DM

Abordagens de aprendizagem profunda para deteção de lesões de pele

DISSERTAÇÃO DE MESTRADO

Jonathan José Camacho Vieira
MESTRADO EM ENGENHARIA INFORMÁTICA

UNIVERSIDADE da MADEIRA
A Nossa Universidade
www.uma.pt

março | 2025

Abordagens de aprendizagem profunda para deteção de lesões de pele

DISSERTAÇÃO DE MESTRADO

Jonathan José Camacho Vieira
MESTRADO EM ENGENHARIA INFORMÁTICA

ORIENTAÇÃO
Fernando Manuel Rosmaninho Morgado Ferrão Dias

COORIENTAÇÃO
Fábio Ruben Silva Mendonça

Faculdade de Ciências Exatas e da Engenharia

Mestrado em Engenharia Informática

Abordagens de aprendizagem profunda para deteção de lesões de pele

Jonathan José Camacho Vieira

Supervisor:

Fernando Manuel Rosmaninho Morgado Ferrão Dias

Co-Supervisor:

Fábio Ruben Silva Mendonça

Funchal - Portugal

Março 2025

Resumo

Nos últimos anos, tem-se observado um aumento substancial nos casos de cancro de pele. Porém, a deteção precoce de cancro é de grande relevância pois aumenta consideravelmente a probabilidade de cura. Neste contexto, disponibilizar uma ferramenta gratuita de auxílio à deteção de lesões na pele pode contribuir para a redução dos riscos para a população, especialmente para os grupos economicamente vulneráveis. Com o avanço da tecnologia, os dispositivos móveis inteligentes (*smartphones*) tornaram-se mais acessíveis à população em geral, dando a possibilidade de utilizar estes equipamentos para fornecer uma ferramenta de classificação de lesões, alertando o utilizador para a necessidade de consultar um especialista. O uso da inteligência artificial, na classificação de imagens, permite realizar uma análise de forma rápida e económica das lesões da pele, podendo ser utilizado como um sistema de triagem inicial na consulta para um especialista. Este trabalho apresenta uma solução multiplataforma para a classificação de lesões de pele, utilizando *transfer learning* com modelos pré-treinados, ajustados para a classificação de lesões da pele em sete categorias, nomeadamente Queratoses Actínicas (*akiec*), Carcinoma Basocelular (*bcc*), Lesões benignas semelhantes à Queratose (*bkl*), Dermatofibroma (*df*), Melanoma (*mel*), Nevos melanocíticos (*nv*) e Lesões Vasculares (*vasc*). Para tal, 38 arquiteturas de redes neuronais profundas foram analisadas e desenvolvimento das mesmas foi realizado utilizando o conjunto de dados com 10.015 imagens. O melhor desempenho foi alcançado usando a arquitetura ConvNeXtXLarge, obtendo uma exatidão e *f1 score* de 87,62% e 76,15%, respetivamente. Desta forma, a aplicação desenvolvida permite ao utilizador submeter uma imagem, que é enviada ao servidor para classificação, garantindo a privacidade da imagem através de encriptação.

Keywords: Cancro de pele, smartphone, inteligência artificial, rede neuronal

Abstract

In recent years, there has been a significant rise in skin cancer cases. Early detection is crucial, as it considerably increases the likelihood of a cure. In this context, providing a free tool for detecting skin lesions could help reduce health risks, particularly for economically vulnerable groups. With the advancement of technology, smartphones have become more accessible to the general population, giving the possibility of using these devices to provide a skin lesion classification tool alerting the user about the need to consult a specialist, improving early diagnosis rates. The use of artificial intelligence in image classification allows for a quick and cost-effective analysis of skin lesions, and can be used as an initial screening system, assisting individuals in determining when professional medical evaluation is necessary. This work presents a multiplatform solution for skin lesion classification, using *transfer learning* with pre-trained models, fine-tuned for classification of skin lesions into seven categories, namely, Actinic Keratoses (*akiec*), Basal Cell Carcinoma (*bcc*), Benign Keratosis-like Lesions (*bkl*), Dermatofibroma (*df*), Melanoma (*nv*), Melanocytic Nevi (*nv*) and Vascular Lesions (*vasc*). For this purpose, 38 Deep Neural Network architectures were analyzed and their development was performed using the dataset with 10.015 images. The best performance was achieved using the ConvNeXtXLarge architecture, obtaining accuracy and *f1* score of 87.62% and 76.15%, respectively. In this way, the developed application allows the user to send an image, which is sent to the server for classification, ensuring the privacy of the image through encryption.

Keywords: skin Cancer, smartphone, artificial intelligence, Deep Neural Networks

Agradecimentos

Agradeço aos meus orientadores, Professor Doutor Morgado Dias, pelo inestimável apoio e orientação ao longo deste percurso, e ao Professor Doutor Fábio Mendonça, pelas valiosas sugestões e conselhos que contribuíram significativamente para o desenvolvimento deste trabalho.

Dirijo um agradecimento especial à minha esposa, Mariana, pela sua paciência, compreensão e constante apoio, fundamentais para a conclusão dos meus estudos.

À minha filha, por me proporcionar tantos momentos de alegria e ternura, especialmente nos períodos mais difíceis deste percurso.

À minha família, os meus pais, à minha irmã e ao meu sobrinho, pelo encorajamento, apoio e palavras de incentivo nos momentos mais desafiadores.

Agradeço a todos os professores pelo apoio, dedicação e partilha de conhecimento ao longo do meu percurso.

.

Índice

Índice de Figuras **viii**

Índice de Tabelas **x**

1 Introdução 1
1.1 Problema 2
1.2 Objetivo 2
1.3 Organização da dissertação 3
2 Estado da Arte - Trabalhos anteriores 4
2.1 Pontos-Chave 6
3 Redes Neuronais 8
3.1 Inteligência Artificial 8
3.2 Redes Neuronais Artificiais 9
3.2.1 Neurónios Artificiais 9
3.2.2 Funções de Ativação 10
3.2.3 Função de ativação Linear 10
3.2.4 Função de ativação Sigmoide 10
3.2.5 Função de ativação de Tangente Hiperbólica 11
3.2.6 Função de ativação Unidade Linear Rectificada . . 11
3.2.7 Função de ativação Softmax 12
3.3 Otimizadores 12
3.3.1 Descida do gradiente estocástica 13
3.3.2 Adagrad 13
3.3.3 RMSProp 14
3.3.4 ADAM 14
3.4 Redes Neuronais Convolucionais 14
3.4.1 VGG 16 15
3.4.2 VGG 19 16
3.4.3 InceptionV3 17
3.4.4 Resnet 18
3.4.5 DenseNet 20

3.4.6 InceptionResNet 21
3.4.7 MobileNet 23
3.4.8 Xception 24
3.4.9 NasNet 25
3.4.10 EfficientNet 28
3.4.11 ConvNext 30
3.4.12 Transferência de conhecimento (*Transfer Learning*) 32
3.5 Pontos-chave 33
4 Arquitetura do sistema 34
4.1 *TFLite* no dispositivo móvel 34
4.2 Arquitetura Cliente-Servidor 34
4.3 Arquitetura implementada 35
4.4 Requisitos Funcionais 36
4.5 Requisitos Não Funcionais 36
4.6 Regulamento Geral sobre a Proteção de Dados 37
4.7 Segurança da arquitetura 37
4.8 Análise de requisitos de segurança 38
4.9 Ambiente de desenvolvimento 38
4.10 Criação do certificado para o servidor 38
4.10.1 Encriptação da imagem 40
4.11 Autenticação via *Token* 43
4.12 Aplicação para Dispositivos móveis 43
4.13 Pontos-chave 44
5 Implementação do Modelo 47
5.1 Materiais 47
5.2 Bibliotecas e Hyperparametros 51
5.2.1 Regularização L2 53
5.2.2 ReduceLROnPlateau 54
5.2.3 EarlyStopping 54
5.3 Implementação do Modelo 55
5.4 Exemplo de um Modelo 58
5.5 Pontos-chave 59
6 Resultados 60
6.1 Métricas 60
6.1.1 Matriz de confusão 60
6.1.2 ACC Macro 64
6.1.3 F1 Macro 66

6.1.4 FNR (False Negative Rate) - Taxa de Falsos Negativos . . . 68
6.1.5 TPR (True Positive Rate) - Taxa de Verdadeiros Positivos . . . 68
6.1.6 FPR (False Positive Rate) - Taxa de Falsos Positivos 69
6.1.7 TNR (True Negative Rate) - Taxa de Verdadeiros Negativos . . . 70
6.1.8 Overall MCC . . . 73
6.1.9 Overall ACC . . . 75
6.1.10 AUC . . . 77
6.1.11 Avaliação de Gráficos de *ACC* e *Loss* . . . 79
6.2 Descrição de Hardware utilizado . . . 84
6.2.1 Limitações de Hardware . . . 84
6.3 Validação dos requisitos . . . 85
6.4 Pontos-chave . . . 87
7 Conclusão . . . 89
7.1 Trabalhos Futuros . . . 90
8 Anexos . . . 91
8.1 Comparação do Tamanho dos Modelos com Regularização L2 91
8.2 Tabela de Camadas e Parâmetros para cada modelo . . . 92
8.3 Código . . . 97

