Deep Learning

Dermatology Problem

Develop a model to address
a dermatology classification problem

Group 45

April 2024

Group Members:

André Filipe, 20230972

Catarina Reis, 20230981

Diogo Almeida, 20230737

João Gonçalves, 20230560

João Pedro Mota, 20230454

Prof. Mauro Castelli | Prof. Yuriy Perezhohin

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação

Contents

1. Introduction

Dermatology plays a crucial role in healthcare, particularly in the early detection and treatment of various skin conditions. In this project, we aim to develop a deep-learning model to address the dermatology classification problem using a dataset sourced from the “FITZPATRICK17” repository.

The dataset comprises images representing a diverse range of skin conditions, annotated with disease information such as malignant/benign classification and URLs of skin lesion images. A key aspect of the dataset is its representation of 114 skin conditions, with varying numbers of images per condition, sourced from the "DermaAmin" [1] and "ATLAS Dermatologico" [2] datasets.

Our primary goal is to build a robust model capable of accurately classify skin conditions from unseen images. To achieve this, we will explore different deep learning architectures and hyperparameter configurations. The model will be trained, validated, and tested using a three-split dataset to ensure reliable performance assessment. This report provides a comprehensive overview of our approach and results, and its analysis.

2. Task Description

The project aims to explore various deep learning architectures, including Convolutional Neural Networks (CNNs), to classify images effectively. The primary objective is to evaluate different model configurations and hyperparameters to identify the optimal architecture for image classification tasks. The project involves data preprocessing, including image augmentation and normalization, to enhance model performance.

Additionally, our project encompasses experimentation with different CNN architectures, such as ResNet, DenseNet, and VGG16, and also a CNN architecture where we have multi-inputs, as images and categorical features. As an evaluation method we will use metrics mostly like loss, and F1 weighted score. The project seeks to identify the most suitable deep learning architecture to get the best classifications possible for the images, having in mind the evaluation methods.

3. Methodology

3.1. Data Pre-processing

The dermatology dataset used in this project contains images representing various skin conditions, each associated with specific disease information, including malignancy or benign status, angle scores, and image URLs. There are metadata for 114 distinct skin conditions, with each condition having between 53 and 653 associated images.

First, we checked the content of our dataset, removing rows that were not associated with images, or rows that had URLs giving errors. As a result, only 0.01 % of the images were dropped, not affecting the dataset. Moving forward to the models, we resized the images to 150 x 150 pixels due to lack of processing power and time restriction. All images were normalized to have pixel values between 0 and 1 to facilitate model training. [3]

3.2. Data Augmentation

Data augmentation [4] is crucial for training models to predict labels from images, as it artificially expands the training dataset with varied transformations, such as rotations and flips. This technique prevents the model from overfitting by exposing it to a wide range of conditions, thereby enhancing its ability to generalize to new images. Therefore, we used data augmentation in all our models, adjusting the parameters to observe how the models would react.

3.3. CNN Architectures

Convolutional Neural Networks (CNNs) are supervised machine learning algorithms designed for image identification and classification, trained using labelled data with their respective classes. In our study, we utilized

custom-built CNNs and various architectures, such as VGG16, ResNet, DenseNet, and a custom-built multi-input model. We applied our dataset to these architectures to evaluate their performance under different criteria. The main idea was to compare the performance of the different models and to observe how the models evolve with various hyperparameters.

3.4. OPTIMIZERS

Optimizers are essential algorithms that adjust weights and learning rates within CNN architectures to enhance performance. In our study, we evaluated three distinct optimizers: Adam (Adaptive Moment Estimation) [5] and SGD [6]. These optimizers are crucial for fine-tuning model parameters during training, improving performance by updating weights and learning rates. Our primary focus was on Adam, as it consistently provided better results compared to the others.

4. RESULTS

4.1. EXPERIMENTAL SETUP

To execute the code, the following Python libraries are required: tensorflow, keras, numpy, matplotlib, sklearn, time, os, cv2(OpenCv), PIL (from PIL import image), BytesIO, and seaborn. The dataset was divided into a 70-15-15 split, for training, validation, and testing, respectively.

4.2. EVALUATION MEASURES

When dealing with an imbalanced dataset, relying solely on accuracy might not reflect model performance. Instead, we can use metrics that consider class imbalances. One such metric is the F1 score, the harmonic mean of precision and recall, which accounts for both false positives and false negatives, making it suitable for imbalanced datasets. The F1 weighted score [7] is particularly useful as it computes the F1 scores for each class, weighted by the number of samples in each class, thereby providing a more balanced performance measure. This is the perfect case to apply this metric, as we have classes with much more examples than others.

