection in digital breast tomosynthesis: Deep convolutional neural network with transfer learning from mammography," *Medical physics*, vol. 43, no. 12, pp. 6654–6666, 2016. 11

[46] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587. 11

[47] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448. 11

[48] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99. 11

[49] R. Sa, W. Owens, R. Wiegand, M. Studin, D. Capoferri, K. Barooha, A. Greaux, R. Rattray, A. Hutton, J. Cintineo *et al.*, "Intervertebral disc detection in x-ray images using faster r-cnn," in *2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. IEEE, 2017, pp. 564–567. 11

[50] J. Liu, D. Wang, L. Lu, Z. Wei, L. Kim, E. B. Turkbey, B. Sahiner, N. A. Petrick, and R. M. Summers, "Detection and diagnosis of colitis on computed tomography using deep convolutional neural networks," *Medical physics*, vol. 44, no. 9, pp. 4630–4642, 2017. 11

[51] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788. 11

[52] R. Platania, S. Shams, S. Yang, J. Zhang, K. Lee, and S.-J. Park, "Automated breast cancer diagnosis using deep learning and region of interest detection (bc-droid)," in *Proceedings of the 8th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*, 2017, pp. 536–543. 11

[53] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37. 11

[54] Z. Cao, L. Duan, G. Yang, T. Yue, Q. Chen, H. Fu, and Y. Xu, "Breast tumor detection in ultrasound images using deep learning," in *International Workshop on Patch-based Techniques in Medical Imaging*. Springer, 2017, pp. 121–128. 11

[55] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988. 11

[56] M. Fenech, N. Strukelj, and O. Buston, "Ethical, social, and political challenges of artificial intelligence in health," *London: Wellcome Trust Future Advocacy*, 2018. 12

[57] D. P. Dos Santos, D. Giese, S. Brodehl, S. Chon, W. Staab, R. Kleinert, D. Maintz, and B. Baeßler, "Medical students' attitude towards artificial intelligence: a multicentre survey," *European radiology*, vol. 29, no. 4, pp. 1640–1646, 2019. 12

[58] E. Vayena, A. Blasimme, and I. G. Cohen, "Machine learning in medicine: Addressing ethical challenges," *PLoS medicine*, vol. 15, no. 11, p. e1002689, 2018. 12

[59] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE transactions on systems, man, and cybernetics*, vol. 9, no. 1, pp. 62–66, 1979. 27

[60] H. Z. U. Rehman and S. Lee, "Automatic image alignment using principal component analysis," *IEEE Access*, vol. 6, pp. 72 063–72 072, 2018. 29

[61] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek, "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation," *PloS one*, vol. 10, no. 7, p. e0130140, 2015. 39, 68

[62] M. Kohlbrenner, A. Bauer, S. Nakajima, A. Binder, W. Samek, and S. Lapuschkin, "Towards best practice in explaining neural network decisions with lrp," in *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2020, pp. 1–7. 39, 68

[63] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, "Multi-view convolutional neural networks for 3d shape recognition," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 945–953. 42, 60

[64] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009. 42

[65] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in *Advances in neural information processing systems*, 2014, pp. 3320–3328. 42

[66] Q. Xie, M.-T. Luong, E. Hovy, and Q. V. Le, "Self-training with noisy student improves imagenet classification," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 10 687–10 698. 47

[67] S. L. Smith, P.-J. Kindermans, C. Ying, and Q. V. Le, "Don't decay the learning rate, increase the batch size," *arXiv preprint arXiv:1711.00489*, 2017. 55

[68] E. Hoffer, I. Hubara, and D. Soudry, "Train longer, generalize better: closing the generalization gap in large batch training of neural networks," in *Advances in neural information processing systems*, 2017, pp. 1731–1741. 55

[69] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural networks*, vol. 12, no. 1, pp. 145–151, 1999. 55

[70] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014. 55

[71] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of adam and beyond," *arXiv preprint arXiv:1904.09237*, 2019. 55

[72] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014. 67

[73] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008. 67

[74] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018. 67

[75] J.-B. Cordonnier, A. Loukas, and M. Jaggi, "On the relationship between self-attention and convolutional layers," *arXiv preprint arXiv:1911.03584*, 2019. 67

[76] F. Wang, M. Jiang, C. Qian, S. Yang, C. Li, H. Zhang, X. Wang, and X. Tang, "Residual attention network for image classification," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 3156–3164. 68

[77] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020. 68

[78] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 618–626. 68

[79] B. Hoover, H. Strobelt, and S. Gehrmann, "exbert: A visual analysis tool to explore learned representations in transformers models," *arXiv preprint arXiv:1910.05276*, 2019. 68