References **125**

Índice de Figuras

1 Inteligência Artificial. 8
2 Modelo não linear de um Neurónio artificial. 9
3 Gráfico da função Linear. 10
4 Gráfico da função Sigmoide. 11
5 Gráfico da função Tanh. 12
6 Gráfico da função Relu. 13
7 Apresentação da arquitetura VGG16. 16
8 Apresentação da arquitetura VGG19. 17
9 Apresentação dos Modulos Inception. a) Modulo Inception Naïve. b) Modulo Inception com redução de dimensão. Adaptado de [1]. . . 17
10 Apresentação da arquitetura Inception V3 [2]. 18
11 Arquitectura Resnet50 adaptado de [3], com imagens do HAM10000 [4]. 19
12 Apresentação da arquitetura Densenet [5]. 20
13 Apresentação da arquitetura InceptionResnetV2 [6]. 22
14 Convoluções separáveis em profundidade *(Depthwise Separable Convolutions)*, adaptado de [7] [8]. 23
15 Apresentação da arquitetura da rede MobileNetV2 [9]. 24
16 Apresentação da arquitetura Xception [10]. 25
17 Apresentação da arquitetura NASNet adaptada de [11]. 26
18 Processo de busca automatizada de células convolucionais [11]. . . 27
19 Exemplo de célula arquitetural gerada [11]. 28
20 Escalonamento da EfficientNet, adaptado de [12]. a) Representação básica. b) Escalonamento em largura. c) Escalonamento em Profundidade. d) Escalonamento em resolução da imagem. e) Escalonamento em Largura, profundidade e resolução. 29
21 Apresentação da arquitetura EfficientNetB0 adaptado de [13]. . . . 29
22 Diferença entre bloco Residual e Bloco Residual Invertido [14]. . . . 30
23 Mecanismos de atenção de [15]. 31
24 Apresentação da arquitetura ConvNext [16]. 32
25 Arquitetura Cliente Servidor. 35

26 Solução proposta para o processo de classificação multiplataforma. . 36
27 Ecrã da aplicação da opção de configuração do *Token*. 43
28 Ecrã inicial da aplicação para classificação de lesões de pele. 44
29 Ecrã da aplicação para a opção de classificação. 44
30 Ecrã de seleção de imagem da galeria. 45
31 Ecrã da aplicação para classificar a imagem selecionada. 45
32 Ecrã da aplicação com o resultado da classificação. 46
33 Tipos de lesões a partir do conjunto de imagens HAM10000 [4]: A) Actinic keratoses, B) Basal cell carcinoma, C) Benign keratosis like, D) Dermatofibroma, E) Melanocytic nevi, F) Vascular Lesion, G) Melanoma. 47
34 Data augmentation, transformações de imagem do HAM10000 [4]. . 49
35 Dropout [17]. 53
36 Precisão e Revocacão [18] 61
37 Resultado da Matriz de confusão com dados de Teste. Sem Regularização L2 62
38 Resultado da Matriz de confusão com dados de Teste. Com Regularização L2 62
39 AUC CURVE por Classe, com Regularizacão L2 78
40 AUC por Classe sem Regularizacão L2 79
41 Treino e Validação - ACC, Loss com Regularização L2 80
42 Treino e Validação - ACC, Loss sem Regularização L2 80
43 Dados Treino com Regularização L2 81
44 Dados Treino sem Regularização L2 81
45 Treino e Validação - ACC com Regularização L2 82
46 Treino e Validação - ACC sem Regularização L2 82
47 Validação - ACC, Loss com Regularização L2 83
48 Validação - ACC, Loss sem Regularização L2 83

Índice de Tabelas

I	Quantidade de imagens por Lesões do conjunto HAM10000 [4].	48
II	Quantidade de imagens para Treino.	50
III	Quantidade de imagens para Validação.	50
IV	Quantidade de imagens para Teste.	51
V	Bibliotecas Utilizadas.	52
VI	Exemplo de Regularização L2.	54
VII	Modelo ConvNeXtXLarge com as camadas e número parâmetros.	59
VIII	Matriz de confusão.	60
IX	Taxa de Classificações Corretas com Regularização L2.	63
X	ACC Macro dos modelos.	65
XI	Resultado da métrica F1 Macro.	67
XII	Resultado das métricas FNR, TNR, FPR, TPR sem L2.	71
XIII	Resultado das métricas FNR, TNR, FPR, TPR com L2.	72
XIV	Resultados da métrica Overall MCC.	74
XV	Resultados da métrica Overall ACC.	76
XVI	Resultados da métrica AUC.	79
XVII	Caraterísticas do PC de treino.	84
XVIII	Caraterísticas do Ambiente Colab de treino.	85
XIX	Resultados Requisitos Funcionais.	85
XX	Resultados Requisitos Não Funcionais.	86
XXI	Tamanho dos Modelos ConvNeXt.	91
XXII	Tamanho dos Modelos EfficientNetB0-B3.	91
XXIII	Tamanho dos Modelos EfficientNetB4-B7.	91
XXIV	Tamanho dos Modelos EfficientNetV2.	91
XXV	Tamanho dos Modelos RESNET.	91
XXVI	Tamanho dos Modelos DenseNet, MobileNet e Xception.	92
XXVII	Tamanho dos Modelos Inception, NasNet e VGG.	92
XXVIII	Camadas das redes VGG16 e VGG19.	92
XXIX	Camadas das redes MobileNet e MobileNetV2.	93
XXX	Camadas das redes NASNetMobile e NASNetLarge.	93
XXXI	Camadas das redes InceptionV3 e InceptionResNetV2.	93

XXXII Camadas das redes DenseNet121 e DenseNet169. 93
XXXIII Camadas da rede DenseNet201. 94
XXXIV Camadas das redes RESNET50 e RESNET50V2. 94
XXXV Camadas das redes RESNET101 e RESNET101V2. 94
XXXVI Camadas das redes RESNET152 e RESNET152V2. 94
XXXVII Camadas das redes ConvNeXtTiny e ConvNeXtSmall. 95
XXXVIII Camadas das redes ConvNeXtBase e ConvNeXtLarge. 95
XXXIX Camadas das redes EfficientNetB0 e EfficientNetB1. 95
XL Camadas das redes EfficientNetB2 e EfficientNetB3. 95
XLI Camadas das redes EfficientNetB4 e EfficientNetB5. 96
XLII Camadas das redes EfficientNetB6 e EfficientNetB7. 96
XLIII Camadas das redes EfficientNetV2-B0 e EfficientNetV2-B1. 96
XLIV Camadas das redes EfficientNetV2-B2 e EfficientNetV2-B3. 96
XLV Camadas das redes EfficientNetV2-S e EfficientNetV2-M. 97
XLVI Camadas das redes EfficientNetV2-L e Xception. 97

Lista de Acrónimos

1D Uma Dimensão

2D Duas Dimensões

3D Três Dimensões

ACC Accuracy

ADAM Adaptive Moment Estimation

ADN Ácido Desoxirribonucleico

AI Artificial Intelligence

akiec Actinic keratosis

ANN Artificial Neural Network

API Interface de Programação de Aplicações

bcc Basal cell carcinoma

bkl Benign keratosis like

CNN Convolutional Neural Networks

df Dermatofibroma

DL Deep Learning

DNN Deep Neural Networks

FN False Negative

FP False Positive

FPR False Positive Rate

GD Gradient Descent

GPU Graphics Processing Unit

IA Inteligência Artificial

IDE Integrated Development Environment

MC Matriz de Confusão

MCC Matthews Correlation Coefficient

mel Melanoma

ML Machine Learning - Aprendizagem Automática

NN Neural Network

nv Melanocytic nevi

ReLU Rectified Linear Unit

TL Transfer Learning

TN True Negative

TNR True Negative Rate

TP True Positive

TPR True Positive Rate

TPU Tensor Processor Unit

RMSProp Root Mean Square Propagation

OvR One-vs-Rest

OvA One-vs-All

OvO One-vs-One

PC Personal Computer

SSL Secure Sockets Layer

vasc Vascular Lesion

1 Introdução

O crescente aumento de doenças dermatológicas, especialmente lesões potencialmente malignas, representam um desafio para a saúde pública global. A deteção precoce das lesões de pele pode ser crucial para um diagnóstico e tratamento atempado a lesões como o melanoma, cancro de pele não-melanoma e outras lesões.

A pele é o maior órgão do corpo humano, atuando como barreira protetora em relação ao meio ambiente que contém bactérias ou vírus.

As células da pele renovam-se constantemente por regeneração celular, onde as células velhas morrem e são substituídas por células novas, originadas pela divisão de outras células. Esse processo pode ser afetado por fatores externos ou internos causando mutações no Ácido Desoxirribonucleico (ADN) das células. Essas mutações podem provocar um descontrolo no ciclo de vida das células que continuam a multiplicar-se de forma descontrolada. Essas células são denominadas de células cancerígenas [19].

O cancro de pele pode ser dividido em dois grupos: não melanoma e melanoma. O melanoma é um tipo de cancro de pele maligno que tem início nos melanócitos. Estes são responsáveis por produzir a melanina, pigmento que dá à pele a sua cor natural. [20].

Recentemente, tem-se observado um aumento substancial nos cancros de pele [19]. Porém, a deteção precoce deste tipo de cancro aumenta a probabilidade de cura. Ao disponibilizar ferramentas abertas que auxiliem na deteção de lesões e alertem aos utilizadores para consultar um especialista, pretende-se facilitar a deteção do cancro na fase inicial. Neste contexto, a convergência entre inteligência artificial, especificamente redes neuronais convolucionais (CNNs), e a tecnologia móvel surge como uma solução para democratizar o acesso a ferramentas de triagem.

Desta forma, surge a necessidade de desenvolver uma aplicação com código fonte aberto, que possa ser utilizada em dispositivos móveis (*smartphones*) para avaliar e analisar lesões de pele, tornando o processo acessível à população em geral. Ao incorporar a Inteligência Artificial (IA) na aplicação, será possível identificar o cancro de pele de forma precoce, ajudando na sinalização das lesões na pele.

A solução proposta consiste numa aplicação multiplataforma que estabelece comunicação com um servidor, para onde a imagem é enviada para ser analisada e classificada pelo modelo, permitindo a classificação de lesões de pele. O modelo implementado no servidor foi selecionado após uma série de testes, sendo este o que apresentou maior exatidão entre todos os modelos treinados e avaliados.

