UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona

Skin lesion analysis on the use of contextual information for melanoma identification in dermoscopic images

Master Thesis
submitted to
Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona
Universitat Politècnica de Catalunya
by

Carlos Hernández Pérez

In partial fulfillment
of the requirements for the master in
Master's degree in Telecommunications Engineering (MET)

Advisor: Verónica Vilaplana Besler
Co-advisor: Marc Combalia Escudero
Barcelona, Date 13/07/2020

Contents

List of Figures

Code Listings

List of Tables

Acknowledgements

I would like to thank my supervisor Verónica Vilaplana for her guidance throughout this research. I would also like to thank my co-tutor Marc Combalia for all his time spent counseling me and letting me use his hardware when I needed to run something in short notice. As a last thank you message, I would like to acknowledge my brother Daniel Hernandez's help wich ensured the quality of this work.

Abstract

Skin lesions are a severe disease globally. Early detection of melanoma in dermatoscopy images significantly increases the survival rate. However, the accurate recognition of melanoma is extremely challenging due to the following reasons: low contrast between lesions and skin, appearances of artifacts, etc. Hence, reliable automatic detection of skin tumors is very useful to increase the accuracy and efficiency of dermatologists. In this MSc thesis we explore the use of contextual information to improve the performance of Deep Learning classifiers in the area of skin lesion image classification. This information was obtained by means of patient embeddings, attention, and a hand-crafted contextual sampler. The proposed methods were evaluated on the ISIC 2020 dataset. Experimental results show that this techniques yield better results but they are still not significant enough. It is necessary to make future research on other ways to aggregate this information.

1 Introduction

In this chapter we will discuss the motivation behind the research done during the development of this project as well as state the main objectives and scope of the work. The chapter ends giving a brief description outlining the rest of the document.

1.1 Project motivation

Skin cancer incidence has increased in the last decade worldwide [2] . Melanoma constitutes the main cause of death due to skin cancer, when considering both its mortality and its incidence. According to the latest statistics, cutaneous melanoma is the sixth most common type of cancer in Europe, with more than 144,000 new cases diagnosed in 2018 [3] . There is the need for early detection of melanoma: the five-year survival rate of melanoma rapidly decreases as its stage advances: from 97% for stage I till only 15-20% for stage IV cancer. A delay in discovering melanoma and starting the treatment increases the risk of death by up to 40% [4] . This trend is combined with a worldwide shortage of dermatologists. In Spain, 3.27 dermatologists per 100,000 citizens [5] , 6.6 dermatologists for every 100,000 citizens in Germany, and 0.55 dermatologists per 100,000 inhabitants in the UK. Even if the excision of an early-stage melanoma is a simple surgical procedure, early-stage melanoma detection is still challenging for expert dermatologists. In high-risk patients, which usually present many atypical moles (dysplastic nevi), the number needed to treat for melanoma is still high: many nevi need to be excised for one melanoma to be detected. Even if a large number of lesions are treated for every patient, the risk of missing a melanoma is still significant [6] . AI-enabled dermatology can extend the reach of dermatological knowledge to non-specialized physicians and improve the diagnostic accuracy of skin cancer [7], [8], [9] , increasing the quality of care and reducing the morbidity and mortality of skin cancer.

To tackle this problem many collectives such as the the International Imaging Skin Collaboration (ISIC) have emerged. ISIC is an international effort to improve melanoma diagnosis, sponsored by the International Society for Digital Imaging of the Skin. The ISIC Archive contains the largest publicly available collection of quality controlled dermatoscopy images of skin lesions and it was used as the main data source for the development of this project. The ISIC 2020 dataset dermatoscopy were obtained from a wide selection of hospitals in multiple continents. This collective has been organizing yearly challenges in order to incentivize dermatoscopic research using artificial intelligence. Unlike previous ISIC Challenges [10] which dealt with multilabel problems. On this years competition the goal is to solve a binary classification problem.

Due to the constant research done by the deep learning community in the field of diagnostic study, Artificial Intelligence algorithms have achieved the *human level milestone* performance for evaluation of dermatoscopy images [17]. The main method of classifications is through the use of Convolutional Neural Networks (CNN), sometimes combined with external metadata. Computed aided diagnosis is becoming an important field of research among data scientists which is met by the increasing demand from the medical field to automate specific tasks which are very time consuming even for highly trained doctors.

1.2 Aims and Scope

The aim of this project is to investigate the potential use of contextual information for the enhancement of Deep Learning classifiers when working with a dermatoscopic dataset. More technically, we want to allow our models to be able to look at other images from the same patient when classifying an input in order to make use of the contextual information.

In this MSc thesis we describe the procedure taken to produce a baseline model for the ISIC 2020 challenge. We make use of a variety of data science tools such as extensive data augmentation, sampling mechanisms and pre-trained state-of-the-art CNNs[23].

Some assumptions are made in this project. Due to time limitations, the pre-processing of image data does not contain image segmentation. This technique has been proven useful for melanoma detection but we only use data augmentation when pre-processing our dataset. This decision reduces the performance when classifying the images. Another assumption is that metadata is mostly ignored. Metadata could improve the results but the focus of this project is make use of the contextual information of a patient's images. The different experiments developed are:

- Train and test a baseline model without using context.
- Design and implement a new type of data sampler to make use of contextual information by feeding multiple images from the same patient to the network at once.
- Implement attention technique [35] to the baseline model.
- Design and implement an embedding technique using only dermatoscopic information to add as an extra feature vector during classification.

1.3 Document structure

This document is presented in a sequential format and is divided into the following chapters:

- Chapter 2 *State-of-the-art* gives an overview of the fundamentals of the Deep learning algorithms used on throughout the project. It also gives a description of the problems that are currently being faced by the Deep Learning community as well as some related work for skin lesion classification.
- Chapter 3 *Methodology* presents the methods employed during the development of the thesis as well as describing the novel approaches that were used on the experiments. The in-and-outs of said approaches can be found on the annexes.
- Chapter 4 *Results* starts by stating the hypothesis being tested and presents the results gathered from a variety of approaches.
- Chapter 5 *Discussion* presents an t-SNE study to shed some light on the results.
- Chapter 6 *Conclusions and Future Work* states the conclusions drawn from the experiments and presents the challenges met alongside possible projections on the research of dermatoscopy imaging processing.

2 Background

In this chapter we will shed some light on dermatoscopic imaging and skin lesions as well as describe the state of the art regarding automated melanoma detection using Deep Learning. We will focus on how this lesions occur as well as how they are captured into images. This task cannot be understood without understanding how Artificial Neural Networks (ANNs) extract relevant features from an image, which we will analyze first.

2.1 Melanoma lesions and dermatoscopic imaging

By definition, skin cancer is the abnormal growth of skin cells. The major and most important exogenous factor causing skin cancer is exposure to UV irradiation through sun exposure [19]. Melanoma, a type of skin cancer, can develop on parts of the body that rarely see light, including the palms, beneath the fingernails or toenails, and the genital area. The causes for this can vary significantly, as can the speed by which cancer develops. Melanoma is a malignant tumour which develops from the pigment-containing cells known as melanocytes [18]. Melanoma can however be cured with prompt excision if diagnosed and detected early. Identification of melanoma from skin lesions using methods such as visual inspection, clinical screening, dermoscopic analysis, biopsy and histopathological examination of skin lesion can be inaccurate and laborious even with experienced dermatologists [20]. This is due to the complex visual characteristics of the skin lesions such as multi-sizes, multi-shapes, fuzzy boundaries, low contrast when compared to the skin and noise presence such as skin hair, oils, air and bubbles. Development of an efficient Computer Aided Diag-nosis (CAD) system for detection and diagnosis of melanoma cancer is thus required. This will improve the diagnosis rate of melanoma and early detection which can facilitate treatment and reduce the mortality rate of the disease.

There has been significant improvement in the research for the development of CAD algorithms and techniques for skin lesion analysis in the recent past. Some of the popular techniques use a rule based on asymmetry, border structure, variegated colour and dermatoscopical structures (ABCDE). This was derived from the rule commonly used by dermatologist for diagnosis of skin cancer. In the ABCDE rule, A stands for asymmetry which means two sides do not match while they match for the symmetry. This can assist in distinguishing benign from malignant skin lesions . B stands for the border structure which is even for benign and uneven for the malignant in most cases. C is for colour variegated; color is always just one colour in the case of benign while the malignant always possess two or more colors. D is the total diameter as general dermatoscopical structure are always very small like half a centimeter for the benign while is always larger than that in the case of malignant. The benign skin lesions are not harmful while the malignant are cancerous and harmful. This rule has always been applied by many hand crafted methods for the analysis of skin lesions images towards the melanoma detection as shown in figure 1.

2.1.1 Acquiring dermatoscopies

Dermatoscopy is the examination of skin lesions with a dermatoscope. Also known as dermoscopy or epiluminescence microscopy, it allows for inspection of skin lesions unobstructed by skin surface reflections. The dermatoscope consists of a magnifier, a light source (polarized

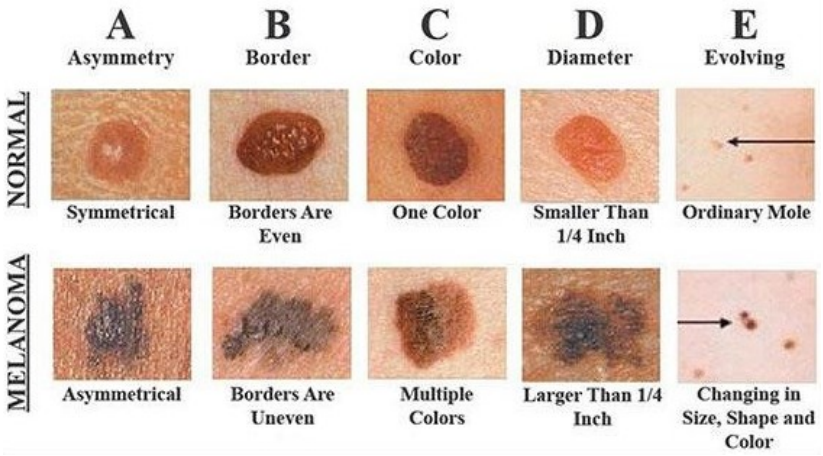

Figure 1: Images comparing benign and malignant melanomas using the ABCDE rule.

or non-polarised), a transparent plate and a sometimes a liquid medium between the instrument and the skin. When the images or video clips are digitally captured or processed, the instrument can be referred to as a digital epiluminescence dermatoscope.

Althought the introduction of this device has improved the efficacy of diagnosis many dermatoscopies suffer from a variety of artifacts. These appear in a variety of forms as represented in figure 2 which shows images from a typical melanoma dataset. These artifacts impose a challenge to any automated diagnosis system as it should learn to ignore them as they are not representative of the pathology.

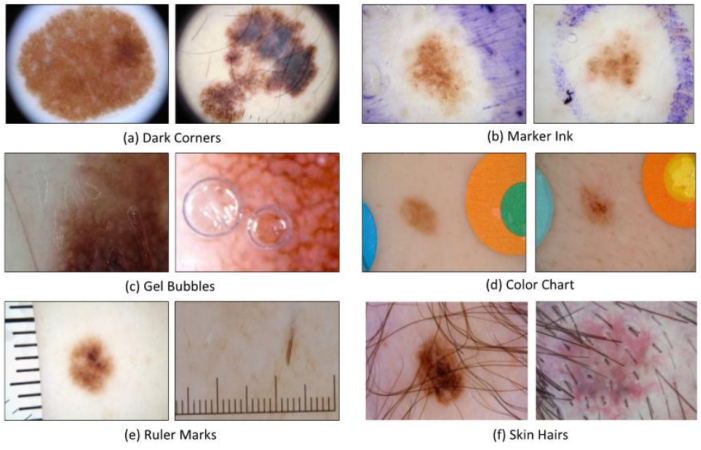

Figure 2: Melanoma images with artifacts.

2.2 Deep Neural Networks

Artificial Neural Networks have been a hot topic since their introduction at the end of the first half of the 20th century. The idea of machines acting in a manner that resembled human intelligence and understanding of the world can be traced back to the Napoleon Era and his use of *turk*

machines to play chess. Many writers had also toyed with the idea of machines and humans interacting as equals in a futuristic society. Even though the scientific community knew that there was a lot to be gained researching on this newly found field, its problem was threefold. On the one side the computational capacity was simply not enough to solve any meaningful problem. On the other side there were no large datasets available nor the technology to store them. To top it all off every connection among neurons needed to be done physically which made it more error prone as well as more difficult to try new configurations. This led to what it is known today as the *AI winter*.

