Deep Learning Project

MASTER DEGREE PROGRAM IN DATA SCIENCE AND ADVANCED ANALYTICS

Group 4

Adriana Costinha 20230567

Ana Filipa Silva 20230577

Beatriz Vasconcelos 20230755

Pedro Barão r20201614

April 2024

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

INDEX

1. Introduction

In this project, we were challenged to build a Deep Learning model that could identify skin diseases. We use features from two dermatology datasets, *"DermaAmin"* and *"ATLAS Dermatologico"* and perform a classification of 114 different skin conditions. We aspire to help doctors spot skin problems more accurately, promoting better health outcomes. To achieve this, we test multiple transfer learning models, multi-input and a convolution neural network built from scratch, with the best being the multi-Input model that uses *EfficientNetB3* for processing image data, having a f1 weighted score of 0.425.

2. Data Exploration and Pre-processing

We used multiple libraries to work on this project, such as *pandas*, *numpy*, *keras*. In the appendix, the full list can be found. Regarding implementation, we mostly used *VSCode*, although for heavier code we ran it on *GoogleColab PRO,* since the tested models required high computational power.

The Dataset is composed of 16 577 images and 9 columns. Upon a first inspection, we noticed in column '*qc*' there were multiple entries labelled as *'3 Wrongly Labelled'*. This implied these entries were dubious, therefore we decided to drop them.

The process of downloading the images raised some challenges. First with the functions we used – and then solved when we included the headers. Then we noticed some URL links were null values or couldn't be accessed – so we decided to drop them. Lastly, we had some issues with saving images all in the same format. We introduced an if statement to force all images to be saved with JPG extension. In total, we kept 16506 images of the dataset. After overcoming these challenges, we worked on splitting the data. We stratified our data into a split of 75-15-10 % for train, validation and test respectively.

2.1 Class Imbalance

With the images downloaded, we were able to observe a significant class imbalance in the target variable *'label'*, as can be seen in *figure 1*. To resolve this imbalance, we analysed some possible options: oversampling with geometrical augmentation, undersampling the majority classes and the usage of class weights.

When testing the viability of oversampling the minority classes with geometrical augmentation – such as rotation, zoom and other types on the images to reproduce new ones – we noticed that the dataset already had its own augmentation of this kind, *figure 2*. Meaning that if we decided to add more geometrical augmentation, we would just be doubling it, leading to much worse model's performance. So, we concluded that it was not a viable option. Not only that, but since there is a considerable difference between the majority and the minority classes, it would be necessary to recreate more than double the existing images for each minority class – inducing low diversity.

On the other hand, to use undersampling to adjust the number of images we would be losing too much data on the majority classes, dropping more than the double of the data.

A common method of dealing with class imbalance is class weights. Not only is it simple, but also effective when handling imbalance without requiring additional data or preprocessing. Therefore, we

adjusted the weights for each class in the model so that each is proportional to the class frequency. [1] Due to this, we had to use different metrics to evaluate our model which are referenced below.

2.2 Preprocessing

For the preprocessing stage of this project, we tested multiple algorithms to help our models' ability to recognize the skin diseases on the dataset. Such as *Clahe*, *Medium Blur* and *DullRazor* . We ended up choosing not to use any of these algorithms, since none helped to improve the models' performance.

The algorithm *Clahe*, Contrast Limited Adaptive Histogram Equalization [2], is applied to improve the contrast in images. In the case of our dataset, since there is a multiple range of different images, this algorithm although did well in many pictures, it also ended up contrasting the skin around, changing it too much. We used different *'clipLimit'* and *'tileGridSize'*, *figures 3 and 6*, and different configurations of the 'RGB2LAB' channels, *figures 4 and 5*, and still didn't show any improvement.

The algorithm *Medium Blur* is used to reduce noise and smooth the edges of an image [3]. Since our dataset is composed of many skin diseases, it's essential to maintain some degree of the initial skin details, which comes in conflict with this algorithm, *figure 7*. So, as we predicted, we didn't achieve great results with it.

The algorithm *Dull Razor* removes "hair" from the images [4]. Nevertheless, to achieve this, it first identifies the dark hair locations by a grayscale operation, then verifies the shape of the hair pixel and replaces the verified pixels by a bilinear interpolation. Finally, it smooths the area with an adaptive median filter. Although this algorithm showed potential, we confirmed it didn't work for most images – *figure 8*.