O modelo desenvolvido é capaz de identificar sete tipos distintos de lesões cutâneas: Actinic keratoses, Basal cell carcinoma, Benign keratosis-like lesions, Dermatofibroma, Melanocytic nevi, Vascular lesions e Melanoma. A classificação é realizada com base em probabilidades, ou seja, o modelo atribui a cada lesão uma distribuição probabilística correspondente à probabilidade de pertencer a cada uma das sete classes.

O modelo final foi selecionado após a avaliação comparativa de 38 arquiteturas diferentes de redes neuronais convolucionais pertencentes a diferentes famílias de redes convolucionas: ConvNeXt, DenseNet, EfficientNet, Inception, InceptionResNet, MobileNet, NASNet, ResNet, VGG e Xception, sendo esta analise não identificada em outros trabalhos anteriores.

1.1 Problema

O crescente aumento de lesões de pele, nomeadamente lesões potencialmente malignas, tem revelado alguns desafios na deteção precoce das lesões. O acesso a um diagnostico especializado pode implicar custos e o avanço tecnológico pode ajudar a um acesso mais democratizado na ajuda da deteção precoce, alertando para a necessidade de consultar um especialista atempadamente.

O problema central reside na ausência de uma ferramenta acessível, que permita realizar uma triagem inicial de lesões de pele, identificando de forma precoce lesões malignas e desta forma democratizar o acesso a tecnologias de diagnóstico acessíveis.

A investigação proposta surge como resposta a este problema, visando desenvolver uma solução tecnológica que mitigue o problema do diagnóstico precoce de lesões, através da inteligência artificial, com a implementação de redes neuronais e tecnologia móvel, classificando 7 tipos diferentes de lesões de pele, com uma solução baseada em *transfer learning*. É utilizada uma arquitetura de rede neuronal convolucional, pré-treinada em um grande conjunto de dados de imagens gerais, e com camadas adicionais para a tarefa específica de classificação de lesões de pele. Esta abordagem permite aproveitar o conhecimento geral de reconhecimento de padrões da rede pré-treinada, para identificar características únicas de diferentes tipos de lesões.

1.2 Objetivo

O presente projeto visa realizar uma avaliação de modelos de CNNs para classificação de lesões de pele, selecionando o modelo com melhor desempenho utilizando a métrica Área sob a curva caraterística de funcionamento do recetor (AUC).

Propõe-se então o desenvolvimento de uma arquitetura Cliente-Servidor, na qual o cliente realizará a captura de imagens de lesões pele, através de um dispositivo móvel, e será transmitida para um servidor que será responsável pelo processamento da imagem, utilizando a CNN selecionada para classificar a lesão. O servidor retornará ao dispositivo móvel as classificações da lesão e correspondente probabilidade para cada uma.

A solução proposta consiste na implementação de um modelo de CNNs que apresente o melhor desempenho, sendo definido como mínimo adequado uma exatidão de 80%, e apresentar uma solução tecnológica que integra a captura de imagem num dispositivo móvel com o processamento num servidor que realiza a classificação utilizando inteligência artificial.

O objetivo é criar uma prova de conceito tecnológica que demonstre a viabilidade de um sistema de deteção de lesões cutâneas baseado em inteligência artificial, com foco na seleção de um modelo de CNN com melhor desempenho integrado numa uma arquitetura cliente-servidor.

1.3 Organização da dissertação

Este trabalho está dividido em 6 capítulos:

- Capitulo 2, apresenta os trabalhos anteriores realizados referente ao tema desenvolvido nesta investigação.
- Capitulo 3, apresenta os conceitos de Inteligência artificial, de transferência de conhecimento e as Redes Neuronais utilizadas neste trabalho.
- Capitulo 4, onde são descritos os requisitos da aplicação e a arquitetura implementada na solução.
- Capitulo 5, onde se explicam os dados utilizados e a implementação do modelo.
- Capitulo 6, apresenta uma análise dos resultados obtidos através das diferentes métricas de avaliação aplicadas ao modelo de classificação de lesões.
- Capitulo 7, que aborda os avanços obtidos durante a pesquisa e apresenta possíveis desenvolvimentos futuros .

2 Estado da Arte - Trabalhos anteriores

A análise do estado da arte permite avaliar os trabalhos anteriores e desta forma extrair informações relevantes que possam ser utilizadas neste trabalho. Verifica-se que existem vários trabalhos na classificação de lesões de pele utilizando redes neuronais com múltiplas camadas escondidas e as características mais relevantes são indicadas nos parágrafos seguintes.

No trabalho de Mahbod et al. [21], para a obtenção de características foram utilizados os modelos pré-treinados AlexNet, VGG16, e ResNet18, para depois realizar a classificação com uma máquina de vetores de suporte. A classificação proposta foi avaliada com 150 imagens obtendo 83.83% de classificações corretas para Melanoma e 97.55% para Keratosis.

No trabalho de Hekler et al. [22], foi utilizado um modelo Resnet50 (rede neuronal convolucional), e explora os benefícios de combinar a inteligência artificial com a intervenção humana na classificação dos cancros da pele. Foram utilizadas 11444 imagens divididas em cinco categorias, foi treinada uma rede neuronal convolucional para classificar as lesões e posteriormente, 112 dermatologistas de 13 hospitais universitários alemães e a CNN classificaram independentemente um conjunto de 300 lesões, as decisões de ambos foram combinadas e no teste de classificação, a combinação alcançou uma precisão de 82,95%, superando a CNN 81,59%. Estes resultados indicam que a colaboração entre especialistas e inteligência artificial pode melhorar a classificação das lesões da pele.

Outro dos trabalhos que foram analisados foi o de Amal Al-Rasheed et al. [23], que propõe um método para a classificação de diversos tipos de cancro da pele, utilizando modelos pré-treinados, nomeadamente VGG16, ResNet50 e ResNet101. Para mitigar o desequilíbrio de classes no conjunto de dados de treino, os autores aplicaram técnicas de aumento de dados, como transformações de imagem, e a geração imagens dermatoscópicas realistas. Os modelos pré-treinados foram ajustados e treinados, os modelos VGG16, ResNet50 e ResNet101 alcançaram uma precisão de 92%, 92% e 92,25% respetivamente. Os resultados sugerem que uma combinação ou *ensemble* de modelos de *transfer learning*, em conjunto com a técnica de aumento de dados, pode melhorar o desempenho na classificação das lesões de pele.

O trabalho de Soenksen et al. [24] a diferença do anterior é que utiliza CNNs para identificar lesões de pele, incluindo aquelas capturadas por câmaras de telemóveis. O sistema analisa imagens de grandes áreas da pele, e deteta lesões potencialmente malignas, este sistema foi treinado com 38283 lesões. Conseguindo uma precisão comparável ao de dermatologistas. O sistema demonstrou uma concordância de 82,96%

com pelo menos uma das três principais lesões, com os especialistas que avaliaram e validaram os resultados.

Similar aos trabalhos anteriores, o trabalho de Danilo Barros Mendes e Nilton Correia da Silva [25], realiza uma classificação de lesões de pele utilizando um modelo de classificação para 12 tipos de lesões. Este modelo é uma arquitetura ResNet-152, treinada com 3797 imagens que foram aumentadas através de transformações de posição, escala e iluminação, obtendo uma AUC de 96% para Melanoma e 91% para Carcinoma Basocelular.

O trabalho realizado por Kartikeya Agarwal e Tismeet Singh [26], efetuam uma classificação binária de lesões pele através de redes neuronais convolucionais, utilizando um conjunto de dados do repositório ISIC (International Skin Imaging Collaboration), composto por 2947 imagens divididas em benignas e malignas. As imagens foram redimensionadas e aplicaram de técnicas de aumento de dados, dividindo o conjunto de dados em 2900 imagens para treino e 350 para teste. Foram treinados varios modelos: DenseNet, XceptionNet, ResNet and MobileNet, obtendo uma exatidão de 86,65% na classificação das imagens.

Ao verificar trabalhos que realizam comparação de múltiplas Redes Convolucionais encontramos o trabalho realizado por Mst Shapna Akter et al. [27], é realizada uma abordagem de deep learning para a classificação de diferentes tipos de lesões de pele. Foi aplicado um modelo de Rede Neuronal Convolucional e seis modelos por *transfer learning* (ResNet50, VGG16, DenseNet, MobileNet, InceptionV3 e Xception), ao conjunto de dados HAM10000 [4]. Os modelos foram treinados obtendo exatidões de 90% para o InceptionV3, 88% para o Xception e DenseNet, 87% para o MobileNet, 82% para o ResNet50 e 77% para o CNN e VGG16. Adicionalmente, desenvolveram modelos em que combinam diferentes arquiteturas, mas apresentaram desempenhos de 78%.

No trabalho de Daniel Alonso Villanueva Nunez e Yongmin Li [28], são utilizadas as redes neuronais convolucionais DenseNet121, VGG16 com normalização em batch e ResNet50 para o diagnóstico de lesões de pele. Os modelos foram treinados para classificar entre lesões benignas e malignas utilizando o conjunto de dados HAM10000. O melhor modelo encontrado foi o ResNet50, que obteve um *recall* para lesão Queratoses Actínicas (*akiec*) de 69%, 93% para a lesão Carcinoma Basocelular (*bcc*) e 76% para a lesão Melanoma (*mel*). Quando o modelo foi ajustado para a classificação binária, o ResNet50 consegue uma sensibilidade de 92,35%, mas o VGG16 consegue uma sensibilidade de 95,40%.

Com o avanço da tecnologia os *smartphones* estão mais acessíveis, permitindo a sua utilização no desenvolvimento de aplicações médicas. No trabalho realizado por Ahmed et al. [29] foi desenvolvida uma aplicação para o sistema operativo iOS. Esta aplicação permitia avaliar lesões utilizando um modelo de CNN denominada MobileNetV2. Foram utilizadas 48373 imagens para treinar o modelo que realizava a classificação das lesões como benigna ou maligna, tendo uma exatidão de 91,33%.