The Neural part of the ANN comes from the idea that a single agent inside a ANN is called a perceptron or neuron. This algorithm was formulated in 1958 by Frank Rosenblatt and it has become the backbone of most ANNs. Its fundamental comes inspired by how biological neurons work and communicate with each other. That being said I personally think that it is detrimental to both Data Science and Medical fields to insist on drawing connections between the two types of neurons, being artificial or biological. Further similitudes in nomenclature will be avoided for the rest of this thesis. A perceptron can be either ON or OFF depending weather the sum of its inputs $\mathbf{x}$ times a learned weight, $\mathbf{w}$ is greater than a certain threshold also known as bias b. The fundamental equation of a perceptron is as follows:

$$a(\mathbf{x}) = b + \sum_{i=1}^{n} w_i x_i = b + \mathbf{w}^T \mathbf{x} \tag{1}$$

Where n is the number of incoming neurons and x_i is the activity of each neuron. The variable $\mathbf{x}$ can represent the set of features of one sample such as a single pixel value of an image. It is common to simplify this equation by adding the bias term as another input with a single weight value of +1 and incorporate one weight which acts as bias. We can then re-write equation 1 as follows:

$$a(\mathbf{x}) = w_0 1 + \sum_{i=1}^{n} w_i x_i = \sum_{i=0}^{n} w_i x_i = \mathbf{w}^T \mathbf{x} \tag{2}$$

Even thought this is the most common computation for the activation of a neuron, other types of ANNs such as the Radial Basis Function (RBF) compute the cuadratic distance of $\left\|\mathbf{w}^T - \mathbf{x}\right\|^2$ instead of the scalar product between the weights $\mathbf{w}$ and the input pattern $\mathbf{x}$. All in all the following figure 3 sums up the mathematical foundations explained:

The only mathematical concept left to explain is the *f()* which represents a non-linear function that is used to determine whether the neuron fires. This non linearity is what makes ANNs able to replicate any distribution. That is because if we are trying to fit non-linear data and only have linear activation functions, our best approximation to the non-linear data will be linear since that's all we can compute. As technology allowed it, perceptrons were assembled into layers with multiple perceptrons on each. On simpler architectures all the perceptrons of a layer are connected to all the perceptrons of the next layer. This creates complex structures which can be dissected into three main components:

1. Input layer: corresponding to the first layer of the network.

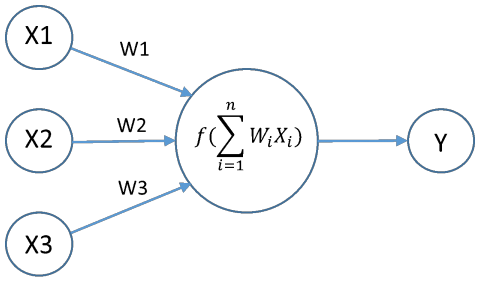

Figure 3: Visual representation of a perceptron.

2. Hidden layers: all layers between the input and the output layer. They are named this way as the user does not see them.
3. Output layer: name given to the last layer of the network.

2.2.1 Convolutional Neural Networks

There are many kinds of architectures that have sprouted from the initial idea of the perceptron although most of them are beyond the scope of this document. Amidst all the existing architectures, the scientific community has adopted CNN [32] for solving image processing problems. A simple example would be, given an image, identify which object it contains. This is an extremely complex task for a regular neural network as any high quality image would need in the order of millions of neurons on its input layer to be able to see the image fully. As this problem would become intractable, CNNs leverage the following ideas to outperforms ANNs for image classification:

- Local connectivity
- Parameter sharing
- Pooling and subsampling hidden units

By exploiting local connectivity each hidden unit is connected only to a sub-region of the image known as the receptive field (it is still connected to all channels). This tactic solves the problem of intractability that fully connected hidden layers would cause. Sharing parameters across certain units reduces even more the number of parameters and forces these units to extract the same features at every position as features are *equivariant*. Units are then organized into feature maps. Hidden units within a feature map cover different positions in an image. Each feature map is then convolved through the image with the equation shown below:

$$y_j = f\left(\sum_i k_{ij} * x_i\right) \tag{3}$$

Where x_i is the ith input channel, k_{ij} is the kernel matrix and y_j is the hidden layer. As per usual on any ANNs a non-linear function $f()$ is also applied. This feature maps, also known as filters are learned during training. It's easy to understand that a single convolutional filter, can't learn to extract the great variety of patterns that compose an image. For this reason, every

convolutional layer is composed of n (hyper-parameter) convolutional filters, each with depth D, where D is the input depth as seen in:

$$O_m(i,j) = f\left(\sum_{d=1}^{D}\sum_{u=-2k-1}^{2k+1}\sum_{v=-2k-1}^{2k+1} F_{m_d}^{(1)}(u,v)I_d(i-u,j-v)\right) \quad m = 1,\cdots,n \tag{4}$$

Therefore, a convolution among an input volume $I = \{I_1,\cdots,I_D\}$ and a set of n convolutional filters $\left\{F_1^{(1)},\cdots,F_n^{(1)}\right\}$ each with depth D, produces a set of n activation maps, or equivalently, a volume of activations maps with depth n. To improve the expressiveness of the network, every convolution is wrapped by a non-linear function *f()* (activation), in that way the training procedure can learn to represent input combining non-linear functions

The pooling and subsampling concept is performed in non-overlapping neighborhoods in order to reduce dimensionality and provide invariance to small local changes. These pooling layers are placed among convolutional layers. It is worth noting that they operate independently over each depth slide of the input volume. They modify the dimensions of height and width but the depth is left intact. Typical pooling methods are max pooling or average pooling.

The anatomy of a CNN can be divided into two parts: the first one is the **feature extractor** which comprehends all the convolutional blocks, that take care of extracting abstract information (in the form of feature maps) from the images. The second part is the **feature classifier**, formed by one or more fully connected layers at the end of the network which give the actual prediction based on the previous features. For recognition, the last fully connected layer ends with a sofmax non-linearity which provides an estimate of the conditional probability of each class.

The weights of both the convolutional blocks and the fully connected layers need to be trained using backpropagation. CNNs' strength relies on the notion that, rather than processing an entire image at once in order to find certain features, they extract features from local sections of the input images. A convolutional layer has a fixed amount of filters that convolve through all the image and extract features. CNNs learn hierarchical representations with increasing levels of abstraction. As an example, the first layers keep track of simple edges and, towards the end of the architecture, the filters could search faces or digit numbers. In short, the network learns features that activate when they *see* some specific type of feature at some spatial position in the input. Stacking activation maps for all filters along depth dimensions forms the full output volume. The rise of this kind of networks emerged at the end of the 90s with models such as LeNet-5 [21].

Computer vision researchers have develop a wide variety of architectures in the last decades. Throughout this project only the EfficientNet [23] was used as it has proven to be on of the stat-of-the-art models. This networks implement more elaborated techniques such as batch normalization or residual connections to fight vanishing gradients among others.

2.2.2 AutoEncoders

An AutoEncoder (AE) is a form of unsupervised learning which distants itself from supervised learning as no label is provided to the data when training the network. An AE is a special construction of a DNN which aims to reduce the dimensionality of the input data via the omission

of uncorrelated data. This structure is composed of two main parts, an Encoder and a Decoder as seen in figure 4. On the input we can find the encoder which transforms the input data into a latent space that has equal or less dimensions. This technique could be seen as an enhanced version of Principal Component Analysis (PCA) which is a dimensionality-reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set. Reducing the number of variables of a dataset usually comes at the expense of accuracy. PCA tackles this issue by projecting its eigenvectores into the largest eigenvalues and thus minimizing the information loss. AE takes this approach one step further by adding non linearities. The Decoder part takes this information into the latent space and aims to reproduce the original data.

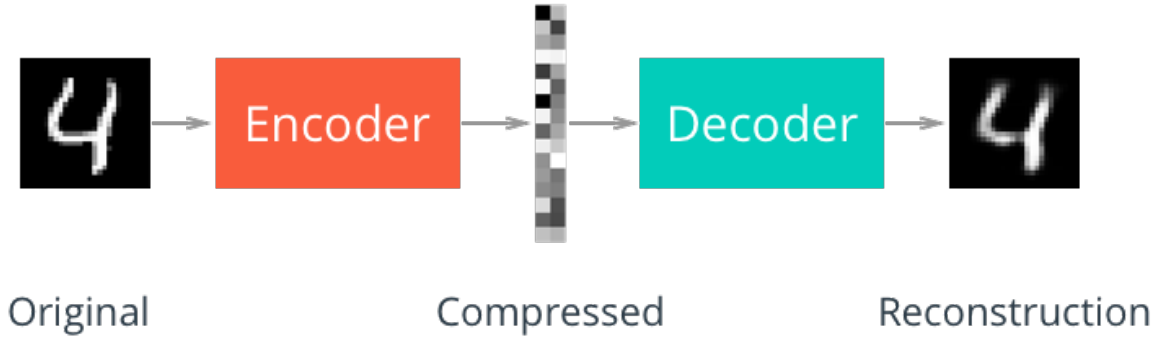

Figure 4: Visual representation of an AutoEncoder.

Therefore the AE does not need any kind of label to train itself as the error is computed via reconstruction loss. This architecture is able to determine how well (or bad) it's doing by comparing its output to its input. The math behind the network is fairly easy to understand. As mentioned before, the network is split into two segments, the encoder and the decoder.

$$\begin{aligned} &\phi : \mathscr{X} \rightarrow \mathscr{F} \\ &\psi : \mathscr{F} \rightarrow \mathscr{X} \\ &\phi, \psi = \underset{\phi,\psi}{\arg\min} \|X - (\psi \circ \phi)X\|^2 \end{aligned} \tag{5}$$

The Encoder function is denoted by ϕ which is responsible to map the original data $\mathscr{X}$ to a latent space $\mathscr{F}$ that represents the bottleneck. On second half of the AE the Decoder, ψ, maps the latent space $\mathscr{F}$ at the bottleneck to the output which should be same as the input function. Thus we are basically trying to recreate the original image after some generalized non-linear compression.

The AE has seen many different variations arise as the scientific community started to make use of its advantages. For dealing with images, AEs tend to not use fully connected layers but rather a set of convolutional layers with only a densely connected layer where the latent representation is. In a nutshell the network learns to reduce the dimensionality via extracting feature vectors as seen in figure 4 which are then used by the decoder to form the original image. A simplified version of an RGB autoencoder can be seen below:

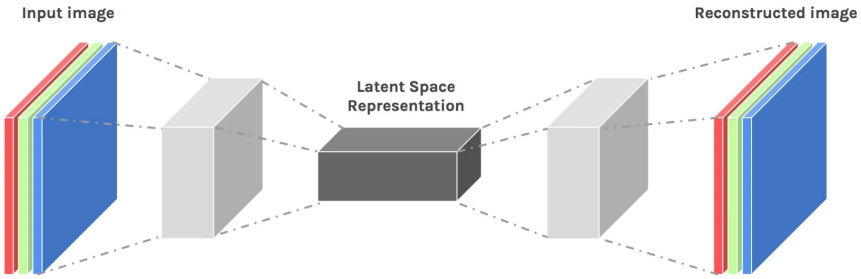

Figure 5: Bird view of a Convolutional AutoEncoder (CAE).

2.3 Related work

On this subsection we will analyze what is currently being done by the scientific community to tackle the problem of melanoma classification using Deep Learning.

2.3.1 Use of CNN for melanoma detection

Deep learning approaches have shown promising results for solving computer vision tasks such as object tracking, and image classification [38]. Deep learning mechanisms can learn sets of high level features from low level ones and gain high accuracy for classification applications without the need for extracting handcrafted features. Especially there has been a trend for taking advantage of superior power of deep learning methods in medical imaging tasks [13]. As explained in an earlier section of this chapter, CNNs are a type of deep learning method where trainable filters and pooling operations are applied on the raw input images, extracting set of complex high level features automatically.

Extracting an effective and discriminative feature set, which precisely differentiate between various classification groups, is a challenging task. On the one hand, there is a pitfall that by using a large set of features we may feed some incoherent traits to the network. On the other hand, by using a small set of features there is a possibility of missing some proper descriptors. Hence, automatic feature extraction systems could be utilized to achieve a discriminative feature set based on a labeled training set, without the need for definition of handcrafted feature extraction procedures.

As ISIC has the biggest publicly available dermatoscopic dataset, we can find many research papers which make use of their images. The dataset size has grown steadily since their first challenge in 2016. In its very first year, the winners already used CNNs to obtain feature maps from the images [12] although their size was shallower. In 2017 a straight forward CNN was proposed for the dermatoscopic feature extraction task [15]. Compared to the 2016's winner, the next year's victor total number of convolutional layers was six-fold, from two to twelve. In subsequent years the importance of acquiring better feature extractors became a main objective. On last year's competition, the ISIC 2019 [11] where the winner of the competition used a compound of deeper CNNs to achieve the highest score in the competition. They used an ensemble with seven EfficientNets and one SENet154 [16] in order to add variability.

2.3.2 Contextual information

The deep learning community has been trying to make use for contextual information for a variety of purposes [36] [37]. Contextual information implies that within the data, there are correlations between data points that which serves as a way to give more information for each input image. The form of the contextual information usually comes hand in hand with fusing the information through the process of classification at some point in the architecture. However, most existing methods either simply concatenate multi-level feature maps or calculate element-wise addition of multi-level side outputs, thus failing to take full advantages of them.

Another way to make use of contextual information came by the hand of Natural Processing Language (NLP) which starting using long short-term memory (LSTM) that helped the network to remember the context when dealing with translation tasks. A more advance version to retrieve contextual information also sprouted on the NLP field with the use of *attention* layers [35]. This specific use will be discussed more thoroughly in the next chapter.

For image processing specifically, we have seen the use of contextual learning for segmentation (which is a pixel-wise classification problem) via concatenating a LSTM network to a 2D CNN from Jinzheng Cai et al [14] and a similar approach using multi-view features also using an LSTM by Hongyu Wang, et al. [37]

2.3.3 Data augmentation

Data augmentation is the creation of altered copies of each instance within a training dataset [26]. Let's unpack this statement in the context of image classification. When we feed image data into a neural network, there are some features of the images that we would like the neural network to condense or summarize into a set of numbers or weights. In the case of image classification, these features or signals are the pixels which make up the object in the picture. On the other hand, there are features of the images that we would not like the neural network to incorporate in its summary of the images (the set of weights). In the case of image classification, these features or noise are the pixels which form the background in the picture or the presence of artifacts.

So, how do we ensure that the neural network can differentiate signal from noise? A very simple solution is to create multiple alterations [25] of each image, where the signal or the object in the picture is kept invariant, whilst the noise or the background is distorted. These distortions include cropping, scaling and rotating the image, among others. Therefore, the network of neurons observes the invariance in the images and encodes this information or signal in the set of weights which summarize the training data. By doing this we are also incrementing the variance [27] of our images and reducing the inherent bias of any dataset as mentioned in a previous subsection.