Additionally, we tried two combinations of the previously mentioned algorithms (*figures 9 and 10*) on our simplest model, *CNN*, but the results didn't improve, so we decided to not use any transformation on the other models.

The *flow_from_directory* function in *Keras* [5] presents a significant challenge due to how it operates: it loads images into the models alphabetically. As a result, if the images are not named or organized in a way that reflects their corresponding labels or categories, there is a risk of incorrect weight association and label assignment. Since it also works with images stored in multiple directories representing distinct labels, we had compatibility problems when working with models such as multi-input, where it is essential to have precise alignment between features data and corresponding images. To overcome these issues, we decided to convert the image data into arrays and allocate a dedicated column within the dataset for their storage.

To export the newly created arrays, we initially attempted to develop a serialization function. However, due to the large file size resulting from the extensive dimensions of the arrays, we opted to use *'pickle'* [6] - a module for converting Python objects into a byte stream for serialization and deserialization.

2.3 Data augmentation

As stated before, we used *image data generator* – geometric augmentation – to create more images, but it wasn't a viable option. We also tested the algorithm *YOLOv3 - You Only Look Once* [7]. It is an object detection algorithm that creates a frame around the object to spatially separate boxes and

associate class probabilities. It was used to try to separate images that were saved as a combination of more than one. We were able to identify some of these cases and divide them. *However,* this didn't work in most cases – *figure 11*. Therefore, we ended up discarding this option as well.

3. Models

In total we studied the performance of eight different models. One CNN, five transfer learning and two multi-input models. For each model we tested the preprocessing algorithms, previously mentioned, but to no avail.

For each model, we implemented *Earlystop callback* to stop running our models in the case that the validation loss didn't improve for two epochs in a row. This way, we could quickly stop models from overfitting.

3.1 Metrics

We have decided to use three different metrics to evaluate the models' performance. Firstly, considering the dataset's class imbalance, we employed the F1 weighted score - which prioritizes classes with more instances. Secondly, to assess the models' performance across all classes, we incorporated the F1 score macro, although it can be misleading if the performance of a rare class is crucial. Lastly, for a granular evaluation, we employed the F1 score micro, which measures the global F1 score of the model, not considering the class of each instance. Additionally, the loss function was also considered when it comes to evaluate if the model was over or underfitting and to quantify the difference between predictions and actual results.

3.2 CNN

We created our own architecture of a convolutional Neural Network, *figure 12*. It's composed of twenty-two layers, seven convolution layers, seven max pooling, five dropout layers and one flatten and two dense layers. After every convolution layer, we followed it with a max pooling to reduce the dimension of feature maps. We added dropout layers throughout the model to help reducing overfitting.

We tried using *AvgPooling* instead of *MaxPooling*, but we didn't obtain better results, so we decided to continue with *MaxPooling*. To decide how many layers to add, we did a try-and-error approach, always keeping in mind to not increase the models' complexity, thereby mitigating the risk of overfitting, while simultaneously ensuring that the metrics remained stable and validation loss did not increase.

Each 2D convolution layer has a *relu* activation function and a parameter *padding* defined as 'same'. This parameter ensures the spatial dimensions of the input and output are consistent. It's important to maintain spatial information all throughout the CNN. We also tried *leakyrelu* as an activation function but didn't get better results.

3.3 Transfer Learning

We tested five transfer learning models, such as: MobileNetV2, ResNet50v2, VGG19 based on [8], EfficientNetB2 and EfficientNetB3. We unfreeze the last two layers on each base model. The one with better performance, as it can be seen in t*able 1*, was EfficientNetB3.

VGG, visual geometric group, is a classical *CNN*. It utilises very small convolution filters, which shows that a significant improvement on the prior configurations can be achieved by pushing the depth to 19 weight layers. *VGG19* has two fully connected layers with an activation function, *relu*. [9]

ResNet-50V2 is an improved version of *ResNet50*, a model that belongs to the ResNet (Residual Networks) family and is renowned for its efficiency in image classification tasks. It's composed of fifty layers, forty-eight convolutional, one *MaxPooling* and one *AvgPooling*. [10, 11]

The *MobileNetV2* model uses a unique inverted residual structure, it employs thin bottleneck layers for both the input and output of the residual block. This model uses lightweight depth wise convolutions to filter features in the intermediate expansion layer. [12]