Outro trabalho que utiliza os *smartphones* na solução é o trabalho de Oztel at al. [30], foi desenvolvida uma aplicação para o sistema operativo Android com o propósito de distinguir as lesões da Varíola dos macacos de outras lesões. Foram utilizadas diferentes redes pré-treinadas e escolhidas as redes com melhor resultado e adaptação as aplicações móveis. A rede ResNet18 foi a rede utilizada com uma exatidão de 74,27% e transformada em formato TensorFlow Lite para ser utilizada na aplicação Android.

Outro dos trabalhos desenvolvidos para Android foi o apresentado por Francese et. al. [31]. Esta aplicação classifica em tempo real as lesões como Melanoma ou "Não Melanoma". Devido ao conjunto de dados que utilizaram não estar balanceado, a exatidão do modelo utilizado foi de 78,8%. No trabalho foram tomadas em conta a assimetria da lesão, o rebordo ou segmentação da lesão, as cores da lesão, o diâmetro e a evolução da lesão. Adicionalmente, foram utilizados os formatos RGB e HSV (*Hue Saturation Value*) das imagens.

Outra solução para aplicações móveis foi proposta por Hameed et al. [32], utilizando uma arquitetura em nuvem, em que o modelo treinado é colocado num servidor e a aplicação do *smartphone* envia a imagem para ser analisada no servidor. O modelo foi desenvolvido usando uma rede neuronal convolucional chamada SqueezeNet, sendo treinada e testada em 1856 imagens. Esta solução apresentava uma exatidão na classificação de 97,21% para 4 categorias: pele saudável, Acne, Eczema e Psoríase.

2.1 Pontos-Chave

Os trabalhos realizados utilizam redes neuronais para classificação de imagens, aproveitando algumas técnicas, como o *transfer learning*, para treinar e melhorar o desempenho dos modelos. A qualidade dos conjuntos de dados é uma peça importante no treino do modelo, no caso de desbalanceamento dos dados, são realizadas transformações das imagens para aumentar o número de amostras a ser processadas pelo modelo, balanceando o conjunto de dados e verificando que as diferenças na iluminação, rotação e qualidade das imagens afetam o desempenho dos modelos. Confirma-se que as CNNs apresentam em geral o melhor desempenho, mas é

necessário analisar qual a arquitetura mais adequada. Outra necessidade é a de ser usada uma base de dados de maior dimensão de forma a validar adequadamente os modelos e a necessidade de recurso de técnicas de aumento dos dados de treino.

No desenvolvimento de aplicações, são realizadas soluções cliente-servidor e modelos transformados que executam localmente nos dispositivos móveis. O processamento em servidor pode garantir que a execução dos modelos originais mantém as métricas avaliadas. O TFlite pode, em alguns casos, apresentar menor desempenho mas oferece vantagens ao executar localmente em dispositivos móveis.

3 Redes Neuronais

Com base no conhecimento das capacidades do cérebro humano, surgiram diversas iniciativas para simular o funcionamento do mesmo. As redes neuronais artificiais são uma das iniciativas e têm contribuído no campo da inteligência artificial com o objetivo de emular seu desempenho e conseguir resolver problemas como o analise e classificação de imagens.

Este capítulo explora os diversos modelos de redes neuronais utilizados, fornecendo uma visão geral das diferentes abordagens em redes neuronais profundas.

3.1 Inteligência Artificial

Nos últimos dez anos, a IA tem sido cada vez mais utilizada em diversas áreas. A IA procura simular a inteligência humana para encontrar soluções através da aprendizagem, nesta área destaca-se a aprendizagem automática (ML), que é uma subárea da IA, conforme indicado na Figura 1. O *ML* permite criar sistemas capazes de aprender com os dados sem serem explicitamente programados para uma determinada tarefa. Um exemplo do seu uso é a condução autónoma de veículos, que utiliza imagens captadas por câmaras e outros sensores para navegar no trânsito [33].

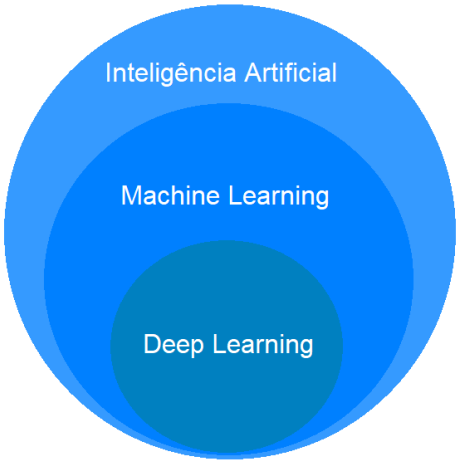

Fig. 1: Inteligência Artificial.

Os sistemas de *ML* podem ser tipicamente enquadrados nas categorias supervisionado, não supervisionado e por reforço. Neste trabalho é utilizada uma abordagem supervisionada, tendo em conta que os dados disponibilizados foram anotados por especialistas.

A aprendizagem supervisionada considera variáveis de entrada **(X)** e de saída **(Y)** e emprega um algoritmo para aprender a função de mapeamento a partir da

entrada para a saída $Y = f(X)$. O objetivo é aproximar a função de mapeamento de forma que ao introduzir novos dados de entrada seja possível prever as variáveis de saída para esses dados [34].

3.2 Redes Neuronais Artificiais

Com base no conhecimento das capacidades do cérebro humano, surgiram diversas iniciativas para reproduzir o o funcionamento do cérebro. O objetivo é emular o seu desempenho nas tarefas, como análise e reconhecimento de imagens, que são um desafio para os computadores.

3.2.1 Neurónios Artificiais

Os neurónios artificiais são as unidades básicas de um modelo de rede neuronal artificial, e são inspirados no funcionamento dos neurónios biológicos.

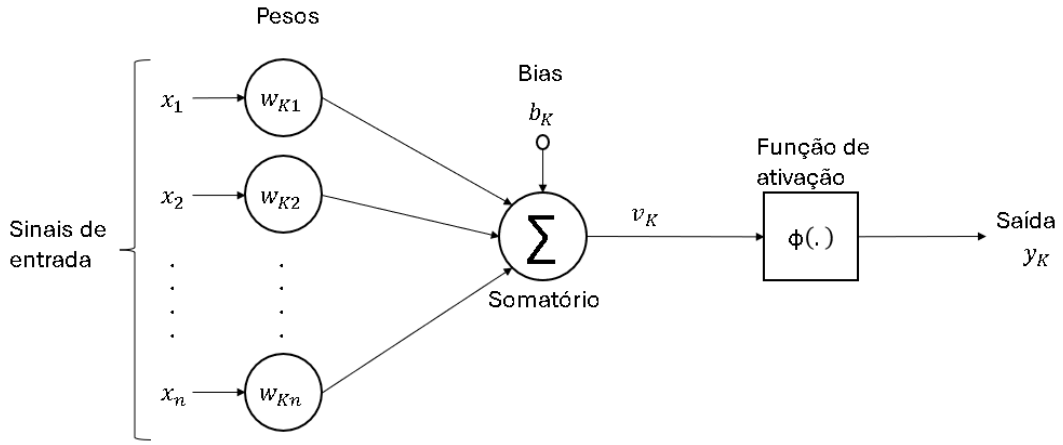

Fig. 2: Modelo não linear de um Neurónio artificial.

Cada entrada x_k é multiplicada pelo seu peso w_k. O neurónio calcula à soma ponderada dessas entradas, adicionando um termo de Bias b. O Bias é um valor que ajuda a ajustar a saída do neurónio, deslocando a função de ativação para cima ou para baixo.

O neurónio artificial k pode ser descrito, em termos matemáticos, por

$$v_k = \sum_{j=1}^{n} (w_{kj} x_j) + b_k. \tag{1}$$

3.2.2 Funções de Ativação

As funções de ativação são funções aplicadas nas saídas do neurónio, que permitem a passagem da soma ponderada dos valores de entrada para a próxima camada, introduzindo a não-linearidade nas saídas dos neurónios, permitindo que a rede aprenda e modele relações complexas entre os dados de entrada e saída. As principais funções de ativação vão ser seguidamente analisadas.

3.2.3 Função de ativação Linear

A função linear aplica um fator de multiplicação ao valor de entrada [35], definida como

$$f(x) = ax + c. \tag{2}$$

A representação gráfica desta função é apresentada na Figura 3. A derivada desta função é constante pelo o gradiente não tende a convergir de forma a encontrar um erro próximo de zero [35].

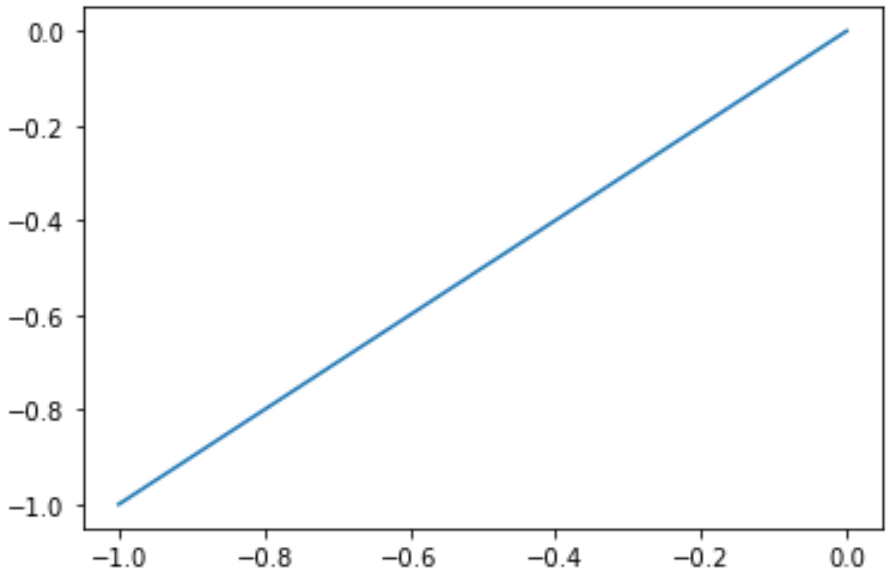

Fig. 3: Gráfico da função Linear.