2.3.4 Class balancing

When dealing with classification problems and Deep Learning architectures the unbalancing of a dataset can present a major drawback to find an optimal solution. When dealing with a multiclass problem we would want the training phase to be as smooth and progressive as possible. In order to achieve this the network should see roughly the same amount of data of each class

on every batch. Datasets rarely present an uniform distribution of samples regarding class distributions. If we were to randomly sample images from this dataset while training the network, it might learn than only outputing the label of the benign class would already make its accuracy over 80% which is a better than random result. This problem has been named *overfitting* by the community. In our case the network would do an above random prediction but these results would not hold in an another dataset, nor even in the validation set.

Another problem might be that the features that the network learns to classify for one class might be different or even counter productive to predict another. The network needs to find a subset of all the features in which all classes are optimally classified. These would not happen if the both classes weren't correlated, that is to say. Their high-dimensional features are orthogonal among each class. Real world applications can not assume this. The model needs to learn to trade-of between all classes.

There are a variety of ways to soften the inconveniences caused by this problem. Some loss functions have also jumped present the possibility to add weights to each class. Giving higher weights to the under-represented class will enhance their losses and thus the network will put more effort to reduce the loss created by those classes. The way that we retrieve samples from our dataset before feeding them to the network, also known as sampling methods, are also widely used. These techniques virtually reduce damage inflicted by class unbalancing.

3 Methodology

This section explains the different methods that were implemented to improve the accuracy of the trained model. It focuses primarily on the sampling techniques, the use of contextual information as well as the architectures used to extract the features. It also gives a description of the dataset in use.

3.1 Dataset

During the development of the experiments explained in the next section we used the ISIC 2020 dataset. The images are provided in DICOM format. This can be accessed using commonly-available libraries, and contains both image and metadata. It is a frequently used medical imaging data format. Images are also provided in JPEG and TFRecord format (simple format for storing a sequence of binary records). Images in TFRecord format have been resized to 1024x1024 pixels. Metadata is also provided outside of the DICOM format and in CSV files which contains the following extra information per image:

- image_name - unique identifier, points to filename of related DICOM image
- patient_id - unique patient identifier
- sex - the sex of the patient (when unknown, will be blank)
- age_approx - approximately patient age at time of imaging
- anatom_site_general_challenge - location of imaged site
- diagnosis - detailed diagnosis information (train only)
- benign_malignant - indicator of malignancy of imaged lesion
- target - binarized version of the target variable (train only)

An image from each class can be found in figure 6. The presence of artifacts can be found in either class as seen in the hairs found on the middle image of both classes.

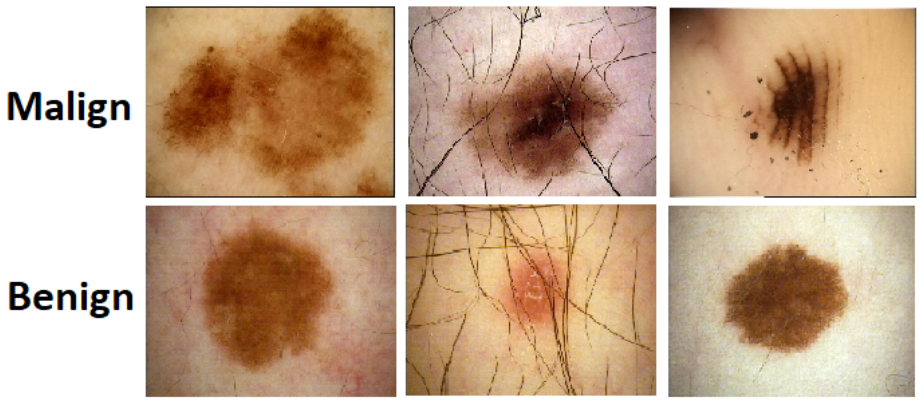

Figure 6: Array of images showing melanoma lesions at the top and benign moles on bottom.

As it can be seen in figure 7 our dataset is not absent of artifacts. Image a) exhibits dark corners that are not present in most of the images. Picture b) shows gel bubbles, c) and d) show a cross made to center the lesion on the image and a background out of focus respectively. That being said, no measures were taken to remove artifacts.

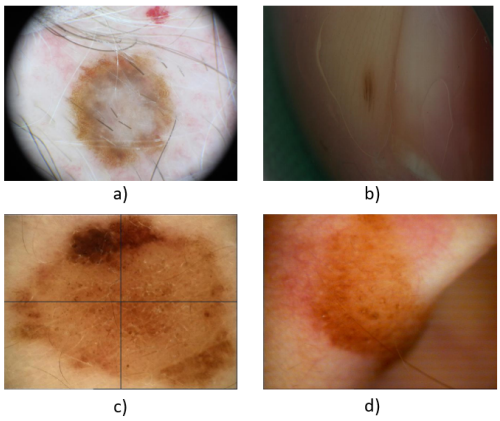

Figure 7: Different artifacts found in the ISIC 2020 dataset.

As it is frequent with medical datasets, the ISIC 2020 repository class distribution is not uniform. The total amount of dermatoscopies per class is shown in table 1.

Class	Samples	Percentage
Benign	32541	98.2%
Malign	584	1.8%

Table 1: Class distribution in ISIC 2020 dataset.

Exploring the dataset further we found that there was a total of 2096 patients. The number of images per patient was diverse, almost half of them have less than 10 images. The number of images decreases exponentially until a maximum of one single patient having 114 dermatoscopies. The full distribution can be found on figure 8. When training, we did not want images from the same patient to appear in both validation and training dataset. We took this into account when making the training and validation split.

Total number of patients	Patients with melanoma	Patients without melanoma
2056	428 (20.9%)	1628 (79.1%)

Table 2: Distribution of patients with melanoma.

As it happened with the number of images per patient, not every patient with melanoma had the same number of malign pictures. Although this distribution's tail is not as large the number of images, it also decays exponentially. Almost three quarters of them had only one picture as seen in figure 9.

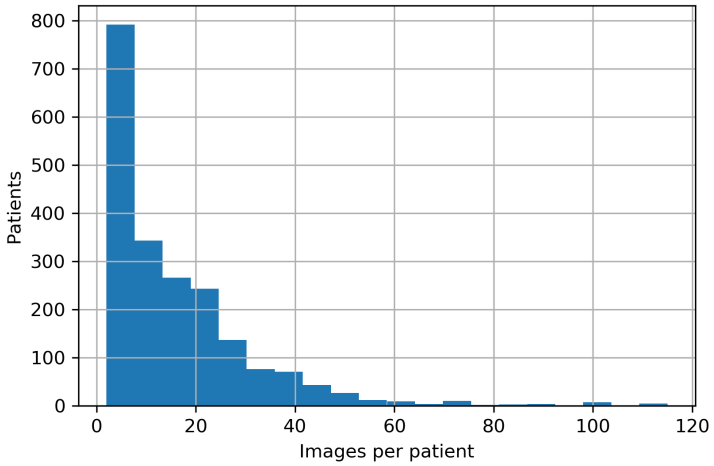

Figure 8: Number of images per patient.

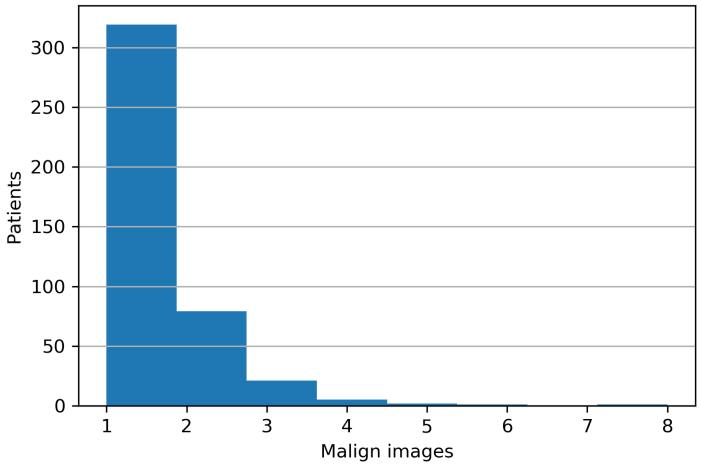

Figure 9: Number of malign images from afflicted patients.

3.2 Sampling

Sampling was extensively used in every step of this project. The goal that we wanted to achieve by sampling was twofold:

- Fight the unbalance property of our dataset
- Enable the use of contextual information

When we talk about sampling we are talking about how we are taking images from our dataset to be later fed to the network. Sampling alters the number of images on each **batch** but the total data stays the same. There are a variety of strategies when it comes to sampling methods [34]. Most of them can be marked down into two categories. On the one hand the most common and

simplest strategy to handle imbalanced data is to **undersample** the majority class. On the other hand we could also **oversample** the minority class in order to make up for its loss of presence in the dataset. A simple visual representation is shown in figure 10. Although the concept behind it is simple to understand it is worth noting that there are multiple ways of achieving this goal and the results will vary depending on which one we take .

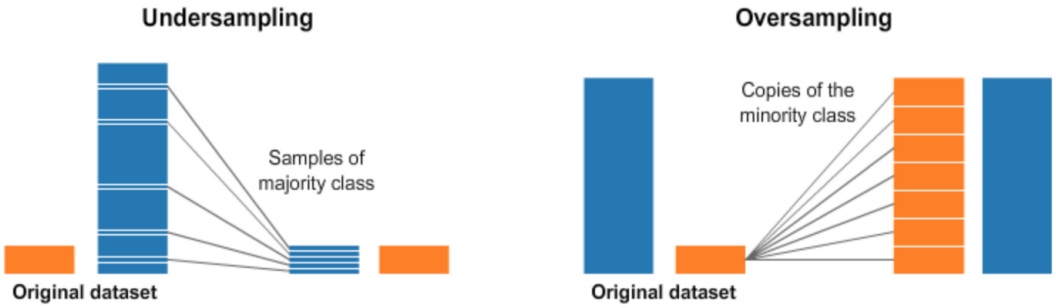

Figure 10: Oversampling and undersampling techniques.

The total number of images in each class is denoted as $\mathscr{U}$ and $\mathscr{O}$ which stand for the under-represented and for the over represented respectively. We then define the sets obtained by using both of this techniques as:

$$\mathscr{U}_{shrank} \in \mathscr{U}$$

$$\mathscr{O} \in \mathscr{O}_{enlarged}$$

Undersampling reduces the total amount of data that your classifier sees and thus it may end up having a negative effect. A more complex undersampling technique would be to sample n amount of data points from the over represented class, being n the number of samples of the under represented class. During training, after a set number of epochs have passed, swap the samples of the over represented class to another n number of new data points. This helps the network see more of your dataset but at the same time it may enhance overfitting on the images from the under represented class (as they are seen many more times). This would make for a $\mathscr{U}_{shrank}$ closer to $\mathscr{U}$ than the simple undersampling technique.

The counter tactic of oversampling tends to yield better results as you are not reducing the total number of samples from your dataset. Even though we are not losing total data points, we are still replicating the under represented class and that might cause overfitting. This is due to the over exposition of the under represented class. A method to mitigate and even resolve this problems comes by the hand of data augmentation. If done right, the network rarely see the same image but rather small permutations of it on different passes.

3.2.1 Weighted Random Sampler

This sampler is already implemented in the Pytorch library and helps to combat class imbalance by over-sampling the under-represented class while exercising the opposite technique for

the over-represented class. The sampler accepts a list of real numbers as an input. Through experimentation we decided to assign each class the inverse of their frequency in the database. Thus we ended up with a vector **w** with length equal to the number of classes where the weight of each class would be $TotalSamples/ClassSamples$ so the final result is:

$$(\mathscr{U} + \mathscr{O})/\mathscr{U}$$

$$(\mathscr{U} + \mathscr{O})/\mathscr{O}$$

These weights make each batch have roughly the same number of samples of each class.

The Weighted Random Sampler takes the input weights and normalizes them in order to create a sort of probability of each class being drawn from the dataset. The sampler uses this vector of probabilities **w** to draw a number of samples from the multinomial distribution inside the vector **w**. The number of samples drawn is equal to the assigned batch size.

By feeding a model with the same number of samples per class we obtained a much more smooth loss curve. This is due to the fact that the model had an equal amount of instances from each class when computing the loss. Thus through back-propagation the network tried to learn how to classify all the classes at the same time.

3.2.2 Contextual Sampler

We wanted to explore an enhanced approach of the Weighted Random Sampler to make use of the contextual information of our dataset. In order to make use of this technique we needed to feed the network with batches of images that contained several images from the same patient. This proved non-trivial as many patients did not have any image belonging to the *malign* class and most of the patients who did only had one. In order to fight the already known class balancing problem we decided to create a custom Pytorch sampler that performed the following operations:

1. Balance the batch so half of the contained patients had at least one image with a melanoma.
2. Balance the images of the patients with melanoma so half of the them would be of benign and the other half of malign dermatoscopies. This meant that most of the time the same melanoma image had to be repeated.
3. Yield a set number patients and images of each patient on every train iteration.

This sampler also contributed to fight the inherit unbalance of the dataset. On every training iterations the percentage of the images went from to 1.8% to 25% (half of the images from half of the patients were melanomas).

During validation, the sampler yielded all the images from a single patient at once. As most of the patients that had malignant melanocytic lesion only had one dermatoscopy from this class, data augmentation techniques were introduced. Some standard vertical and horizontal flips were performed alongside rotations and random color adjustments in order to enhance the variability of the dataset. We also tried another colour constancy algorithm known as Shades of Gray [39] which proved to yield the same gain in accuracy as our random colour permutations but it was more computationally expansive and thus it was discarded.