EfficientNet is a CNN that is capable of uniformly scaling all dimensions of depth, width and resolution using a fixed compound coefficient in a principled way. Its base network, *EfficientNetB0*, is based on the inverted bottleneck residual blocks of *MobileNetV2*, with the addition of other blocks. *B2* and *B3* are improved versions of *B0*. Analysing *table 1*, we can conclude that the best transfer learning model is *EfficientNetB3* as it outperforms the other transfer learning models in all the metrics having the lowest validation loss. [13]

3.4 Multi-Input Models

Regarding multi-input models, two different approaches were followed. The first approach combined the *EfficientNetB3*, the best transfer learning model found, to process the images alongside a multilayer perceptron to process the categorical data associated with each image. This categorical data consisted of four variables, two of which represented the colour of the skin presented in the image in different scales. As both provided the same information, only one was kept, the '*fitzpatrick_scale*', as it had a more detailed explanation in the referenced paper [14]. The extracted image features are then concatenated with the processed categorical features, which pass through some dense layers before coming to a final prediction. The second approach used the same CNN to process the image data and make predictions, while using an *XGBoost* to make predictions according solely to the categorical data. A grid search was conducted to optimize the hyperparameters of the model. Both predictions were then passed through dense layers, arriving at the final output. Analysing *table 2,* we can conclude the first approach has the best results.

3.5 Exploring hyperparameters

For image sizes, we tried a rectangular – 290x192 – and squared sizes – 224x224 and 300x300. To strike a balance between computational efficiency and large enough images for the models to be able to learn, this rectangular scale was found by searching the images for the size of the smallest 5%. The squared sizes were decided based on the input sizes of the transfer learning models used. For the batch size: 64 and 32 were tested [15, 16], and for the learning rate: 1E-3, 1E-4 and 1E-5 [17]. Three optimizers were assessed - *Adam*, *RMS* and *Adagrad*.

3.6 Choosing the Best Model

In order to choose our best model, we needed to evaluate each metric with close attention between all models. Knowing the best transfer learning model was EfficientNetB3, we only had to compare it to the CNN and Multi-input model. After analysing the *table 3 and figure 13*, we can conclude the best

model is the simple multi-input once it scores the highest in all the metrics, while having the lowest validation loss.

With the data imbalance of this dataset, we applied class weights – explained previously – to EfficientNetB3 and to the simple multi-input model, and noticed some classes had a zero recall and precision, therefore a zero f1score. Due to this odd behaviour, we decided to check the images of each label: the labels '*sun damaged skin*', '*pustular psoriasis*', '*ichthyosis vulgari*' are from minority classes with where all images are so different – we believe our models aren't able to find a pattern, meaning that the weight distribution could need some adjustments. With the label '*langerhans cell histiocytosis*' it's the opposite, since they're all so similar, we believe our models are struggling to associate the label to possibly different images in the test set. Resulting in a zero F1score. Analysing *table 4* and *table 5*, we can still conclude the best model is the simple multi-input without class weights and with *Adam* optimizer.

The multi-input model went through several iterations before becoming the *de facto* best model. One of the encountered problems was the already mentioned class imbalance, which affected mainly the prediction of the underpopulated classes. To overcome this, a tentative build with class weights was attempted, although it found no success. Another problem was overfitting, the model was quickly reaching high scores for the training data (+0.95 weighted f1 score), while the same scores for validation data were more than their half. By changing the model architecture – adding more dropout layers, increasing its rate and introducing early stopping – this problem was solved.

4. Conclusion

With this Deep Learning project, we were capable of working with multiple different models to accurately predict skin diseases. It is important to state that not only does this dataset has a big problem with imbalanced data, but it also lacks a diversity of skin colour range.

For a future possibility, we think testing *generative adversarial networks* (*GANs*, [18]) would be a good idea, because it can produce synthetic data samples increasing the size of the dataset and balancing the class distribution, as well as testing other algorithms for preprocessing, like cropping the images. Another algorithm to test, would be *GradCam,* where it essentially tracks where the models are "looking" and that way we could fix it to get better results.

Additionally, in the project we focused more on using the *Keras* library, but if we had more time, we'd have tried to use the the *PyTorch* library, and try new models like *AlexNet*.

In summary, this project not only achieved its primary objective, but also laid the groundwork for continued exploration and improvement.

