Skin Cancer Classification:

A deep learning approach

Deep Learning

Bruna Faria | 20211529; Catarina Oliveira | 20211616; Inês Vieira | 20211589;
Joana Rosa | 20211516; Rita Centeno | 20211579

Introduction

Skin cancer is one of the most common types of cancer worldwide and according to World Health Organization, one in three people might even suffer from it.[1] Several types of skin lesions exist, and an accurate and early diagnosis is essential for the treatment to be successful. [2] However, one of the main problems in this diagnosis is the similarity between benign lesions and malignant ones. In fact, without any technical support dermatologists only have an accuracy rate of around 65-80% in diagnosing melanoma, which ranks as the deadliest type of skin cancer; nevertheless, it can be cured in around 95% of the cases if detected early. [1,3] On suspicious cases, one common approach is to photograph the lesion with a magnifying high-resolution camera, with controlled light and a filter to reduce reflections; not only does this approach increase the accurate diagnosis rate of dermatologists but it also generates a large quantity of detailed images of skin lesions. [3,4] Deep Learning approaches, in particular with Convolutional Neural Networks (CNNs) have been demonstrating great ability in skin lesion classification, especially in binary problems.[4] On Multiclass problems, previous projects have been using techniques such as data augmentation, transfer learning and fine-tuning to achieve higher than 80% accuracies or f1-scores. [1,3] The goal of this project is to achieve a high-performance model that combines multimodal inputs, specifically images and tabular data (with age, location of the lesion, among others), in order to classify the skin lesions in 7 different categories.

Methodology

Before diving deep into the steps followed it is important to mention the creation of path.py**.** This file represented the path where the main data folder was defined and was personal to every member of the group. Since the data available had two different data types, it had to be pre-processed differently. Therefore, the next steps are also divided between the steps in common for both the images and tabular data (conveniently mentioned as metadata).

Common Steps

- Storing Images and Labels

The first step in what regards dealing with the provided data was to open the images and store them as arrays in a list, by using *os, pil* and *numpy* libraries. Additionally, the labels for each image were stored in another list in the corresponding index. These lists were directly separated into train and test - a total of 4 lists (training images, training labels, test images and test labels). The metadata was also divided into two smaller datasets, one for training and one for test data. However, before this division was made, the original indexes from the initial

metadata were stored in a new column to later check if the preprocessing stages were correctly done. For everything regarding Dataframes, *pandas* was the chosen library.

Images

- Initial Preprocessing

Regarding the images, both from the train and test sets, were subjected to some initial transformations. DullRazor was the first preprocessing technique used. The goal was to remove hair that was present in the image that could produce noise which could prejudice the model. Additionally, since the next techniques would affect the images resolution (by creating distortions in the images); therefore, impacting the performance of the hair-removing tool, it was important that it was applied before any other step. The code was retrieved from git hub repository and is referred to in the references. [5] Next, images were resized into a smaller format (71,71) to save some computational resources and for convenience to the pre-trained model. Images were also normalized so that every pixel had a value between 0 and 1, and features across different images had similar scales. Manipulation of images was mainly achieved through the use of the *cv2* library. Labels were also encoded using LabelEncoder from *sklearn.* This last library was used multiple times from now on to apply multiple pre-processing and machine-learning related techniques. To avoid data leakage from the test set, a train-test-split was performed on the training data, separating it in train and validation sets. The following steps were only performed on the training set.

Some of the lesion's images were repeated (even if with slight differences) so it was decided to drop duplicates and keep only one picture of each lesion. If not, the model would likely overfit (especially given that oversampling was performed later) since duplicated images would be more representative than others. Therefore, to ensure fairness in the images' importance distribution, this was the best approach found. As mentioned previously, oversampling was implemented to increase the representativeness of the minority classes. As the number of observations for the majority class was higher than every other class, a threshold was established to increase the number of observations of each minority class to half of the number of observations belonging to the majority one. In this way, a fairer representation among classes was ensured, while still guaranteeing that no severe computation problems would emerge from this data creation. That said, this oversampling technique also had its disadvantages. Having a copy of the same images repeatedly, in the same dataset, would highly bias the model towards the repeated images. Therefore, to mitigate this issue, the *Image Data Generator* from *keras* was used to allow for some differences among the copied images. Changes such as rotation, reflection, shifting, and zooming were then performed during training.