All in all, the sampler enabled for a use of contextual information but further worked needed to be done. By only using this sampler, the model would extract the features from the given images but the prediction of each image would not be affected by the same patient's images. On the next chapter we present two methods to tackle this problem

3.3 Models used

The experiments were performed using two main components. A CNN named EfficientNet and a Convolutional AE. This subsection gives a small description of each.

3.3.1 EfficientNet

EfficientNet is a model developed by Google AI, its architecture was found via Compound Model Scaling. In the paper *"EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks"* they propose a novel model scaling method that uses a simple yet highly effective compound coefficient to scale up CNNs in a more structured manner. Unlike conventional approaches that arbitrarily scale network dimensions, such as width, depth and resolution, their method uniformly scales each dimension with a fixed set of scaling coefficients.

In order to understand the effect of scaling the network, the team at GoogleAI systematically studied the impact of scaling different dimensions of the model. While scaling individual dimensions improves model performance, they observed that balancing all dimensions of the network—width, depth, and image resolution—against the available resources would best improve overall performance. The first step in the compound scaling method is to perform a grid search to find the relationship between different scaling dimensions of the baseline network under a fixed resource constraint (e.g., 2x more FLOPS).This determines the appropriate scaling coefficient for each of the dimensions mentioned above. They then apply those coefficients to scale up the baseline network to the desired target model size or computational budget.

EfficientNet posseses different versions according to its depth. This architecture was compared with other existing CNNs on ImageNet. In general, the EfficientNet models achieved both higher accuracy and better efficiency over existing CNNs. This means that the model occupies less space on the GPU thus more images can be loaded on each batch further increasing the speed in training. On the figure 12 below we can find a full comparison.

3.3.2 Auto-encoder

The main idea to use an AE was to reduce the correlation among the features extracted from the images and the features used for the contextual information. As explained in chapter 2 AEs learn to compress the data that is being fed to them and how to reconstruct the same data using the latent variables stored in the interconnection between encoder and decoder. This process differs from the way that the aforementioned EffNet extracts features from the images. The CNN does not care to handle the reconstruction task, it only focuses on using the extracted features to solve a classification problem.

Both the AE and the CNN are able to effectively reduce the dimensionality of the images but their approach to get them is different. Thus we are able to effectively compute image features that represent the same high dimensional image but their correlation is minimized.

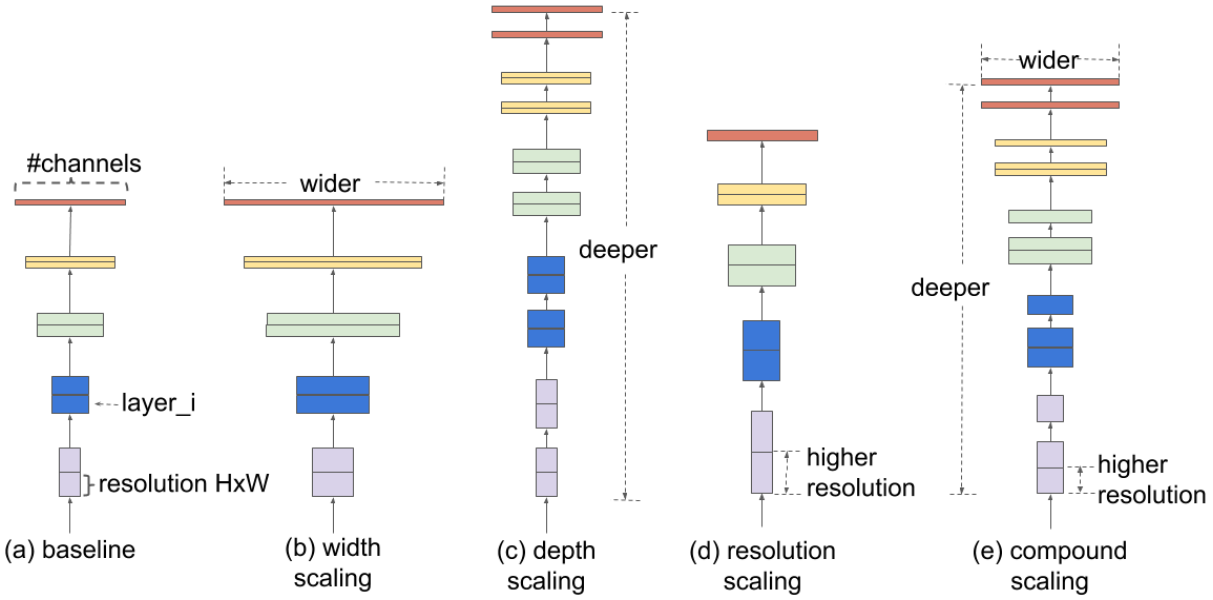

Figure 11: Comparison of different scaling methods. Unlike conventional scaling methods (b)-(d) that arbitrary scale a single dimension of the network, the EfficientNet scaling method uniformly scales up all dimensions in a principled way.

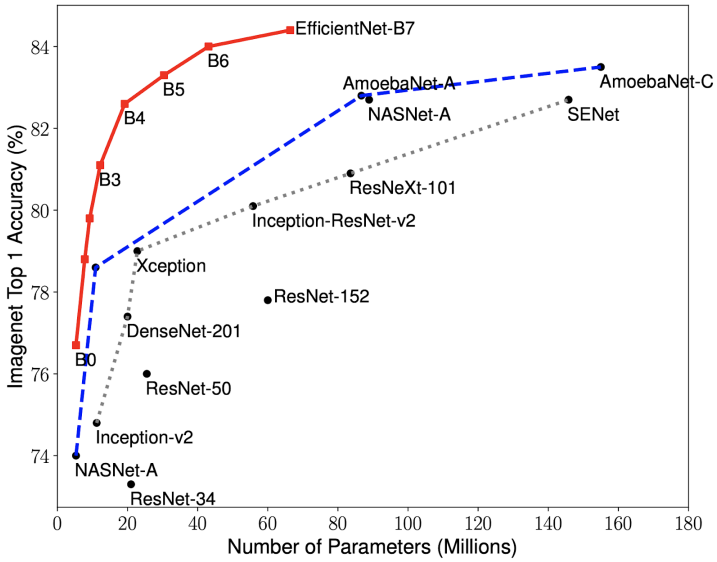

Figure 12: Model Size vs. Accuracy Comparison. EfficientNet-B0 is the baseline network developed by AutoML MNAS, while Efficient-B1 to B7 are obtained by scaling up the baseline network.

Two AE were trained during the development of this thesis. The first simple model code was developed at the beginning of the work. It handled the reconstruction problem correctly but the extracted features were not of much use as it only contained three convolutional layers in the downsampling and another three in the upsampling. Some months later a more complex architecture found on the open GitHub repository of the user Foamliu [40]. On his code there

were 5 downsampling and 5 upsampling segments with several layers each. It also showcases the sharing of pooling indices which helps the network remember in which position of the kernel it found the maximum value when downsampling. The result is a better reconstructions. A more detailed visual explanation can be found in figure 13.

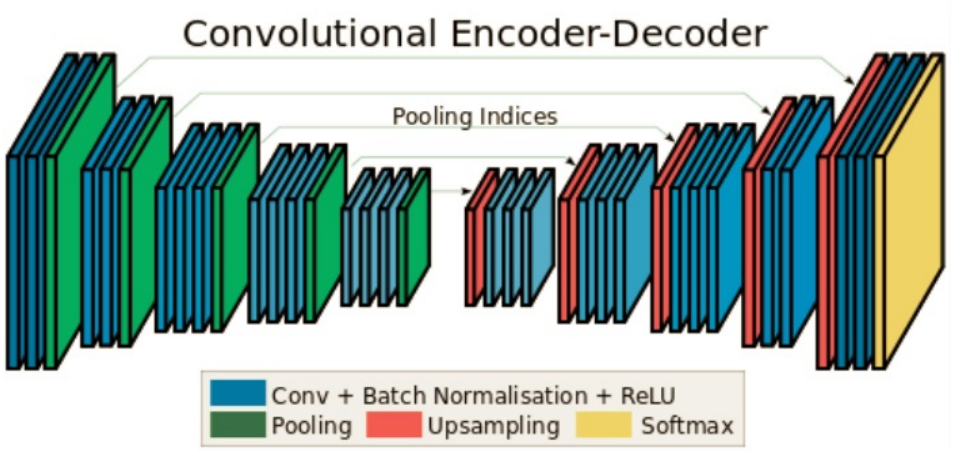

Figure 13: Layer scheme of the CAE used.

3.4 Use of contextual information

As mentioned before, the ISIC 2020 dataset contains several images from almost every patient. The main objective was to create an architecture in which we would input N images from the same patient and compute N labels. The trick is to use the information contained on one image to influence on the decision making of the other images. We explored two main ways of achieving this goal which are explained below.

3.4.1 Context through a patient embedding

We wanted to implement a form of *patient feature embedding*. To achieve this we trained the models explained in the past subsection and then passed all our dataset through them to extract the features of each image. With this information several classification methods were tried out. The description of this algorithms alongside their results are thoroughly explained in the next chapter. With the extracted features we computed the mean of every feature and we added it with late fusion to our model during training. Two main techniques were used:

- Media from the Convolutional Neural Network
- Media from the AutoEncoder
- Use of the Contextual Sampler

The idea behind it was to capture the information of a patient in one single vector of features in order to facilitate the classifier's task of predicting wether an image had a bening or malign dermatoscopy. This vector could either be already computed from an already pre-trained network

or be updated during training. This information is added through late fusion after the convolutional layers of the EfficientNet and before the fully conected layers that form the **feature classifier** as seen in figure 14.

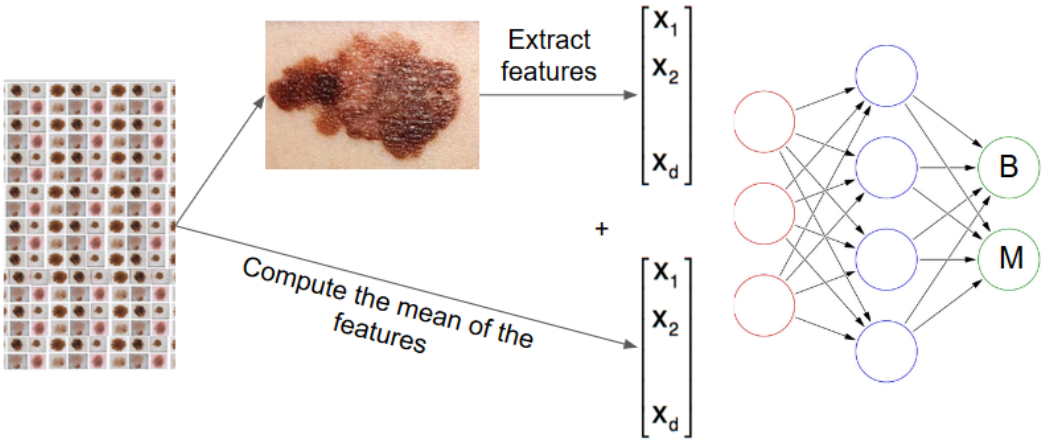

Figure 14: Architecture of patient feature approach.

3.4.2 Context through attention

We hypothesized that there was improvement to be gained by using the contextual information brought by having multiple images from the same patient. Intuitively when using attention we are adding extra information retrieved from other images. The contribution of these images is weighted according to how important they are to solve the task at hand, in our case, solve a a classification problem.

After extracting the features of the images we used query-key value pairs to find the attention value that each image had to give. We employ a residual connection around the the attention layer followed by a layer normalization as explained in *Attention is all you need* [35]. Each attention score was computed with the following equation:

$$\text{Attention}\,(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{6}$$

Where Q, K and V are query, key and value matrices respectively and d_ks the dimension of the key. These matrices are computed by multiplying the features of each image by three sets of trainable parameters of the network. Following the path of the aforementioned publication, we used a Multi-Head Attention which consists of having multiple projections of the query, key and value. These are all concatenated into and projected once more leading to the final value obtained as depicted in equation 7.

$$\begin{aligned}\text{MultiHead}\,(Q,K,V) &= \text{Concat}(\text{head}_1,\ldots,\text{head}_h)\,W^O \\ \text{where head}_i &= \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right)\end{aligned} \tag{7}$$

The projections are done via the trainable set of parameter matrices: $W_i^Q \in \mathbb{R}^{d_{\text{madel}}} \times d_k, W_i^K \in \mathbb{R}^{d_{\text{madel}}} \times d_k, W_i^V \in \mathbb{R}^{d_{\text{madel}}} \times d_v$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{modat}}}$

In our work we employed 8 heads and a model depth of 1280 which were the number of features extracted from the convolutional section of our model.

3.5 Framework

The majority of the work done on this thesis was performed on computers. On the hardware side, there were two main workstations: my personal computer and the GPI resources. The work on my laptop was mostly done in the Visual Studio Code IDE and Jupyter Notebooks. The nature of this work focused mainly with dealing with data organization and small tweaks to the network. All the resource-heavy experiments were computed at the GPI server. This is due to limitations that my personal computer imposed. Using the UPC facilities, I was able to train multiple NN with their ability run a code for up to 24 hours as well as make small proof of concepts on their more time limited servers. All the code on these servers was written using Pycharm IDE. Most of the work of this thesis was coded using Python programming language and the Pytorch, Scikit-learn and Pandas libraries.

The servers at the UPC offered the following specs:

1. 12 GB GPUs with cuda/10.0 drivers
2. CPU with up to 15 threads
3. 15GB of memory per test
4. Up to 150GB of hard disk to store all my data

4 Results

Now that we have explained all the methods used, on this chapter we will present the experiments that were carried out alongside their results. First we will focus on single image classification using EfficientNets as well as a the classification of the image features using simpler ML algorithms.