Additionally, we monitored the loss [8] metric, which is crucial for evaluating potential overfitting. In our case we used the categorical cross-entropy loss function because there are more than two label classes to predict. Monitoring loss function provides essential feedback on the model's learning effectiveness, indicating how well the model's predictions match the actual data. It is particularly useful in identifying overfitting, especially in datasets where certain classes are dominant. In our case, we have many classes and some of them are predominant. A significant divergence between training and validation loss suggests that the model is memorizing the training data, rather than learning to generalize from it. Following the professor's guidance, we considered any model exhibiting a difference greater than 0.5 decimals between training and validation loss as overfitting. By keeping a close watch on the loss, we were able to adjust training parameters in real-time to mitigate this issue.

After going through these steps and determining that the model was feasible, we tested it on the test set to evaluate its performance. This was the process we used to determine the importance of the model.

4.3. APPROACH OF INTERMEDIATE MODELS

In this part of the report, we present the evaluation of a wide range of Convolutional Neural Network (CNN) models [9] trained for our classification task. The goal was to identify the most effective model architecture and hyperparameters for the given dataset, having in mind our process of selection and evaluation. In our study, we explored different configurations to fine-tune and optimize the performance of our models. Throughout the development of these models, the improvements largely reflected the power of trial and error, as well as a growing sensitivity to how each parameter impacts the model's performance. Here are the main approaches we employed and considered relevant for the project and acquire knowledge for ourselves:

VGG16

Model VGG16(first) - [10] **Fine-tuning the Last Few Layers:** In this setup, we have chosen to freeze all but the last four layers of the VGG16 base. This approach keeps the weights of the earlier layers unchanged. This approach aimed to adapt the pre-trained model more closely to the specific nuances of the skin disease images, enhancing model performance relative to the baseline.

Model VGG16(second) - Transfer Learning [11] **with Feature Extraction:** Here our VGG16 model was employed as a feature extractor. The original fully connected layers were removed and replaced with new layers tailored to the classification of 114 skin diseases. New fully connected layers were added with specified neuron counts to fit the classification needs. Only these newly added layers were trained. [*Tabela I - Loss/Accuracy]*

Custom-Built CNN

Model 4(first) - Baseline CNN [12] **Approach**: The model begins with two 32-filter convolutional layers, each utilizing 3x3 kernels with 'relu' activation and L2 regularization, designed to process images of size 70x70 with 3 colours channels. These layers are followed by a max-pooling layer to reduce spatial dimensions. The network then employs successive convolutional layers with increasing filter sizes—64, 128, and 256—each paired with max-pooling and continued regularization. After processing through these layers, the model transitions to classification with a flattening layer, a dropout rate of 0.25 to reduce overfitting, and dense layers (256 and 128 neurons) culminating in a SoftMax-activated output layer. It utilizes the Adam optimizer with a learning rate of 0.0001 for fine-tuned training.

Model 5(second) - Adding complexity: The transition from the previous model to this one introduced some modifications aimed at enhancing the network's ability to process larger images. First, this model increases the input image size to 150x150 pixels, compared to the previous model which processes 70x70 pixel images. This adjustment allows it to handle more detailed images. Unlike the previous model, this one incorporates batch normalization after each convolutional layer, except the first. This addition helps stabilize the learning process by normalizing the activations, which can lead to faster convergence and improved training stability. Additionally, this model varies the dropout rates throughout the network, ranging from 0.15 to 0.3 after specific layers. This strategy contrasts with the previous model, where a single dropout rate of 0.25 was used after flattening. The varied rates help better control overfitting, especially as the network's depth increases. It also uses a higher learning rate of 0.001 with its Adam optimizer, compared to the previous model which used a lower rate of 0.0001.

Model 6(third) – The best fine tuning: The final iteration of our model emerged as the best choice, exhibiting superior performance on the test set while effectively managing overfitting. In terms of architecture, we reduced one convolutional layer and we still implemented batch normalization after each one to promote more stable learning. We carefully adjusted the dropout rates to find the ideal balance between learning efficiency and overfitting prevention. To further enhance regularization, we continued to apply L2 regularization across all convolutional layers, penalizing complex weights to curb overfitting. For optimization, we opted for a reduced learning rate of 0.0001, and we kept using the Adam optimizer. [12] [*Tabela V - Model6 CNN Loss/Acc][Tabela IX - Model6(Custom Built) Architeture]*

DenseNet and ResNet

Model - DenseNet [13] [14] **and ResNet** [15] **Approach**: Our objective was to evaluate two models, ResNet101 and DenseNet121, under nearly identical conditions, with only slight differences in data augmentation parameters. We aimed to assess how these subtle variations could impact performance outcomes. Surprisingly, the results varied significantly. Using the DenseNet121 architecture, we achieved one of the best performance among all tested models