- Modelling

The first approach was to build a model by hand, with Convolutional, Pooling, Flatten, Dropout and Dense layers from *Keras*. Since the performance of these models was quite poor, pre-trained models were introduced. This approach allowed for the model to have some previously acquired knowledge which would be applied to the data at hand; hence, preventing the model from having to start its learning journey from zero. Two pre-trained models were tested: Xception and MobileNet, due to literature presenting them with the best results in similar tasks.[6] Finally, to better adapt the pre-trained model to the HAM10000 dataset, a fine-tuning approach was also implemented, including the Image Data Generator and Early Stopping Callback (Reduce LR on Plateau was also tried). To prevent a greater overfit, a very low learning rate was introduced.

Metadata

- Preprocessing

Regarding the tabular data, the variables encountered were *dx_type*, *age*, *sex*, *localization*, *lesion_id*, *image_id* and the target *dx*. The ones regarding the id of the image and the lesion were dropped, since they wouldn't

provide any relevant information to the model. Missing values were identified in the variable *age*. However, some more were discovered along the exploration of the data. These were in the variables *localization* and *sex* and were described as 'unknown'. The strategy to deal with these was to use KNN Imputer from *sklearn*, since it replaces the missing values according to similar observations (neighbors). Before doing so, it was necessary to encode the categorical variables (*sex*, *localization* , *dx_type* and *dx*). *Localization* was encoded following the order head to toe, and in regards to *dx_type,* One Hot Encoding was applied. As for the target, the strategy was just to encode it as in the image processing, with LabelEncoder. Since KNN uses distance, it was necessary to scale the variables and the missing values in the variable age were imputed temporarily with 0; the chosen scaler was MinMax. Just before the imputation, missing values were requalified as NaN (to be recognized by the imputer) and imputed with KNN. Given that this table was already divided in train, validation and test due to the processing of the images, train test split wasn't necessary and so, only the target variable was separated from the rest in each dataset. Finally, the arrays were converted to tensors with *tensorflow*.

- Modelling

The first modelling approach in the tabular data was to try some models, the best and final one had 5 dense layers, and a dropout layer. The insights from these tries were then used to run keras hyperband tuner, always with an early stopping callback activated, since it is a quite efficient and scalable fine-tuning algorithm.

Functional API

As an attempt to improve the existing models in the project, it was decided to try to make use of both the images and metadata. This resulted in the application of a functional API that allowed the building of a more complex model which used both previous models (image classification along with the tabular data model) as functional blocks to apply transfer learning into this new model.

Results and Discussion

Prior to the discussion of results it is relevant to mention that the original dataset was imbalanced. Due to this, and even though oversampling was applied to mitigate these effects, it was not possible to guarantee at 100% a perfectly balanced dataset, as it would greatly increase the required computational power, which was already a problem being faced. Therefore, all of the model's performances were evaluated based on the f1-score, since it represents a harmonic mean of both precision and recall (two distinct metrics that provide different insights into how the models are performing). At the same time, accuracy was also checked, since it gives a slightly different understanding of the proportion of correct predictions the model made.

Images

When fitting the pre-trained model to the HAM100000 images, the team tried to achieve performance values as close as possible between the train and validation sets, achieving a model that did not overfit nor underfit significantly. This was done by performing slight alterations to the layers added at the end of the pre-trained model. This base model had f1-scores of 0.60 and 0.65 and accuracies of 0.61 and 0.64, for the training and validation sets, respectively. In the fine-tuning process of this model, the results tended to show a lot of overfitting, which was also mitigated with alterations in the added layers – add dropout layers, increase the percentage of frozen neurons, and lower the learning rate. In the end, the overfitting was reduced to only 0.04.

The final model had the following architecture: Xception (CNN that is 71 layers deep) + Dense(256) + Dropout(0.1) + Dense(128) + Dense(7). The fully connected layers used a *relu* activation function, except the last one which used *softmax*, as it assigns probabilities to every class. This model was compiled using the *adam* optimizer and the *sparse_categorical_crossentropy* as the loss function, due to the fact that, in the end, the target variable consisted of several integer values (given that it was encoded using *LabelEncoder*).