5. References

[1] Kumar, A. (2023, December 6). *Handling Class Imbalance in Machine Learning: Python Example*. Analytics Yogi. https://vitalflux.com/class-imbalance-class-weight-python-sklearn/

[2] amritpal333. (2021, February 23). *CLAHE augmentation -Ranzcr comp*. Kaggle.com; Kaggle. https://www.kaggle.com/code/amritpal333/clahe-augmentation-ranzcr-comp

[3] GeeksforGeeks. (2019, April 17). *Python Image blurring using OpenCV*. GeeksforGeeks; GeeksforGeeks. https://www.geeksforgeeks.org/python-image-blurring-using-opencv/

[4] BlueDokk. (2021). *GitHub - BlueDokk/Dullrazor-algorithm: Pre-processing technique called DullRazor for the detection and removal of hairs on dermoscopic images.* GitHub. https://github.com/BlueDokk/Dullrazor-algorithm

[5] *Keras ImageDataGenerator with flow_from_directory() – Study Machine Learning*. (2024). Studymachinelearning.com. https://studymachinelearning.com/keras-imagedatagenerator-with-flow_from_directory/

[6] *pickle — Python object serialization*. (2024). Python Documentation. https://docs.python.org/3/library/pickle.html

[7] Redmon, J. (2018). *YOLO: Real-Time Object Detection*. Pjreddie.com. https://pjreddie.com/darknet/yolo/

[8] Houda Bichri, Adil Chergui, & Hain, M. (2023). Image Classification with Transfer Learning Using a Custom Dataset: Comparative Study. *Procedia Computer Science*, *220*, 48–54. https://doi.org/10.1016/j.procs.2023.03.009

[9] *Papers with Code - VGG Explained*. (2022). Paperswithcode.com. https://paperswithcode.com/method/vgg

[10] Petru Potrimba. (2024, March 13). *What is ResNet-50?* Roboflow Blog; Roboflow Blog. https://blog.roboflow.com/what-is-resnet-50/

[11] *ResNet-50: The Basics and a Quick Tutorial*. (2023, May 22). Datagen. https://datagen.tech/guides/computer-vision/resnet-50/

[12] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. ArXiv.org. https://arxiv.org/abs/1801.04381v

[13] Tan, M., & Le, Q. V. (2019). *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. ArXiv.org. https://arxiv.org/abs/1905.11946v5

[14] Groh, M., Harris, C., Soenksen, L., Scale, F., Francisco, S., Scale, R., Scale, A., Koochek Banner, A., Phoenix, H., & Badri, O. (n.d.). *Evaluating Deep Neural Networks Trained on Clinical Images in Dermatology with the Fitzpatrick 17k Dataset*. Retrieved April 28, 2024, from https://workshop2021.isic-archive.com/paper_groh.pdf

[15] Kandel, I., & Castelli, M. (2020). The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset. *ICT Express*, *6*(4), 312–315. https://doi.org/10.1016/j.icte.2020.04.010

[16] Chollet, F. (2018). Deep Learning with Python. Manning Shelter Island. 361 pages. - batch size

[17] Kandel, I., Castelli, M., & Aleš Popovič. (2020). Comparative Study of First Order Optimizers for Image Classification Using Convolutional Neural Networks on Histopathology Images. *Journal of Imaging*, *6*(9), 92–92. https://doi.org/10.3390/jimaging6090092

[18] Catur Supriyanto, Salam, A., Junta Zeniarja, & Wijaya, A. (2023). Two-Stage Input-Space Image Augmentation and Interpretable Technique for Accurate and Explainable Skin Cancer Diagnosis. *Computation*, *11*(12), 246–246. https://doi.org/10.3390/computation11120246

6. Appendix

The libraries we used to implement our models were:

- *Pandas* for its versatility, primarily for reading CSV files, conducting data analysis, and facilitating data manipulation.
- *Math* and *Random* for random division of images.
- *NumPy* and *Scikit-learn* to convert images into arrays.
- *OS* and *Requests* for retrieving/downloading images from URLs in the dataset.
- *PIL* and *OpenCV* (cv2) for reading and processing image files, as well as applying various transformations.
- *TensorFlow* and *Keras* for data preprocessing, model building, and evaluation.
- *Matplotlib* for plotting graphs for metric evaluation and visualizing images with applied transformations.