[1] For every model presented the activation function used was “SoftMax”. [17]

[2] Since we are limited to five pages of report, we can get in much detail about the value of each parameter and its explanation, however, is present in the code file.

based on the F1 weighted score on the test set, although it's important to note significant overfitting in the loss function beyond the fifth epoch, indicating a potential weakness in the model despite high performance metrics.[*Tabela II-DenseNet Loss/Acc]* Conversely, the ResNet101 model showed one of the poorest performances in terms of the F1 weighted score in the test set, suggesting better generalization despite lower F1 scores.[*Tabela III-ResNet Loss/Acc]* Both models were configured with the Adam optimizer, a learning rate of 0.001, and were trained over 20 epochs, using transfer learning with identical layers.

These two models provided us with a crucial point of comparison for the CNN models we created. The main difference between these models lies in their architectural designs and how they handle feature propagation. ResNet101 utilizes residual connections, which allow it to skip one or more layers. This helps alleviate the vanishing gradient problem and enables the training of deeper networks without performance degradation. In contrast, DenseNet121 employs densely connected layers, where each layer receives additional inputs from all preceding layers and passes its own feature-maps to all subsequent layers, enhancing feature reuse and reducing the number of parameters. This architectural difference fundamentally influences their performance, particularly in terms of overfitting and generalization capabilities.

Multi Input

Model - Multi Input [16]**:** In this model, we integrate categorical features such as the 'fitzpatrick_scale', 'nine_partition_label', and 'three_partition_label' with image data. This approach aims to provide the model with a more comprehensive set of information, enhancing its potential beyond what the Custom-Built CNN model can achieve. Initially, we tested a similar multi-input model configuration before finalizing this design.

The image processing component of the model involves multiple convolutional layers with increasing filter sizes from 32 to 64, and finally to 128. Each layer includes batch normalization and dropout layers to promote generalization and prevent overfitting. Max pooling layers follow the convolutional stages to reduce the spatial dimensions of the images, focusing on the most significant features. The processed data is then flattened, making it suitable for concatenation with the tabular data.

For the tabular data, we begin by normalizing the input through a custom lambda layer to ensure all data is on a comparable scale. This is crucial for effective learning and seamless integration with the image data. The normalized data then passes through several dense layers, each featuring batch normalization, moderate dropout rates, and L2 regularization to enhance the model's robustness.

The outputs from both the image and tabular data streams are concatenated, forming a comprehensive feature set that merges visual and structured data. This combined data is further processed through additional dense layers equipped with dropout, leading up to a SoftMax output layer. This layer classifies the inputs into multiple classes based on the learned features, providing a probabilistic representation of class membership. [*Tabela VII- Multi Input]* Finally, the model uses the Adam optimizer with a learning rate of 0.0001, and the loss function is categorical cross-entropy, which is suitable for the multi-class classification tasks at hand.[*Tabela IV- Multi Input Loss/Acc]*

5. Evaluation of Best Approach

After going through our selection process, we decided to choose the multi-input model as our best option. It shows a little sign of a very small overfitting on the loss in the first epochs, however, when tested on the test set, it achieved a respectable Weighted F1 score of 0.32. Therefore, in comparison with other models, we decided to select this one as the best, it gives the best predictions with just a little of overfit. *[Tabela IV- Multi Input Loss/Acc]* Below on the annex*[Tabela VI- Results from the evaluated models and some other],* we have provided a table from excel with some acceptable models, although not showing all that we built which are present in the code file, and in some way performed poorly. This format makes it easier to visually compare the results and clarify the parameters we used.

6. Error Analysis

In this topic, we aim to discuss which aspects our best model is misclassifying and understand what the model is failing to learn. After analysing the error percentages in each class, we observe a pattern: classes with fewer images for the model to learn from tend to have more misclassifications. As predicted, having fewer images adversely affects the model's performance. Therefore, it is evident that we could enhance the model's performance by ensuring an equal and substantial number of images for each class. However, there are some classes that do not follow the pattern, having low number of examples on training but still get well classified. In the code there are present classes with more error percentages and the relations with the number of images in the class. [*Tabela VIII - Error rate/Identified Pattern]*

An error we made during the project was using the final score of the weighted F1 in the test set, which corresponds to the last epoch of the model. However, we should have used the value from the last epoch before overfitting occurred in the loss function, instead of using the value from the very last epoch. When we noticed the error, we started using the model that was already recorded in the model checkpoint callback.

Another mistake we noticed later was the custom function we used to evaluate the weighted F1 score during training and validation, which yielded results slightly different from what was correct, this only happened in the Custom-Built CNN but not on the Multi-Input model. Consequently, we focused solely on the Weighted F1 score in the results of the test set to evaluate the Custom-Built CNN. For the multi-input model, the Weighted F1 Score during training and validation can be interpreted correctly.