3.2.4 Função de ativação Sigmoide

A função Sigmoide gera valores no intervalo [0,1]. Quanto maior for o valor de entrada, mais próximo o valor retornado será de 1 e quanto menor for a entrada o valor de saída mais próximo o valor da saída será de 0 [35] [36]. Esta função é dada por

$$f(x) = \frac{1}{1 + e^{-x}}. \tag{3}$$

O gráfico desta função é apresentado na Figura 4.

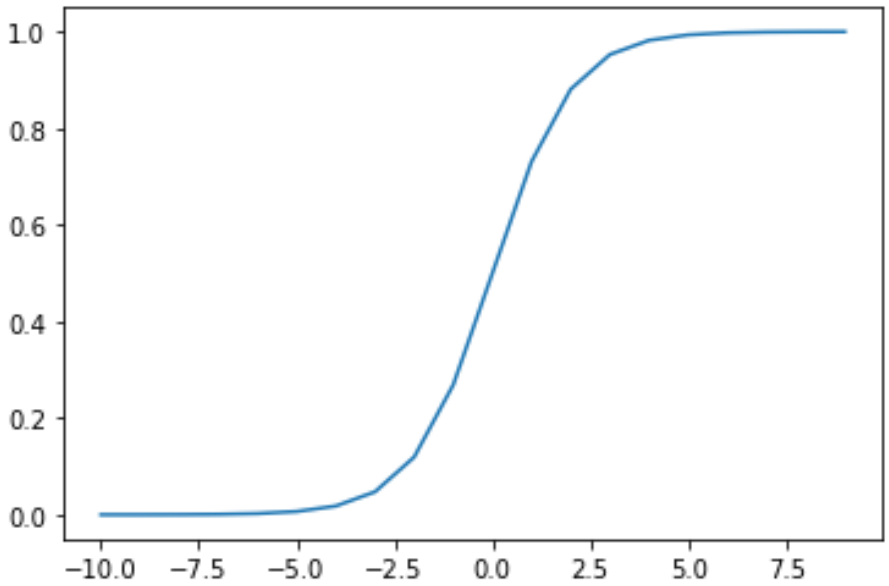

Fig. 4: Gráfico da função Sigmoide.

3.2.5 Função de ativação de Tangente Hiperbólica

A função tangente hiperbólica produz valores no intervalo [-1, 1], sendo similar a função de ativação Sigmoid. Quanto maior for o valor de entrada o valor retornado será próximo de 1 e quanto menor for a entrada o valor de saída será próximo de -1 [36]. Esta função é dada por

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}. \tag{4}$$

O gráfico desta função é apresentado na Figura 5.

3.2.6 Função de ativação Unidade Linear Rectificada

A Unidade Linear Rectificada (ReLU) retorna valores entre 0 e infinito. Para valores de entrada inferiores a zero a função retorna 0, e para entradas superiores ou igual

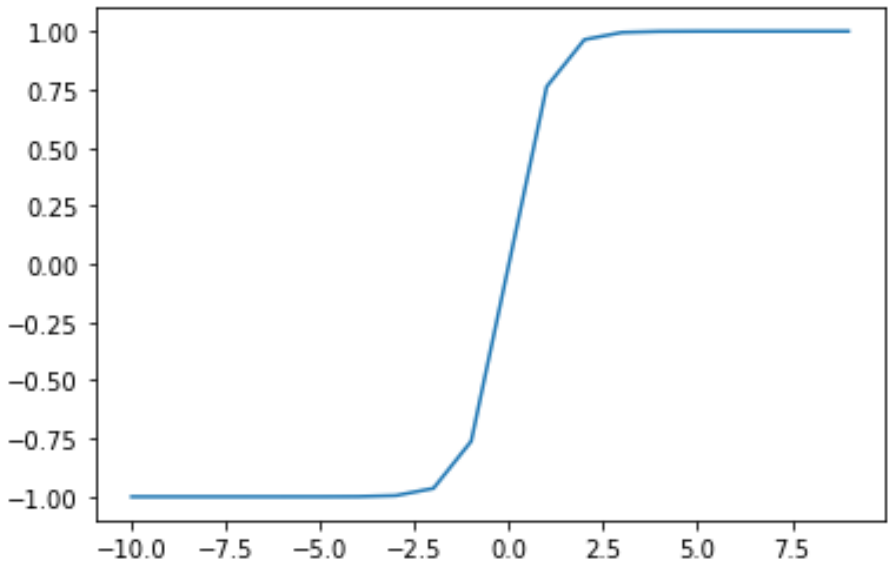

Fig. 5: Gráfico da função Tanh.

a zero é retornado o valor de entrada. Esta é definida como

$$output = f(x) = \begin{cases} 0, & \text{if } x < 0; \\ x, & \text{if } x \geq 0. \end{cases} \tag{5}$$

O gráfico desta função é apresentado na Figura 6.

3.2.7 Função de ativação Softmax

A função Softmax retorna valores no intervalo [0, 1]. É utilizada para a classificação, onde a soma dos valores das classes da 1. O maior valor representa a classe mais provável da classificação. É uma generalização da função Sigmoid para múltiplos casos [35] [36]. Esta função é definida como a função exponencial de cada elemento x_i, do vetor de entrada x, e normaliza estes valores dividindo pela soma de todas as exponenciais garantindo que a soma das componentes do vetor de saída é 1.

$$f(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}. \tag{6}$$

3.3 Otimizadores

Os otimizadores são algoritmos que ajustam os pesos da rede neuronal para minimizar a função de perda durante o treino.

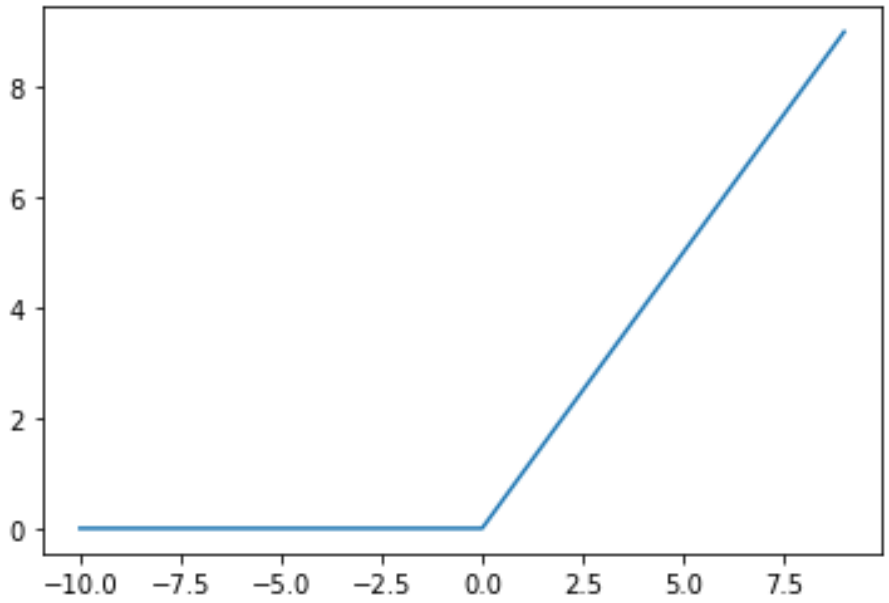

Fig. 6: Gráfico da função Relu.

A maioria dos otimizadores utilizam o método da análise da descida do gradiente padrão ou variantes desse método. No processo, o otimizador calcula os gradientes da função de perda em relação aos pesos e parâmetros da rede. Após o calculo do gradiente, atualiza os pesos com base numa regra específica, que pode variar dependendo do algoritmo utilizado. Podem também ajustar a taxa de aprendizagem (*Learning Rate*), que determina o tamanho do passo em cada iteração.

Em problemas complexos, o otimizador precisa de lidar com a possibilidade de encontrar mínimos locais em vez do mínimo global, evitando ficar onde o gradiente é pequeno ou nulo. Seguidamente são analisados os otimizadores mais comuns

3.3.1 Descida do gradiente estocástica

Este otimizador atualiza os pesos usando um pequeno subconjunto de dados (*batch*), atualizando os mesmos no sentido do gradiente negativo da função de perda, isto é, para minimizar a perda [37]. Pode também usar abordágem com *Momentum*, que adiciona uma "memória" às atualizações do algoritmo para reduzir oscilações, acelerando a convergência [37].

3.3.2 Adagrad

Adagrad adapta a taxa de aprendizagem para cada parâmetro, ajustando-a com base nas iterações anteriores. Isso é feito acumulando os quadrados dos gradientes anteriores e resulta numa taxa de aprendizagem maior para parâmetros raramente

atualizados e uma taxa de aprendizagem menor para parâmetros frequentemente atualizados [37].

3.3.3 RMSProp

RMSprop foi criado para resolver o problema de taxas de aprendizagem, que diminuem muito rapidamente no AdaGrad. Utiliza uma média móvel exponencial dos quadrados dos gradientes, mantendo a taxa de aprendizagem mais estável ao longo do tempo [37].

3.3.4 ADAM

O combina características de RMSprop e a descida do gradiente estocástica com *momentum*, adaptando as taxas de aprendizagem individualmente para cada parâmetro. Também usa médias móveis dos gradientes de primeira e segunda ordem, resultando numa taxa de aprendizagem mais robusta e eficiente [37].

3.4 Redes Neuronais Convolucionais

As CNNs são um tipo especifico de redes neuronais artificiais, amplamente utilizadas para o processamento de imagens. Inspiradas na estrutura do neuronio humano, as CNNs são capazes de aprender características relevantes das imagens, reduzindo a necessidade de extração manual das características.