When training what ended up to be chosen as the final model the following f1-score results were obtained: 0.78 on the training set; 0.74 on the validation set; 0.73 on the test set (when trained on only part of the original training data – training data with oversampling, also mentioned as X_train_over) and finally 0.77 on the testing set when the model was trained with all the original training data.

Metadata

As mentioned previously, the metadata was also target of exploration in order to possibly increase the predictive power of the image classification model. In this sense, when building the first model, results showed some underfitting. As so, adjustments in terms of number of neurons and layers were made in order to achieve a minimum overfit. These insights were then used to run a hyperparameter fine-tuning with keras hyperband tuner on each of the studied models, as it is an efficient and scalable tuning method.

The best-found initial model for this situation was one with: 5 dense layers and a dropout one in the middle (Dense (512) + Dense (512) + Dropout (0.1) + Dense(512) + Dense(128) + Dense(7), was trained over 100 epochs); additionally, it was compiled with the *adam* optimizer and *sparse_categorical_crossentropy*, for the same reason as explained in the *Images* section. Such a model was able to achieve, an accuracy of 0.64 and 0.61, as well as loss of 0.86 and 0.95, on the training and validation sets, respectively. Similarly, the f1-score was of 0.62 for training and 0.65 in validation. As mentioned previously hyperparameter tuning was performed and the final model's suggested architecture was slightly different (Dense(512) + Dense(512) + Dense(64) + Dense(256) with a learning rate of 0.001 trained over 12 epochs). This model was able to get to an f1-score of 0.66 on the training and 0.68 on the validation data.

Functional API

When the two previously described models (image and tabular data classification models) were combined through the use of a functional API the model was able to reach f1-score values of 0.64 and 0.69 for the training and validation data, respectively. Besides, it had a training loss of 0.69 with an accuracy of 0.8 and a validation loss of 0.89 with an accuracy of 0.67. From such results, it is clear that the model showed a lower predictive capability than the first model just classifying on its own.

Overall, the multi-input model (receiving both image and tabular data) seemed to not be able to predict skin cancer classes as well as the model which was trained only with skin lesion images. This can be due to the fact that the tabular data available did not have many features that could be relevant for the prediction of skin cancer; if perhaps there were more patient's data this model with functional API could have obtained better results.

Finally, it might be relevant to mention that the models were not able to equally identify all of the classes, and it was also clear that the class with more data/representativeness (Melanocytic nevi) was the one in which the model performed best.

Summary

Throughout the modelling process, some errors in the project development strategy were identified. The problem which the team members were faced the most with was the over or underfitting of the models. Because due to the high computational power that the fine-tuning step took it was decided that the best approach would be to only run it when a model did not seem to over nor underfit too much on the tried architecture. In some of these cases, the models ended up being too restricted by either increasing the dropout rate of the dropout layer or/and decreasing the number of neurons in the dense layers of the models.

Additionally, an example of the classified images and some mispredictions were represented in Figure 1. Through this image and the data combined from the confusion matrix on Figure 2 it is possible to have some clues on where the model is failing. Looking at it, one can infer that the Melanoma (mel) is the most wrongly

predicted class, meaning that when the model fails to predict the class at hand, it's common that it classifies it as melanoma. Moreover, by looking at Figure 1, it seems that this might be due to the lesion having a brown color tone, which is more common in melanoma images. Furthermore, Melanoma and Melanocytic nevi (nv) seem to be two classes that the model "frequently" mixes up. These are in fact, quite similar even to the human eye and in some cases the difference can be so subtle that only a biopsy can tell. The same happens in the classes Benign keratosis-like lesions (bkl) and nv.

Recommendations

After careful examination of all the decisions throughout the project, it was identified, that in the future, and if there was more time and computational power available more preprocessing techniques could be tried such as not reducing so much the size of the image. In addition, in the oversampling process, there might have been developed a better calculus behind the number of observations to assign to each class. Another possible option, if the computers that were used had more computational power, would have been trying a total oversampling. Undersampling could also have been an option to try; however, losing more images could result in unfairness between classes and so it wasn't tested in the limited time available. In addition, instead of dropping duplicated images, one could've searched better which one had the best resolution and only keep that one, instead of just leaving the first one that appeared.