Afterwards we show the results of the experiments that were built on top of the EfficientNet to try and exploit the contextual information. We propose three different strategies using our own contextual sampler, an expectation maximization algorithm and the use of attention.

4.1 Single image classification

In order to properly compare our models we first trained three models each one with increasing expressiveness. They only used single image classification and were not built to make us of contextual information.

4.1.1 EfficientNet Vanilla

The most basic implementation of the EfficientNet only had the next items as added features to the network:

- Pre-trained on ImageNet
- Adam optimizer [41]
- Cosine Annealing learning rate scheduler
- Cross entropy loss function

Cross entropy performs comparably better than other loss functions such as the square error which tends to give too much emphasis to incorrect outputs. It works by predicting a probability distribution $p(y=i)$ for each class $i=1,2,...,C$. The formula is as follows:

$$\text{loss}(x,\ \text{class}\ ) = -\log\left(\frac{\exp(x[\ \text{class}\ ])}{\sum_j \exp(x[j])}\right) = -x[\ \text{class}\ ] + \log\left(\sum_j \exp(x[j])\right) \tag{8}$$

Where $x[j]$ is the probability that was outputted by the neural network for each class and $x[class]$ is the ground-truth probability of the correct class. The losses are averaged across observations for each mini-batch.

We also used a two phase approach during the training phase:

1. Train only the final classifier for a few iterations in order to have better than random initial values
2. Unfreeze all the network

The accuracy and the loss obtained during the training are shown in figure 15. In it we see the loss function and the accuracy plateauing. The 50% balanced accuracy means that the classifier is not able to generalize from the training data.

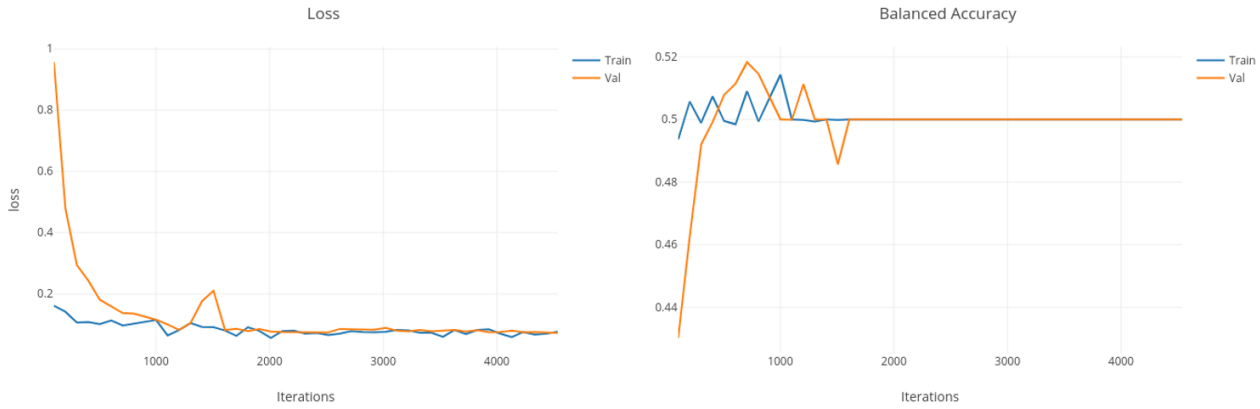

Figure 15: On the left we can see the loss of the network during training. On the right we can find the balanced accuracy.

We computed the confusion matrices in figure 16 and we can observe that the network has learned to always classify its inputs as the benign class. This is due to the network scarcely seeing images with the malign label.

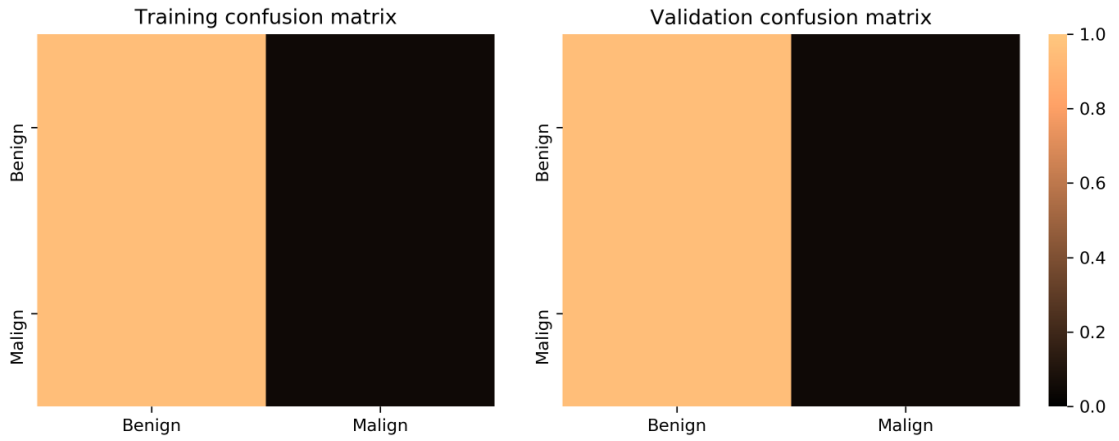

Figure 16: Confusion matrices of the training (left) and validation (right) splits of the vainilla architecture.

4.1.2 EfficientNet with weighted cross entropy

The cross entropy function allows to add a weight to each class with which to ponder the loss. By adding higher weights to a certain class, we are incentivizing the network to focus its efforts into reducing the error for that class. Mathematically, the cross entropy loss equation 8 becomes:

$$\text{loss}(x,\ \text{class}\ ) = \text{weight}[\ \text{class}\ ]\left(-x[\ \text{class}\ ] + \log\left(\sum_j \exp(x[j])\right)\right) \tag{9}$$

With this technique the performance of the network improved considerably as seen in figure 17. The training and the accuracy gain is still pretty noisy but the balanced accuracy went from 50% up to 80%. The number of iterations trained compared to the previous experiment was six-fold. Even with the extra time did not present signs of being able to improve further.

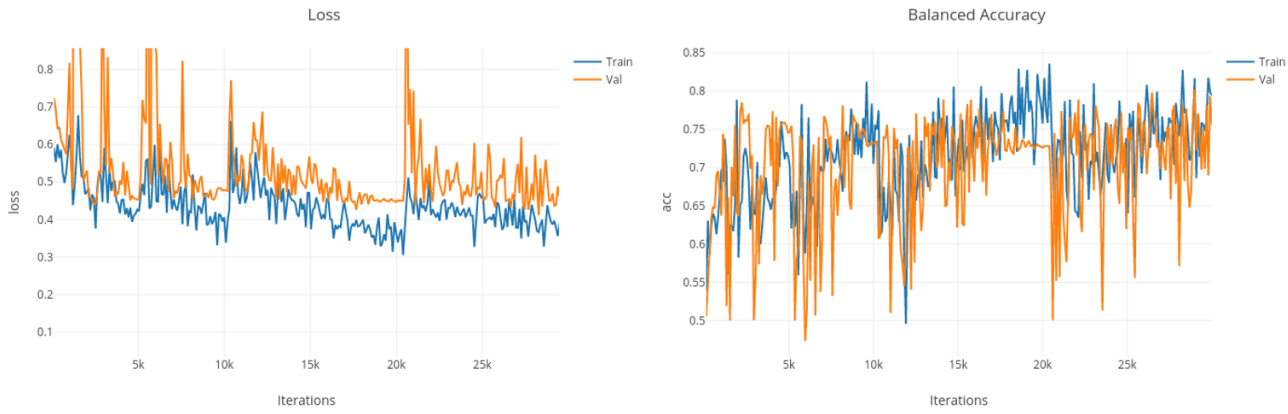

Figure 17: Loss and accuracy plots over number of iterations using a weighted cross entropy loss.

4.1.3 Weighted Random Sampler

As explained in the previous chapter, the Weighted Random Sampler helped to achieve a uniform distribution of both classes on every batch. As the weighted cross entropy helped reach a higher accuracy score so did the weighted random sampler.

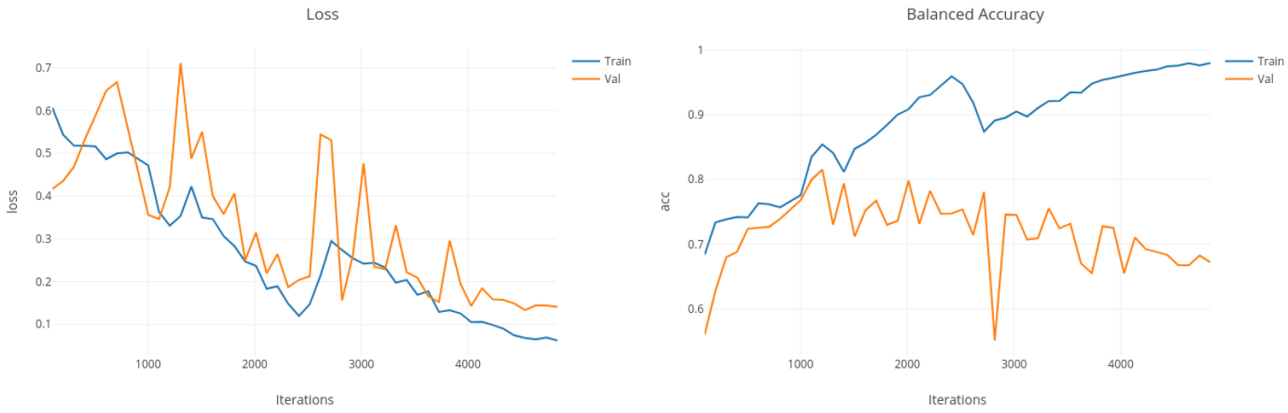

Figure 18: Loss and accuracy plots using over number of iterations using a weighted random random sampler.

Not only did the network achieve a similar result as the weighted cross entropy but it did so much quicker. With only about 1,3K iterations, the network started to overfit on the training

data. At about 5000 iterations the network had almost memorized the dataset. This improvement stems from the fact that the network was seeing the same amount of images from each class on each forward pass. Thus it gained the ability to learn much faster from our data.

Taken a look once more into the confusion matrix of this last experiment we can see that the network is struggling with the malign class. Most benign images were correctly classified but more than half of the melanomas were given the wrong label.

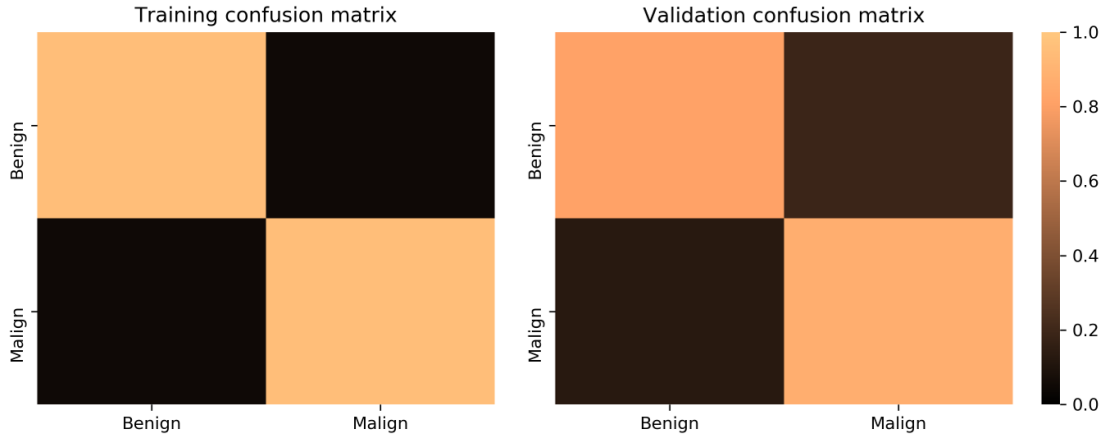

Figure 19: Confusion matrices of the training (left) and validation (right) splits. Network trained using a Weighted Random Sampler.

4.1.4 Use of Random Forest algorithms with the extracted features

After training the CNN and then the CAE we used both models to convert the dataset from images to features. We wanted to see how a traditional Machine Learning (ML) algorithm would perform with the different embeddings. We decided to use an Extra Tree classifier: This algorithm implements a meta estimator that fits a number of randomized decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control overfitting. We used the features of the CNN trained with the Weighted Random Sampler. The result is shown in table 3:

Model features	Roc AUC score	Precision	Recall
CAE	90.2%	67.3%	49.7%
EffNet-WRS	65.6%	41.4%	2.8%

Table 3: ExtraTree algorithm performance given CAE and CNN extracted features.

4.2 Using contextual information

For the next experiments we re-used the weights of the best model, trained with the weighted random sampler, in order to speed up the training of the following architectures:

We tried two main approaches, one through the use of the multi-headed attention explained in the previous chapter. The other approach would compute the mean of the features from the same patient and concatenate the result to the image feature vector before classification.

Attention should answer the following question: Which input, except from the image we are currently classifying, should we focus on? The multiheaded attention is trained to focalize on the images that help to reduce the loss function. Once again we are feeding the network with images from the same patient and allow it to use these images to improve its capacity. We tried two main experiments with this concept. First we tried the contextual sampler of our own devise and then we also tried an expectation maximization algorithm in which all the images features of a patient were fed to the attention layers.

4.2.1 Contextual Sampler

With this approach we used the sampling technique explained in the previous chapter. Our CNN first extracted features from the images and we then used these features as input to a multi-headed attention layer. It is important to keep in mind that every batch of images had the shape:

$$[Total\ patients, Images\ per\ patient, Colour\ channels, Width, Height]$$

The features of every image were extracted independently from one another, it was only after that we fed the subset of images from each patient to the attention layers. After the non-linearities of the inner fully connected layers within the multi-headed attention we end up with an attention vector that we then sum to the residual of the last layer. By doing so we are adding a percentage of how much of each of the other images in the batch should we look at to make the correct classification.