As CNNs são compostas por diferentes tipos de camadas, as principais camadas são:

- Camada Convolucional, aplica filtros (*kernels*) sobre a imagem de entrada para extrair características, cada filtro desliza pela imagem aplicando operações de convolução e gerando um mapa de características (*Feature Map*).
- Camada de *Pooling*, esta camada reduz a dimensão do *Feature Map*, diminuindo o número de parâmetros. Os camadas mais comuns de *pooling* são:
  - Max Pooling, esta camada seleciona o valor máximo de cada região da matriz.
  - Average Pooling, esta camada seleciona o valor médio de cada região da matriz.
- Camadas Densas (*Fully Connected Layers*), estas camadas tomam a decisão final, utilizando funções de ativação para classificação.

Nesta serão analisadas as arquiteturas de CNNs que foram testadas neste projeto, utilizando a biblioteca *Keras*, que é uma Interface de Programação de Aplicações

(API) da plataforma *Tensorflow*. Foram testados os modelos convencionais de redes neuronais convolucionais disponibilizados para *Transfer Learning* com e sem mecanismos de atenção. Não sendo utilizados os *Transformers* [15], sendo os modelos pré-treinados facilmente adaptados para classificação de pele através de *Transfer Learning*, e os Transformers são modelos mais complexos que exigem recursos computacionais maiores.

3.4.1 VGG 16

A VGG16 é uma arquitetura de rede neuronal convolucional desenvolvida em 2014 pelo *Visual Geometry Group* da Universidade de Oxford, possui 16 camadas, das quais 13 são camadas convolucionais e 3 camadas totalmente conectadas (*fully connected*).

A imagem de entrada, para a rede VGG16, é de tamanho fixo de 224×224 pixels, onde as camadas convolucionais utilizam filtros de tamanho 3×3, que extraem as características da imagem.

Na figura 7 está representada a CNN VGG16, com 16 camadas divididas da seguinte forma:

- Duas camadas convolucionais de 64 filtros de tamanho 3×3, seguidas por uma camada de *pooling*.
- Duas camadas convolucionais de 128 filtros de tamanho 3×3, seguidas por outra camada de *pooling*.
- Três camadas convolucionais de 256 filtros de tamanho 3×3, seguidas por mais uma camada de *pooling*.
- Dois conjuntos de três camadas convolucionais de 512 filtros, de tamanho 3×3, cada um seguido por uma camada de *pooling*.
- Três camadas totalmente conectadas. As duas primeiras têm 4096 neurónios cada, e a última camada é uma *softmax*, que corresponde ao número de classes no conjunto de dados

As camadas de *pooling* utilizam a operação de *max-pooling* com uma janela de 2×2, e um *stride* de 2, esta ultima opção indica o número de pixéis que o kernel ignora enquanto se move ao longo da largura e altura da imagem, o que reduz a dimensão das características extraídas.

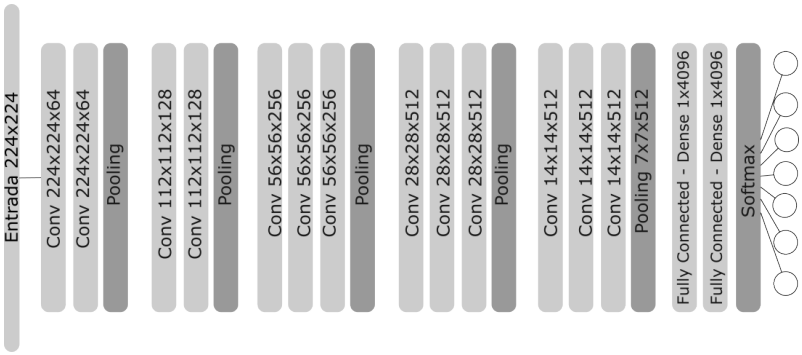

Fig. 7: Apresentação da arquitetura VGG16.

Todas as camadas convolucionais e totalmente conectadas utilizam a função de ativação ReLU (Rectified Linear Unit), que ajuda a introduzir não-linearidade na rede.

3.4.2 VGG 19

A VGG19 é uma variante da arquitetura de rede neuronal VGG16, e é composta por 19 camadas treináveis, acrescentando mais camadas para melhorar a capacidade de extração de características. A arquitetura de VGG19 é a seguinte:

- Duas camadas convolucionais de 64 filtros de tamanho 3×3, seguidas por uma camada de *pooling*.
- Duas camadas convolucionais de 128 filtros de tamanho 3×3, seguidas por outra camada de pooling.
- Quatro camadas convolucionais de 256 filtros de tamanho 3×3, seguidas por uma camada de *pooling*.
- Quatro camadas convolucionais de 512 filtros de tamanho 3×3, seguidas por uma camada de max-*pooling*.
- Por ultimo, quatro camadas convolucionais de 512 filtros de tamanho 3×3, cada um seguido por uma camada de *pooling*.
- Após as camadas convolucionais e de *pooling*, possui três camadas totalmente conectadas. As duas primeiras têm 4096 neurónios cada, e a última camada é uma *softmax*, que corresponde ao número de classes no conjunto de dados.

Similar a VGG16, as camadas de *pooling* utilizam a operação de *max-pooling* com uma janela de 2×2 e um *stride* de 2.

Todas as camadas utilizam a função de ativação ReLU, que ajuda a introduzir não linearidade na rede.

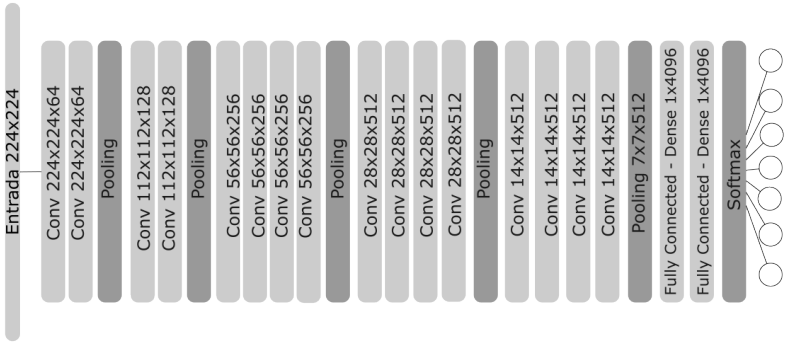

Fig. 8: Apresentação da arquitetura VGG19.

3.4.3 InceptionV3

Os modelos InceptionV1, InceptionV2 e inceptionV3 foram desenvolvidos pela Google. O InceptionV1 foi apresentado em 2014 e introduz o conceito dos módulos Inception, estes módulos permitem a utilização de diferentes tamanhos de filtros convolucionais em paralelo [1] como apresentado na figura 9.

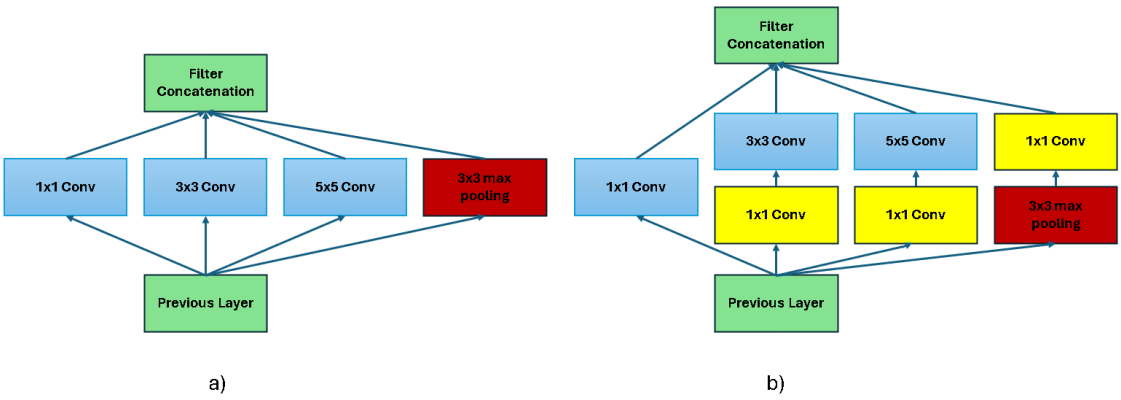

Fig. 9: Apresentação dos Modulos Inception. a) Modulo Inception Naïve. b) Modulo Inception com redução de dimensão. Adaptado de [1].

O objetivo principal dos módulos Inception é o de melhorar a capacidade de extração de características, permitindo que múltiplos filtros de tamanho diferente operem em paralelo sobre a mesma entrada. Isto permitiu reduzir o número de parâmetros dentro da rede. Com os tamanhos diferentes dos filtros permite extrair

múltiplas caraterísticas, capturando padrões diferentes e ao combinar operações em paralelo e técnicas de redução de dimensionalidade, reduz o número de parâmetros.

O modulo *naïve*, foi o primeiro módulo Inception, que apresentava convoluções de 1×1, 3×3 e 5×5 e uma operação de *Max Pooling* 3×3 aplicadas em paralelo, sendo as saídas das operações concatenadas.

Para reduzir o custo computacional da versão naïve, foi introduzida a redução de dimensionalidade no InceptionV2, onde aparecem convoluções 1×1 antes das convoluções 3×3 e 5×5, reduzindo o número de canais de entrada antes de aplicar filtros maiores. Após o *Max Pooling*, é aplicada uma convolução 1×1 para manter as informações mais relevantes.

Na versão InceptionV3, as convoluções de 5×5 foram fatorizadas em convoluções menores, duas convoluções 3×3, reduzindo o número de cálculos. E as convoluções 3×3 são substituídas por convoluções separáveis, isto é uma convolução 3×1 seguida de uma convolução 1×3. Mantendo as convoluções 1×1 para redução de dimensionalidade.