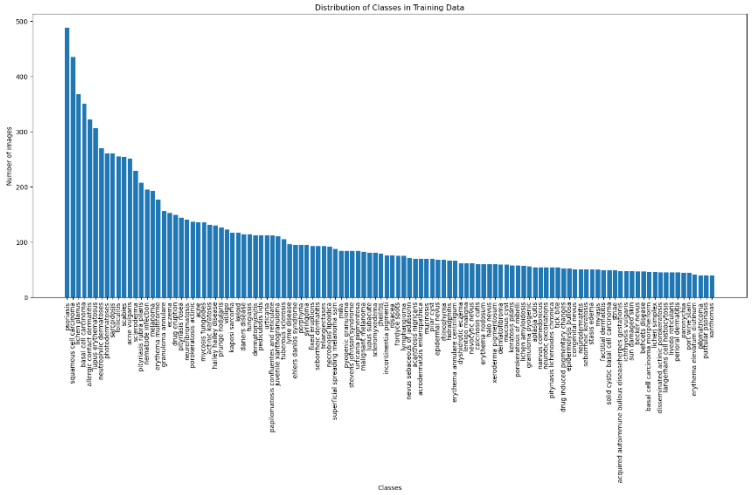

Figure 1 – Class distribution showing a clear imbalance.

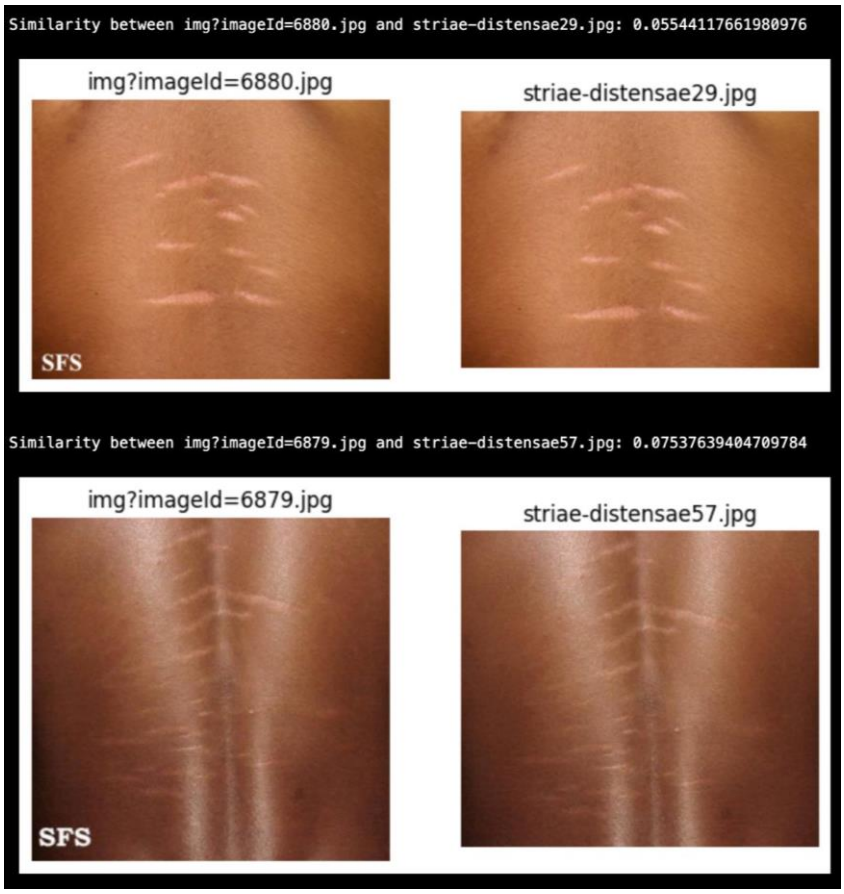

Figure 2 – Pre-augmentation present in the dataset.

Figure 3 – Representation of Clahe, with clipLimit = 2.0 and tileGridSize = (7,7).

Figure 4 – Representation of Clahe, with clipLimit = 3.0 and RGB2LAB.

Figure 5 – Representation of Clahe, with clipLimit = 5.0 and RBG2LAB enhanced.

Figure 6 – Representation of Clahe, RBG2LAB, clipLimit = 2.0 and tileGridSize = (1,1).