It is worth noting that with this approach we are not passing all the images of a patient at every pass but rather a fixed amount for every patient. As discussed in Chapter 3, the patients that had a malign image, have class balanced batches. The reason why we did not input every image per patient is due to limitations in the amount of GPU.

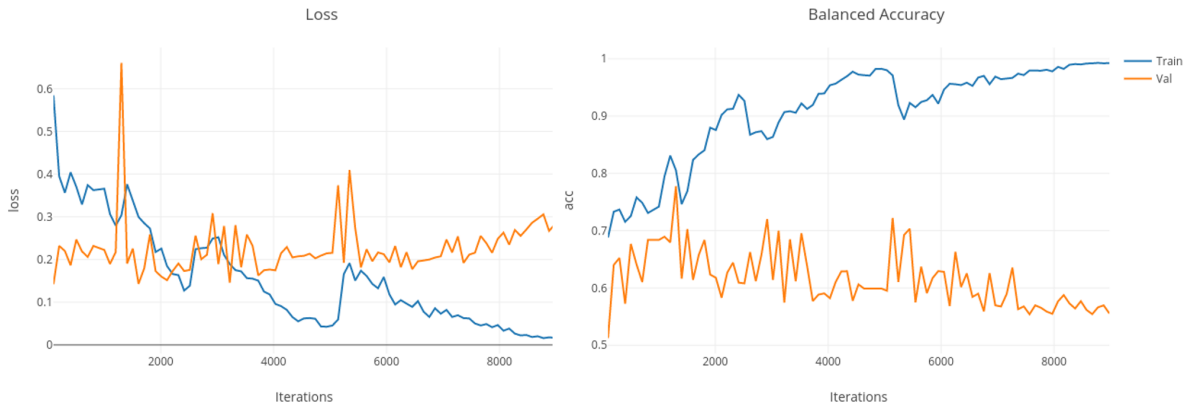

Figure 20: Loss and accuracy plots over number of iterations using our contextual sampler.

As seen in figure 20 the contextual sampler did not help to improve the network's capacity. Our

intuition tells us that even thought we were balancing patients and classes, not having equal representation of each class during training may have worsen the total performance.

4.2.2 Context through patient attention

As we wanted to check how the attention model would perform if we fed all the images of a patient at the same time we needed to do something different. A way to bypass the limitations in GPU memory was to pre-compute the features of the images beforehand. By doing this we would follow an expectation maximization algorithm, swapping between updating the network to extract better features and also using those features for the attention layers.

The main idea was to train the network for a set amount of iterations and then make the feature extraction. The extracted features of all the images (except the one being fed to the network) would be given to the attention layer and then added to the one image being classified (repeated for every image in the batch). The complete algorithm 1 for expectation maximization can be found below:

Algorithm 1 Expectation maximization algorithm

procedure EXTRACT-FEATURES($model, dataloader$) ▷ Dataloader contains the features
  for $images$ in $dataloader$ **do**
    $features \leftarrow model.extract\_features(images)$
    $features_{list} + \leftarrow features$
  $dataloader_{features} \leftarrow features_{list}$
procedure TRAIN-MODEL($model, optimizer, dataloader$)
  ▷ This whole procedure is repeated a set number of iterations

  $dataloader_{train} \leftarrow dataloader['train']$
  $dataloader_{validation} \leftarrow dataloader['validation']$
  $iteration \leftarrow 0$
  for $images, labels, features$ in $dataloader_{train}$ **do**
    $predictions_{train} \leftarrow train\_iteration(images, labels, features, optimizer)$
    $predictions_{total} + \leftarrow predictions_{train}$
    $iteration + \leftarrow 1$
    if $iteration > validation_{threshold}$ **then**
      for $images, labels, features$ in $dataloader_{validation}$ **do**
        $predictions_{validation} \leftarrow validation\_iteration(images, labels, features)$
        $predictions_{total} + \leftarrow predictions_{validation}$
      $accuracy_{train}, accuracy_{validation} \leftarrow compute\_accuracy(predictions_{total})$
      $extract\_features(model, dataloader)$ ▷ Updates image features
      $iterations \leftarrow 0$

the results can be seen in the figure 21. From it we cannot see consistent improvement, there is a peak right at the beginning of the training but the architecture is not able to learn further.

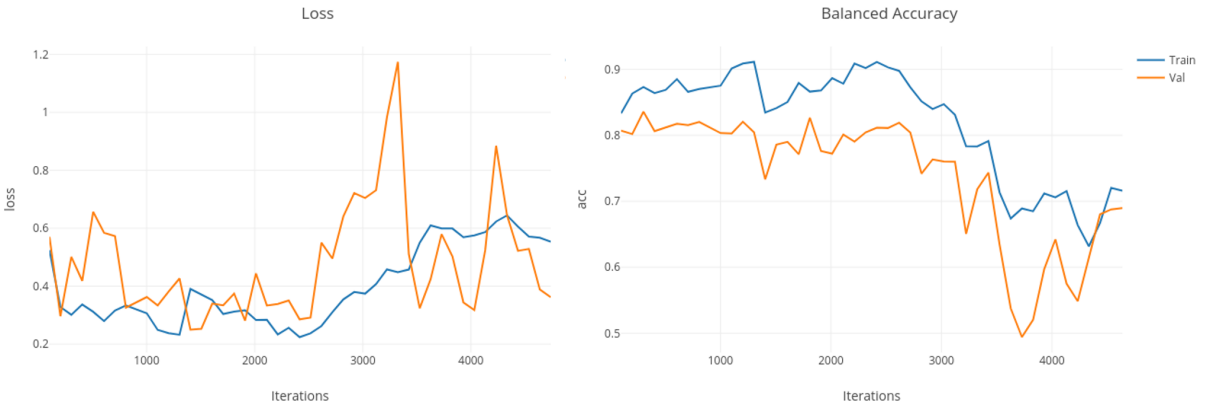

Figure 21: Loss and accuracy plots over number of iterations using an attention architecture.

4.2.3 Static attention cheat

In order to ensure that our implementation of the attention algorithm was correctly implemented, we design an experiment in which we would give an extra feature vector to all the patients that had malign images. This feature vector consisted of maxed values which helped the network achieve extremely high values of accuracy in the validation set. The accuracy curve of this experiment can be seen in figure 23.

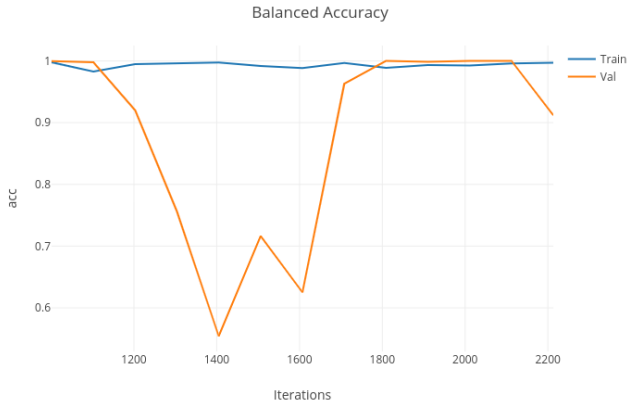

Figure 22: Accuracy of the network over iterations. The network shown uses an extra feature vector for the patients with malign dermatoscopy.

Exploring the attention weights of the network, when classifying melanoma images, our model had learned to look for the synthetic feature vector which acted as a red flag. This experiment confirmed that the attention was indeed working correctly but it is obvious that it is not useful for actual classification.

4.2.4 Using patient embedding

The last way of tackling contextual information that we tried was the computation of patient embeddings by using the mean of the image features that the network extracted. As explained

in the last chapter, the mean uses all the image features of a patient, making use of the algorithm 1 in order to bypass GPU limitations. This extra information is concatenated to the features vectors before the classification layer.

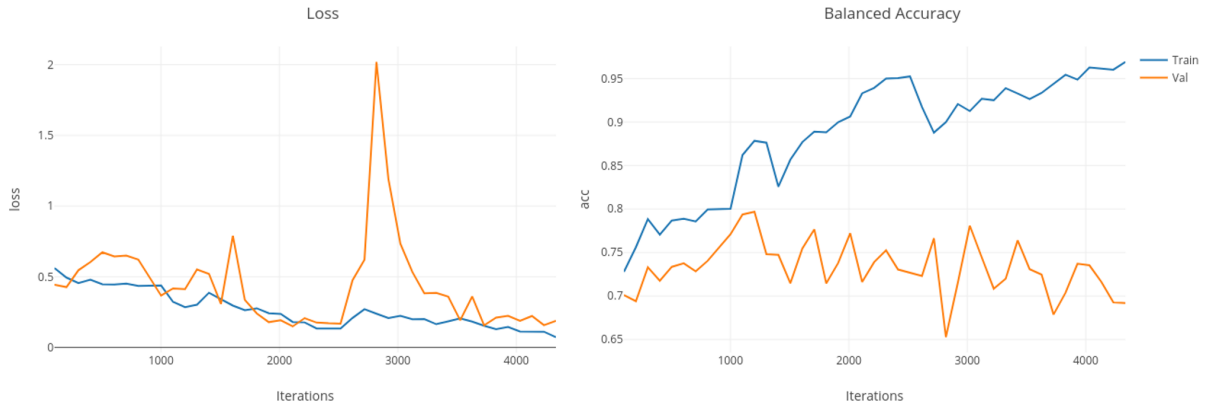

Figure 23: Loss and accuracy plots over number of iterations using patient embedding architecture.

As it happened with the contextual sampler, the network sees a spike at the beginning of the training but it quickly starts to overfit on the data.

4.3 Final Comparison

After we had all the models trained we forwarded our dataset through them to extract their predictions. By doing this we would have a fair way to compare their results apart from the plots shown on last subsections. The data was the combination of the training data plus the validation data. The only architecture that saw all the data from the beginning was the Convolutional AutoEncoder.

We computed three metrics: ROC AUC Score, precision and recall. ROC AUC Score in our binary case serves as a better substitute for accuracy because it evaluates the outputted probabilities of the network, instead of binary labels. The result can be found on table 4.

Model	Roc AUC score	Precision	Recall
EffNet - Weighted Cross Entropy	88.8%	6.3%	84.9%
EffNet - Weighted Random Sampler	91.5%	7.6%	87.2%
CAE features + ExtraTree classifier	90.2%	**67.3**%	49.7%
EffNet - Contextual Sampler	87.9%	9.3%	86.9%
EffNet - Attention + expectation maximization	92.2%	8.2%	88.2%
EffNet - Static Mean + expectation maximization	**92.7**%	7.7%	**89.9**%

Table 4: Model comparison for three different metrics.

From the table we can see that the use of static context and the use of attention only helped to improve the performance of the network slightly. The order in which the architectures are

explain in this section follows the order in which they are presented on the table with the exception of the simplest EfficientNet training. It is worth noting that the CAE, which was trained in an unsupervised manner, performed rather well. Although there is a caveat, when tested against pictures that had not being fed during training, its performance dropped considerably.

5 Discussion

Throughout this work an analysis on the potential of Deep Learning applied to the field of dermatoscopic imagining has been carried out. On top of that there was an extra condition of trying to achieve a better performance than previously published work [7] [17] by exploiting the contextual information of the dataset. There were certain restrictions that we applied to our models in order to reduce the amount of pre-processing performed on the dataset, such as not using image segmentation. The aim of the project was to improve the model's accuracy when dealing with a binary classification problem. Despite intensive investigations into data representations including various approaches to obtain patient embeddings and or implementing attention to the networks, we could not find clear results that proved our hypothesis true. During the development of this thesis we believed that there was improvement to be gained by using said contextual information in the context of dermatoscopic images. After proving empirically that this was not the case we decided to run a small interpretability study using t-SNE projections of our extracted features. This study does not pretend to give a complete explanation as to why our approach did not work but rather a visual intuition as to why our experiments did not perform as expected.

5.1 t-SNE study

T-distributed Stochastic Neighbor Embedding is a machine learning algorithm for visualization developed by Laurens van der Maaten and Geoffrey Hinton. It is a nonlinear dimensionality reduction technique well-suited for embedding high-dimensional data for visualization in a low-dimensional space The work done with the t-SNE algorithm had as its input the extracted features from the CAE and also from the CNN. We first explain how we computed the projection presented in figures 24, 26 and 25 and then discuss the results.

5.1.1 Extracted features

The first thing we needed was to reduce the dimensionality of the extracted features so we could visualize it on a 2D graph. Below, in figure 24, we can see the 2D projection of the features extracted by the CAE. As it is common practice to use PCA on your data first before feeding it to t-SNE.

Two major clusters are formed, the malign image's feature have less variance than their benign counterpart and they are clustered together on the top right of the plot. Although some of them do not adhere to this rule and we believe that those are the ones dragging the performance of the classifier.

A similar feature space appeared when we used the features extracted from the CNN instead of the CAE as seen in figure 25. In the case of the CNN the clusters boundaries are not as clear as they were with the AutoEncoder. This might be due to the CNN working with a much higher number of parameters which implies higher dimensionality features which in exchange means that PCA and t-SNE will have a tougher problem when making data dimensionality reduction.