The group also noticed that the absence of setting the seed will represent a problem in reproducing the code, so the results aren't always the same.

7. Future Work

To enhance our model's ability to identify various skin conditions in diverse images with varying lighting, backgrounds, and skin tones, we recognize the need for a more extensive and diverse training dataset. By exposing the model to a wider range of real-world scenarios, it can better generalize and adapt to the complexities of new images. Moreover, leveraging additional computational resources would enable us to explore a broader spectrum of hyperparameters through techniques like grid search. This thorough exploration could potentially uncover optimal configurations for enhancing model performance. Experimenting with different data splitting strategies and analysing their impact on model performance could also provide valuable insights into our study. Lastly, given more time, we could delve deeper into exploring a variety of architectural variations. This exploration could involve experimenting with different CNN architectures and if available, incorporate other types of data such as patient metadata or previous medical history, that could provide more context to the image-based predictions.

8. Conclusion

Throughout this project, we embarked on a comprehensive exploration of both pre-trained and custom-built CNN models. From the outset, the extensive range of class labels coupled with the sparse data hinted at potential challenges in achieving optimal results. Indeed, while we are relatively pleased with the outcomes of our various experiments, the models we developed have not yet reached a standard suitable for clinical deployment, primarily due to an unsatisfactorily low F1 score when talking about health. Given the critical nature of medical diagnostics, settling for such a level of performance is not an option.

To enrich our project and insert some extra novelty in our work, we incorporated a multi-input model. Looking forward, with enhanced computational resources, we are optimistic that we could refine a little bit these models, however we know that wouldn´t be enough to be applied in a real-world problem.

9. References

[1] "Dermaamin," [Online]. Available: https://www.dermaamin.com/site/. [Accessed 26 April 2024].

[2] "Atlas Dermatology," [Online]. Available: https://atlasdermatologico.com.br/. [Accessed 26 April 2024].

[3] "TensorFlow," [Online]. Available: https://www.tensorflow.org/tutorials/load_data/images?hl=pt-br. [Accessed April 2024].

[4] "TensorFlow," [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/SGD. [Accessed April 2024].

[5] "TensorFlow," [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam. [Accessed 25 April 2024].

[6] "TensorFlow," [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/SGD. [Accessed 25 April 2024].

[7] "Scikit-learn," [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html. [Accessed 25 April 2024].

[8] "Keras," [Online]. Available: https://keras.io/api/losses/probabilistic_losses/#categoricalcrossentropy-class. [Accessed 25 April 2024].

[9] “Keras,” [Online]. Available: https://keras.io/api/. [Acedido em 25 April 2024].

[10] "Keras," [Online]. Available: https://keras.io/api/applications/vgg/. [Accessed 25 April 2024].

[11] "Keras," [Online]. Available: https://keras.io/guides/transfer_learning/. [Accessed 25 April 2024].

[12] "TensorFlow," [Online]. Available: https://www.tensorflow.org/tutorials/images/cnn?hl=pt-br. [Accessed 25 April 2024].

[13] "TensorFlow," [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/applications/DenseNet201. [Accessed 25 April 2024].

[14] "AnalyticsVidhya," [Online]. Available: https://www.analyticsvidhya.com/blog/2022/03/introduction-to-densenets-dense-cnn/. [Accessed 25 April 2024].

[15] "TensorFlow," [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/applications/ResNet101. [Accessed 25 April 2024].

[16] G. Hogg, 25 April 2024. [Online]. Available: https://www.youtube.com/watch?v=4-O14gOdRso&t=1089s&ab_channel=GregHogg.

[17] "TensorFlow," [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/activations/softmax. [Accessed 25 April 2024].