A arquitetura InceptionV3 é composta por um total de 48 camadas, ver figura 10. Nela foi adicionado o uso de RMSProp *(Root Mean Square Propagation)* como otimizador, regularização do modelo com *Label Smoothing*, que é uma técnica de regularização que ajuda a evitar o *overfitting* e a fatorização de convoluções 7×7.

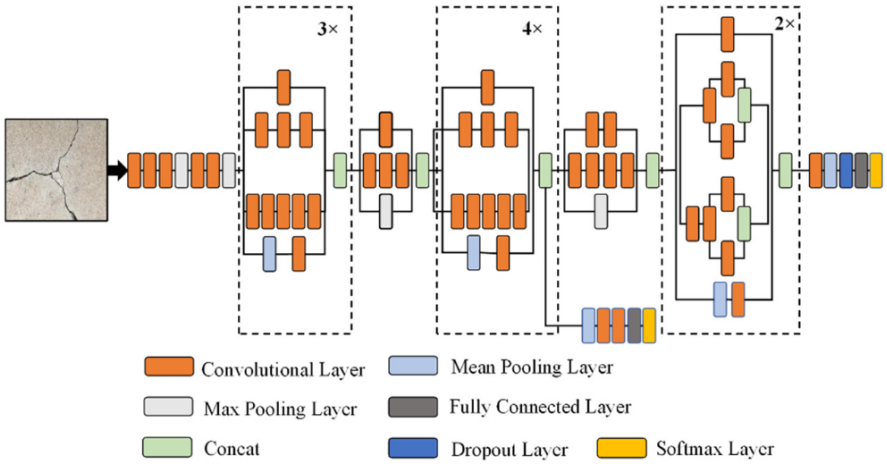

Fig. 10: Apresentação da arquitetura Inception V3 [2].

3.4.4 Resnet

A ResNet é uma rede neuronal artificial com 50 camadas, apresentada em 2015 por investigadores da Microsoft Research [38], destaca-se pela capacidade de treinar

redes muito profundas sem ser substancialmente afetada pelo problema do desvanecimento do gradiente até ao uso de centenas de camadas, onde este problema ocorre quando os gradientes utilizados para atualizar os pesos se tornam extremamente pequenos, dificultando o treino.

A principal inovação da ResNet é a introdução de ligações residuais (*skip connections*), conectando a entrada de um bloco residual diretamente à sua saída, saltando uma ou mais camadas intermédias, resolvendo o problema do desaparecimento do gradiente em redes muito profundas.

Depois de cada camada convolucional é realizada uma normalização por lotes das ativações (*Batch Normalization*). Cada bloco residual usa uma estrutura *Bottleneck* de 3 camadas:

- Convolução 1×1 para reduzir dimensionalidade
- Convolução 3×3 para processamento
- Convolução 1×1 para restaurar dimensionalidade

Nos blocos residuais, a entrada x é passada através de várias camadas de convolução para produzir uma saída $F(x)$. Em seguida, a entrada original é adicionada à saída para produzir a saída final do bloco y, de acordo com $y = F(x) + x$.

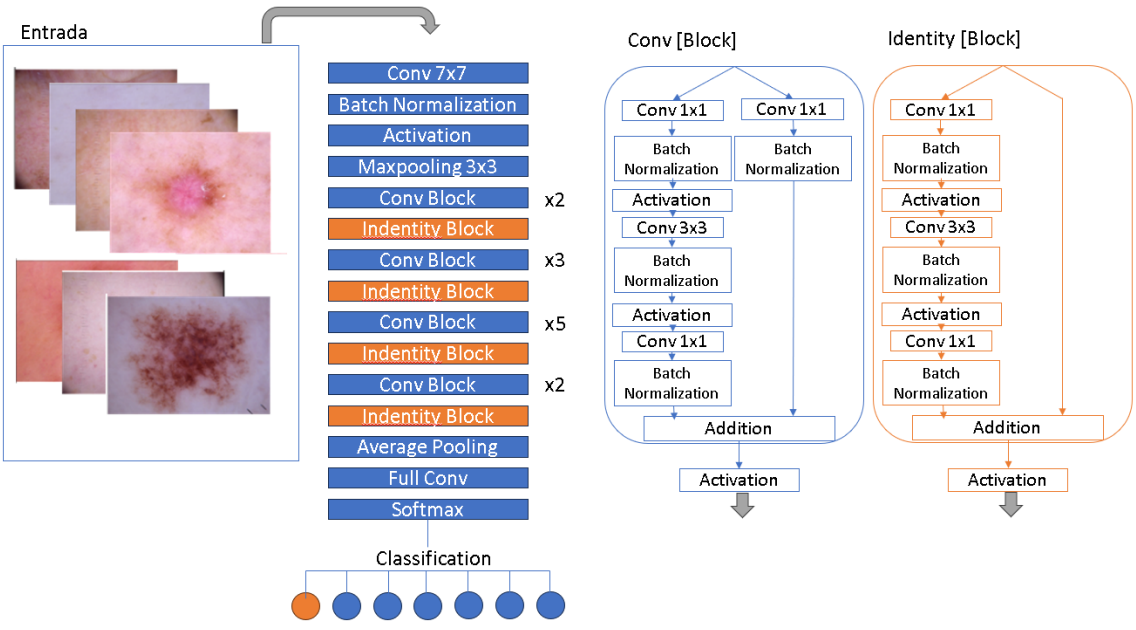

Fig. 11: Arquitectura Resnet50 adaptado de [3], com imagens do HAM10000 [4].

Essa soma direta da entrada à saída é o que permite treinar redes muito profundas sem sofrer substancialmente do problema do desvanecimento do gradiente, proporcionando um "atalho" para o gradiente fluir.

Existem variantes da ResNet 50, cada uma delas com um número diferente de camadas:

- ResNet-50: 50 camadas
- ResNet-101: 101 camadas
- ResNet-152: 152 camadas

3.4.5 DenseNet

A DenseNet é um tipo de CNN que se destaca por suas conexões densas entre as camadas, cada camada está conectada a todas as camadas anteriores, isto é, a entrada de cada camada é a união das saídas de todas as camadas anteriores.

A DenseNet é composta por blocos densos, onde cada bloco contém várias camadas convolucionais. Entre esses blocos, existem camadas de transição que reduzem a dimensão dos dados.

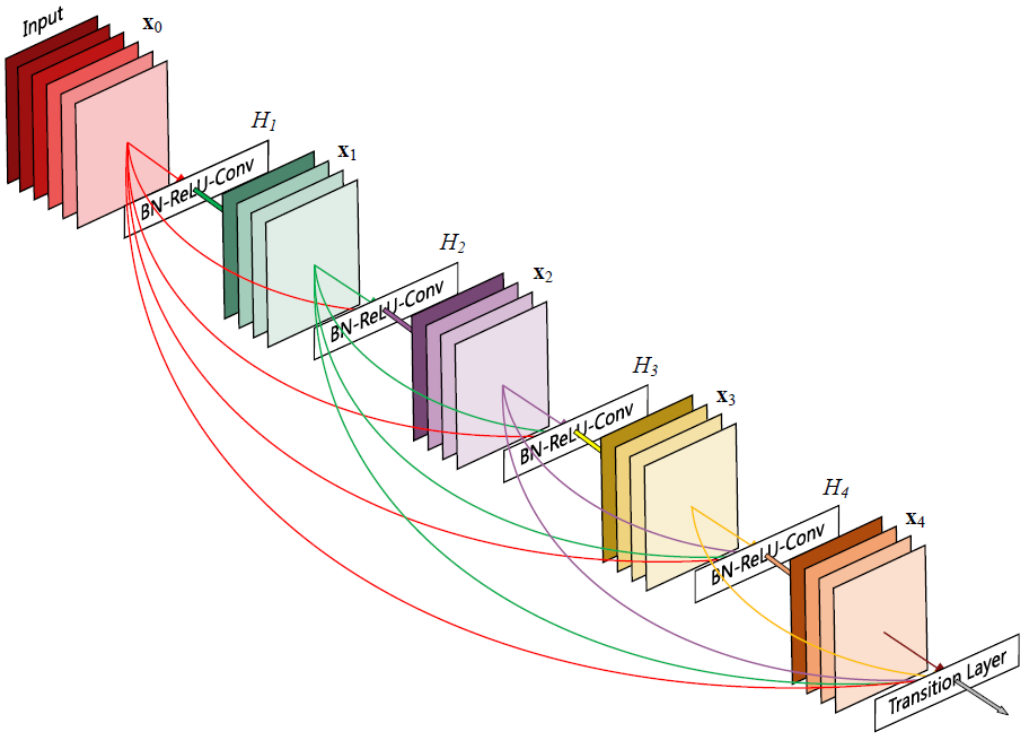

Fig. 12: Apresentação da arquitetura Densenet [5].

Dentro de cada bloco denso, cada camada recebe como entrada todas as saídas das camadas anteriores, concatenando as saídas das camadas anteriores [5]. Entre os blocos densos, existem camadas de transição que reduzem a dimensão dos dados usando operações como convoluções 1×1 e *Pooling*. A cada nova camada adicionada, a dimensão das características aumenta, pois está concatenando novas saídas.

3.4.6 InceptionResNet

A InceptionResNet combina elementos das arquiteturas Inception e ResNet, tendo sido desenvolvida para melhorar a eficiência computacional das CNNs, utilizando módulos Inception que aplicam convoluções de diferentes tamanhos em paralelo.

A arquitetura ResNet, como explicado anteriormente, introduz conexões residuais, que são atalhos que saltam uma ou mais camadas. Essas ligações ajudam a mitigar o problema do desaparecimento do gradiente, facilitando o treino de redes muito profundas.

Os módulos Inception são a base da arquitetura. Cada módulo aplica várias convoluções com diferentes tamanhos de filtro (1×1, 3×3, 5×5) e uma operação de *Pooling* em paralelo. Isso permite que a rede capture características em diferentes escalas e resoluções. A saída dessas operações é concatenada, formando a entrada para o próximo módulo.