Figure 7 – Representation of Medium Blur.

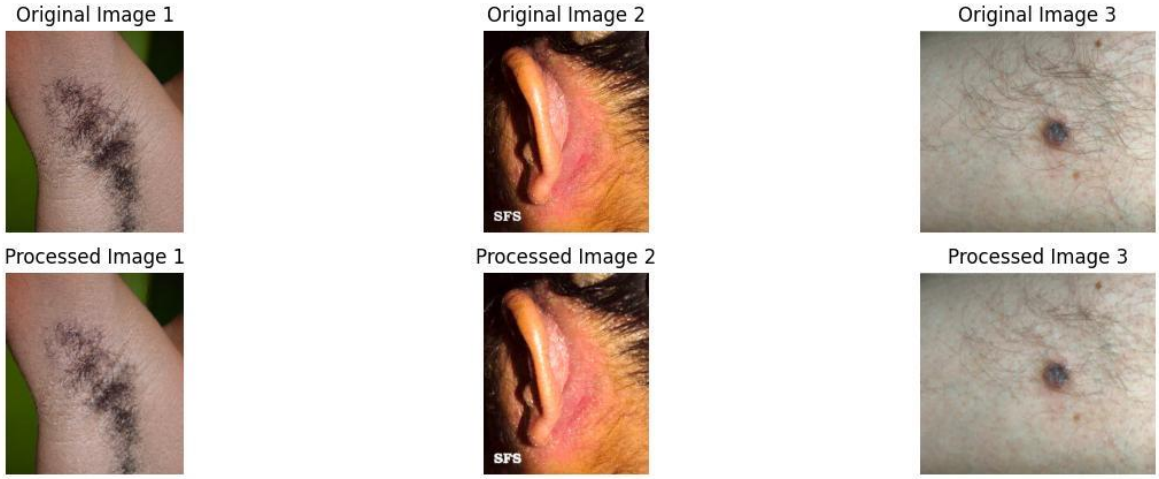

Figure 8 – Representation of Dull Razor.

Figure 9 – Representation of Clahe of figure X and Medium blur.

Figure 10 – Representation of Clahe of figure X and Dull Razor.

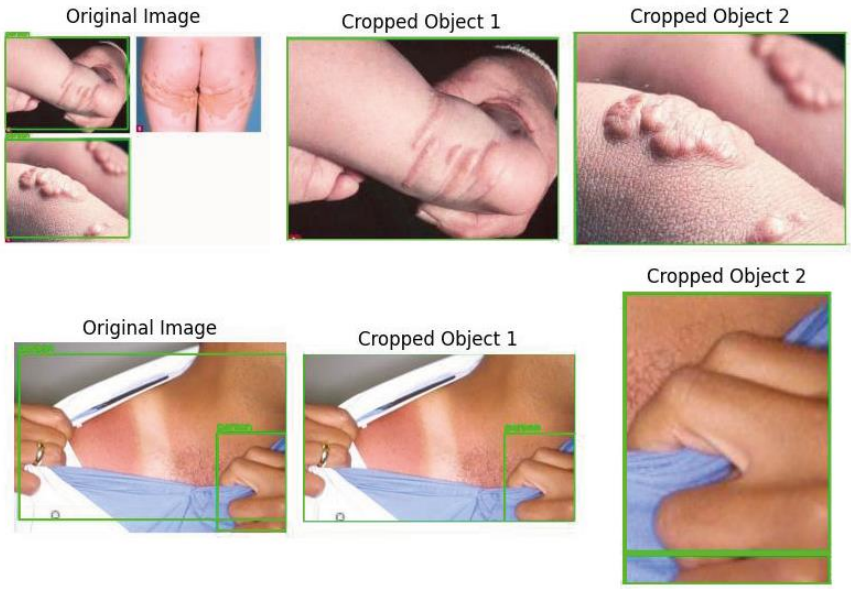

Figure 11 – Representation of YOLO's behaviour. On the top: the algorithm identifies correctly two objects in the original image, but not the third. On the bottom: the algorithm is recognizing more than one object in frame while it should only identify one 'person'.