10.Annex

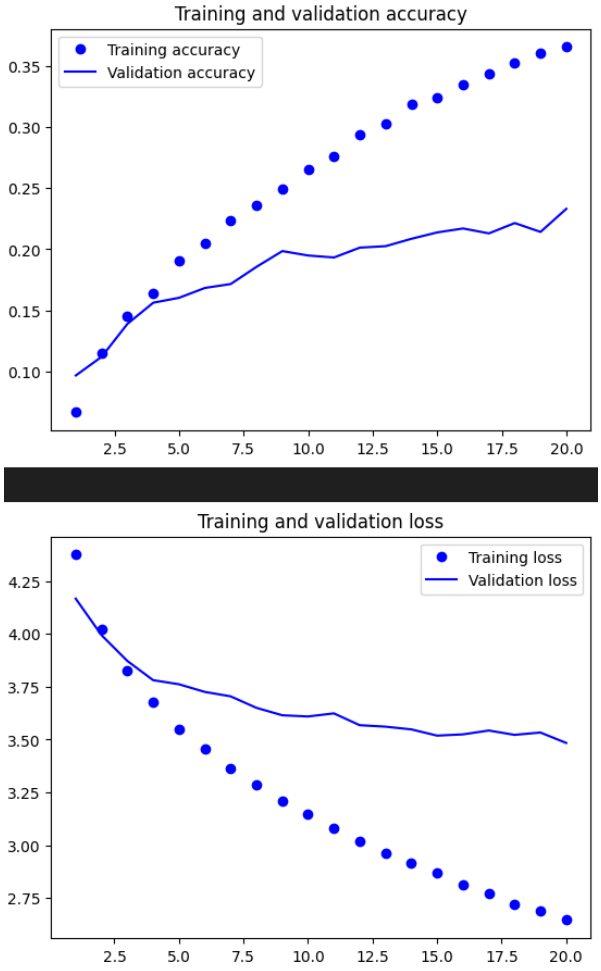


Tabela I - Loss/Accuracy

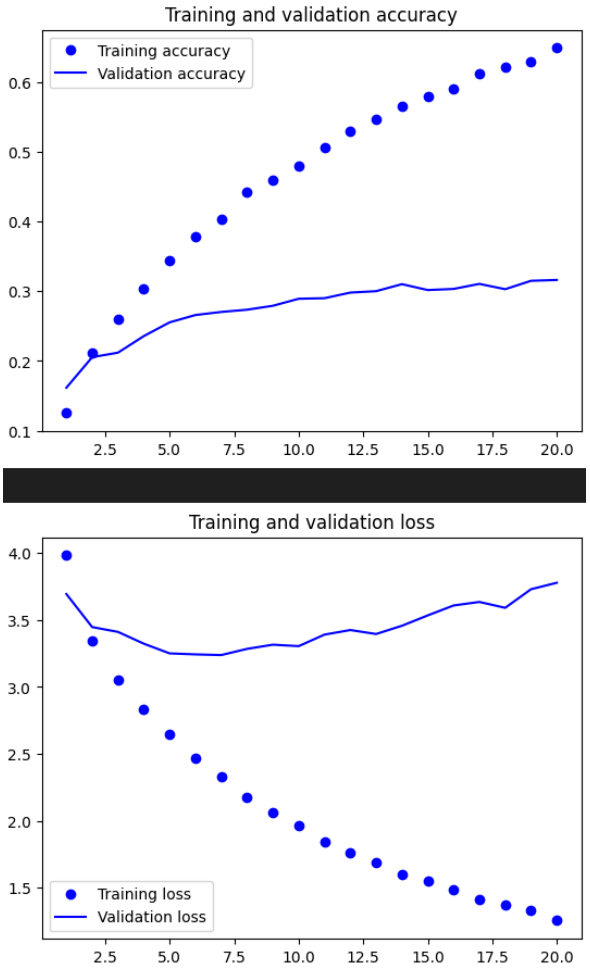
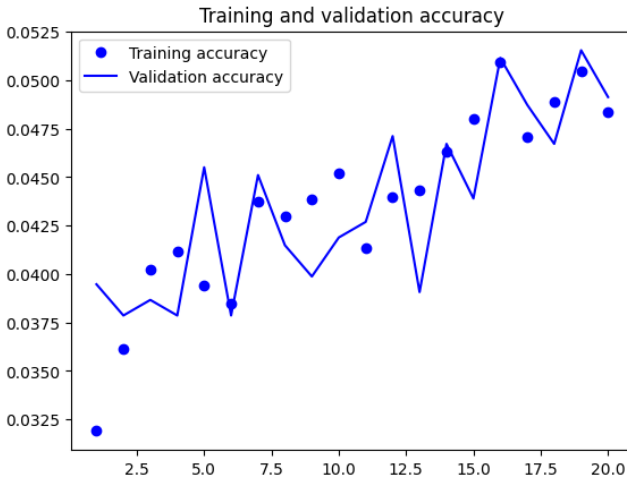


