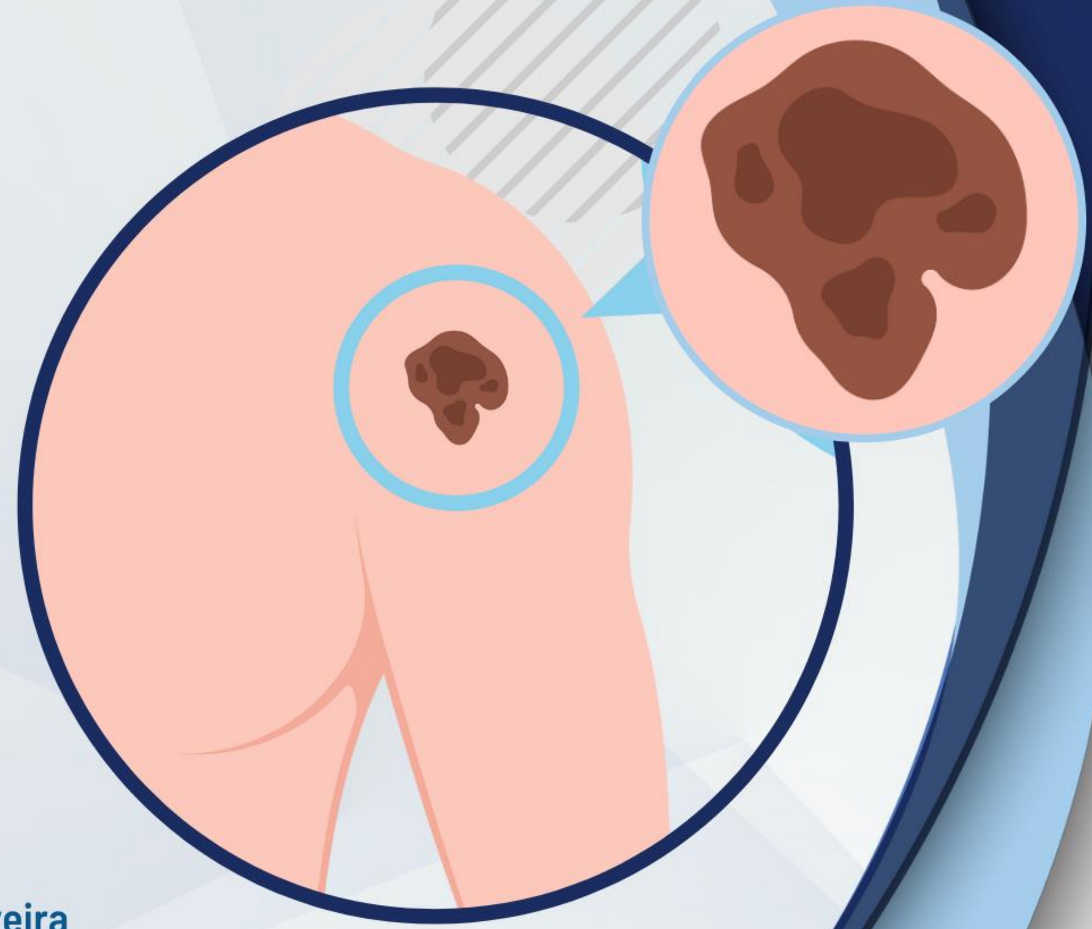
NOVA IMS Information Management School

DEEP LEARNING
DATA SCIENCE DEGREE
With the orientation of professors
Mauro Castelli and Yuriy Perezhohin

DEEP LEARNING FOR SKIN CANCER CLASSIFICATION: MODEL DEVELOPMENT AND EVALUATION

Project Report



20211595 | António Oliveira
20211628 | David Martins
20211637 | Mariana Ferreira
20211619 | Mariana Takimura
20211639 | Rui Lourenço

Table of Contents

Introduction

The aim of this project is to develop a Deep Learning Model able to correctly classify skin cancer images. We used a dataset that contains several images, where each image corresponds to a skin cancer type. We were also given metadata about each image, regarding the age and gender of the patient, as well as the location of the cancer.

Our goal is to use this information to accurately predict which cancer type is displayed in an image.

Methodology

Since we were given the data, we started by taking a closer look at some of the images, while trying to find similarities between them, which could make their pre-processing easier. To better understand which cancer images regarded each cancer types, we decided to split the data into folders based on the cancer labels. We also decided to display the tabular data, to better understand it. When doing so, we realised that the dataset contained information about both the train and test folders (which contained the images). To overcome this issue, we created a new column that identified which folder the observation belonged to and split the dataset into train and test.