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3,3), activation = 'relu', input_shape = (290, 192, 3), padding = 'same'))
model.add(layers.MaxPooling2D((2,2)))

model.add(layers.Conv2D(64, (3,3), activation = 'relu', input_shape = (290, 192, 3), padding = 'same'))
model.add(layers.MaxPooling2D((2,2)))

model.add(layers.Conv2D(128, (3,3), activation = 'relu', input_shape = (290, 192, 3), padding = 'same'))
model.add(layers.MaxPooling2D((2,2)))

model.add(Dropout(0.25))
model.add(layers.Conv2D(256, (3,3), activation = 'relu', input_shape = (290, 192, 3), padding = 'same'))
model.add(layers.MaxPooling2D((2,2)))

model.add(Dropout(0.3))
model.add(layers.Conv2D(512, (3,3), activation = 'relu', input_shape = (290, 192, 3), padding = 'same'))
model.add(layers.MaxPooling2D((2,2)))

model.add(Dropout(0.35))
model.add(layers.Conv2D(1024, (3,3), activation = 'relu', input_shape = (290, 192, 3), padding = 'same'))
model.add(layers.MaxPooling2D((2,2)))

model.add(Dropout(0.4))
model.add(layers.Conv2D(2048, (3,3), activation = 'relu', input_shape = (290, 192, 3), padding = 'same'))
model.add(layers.MaxPooling2D((2,2)))

model.add(layers.Flatten())
model.add(layers.Dense(512, activation = 'relu'))
model.add(Dropout(0.4))
model.add(layers.Dense(114, activation = 'softmax'))
```

Figure 12 – Convolutional Neural Network architecture with the number of neurons for each layer.

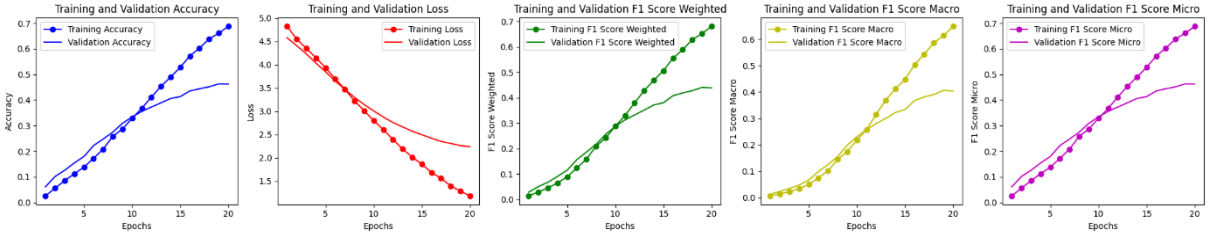

Figure 13 – Training and Validation Accuracy, Loss, Weighted, Macro and Micro F1 scores for each epoch of the best model, multi-input with Adam optimizer.

Table 1 – Performance Metrics for Transfer Learning Models.

Model	Test F1score micro	Test F1score macro	Test F1score weighted	Validation Loss
VGG19	0.038	0.002	0.007	4.512
ResNet50v2	0.011	0.007	0.007	4.718
MobileNetV2	0.296	0.216	0.265	3.275
EfficientNetB2	0.300	0.231	0.277	3.358
EfficientNetB3	0.362	0.321	0.34	2.872

Table 2 – Performance Metrics for Multi-Input Models.

Model	Test F1score micro	Test F1score macro	Test F1score weighted	Validation Loss
Multi-Input simple	0.416	0.34	0.393	2.39
Multi-Input with XGBoost	0.053	0.006	0.016	4.683

Table 3 – Performance Metrics for the best Models of each type.

Model	Test F1score micro	Test F1score macro	Test F1score weighted	Validation Loss
EfficientNet B3	0.362	0.321	0.340	2.872
Multi-Input simple	0.416	0.340	0.393	2.390
CNN	0.190	0.149	0. 170	3.570

Table 4 – Performance Metrics for the models with class weights.

Model	Test F1score micro	Test F1score macro	Test F1score weighted	Validation Loss
EfficientNetB3 with class weights	0.358	0.343	0.347	3.045
Multi-Input simple with class weights	0.340	0.326	0.326	2.720
Multi-Input simple without class weights	0.416	0.340	0.393	2.390

Table 5 – Performance Metrics for the best Models of each type.

Simple multi-input with different optimizers	Test F1score micro	Test F1score macro	Test F1score weighted	Validation Loss

Adam	0.446	0.384	0.425	2.242
Adagrad	0.027	0.008	0.020	4.659
RMS	0.416	0.340	0.393	2.390