Tabela II-DenseNet Loss/Acc



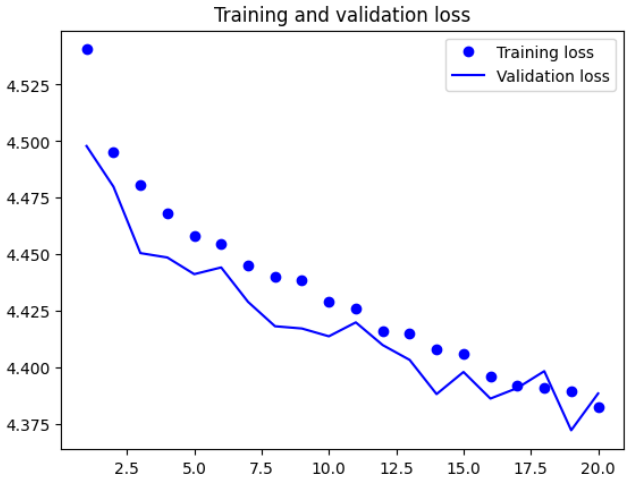
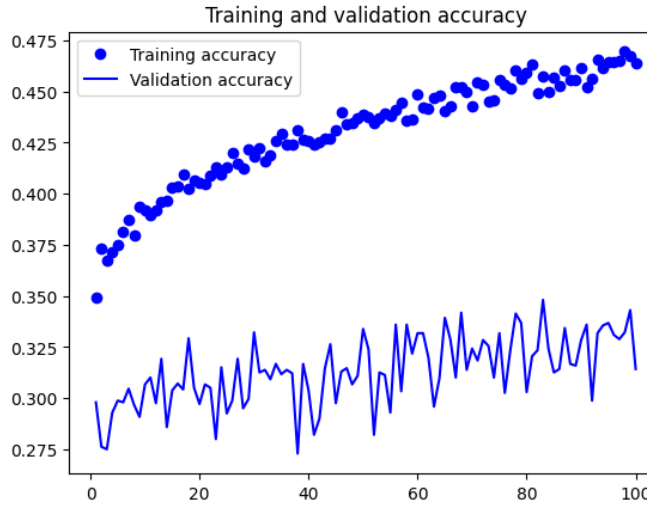


Tabela III- ResNet Loss/Acc



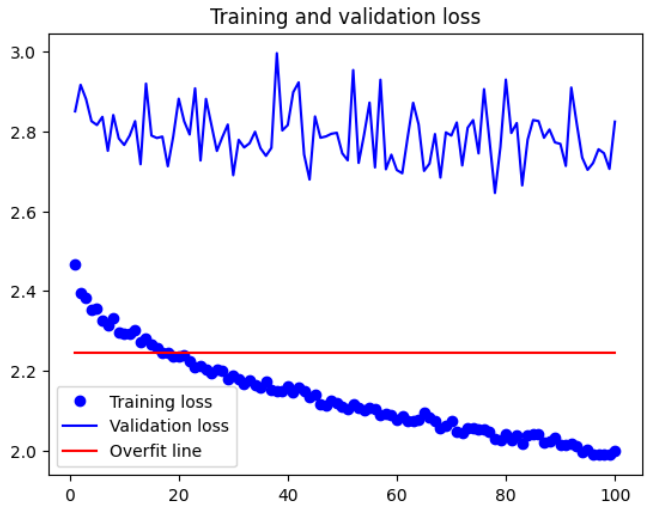
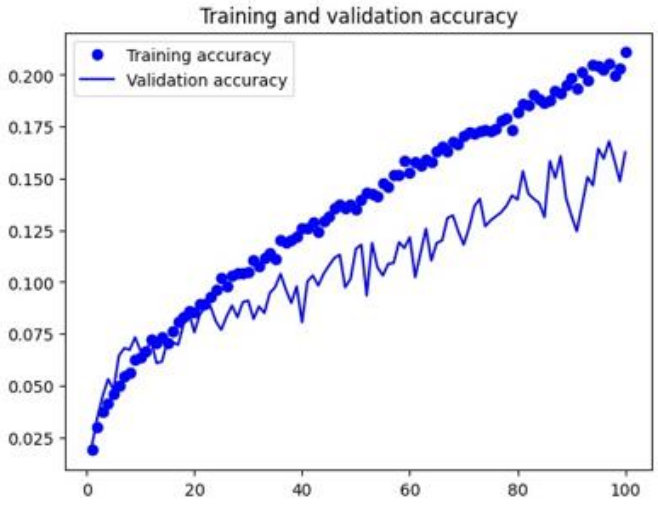