Exploratory Data Analysis

With this issue dealt with, our next step was to perform Exploratory Data Analysis. We used methods like *describe* and *info* to explore our data. Then, we decided to display the unique values of each column, which showed us, for example, the name of the cancer locations present in our dataset. Then we took a similar approach but using the method *value_counts*. This method allowed us to conclude that the Melanocytic Nevi (nv) class had considerably more observations than the rest of the classes. This method also confirmed that we had 57 observations with unknown gender.

After this brief exploration, we moved on to look for missing values. Using the method *isnull* we were able to conclude that there were 57 observations with their age missing values. Before imputing them with the mean of this variable, we decided to check if these missing values corresponded to the same observations that had *unknown* values in the *sex* feature.

Having treated the missing values, we decided to check for incoherencies. As previously mentioned, there are incoherent values in the *sex* column. As we believe the observations in question may belong to people that for privacy reasons decided not to share their gender, we decided not to treat these values. We also looked for incoherent values in the variable *age*. We discovered that there were 39 observations with age equal to zero. Since the age column is not used to train the models, we decided to keep these values, as the images correspondent to these observations may be useful for model training.

This was followed by Feature Engineering. This step consisted of using *Label Encoder* to encode both the cancer label and its location, even though we later decided not to use the location variable for model training. We did this because we know that it is easier for models to deal with numeric values than with strings.

The next step was to look for outliers. We decided to plot a boxplot for the column *age*, which was the only numeric column we had. This boxplot did not show any outliers.

The final step of data exploration is the Visualisations section. Here we decided to plot the average age per sex, which showed us that males were on average a little older than females. We also plotted the average age of each person per cancer location. This graph showed us that, on average, people with cancer in the acral area are younger than, for example, people with cancer in their faces. Then we plotted a stacked bar chart that showed us the diagnosis made per location. This did not allow us to take many insights, as most diagnosis lead to the same class, independently of the cancer's location.

Classification (1)

With our tabular data treated, we decided to reproduce some simple models, just to see what their results would be like without any image pre-processing. To deal with our imbalanced dataset, we decided to use a Data Generator, which generated images for all our classes. Then we decided to use Stratified K-Fold, which is adequate for imbalanced data. We created a model, using the Model Checkpoint and Early Stopping callbacks, and made predictions. This Convolutional Neural Network (CNN) model was based on a model taken from the theoretical class slides, and used a combination of Convolution, Max Pooling and Dropout layers, where the initial number of neurons was small (32) and kept increasing until the end (256). This happens because having more units in the fully connected layers, towards the end of the network, allows the model to learn more complex relationships, which can be crucial for the final classification. This model also used a flatten layer and two dense ones. Finally, we compiled it using the *adam* optimiser and categorical crossentropy for the loss.

This model was able to predict all classes but *Dermatofibroma*, with a weighted average of 65%. It is worth noting that this class is the one with the least number of observations.

Image Preprocessing

Following this experiment, we decided to move on to what we expected would be a crucial step, the image pre-processing. As this step could have a very significant impact on the model's ability to correctly make predictions, we started by researching some commonly used methods to treat images. After implementing each pre-processing technique, models were trained, and their results were assessed.

The first image pre-processing technique we decided to use was blur. To implement this we created a function, *billiniar_blur*, that started by resizing the image and converting it from BGR to RGB format. The latter was done as the used *cv2* library required a RGB colour space to read the images. Then, we defined a circle in the centre of the image and created a circular mask that started by creating a black image and a white circle. Then, this mask was inverted, turning the white circle into black and vice-versa. Following this, a bilateral filter was applied to a copy of the original image. This filter will then blur the pixels outside the circular mask. The original image and its blurred version were then displayed by the *show_blur* function, using the *matplotlib* library. This allowed us to be sure these functions were working as expected.

Following the usage of Blur, we decided to try to create a function that produced a circular mask, where everything outside it was black, isolating the cancerous region. This would particularly benefit the pre-processing of the images that contained black corners, like the ones taken from a microscope, allowing for a potential model performance increase later in the project. We called this function *circle_mask*, that again started by resizing the image. Then, the centre of the image was computed, along with the circle's radius. Then a mask was created and applied to the image. Both images were then displayed for us to be able to visually understand if the function was working properly, [4].

The next pre-processing technique we will use works by selecting pixels based on a colour threshold. It starts by reading and resizing the image, followed by a conversion from BGR to LAB colour space. Then a colour threshold was chosen for skins with darker regions, followed by a creation of two masks that were later combined and inverted. Finally, some morphological transformations are applied, as well as the Gaussian blur to the regions selected by the mask. This method did not achieve great results. As such, it is not used in the notebook, but this function remains in the external utils.py file.