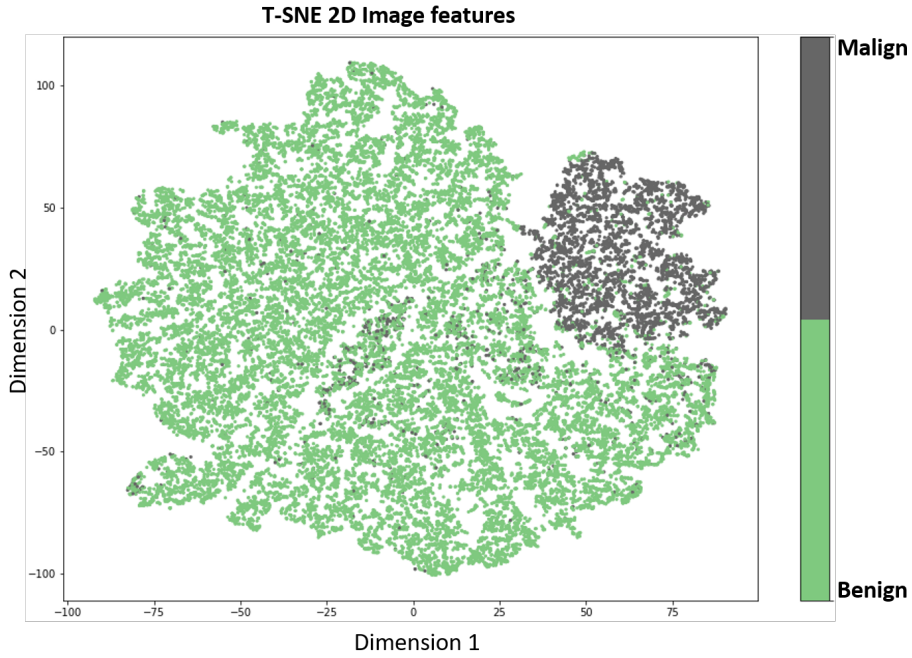

Figure 24: 2D image features extracted with the CAE.

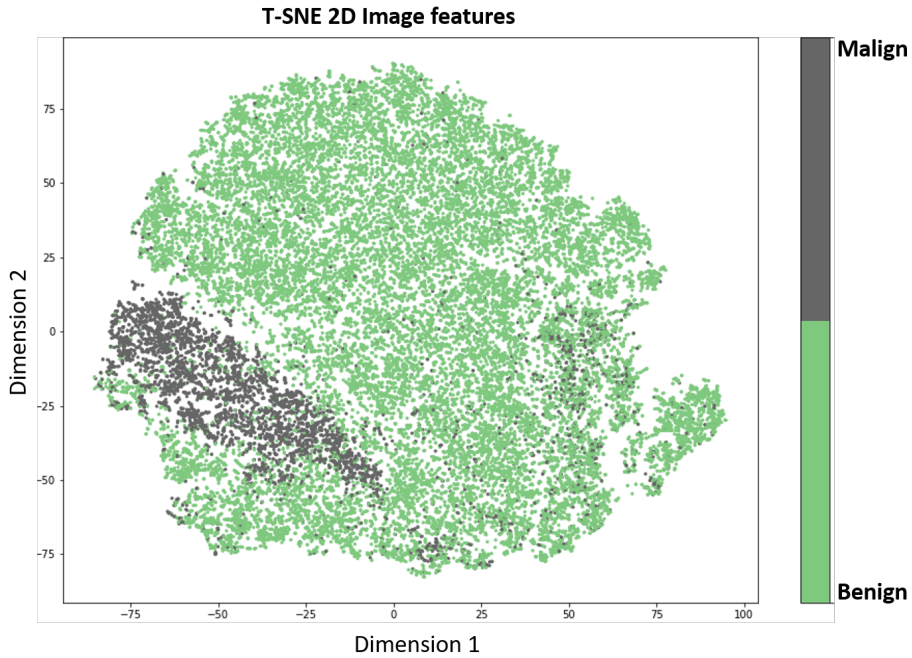

Figure 25: 2D image features extracted with the CNN.

5.1.2 Feature space of the patient embeddings

The difference between classes on graphs 24 and 25 are clear to the naked eye. Now that we could see the classes, we computed the mean of each patient's image with benign label. If

a patient's malign images were somehow correlated with their benign dermatoscopies, said correlation should show in the graph. We computed the following three steps per patient:

- Put aside the images with the malign label
- Group the other dermatoscopies features
- Compute the mean

After we had converted from images to patient embeddings we applied dimensionality reductions with PCA and t-SNE once more. The result in figure 26 shed some light as to why the contextual approach was not improving the performance of the model

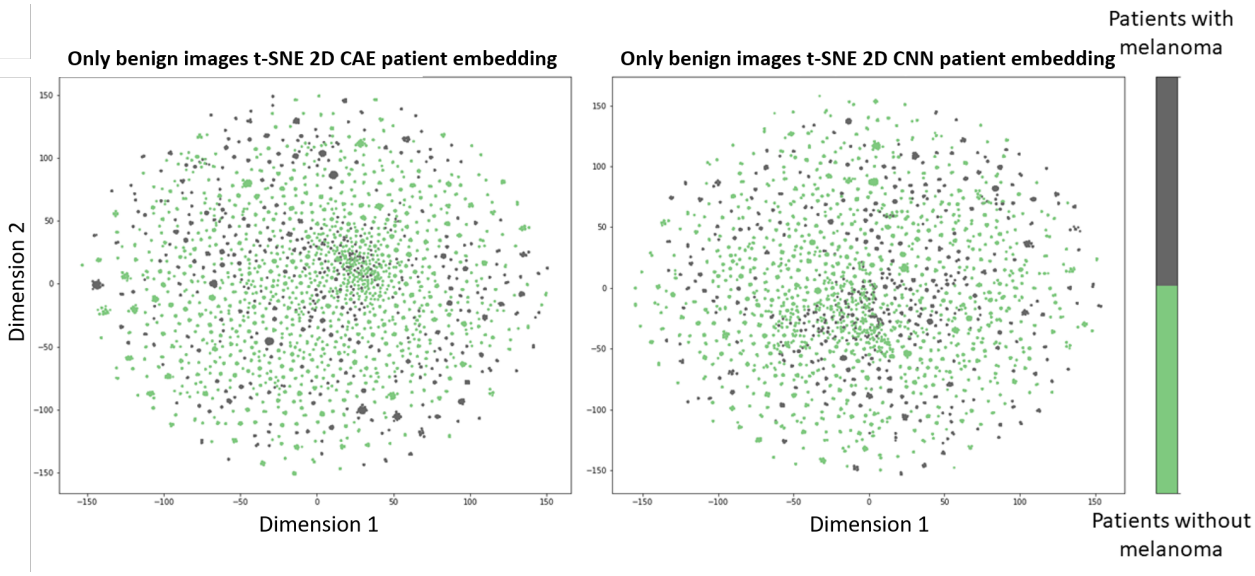

Figure 26: 2D patient embeddings of the CAE (left) and the CNN (right).

The graph 26 shows an almost uniform cloud of points with higher density at the center. This representation that does not provide any clear distinction between the patient embeddings of patients with malign images compared to those which do not have them.

6 Conclusion and Future Work

The results of this work provide positive results in the use of contextual patient information for the classification of dermatoscopic images. Even though the performance increased they did not reach our expectations. We have also proposed a new sampling paradigm which builds on top of the already implemented Weighted Random Sampler to provide balanced batches with additional conditions which we named Contextual Sampler. This novel sampling technique did not perform better than its predecessor but it may proof useful when dealing with a different type of dataset. On this thesis we have successfully implemented multi-headed attention even thought it does not seem to yield substantial improvement. The best proven method has been a balancing on the classes while training the neural network. A small interpretability study has also been rendered to give an explanation unto why we found this lack of substantial improvement.

All in all there are certain aspects that could be explored in order to shed some light on the interpretability of this project as well as, hopefully, improve its performance:

- Explore the use of metadata, adding it to the feature space before and after the attention layer to see how the model behaves. If added before the attention layer the model might be able to find some correlation between features that could not be found just by using image features.
- Bigger interpretability study in order to provide a more extensive explanation as to why our intuition failed to meet our expectations.
- As discussed in Chapter 2 segmentation is a really useful tool for melanoma classification. The implementation of this technique could help the classification performance via eliminating sections of the image which do not add any additional information and could detriment the performance of the model by finding spurious correlations hidden amongst the data.
- Divide the image into different sections. After extracting the features we could apply patch attention in order for the network to give an attention to specific parts of those images. This approach is depicted in the figure 27.

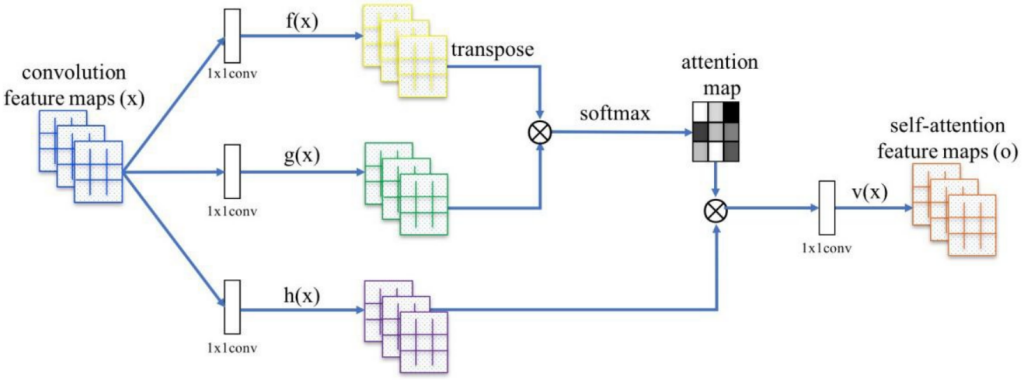

Figure 27: Multi headed attention using image patches instead of whole images.

References

[1] Barata, Catarina, M. Emre Celebi, and Jorge S. Marques. ”Melanoma detection algorithm based on feature fusion.” 2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC). IEEE, 2015.

[2] M. Akdeniz, E. Hahnel, C. Ulrich, U. Blume-Peytavi, y J. Kottner, «Prevalence and associated factors of skin cancer in aged nursing home residents: A multicenter prevalence study», PLOS ONE, vol. 14, n.o 4, p. e0215379, abr. 2019, doi: 10.1371/journal.pone.0215379.

[3] F. Bray, J. Ferlay, I. Soerjomataram, R. L. Siegel, L. A. Torre, y A. Jemal, «Global cancer statistics 2018: GLOBOCAN estimates of incidence and mortality worldwide for 36 cancers in 185 countries», CA. Cancer J. Clin., vol. 68, n.o 6, pp. 394-424, 2018, doi: 10.3322/caac.21492.

[4] R. Z. Conic, C. I. Cabrera, A. A. Khorana, y B. R. Gastman, «Determination of the impact of melanoma surgical timing on survival using the National Cancer Database», J. Am. Acad. Dermatol., vol. 78, n.o 1, pp. 40–46, 2018.

[5] A. Conde Taboada et al., «Distribución geográfica de dermatólogos y plazas MIR de dermatología en España», Piel Form. Contin. En Dermatol., vol. 18, n.o 9, pp. 477-480, nov. 2003, doi: 10.1016/S0213-9251(03)72761-0.

[6] A. J. Petty et al., «Meta-analysis of number needed to treat for diagnosis of melanoma by clinical setting», J. Am. Acad. Dermatol., vol. 82, n.o 5, pp. 1158-1165, may 2020, doi: 10.1016/j.jaad.2019.12.063.

[7] A. Esteva et al., «Dermatologist-level classification of skin cancer with deep neural networks», Nature, vol. 542, n.o 7639, pp. 115–118, 2017.

[8] «Man against machine reloaded: performance of a market-approved convolutional neural network in classifying a broad spectrum of skin lesions in comparison with 96 dermatologists working under less artificial conditions - Annals of Oncology». https://www.annalsofoncology.org/article/S0923-7534(19)35468-7/fulltext

[9] P. Tschandl et al., «Comparison of the accuracy of human readers versus machine-learning algorithms for pigmented skin lesion classification: an open, web-based, international, diagnostic study», Lancet Oncol., vol. 20, n.o 7, pp. 938–947, 2019.

[10] Noel Codella, Veronica Rotemberg, Philipp Tschandl, M. Emre Celebi, Stephen Dusza, David Gutman, Brian Helba, Aadi Kalloo, Konstantinos Liopyris, Michael Marchetti, Harald Kittler, Allan Halpern: “Skin Lesion Analysis Toward Melanoma Detection 2018: A Challenge Hosted by the International Skin Imaging Collaboration (ISIC)”, 2018; https://arxiv.org/abs/1902.03368

[11] Skin Lesion Classification Using Loss Balancingand Ensembles of Multi-Resolution EfficientNets

[12] Nasr-Esfahani, Ebrahim, et al. ”Melanoma detection by analysis of clinical images using convolutional neural network.” 2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC). IEEE, 2016.

[13] Suk, Heung-Il, and Dinggang Shen, “Deep learning-based feature representation for AD/MCI classification,”Medical Image Computing and Computer-Assisted Intervention–MICCAI 2013. Springer BerlinHeidelberg, 2013. 583-590.

[14] Cai, Jinzheng, et al. ”Improving deep pancreas segmentation in CT and MRI images via recurrent neural contextual learning and direct loss function.” arXiv preprint arXiv:1707.04912 (2017).

[15] Li, Yuexiang, and Linlin Shen. ”Skin lesion analysis towards melanoma detection using deep learning network.” Sensors 18.2 (2018): 556.

[16] Hu, Jie, Li Shen, and Gang Sun. ”Squeeze-and-excitation networks.” Proceedings of the IEEE conference on computer vision and pattern recognition. 2018.