Tabela IV- Multi Input Loss/Acc



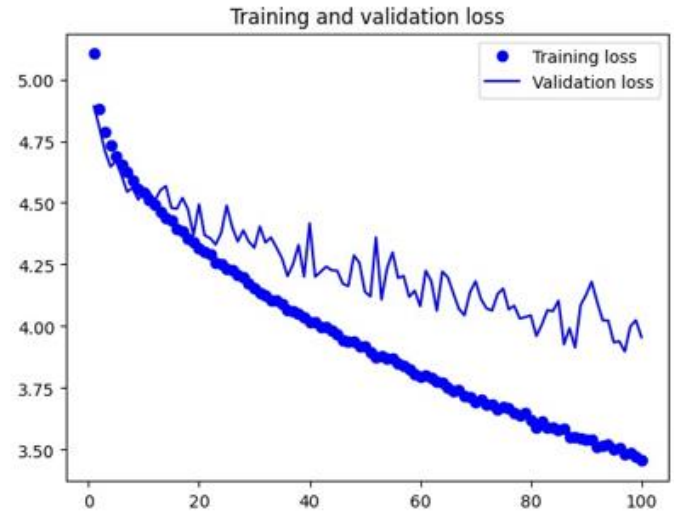


Tabela V - Model6 CNN Loss/Acc

Model Name	layers	epochs	epoch_when_model_saved	filters	optimizer	batch	loss	f1_weighted	Accuracy	DIFF BETWEEN LOSS	DIFF BETWEEN F1	val_loss	val_f1_weighted	val_accuracy	loss_test	f1_weight_test	accuracy_test	Observações
model2	2 layers(256, 128)+conv+output	100	68	(32,64,128,128)	SGD(lr=0.0001, momentum=0.5	32	3.61666	0.0435	0.135	0.51564	0.0099	4.1323	0.0342	0.116	4.057742596	0.0052	0.11572581	
model6	2 layers(256, 128)+conv+output	100	83	32, 32,64,64,128	Adam(lr=0.0001)	32	3.2492	0.14855	0.25677	0.4021	-0.00235	3.6513	0.1509	0.21	3.782980032	0.2256	0.23145182	
model6_3	1 layer 128+conv+output	100	56	32, 32,64,64,128	Adam(lr=0.0001)	128	3.2578	0.1427	0.2598	0.4492	0.0156	3.707	0.1271	0.23	3.943766932	0.2016	0.21532258	
model6_5	2 layers(256, 128)+conv+output	100	64	32,32,64,64	Adam(lr=0.0001)	32	2.8619	0.2406	0.3138	0.7376	0.0618	3.6196	0.1794	0.23	4.46445412	0.2387	0.16411291	
model6_4	1 layer 128+conv+output	100	46	32,32,64,65	Adam(lr=0.0001)	32	3.0631	0.1759	0.2795	0.7376	0.0405	3.8007	0.1354	0.21	4.087746143	0.2898	0.21532258	No loss encontramos overfit, mas nas outras métricas até está balanceado.
model6_9	2 layers(256, 128)+conv+output	100	84	32, 32,64,64,128	SGD(lr=0.0001, momentum=0.9	32	3.3282	0.1347	0.2442	0.3625	0.0211	3.6907	0.1136	0.2	3.899428606	0.2301	0.18185484	

Modelos Multi-input	Branch Convulução	Fully Connected Brunch	Depois de concatenar	epochs	batch	optimizer	loss	f1_weighted	accuracy			val_loss	val_f1_weighte	val_accur			
modelo 1	Input+RandTranslation+RandomRotation+RandomZoom+Conv2D(32)+Conv2D(32)+Conv2D(64)+Conv2D(32)+Conv2D(128)+	Input+Lambda+Dense(128)+Dense(128)+Dense(128)+Dense(128)	128+114	60		Adam(lr=0.0001)	2.1145	0.428	0.4378	0.6049	0.1099	2.7194	0.3181	0.3	2.711	0.3045	0.3233
modelo 2	Input+RandTranslation+RandomRotation+RandomZoom+Conv2D(32)+Conv2D(32)+Conv2D(64)+Conv2D(32)+Conv2D(128)+	Input+Lambda+Dense(128)+Dense(128)+Dense(128)+Dense(128)	128++(existe um dropout)+114	100(resultados da 78)	32	Adam(lr=0.0001)	2.0274	0.4503	0.4602	0.6183	0.1246	2.6457	0.3257	0.34	2.6044	0.3393	0.3513

Model name	layers	batch size	epochs	optimizer	train_loss	val_loss	val_accuracy	DIFF BETWEEN LOSS	f1_weighted test	accuracy_test
V0016 Fine-tuning	Freeze until the last 4 layers+Flatten()+D	32	20	Adam(learning_rate=0.0001)	2.6224	3.4841	0.2332	0.8617	0.2092	22.78%
V0016 Transfer Learning	Freeze all+Flatten()+Dense(512, activati	32	20	Adam(learning_rate=0.0001)	2.6373	3.4766	0.2259	0.8393	0.2081	22.74%
ResNet101	Freeze all+GlobalAveragePooling2D()+D	32	20	Adam(learning_rate=0.001)	4.3873	4.3885	0.0491	0.0012	0.0165	4.80%
DenseNet121	Freeze all+GlobalAveragePooling2D()+D	32	20	Adam(learning_rate=0.001)	1.185	3.7778	0.3161	2.5928	0.3066	31.01%