Our final effort to pre-process the images was to implement *dullRazor.* This was a method we found when researching for pre-processing techniques. However, to be able to use it, we needed to use dullRazor.exe, which we were not able to integrate with our python code. Nonetheless, as we wanted to still use this method, we looked for an alternative, having found a script of its code on GitHub. This function consisted of the attempt to remove imperfections, like hair, from the images. It started by converting the Images to grayscale. After doing so, it applied the black hat filter, which highlights the darks regions against a brighter background. Gaussian blur was then applied to the result of this filter, and a mask was created. Then, inpainting was done to replace the pixels in the original pictured based on the mask results. The image is converted back to RGB and displayed thanks to the function *show_dullRazor*. This method worked quite well for removing dark hairs but had trouble removing brown or lighter hairs, [1] [2] [3] .

This was the technique that showed better results, and the one we used for image pre-processing for the final scores. We used *img_modifier* to generate new images with this pre-processing, which were saved in the folder *new_data*.

Classification (2)

After implementing a handful of Image Preprocessing techniques, we must evaluate their results by implementing a Neural Network model. We started by defining the callbacks, as was done in Classification (1). We only defined two callbacks, one that was able to save the best parameters (model checkpoint) and the other that stopped the model if it was not improving (early stopping). Then, using the data generator for training and validation, and a Stratified K-Fold for splitting imbalanced data, we experimented with several models and parameters. To find the optimal parameters for our model we took advantage of the Keras Tuner Hyperband, which allocates resources to promising hyperparameter configurations. It is also well-suited for large search spaces, as ours. This tuner is often more computationally efficient when compared with a *GridSearch* or *RandomSearch*.

In the end we decided to keep some of the models we used in this section in the *models.py* file. The first one was already described in Classification (1). The second consisted of a model that did not use dropout layers, with a similar combination of Convolution and Max Pooling layers. However, for testing purposes, this time we decided to start with a large number of neurons (160) and end with a smaller one (20). Then we used a flatten and three dense layers, as well as the same optimizer as it appeared to output the better results, and the same loss function. We also kept a third model, *create_model_1*, which uses dropout layers, along with the Convolution and Max Pooling layers. This model starts with 256 neurons and the number of neurons is divided by two until it ends with 32 neurons.

The final and best model contains the optimal parameters outputted by the *Keras Hypertuner*. It is also composed by a combination of Convolution, Dropout and Max Pooling layers, with a flatten and dense layers in the end. These layers are composed in a similar way as it was done in our first model.

With our final model defined (model_tuned), trained and validated on the initial train set, we must now pre-process and make predictions based on never seen data, from the test set. The pre-process technique applied to these images was the same as the one applied to the train data, *dullRazor*. Before doing so, we decided to make predictions on the images in the training set and used the *classification report* from the *sklearn.metrics* library. We also decided to display two visualizations that showed the change in the train and validation accuracy and losses throughout the epochs.

With these results understood, we applied our model in the test set and made the final predictions.

Results

In our initial phase, we trained a simple model, without any image pre-processing, which did not achieve great results, as expected. We moved on from there to test several image pre-processing techniques, where we were able to identify *dullRazor* as the best one. This step proved to be as crucial as we expected, as the results improved considerably from the first experimentation, reaching a maximum weighted average of the f1-score of 76% in the training data and 74% when making predictions in the test set. Of course, these values were also highly influenced by the model we were using, which suffered several variations across the development of this project. It is also worth noting that the final model is not always able to predict the Dermatofibroma (df) class, since this class has so little observations. This can affect the f1-score for about 2%.

We also believe that if we were to have as many observations from the other classes as we have from the Melanocytic Neva (nv) class, it would be possible to achieve better results. If we were to improve our work, we could use a U-Net model for image segmentation and try to improve the *dullRazor* function to better detect lighter hairs.

On a global overview, we are satisfied with our work. However, we are somewhat disappointed not to have greater results. Despite this, we believe we have put a lot of effort into the image pre-processing techniques and model development sections of this project.

References

Castelli, Mauro in Deep Learning Classes Powerpoints

Perezhohin, Yuriy in Deep Learning Classes Colab Notebooks

[1] - Tim Lee, Vincent Ng, Richard Gallagher, Andrew Coldman, David McLean in Dullrazor®: A software approach to hair removal from images

[2] - DullRazor downlaod - http://www.derm.ubc.ca , consulted in 27/11/2023

[3] - Python version of DullRazor - https://github.com/BlueDokk/Dullrazor-algorithm, consulted in 29/11/2023

[4] - Python Script Base for Circle Mask - https://stackoverflow.com/questions/67640095/create-a-circular-masked-image-then-place-it-on-another-image-in-opencv, consulted in 04/12/2023