[17] Tschandl, P., Codella, N., Akay, B.N., Argenziano, G., Braun, R.P., Cabo, H.,Gutman, D., Halpern, A., Helba, B., Hofmann-Wellenhof, R., et al. (2019) Com-parison of the accuracy of human readers versus machine-learning algorithms forpigmented skin lesion classification: an open, web-based, international, diagnosticstudy. The Lancet Oncology

[18] PDQ Adult Treatment Editorial Board. PDQ Melanoma Treatment. Bethesda, MD: National Cancer Institute. Updated 04/11/2019 Available at: https://www.cancer.gov/types/skin/hp/melanoma-treatment-pdq.

[19] Curtin JA, Fridlyand J, Kageshita T, Patel HN, Busam KJ,Kutzner H, et al. Distinct sets of genetic alterations in melanoma.N Engl J Med 2005;353(20):2135e47

[20] Abbas, Qaisar, M. Emre Celebi, Carmen Serrano, Irene FondoN GarciA, and Guangzhi Ma. ”Pattern classification of dermoscopy images: A perceptually uniform model.” Pattern Recognition 46, no. 1 (2013): 86-97

[21] LeCun, Yann, et al. ”Gradient-based learning applied to document recognition.” Proceedings of the IEEE 86.11 (1998): 2278-2324.

[22] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[23] Tan, Mingxing, and Quoc V. Le. ”Efficientnet: Rethinking model scaling for convolutional neural networks.” arXiv preprint arXiv:1905.11946 (2019).

[24] D. Bisla, A. Choromanska, R. S. Berman, J. A. Stein,and D. Polsky. Towards automated melanoma detec-tion with deep learning: Data purification and aug-mentation. InIEEE Conference on Computer Visionand Pattern Recognition Workshops (CVPRW), 2019.

[25] Wang, Jason, and Luis Perez. ”The effectiveness of data augmentation in image classification using deep learning.” Convolutional Neural Networks Vis. Recognit 11 (2017).

[26] Shorten, Connor, and Taghi M. Khoshgoftaar. ”A survey on image data augmentation for deep learning.” Journal of Big Data 6.1 (2019): 60.

[27] Mikołajczyk, Agnieszka, and Michał Grochowski. ”Data augmentation for improving deep learning in image classification problem.” 2018 international interdisciplinary PhD workshop (IIPhDW). IEEE, 2018.

[28] Okabe, Koji, Takafumi Koshinaka, and Koichi Shinoda. ”Attentive statistics pooling for deep speaker embedding.” arXiv preprint arXiv:1803.10963 (2018).

[29] Gessert, N., Sentker, T., Madesta, F., Schmitz, R., Kniep, H., Baltruschat, I.,Werner, R., Schlaefer, A. (2018) Skin lesion diagnosis using ensembles, unscaledmulti-crop evaluation and loss weighting. arXiv preprint arXiv:1808.01694

[30] Russakovsky O, et al. (2015) Imagenet large scale visual recognition challenge.Internationaljournal of computer vision115(3):211–252.

[31] Lin T Y, Maire M, Belongie S, et al. Microsoft coco: Common objects in context[C]//European conference on computer vision. Springer, Cham, 2014: 740-755

[32] Fei-Fei Li, Justin Johnson, and Serena Yeung. Cs231n: Convolutional neural networks for visual recognition.2017. URL http://cs231n.stanford.edu/.

[33] InProceedingsBissoto 2020 CVPR Workshops: Bissoto, Alceu and Valle, Eduardo and Avila, Sandra ”Debiasing Skin Lesion Datasets and Models? Not So Fast”, The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops June 2020

[34] Ando, Shin, and Chun Yuan Huang. ”Deep over-sampling framework for classifying imbalanced data.” Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Springer, Cham, 2017.

[35] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł. and Polosukhin, I., 2017. Attention is all you need. In Advances in neural information processing systems (pp. 5998-6008).

[36] Xiao, Cao, et al. ”Readmission prediction via deep contextual embedding of clinical concepts.” PloS one 13.4 (2018): e0195024.

[37] Wang, Hongyu, et al. ”Breast mass classification via deeply integrating the contextual information from multi-view data.” Pattern Recognition 80 (2018): 42-52.

[38] D.Ciresan,U.Meier,and J.Schmidhuber,“Multi-column deep neural networks for image classification,”in IEEE Conference onComputer Vision and Pattern Recognition (CVPR), 2012, 2012, pp. 3642-3649

[39] Finlayson, G.D. and Trezzi, E., 2004, January. Shades of gray and colour constancy. In Color and Imaging Conference (Vol. 2004, No. 1, pp. 37-41). Society for Imaging Science and Technology.

[40] GitHub repository for the user Foamliu and his AutoEncoder: https://github.com/foamliu/Autoencoder

[41] Kingma, Diederik P., and Jimmy Ba. ”Adam: A method for stochastic optimization.” arXiv preprint arXiv:1412.6980 (2014).

[42] Brown, Tom B., et al. ”Language models are few-shot learners.” arXiv preprint arXiv:2005.14165 (2020).

Appendices

A Contextual sampler

This first appendix shows the Pytorch code used to create the contextual sampler. It recieves a dataframe with the information of the dataset and the batch size.

```
class ContextualSampler(Sampler):
    r"""
    Custom sampler that outputs balanced batches from each patient.
    """
    def __init__(self, df, replacement=True, batch_size=4, val=False):
        '''
        df = Dataframe containing the dataset to be used.
        replacement = With replacement=True, each sample can be picked
                                            in each draw again.
        batch_size = int which chooses the size the size of the batch
        val = If validating the dataset will take all the images of a
                                            patient.
        '''

        self.replacement = replacement
        self.df = df
        self.batch_size = batch_size

        def has_mel(df):
            has_mel = 1 in df['label'].unique()
            row = df.iloc[0]
            row['has_mel'] = int(has_mel)
            del row['label'], row['filename']
            return row

        self.pat = df.groupby('pat_id').apply(has_mel).reset_index(drop=
                                            True)

        self.val = val

        if not isinstance(self.num_samples, _int_classes) or isinstance(
                                            self.num_samples, bool) or \
                self.num_samples <= 0:
            raise ValueError("num_samples should be a positive integer "
                             "value, but got num_samples={}".format(self
                                                                    .
                                                                    num_samples
                                                                    ))


        if not isinstance(self.replacement, bool):
            raise ValueError("replacement should be a boolean value, but
                                                got "
                             "replacement={}".format(self.replacement))
```

```
    def __iter__(self):
        pat_weights, _ = make_weights_for_balanced_classes(self.pat['
                                                has_mel'])
        pat_weights = torch.FloatTensor(pat_weights)

        pat_ids = torch.multinomial(pat_weights, len(pat_weights), self.
                                                replacement).tolist()
        sampled_pat = self.pat.iloc[pat_ids].copy()

        if self.val:
            sampled_pat = self.pat.copy()

        counter = 0
        ret_lesions = []
        for pat_id in sampled_pat['pat_id']:
            lesions = self.df[self.df['pat_id'] == pat_id].copy()

            les_weights, _ = make_weights_for_balanced_classes(lesions['
                                                    label'])
            les_weights = torch.FloatTensor(les_weights)

            les_ids = torch.multinomial(les_weights, 1, self.replacement
                                                    ).tolist()

            sampled_lesions = lesions.iloc[les_ids].copy()

            if self.val:
                sampled_lesions = lesions

            return_lesions = sampled_lesions.index
            ret_lesions += return_lesions.tolist()
            counter += 1

            # Patients per batch
            if counter >= 8:
                yield ret_lesions
                counter = 0
                ret_lesions = []

    def __len__(self):
        return len(self.df)
```

B Auto embedding effnet

This appendix shows the Pytorch class that we created to make an autoembedding

```
class autoEmbeddingEff(nn.Module):
    def __init__(self, with_feat=False):
        super(autoEmbeddingEff, self).__init__()
        self.eff_net = EfficientNet.from_pretrained('efficientnet-b0',
                                                    num_classes=2)
        self.eff_net._fc = nn.Identity()
        self.with_feat = with_feat

        if with_feat == True:
            self._fc = nn.Linear(1280 * 2, 2)
        else:
            self._fc = nn.Linear(1280, 2)

    def forward(self, x):
        # Stacks a tensor upon itself
        def tile(tensor, dim, n):
            if dim == 0:
                return tensor.unsqueeze(0).transpose(0, 1).repeat(1, n,
                                                    1).view(-1, tensor.
                                                    shape[1])
            else:
                return tensor.unsqueeze(0).transpose(0, 1).repeat(1, 1,
                                                    n).view(tensor.shape
                                                    [0], 1)
        # Extract features
        x = self.eff_net(x)

        if self.with_feat == True:
            # Computes the mean of the tensor
            # Outputs [1,1280]
            pat_feat = torch.mean(x, dim=0, keepdim=True)
            # Outputs [num_images ,1280]
            pat_tiled = tile(pat_feat, 0, x.size(0))
            x = torch.cat((x, pat_tiled), dim=1)
        x = self._fc(x)
        return x
```

C Attention network

This appendix shows the Pytorch class that was created to make use of the attention layers provided in the multi-headed attention layer when using our contextual sampler. After extracting the features the tensor is then permuted to fit the attention layer's dimensions. Afterwards a residual connection is performed between the extracted features and the attention layer's output.

```
class TransformerAttentionEff(nn.Module):
    def __init__(self, with_feat=False, model_path = None):
        super(BettertransformerAttentionEff, self).__init__()
        self.eff_net = EfficientNet.from_pretrained('efficientnet-b0',
                                            num_classes=2)

        checkpoint = torch.load(model_path, map_location='cpu')
        self.eff_net.load_state_dict(checkpoint)

        self._fc = self.eff_net._fc

        self.eff_net._fc = nn.Identity()
        self.norm1 = nn.LayerNorm(1280)

        self.hydraAttention = nn.MultiheadAttention(embed_dim=1280,
                                            num_heads=8, dropout=0)

    def forward(self, x, train=False):

        patients = x.size(0)
        images = x.size(1)

        x = x.view(patients*images, 3, 224, 224)
        features = self.eff_net(x) # -> [num_images,1280]
        features_att = features.view(patients, images, 1280).permute(1,
                                            0, 2)

        features_att_out, att_weights = self.hydraAttention(features_att
                                            , features_att, features_att
                                            )


        if train:
            print(att_weights.cpu().detach().numpy())

        features_att.view(num_pat, num_im_pat, -1)
        features_att = features_att + features_att_out
        features_att = self.norm1(features_att)

        features_att = features_att.permute(1, 0, 2)

        x = self._fc(features_att)
        x = x.view(-1, 2)

        return x, att_weights
```

D Static mean dataloader

This appendix shows one of the custom datasets used during the development of the thesis. This is the only one shown due to it having more differences with the usual structure of other datasets. It presents a function named get_pat_feat which is called every set number of iterations and its used to provide the attention layer with all the image features of every patient.

```
class ISIC_Dataset_PatFeat(Dataset):
    def __init__(self, df, transform=None):

        # Initialize Dataset class from a dataframe df, and by default
                                                no transformations
        self.df = df.copy()
        self.transform = transform

    def __len__(self):

        # return length of the df as length of the dataset class
        return len(self.df)

    def get_pat_feat(self, filename):
        pat_id = self.df[self.df['filename'] == filename]['pat_id'].iloc
                                                [0]
        features = self.df[self.df['pat_id'] == pat_id]['features'].
                                                values.tolist()
        features = np.array(features)

        return features

    def __getitem__(self, idx):
        # Return the row (row has the path to the image)
        row = self.df.iloc[idx]
        # Open the image with that name
        try:
            sample = Image.open(row["filename"])
            sample = sample.convert('RGB')

            pat_feat = torch.FloatTensor(self.get_pat_feat(row['filename
                                                    ']))

            # Transform if there are any transformations to do
            if self.transform:
                sample = self.transform(sample)
        except Exception:
            import traceback
            traceback.print_exc()
            return self.__getitem__(random.randint(0, self.__len__()))

        # Return the sample image and its label, if there was any more
                                                data it would also be
                                                returned here
        return sample, row["label"], pat_feat, row["filename"]
```

E Static mean attention

The presented class makes use of the previous appendix dataset class to use all the images of a patient when using an attention layer.

```
class EfficientNetPatAtt(nn.Module):

    def __init__(self, model_path=None):
        super(EfficientNetPatAtt, self).__init__()
        self.model = EfficientNet.from_pretrained('efficientnet-b0',
                                                  num_classes=2)

        # If a model is given it will load its weights first
        checkpoint = torch.load(model_path, map_location='cpu')
        self.model.load_state_dict(checkpoint)

        self._fc = self.model._fc

        self.model._fc = nn.Identity()
        self.norm1 = nn.LayerNorm(1280)
        self.hydraAttention = nn.MultiheadAttention(embed_dim=1280,
                                                  num_heads=4, dropout=0.6)

    # Method called to extract features for the expectation
    # maximization algorithm
    def extract_features(self, x_step):
        x_step = self.model(x_step)
        return x_step

    def forward(self, x_step, pat_feat, attention):
        extracted_features = self.model(x_step)

        if attention:
            res = []
            res_att = []
            for aux_x, aux_feature in zip(extracted_features, pat_feat):
                feat = aux_feature.cuda()
                feat = feat.unsqueeze(dim=1)
                aux_x = aux_x.unsqueeze(dim=0)
                aux_x = aux_x.unsqueeze(dim=0)

                features_att_out, att_weights = self.hydraAttention(
                                                    aux_x, feat, feat)
                res_att.append(att_weights.max())

                features_att = aux_x + features_att_out
                features_att = self.norm1(features_att)

                features_att_out, att_weights = self.hydraAttention(
                                                    features_att, feat,
                                                    feat)

                features_att = features_att + features_att_out
                features_att = self.norm1(features_att)
```

```
            features_att = features_att.squeeze()
            res.append(features_att)

        extracted_features = torch.stack(res)

    x = self._fc(extracted_features)

    return x
```