Tabela VI- Results from the evaluated models and some other models

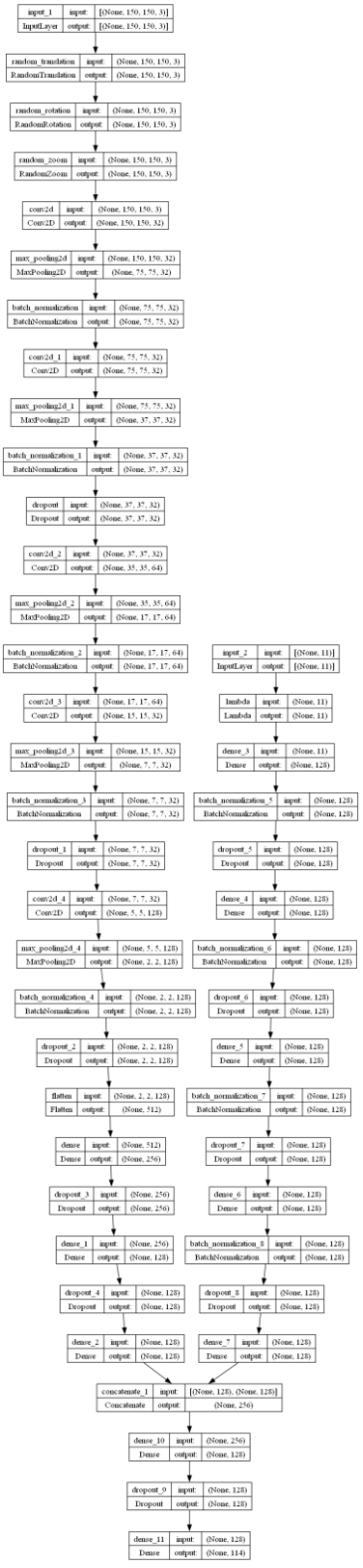


Tabela VII- Multi Input Architecture

```
'seborrheic dermatitis': {'exemplos_treino_classe_seborrheic dermatitis': 88,
 'vezes_que_acertou': 2,
 'taxa_de_erro': 0.8888888888888888,
 'vezes_que_errou': 16,
 'classe_mais_errada': 'allergic contact dermatitis',
 'vezes_classe_mais_errada': 5,
 'treino_exemplos_classe_allergic contact dermatitis': 294},
'pilomatricoma': {'exemplos_treino_classe_pilomatricoma': 34,
 'vezes_que_acertou': 1,
 'taxa_de_erro': 0.875,
 'vezes_que_errou': 7,
 'classe_mais_errada': 'telangiectases',
 'vezes_classe_mais_errada': 3,
 'treino_exemplos_classe_telangiectases': 84},
'malignant melanoma': {'exemplos_treino_classe_malignant melanoma': 74,
 'vezes_que_acertou': 2,
 'taxa_de_erro': 0.875,
 'vezes_que_errou': 14,
 'classe_mais_errada': 'superficial spreading melanoma ssm',
 'vezes_classe_mais_errada': 7,
 'treino_exemplos_classe_superficial spreading melanoma ssm': 81},
'perioral dermatitis': {'exemplos_treino_classe_perioral dermatitis': 43,
 'vezes_que_acertou': 1,
 'taxa_de_erro': 0.875,
 'vezes_que_errou': 7,
 'classe_mais_errada': 'allergic contact dermatitis',
```

Tabela VIII - Error rate/Identified Pattern(Some Examples)

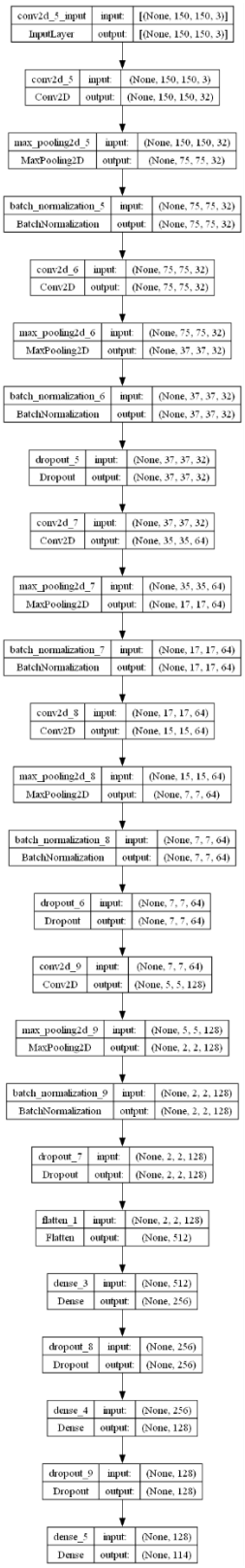


Tabela IX - Model6(Custom Built) Architeture