Another possible recommendation for future works, would be including lesions on dark skin; since this and other similar models in the literature were only trained on white skin lesions they would likely fail in predicting the same lesions in a different colored skin.

Conclusion

Finally, although several different approaches were studied, the model which was found to be the best performing one to predict the type of skin cancer a patient has based on a single image of the skin lesion was a CNN pre-trained model (Xception) combined with three extra Dense layers, a Dropout layer between these first two along with fine-tuning. This model was able to have a final f1-score in the test set, when being trained with all of the training data, of 77%. In the end, the results achieved through a deep learning approach can be hopeful to the future of cancer identification in its early stages; hence, increasing the chances of better healthcare tools and survival rates for this deadly disease.

Annex

Figure 1 - Some Examples of Final Predictions

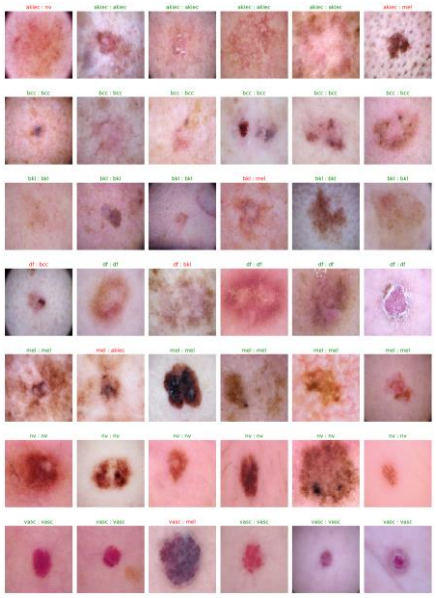
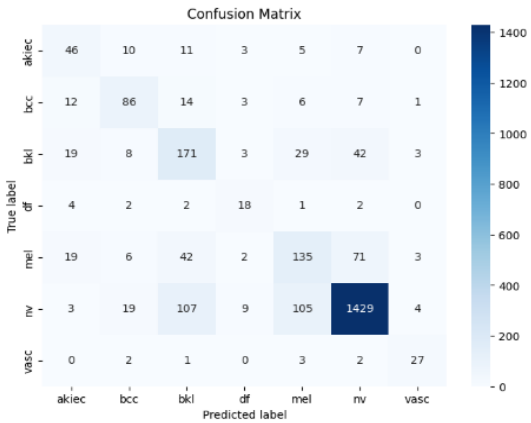


Figure 2 - Confusion Matrix of the Final Predictions



References

1. Ali, K., Shaikh, Z. A., Khan, A. A., & Laghari, A. A. (2022). Multiclass skin cancer classification using EfficientNets – a first step towards preventing skin cancer. *Neuroscience Informatics*, *2*(4), 100034.
2. Dorj, U. O., Lee, K. K., Choi, J. Y., & Lee, M. (2018). The skin cancer classification using deep convolutional neural network. *Multimedia Tools and Applications,* 77(8), 9909–9924. https://doi.org/10.1007/s11042-018-5714-1
3. Brinker, T.J., Hekler, A., Utikal, J.S., Grabe, N., Schadendorf, D., Klode, J., Berking, C., Steeb, T., Enk, A.H., Von Kalle, C. (2018). Skin Cancer Classification Using Convolutional Neural Networks: Systematic Review. *J Med Internet Res 2018*; 20(10):e11936
4. Lee, Y.C., Jung, S., & Won, H. (2018). WonDerM: Skin Lesion Classification with Fine-tuned Neural Networks. *ArXiv,* abs/1808.03426.
5. Velasquez, J. (2020). Dullrazor algorithm. GitHub Repository. https://github.com/BlueDokk/Dullrazor-algorithm/tree/main
6. Jain, S., Singhania, U., Tripathy, B., Nasr, E.A., Aboudaif, M.K., Kamrani, A.K. (2021) Deep Learning-Based Transfer Learning for Classification of Skin Cancer. *Sensors (Basel).* 21(23):8142. doi: 10.3390/s21238142. PMID: 34884146; PMCID: PMC8662405.