Combinação Inception e ResNet

Na InceptionResNet, os módulos Inception são combinados com as conexões residuais, adicionando a saída de um módulo Inception à entrada original.

Existem várias versões da InceptionResNet, a Inception-ResNet-v1 e Inception-ResNet-v2, cada uma com pequenas modificações para melhorar o desempenho e a eficiência.

A primeira versão é uma combinação dos módulos *Inception* com as conexões residuais, projetada para ser uma rede profunda e eficiente, mas com uma estrutura relativamente simples em comparação com a versão 2, a qual introduz melhorias na estrutura dos módulos *Inception* e nas conexões residuais, originando uma rede mais complexa e profunda.

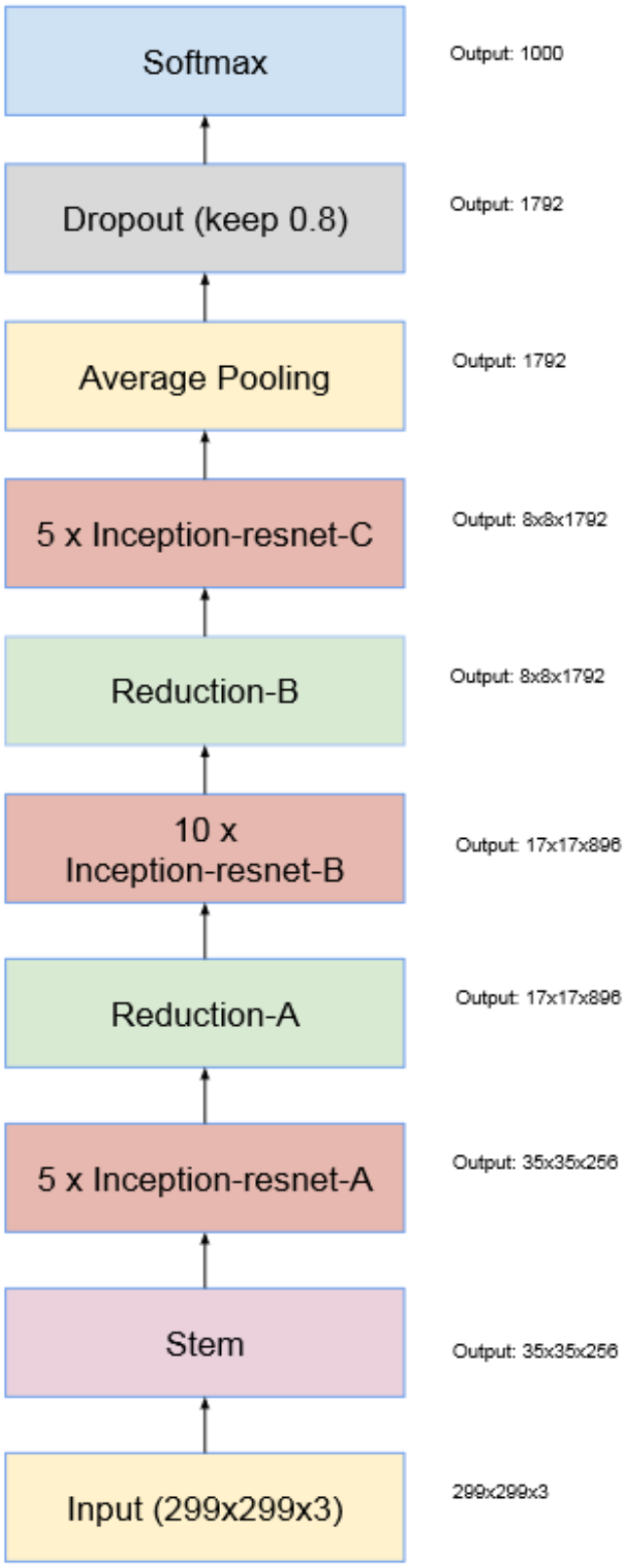

Fig. 13: Apresentação da arquitetura InceptionResnetV2 [6].

3.4.7 MobileNet

A MobileNet foi desenvolvida para ser eficiente em termos de velocidade e tamanho em dispositivos móveis [39]. Esta foi desenvolvida pelo Google, sendo tão precisa como a arquitectura VGG16 mas é 32 vezes mais pequena e 27 vezes menos intensiva em processamento [39]. A principal inovação da MobileNet é o uso de convoluções separáveis em profundidade *(depthwise separable convolutions)*, que reduzem substancialmente o número de parâmetros e cálculos necessários.

O modelo MobileNet baseia-se em convoluções separáveis em profundidade, que é uma factorização numa convolução padrão em profundidade e uma convolução 1×1, chamada convolução pontual. A convolução em profundidade aplica um único filtro a cada canal de entrada (R,G,B) e a convolução pontual aplica uma convolução 1×1 para juntar as saídas da convolução em profundidade, na figura 14 .

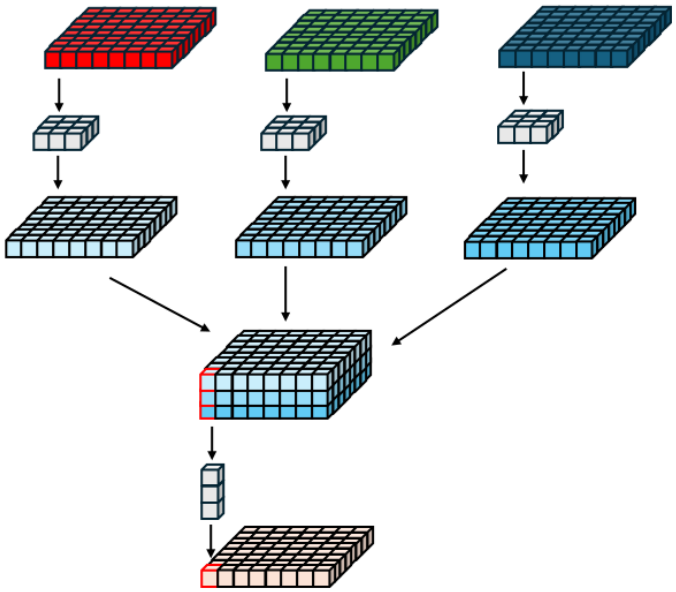

Fig. 14: Convoluções separáveis em profundidade *(Depthwise Separable Convolutions)*, adaptado de [7] [8].

As convoluções separáveis em profundidade dividem a convolução padrão em duas operações mais simples, reduzindo a complexidade computacional. Em vez de aplicar uma convolução 3D, é aplicada uma convolução 2D *(Depthwise)* e a seguir é aplicada uma convolução 1D *(Pointwise)*, estas operações reduzem o número de parâmetros e operações, tornando a rede mais leve e rápida.

Na figura 15 pode ver-se a arquitetura da MobileNetV2, onde a entrada é processada por uma convolução 3×3, para depois ser processada por 7 blocos invertidos o

bottleneck layer, onde cada bloco é composto por três camadas principais: uma camada de expansão 1×1, uma convolução *Depthwise* 3×3 e por ultimo uma camada que reduz a dimensão. À saída dos 7 blocos é aplicada uma convolução que ajusta a dimensão das características para a camada de classificação, e por ultimo reduzir as dimensões da saída da última camada convolucional para aplicar a classificação com uma função *Softmax*.

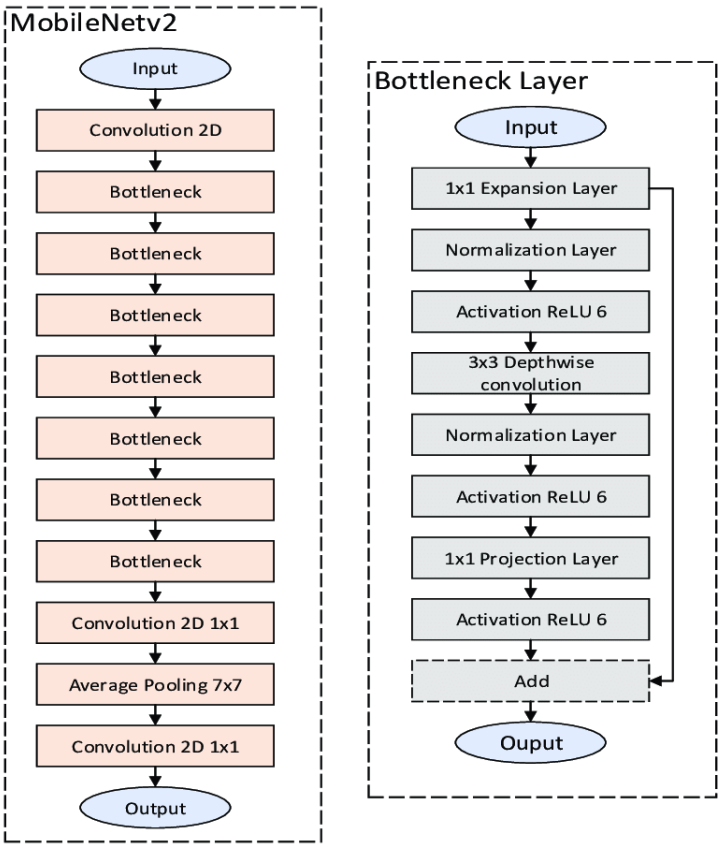

Fig. 15: Apresentação da arquitetura da rede MobileNetV2 [9].

3.4.8 Xception

A arquitetura Xception, é uma rede neural convolucional profunda baseada na ideia de convoluções separáveis em profundidade (*depthwise separable convolutions*). Este modelo é uma evolução da arquitetura Inception, propondo uma modificação nas convoluções, substituindo os módulos Inception por convoluções separáveis em profundidade. Consiste em 36 camadas convolucionais estruturadas em 14 módulos, onde todas as camadas convolucionais têm ativação ReLU, exceto a ultima camada. Para reduzir o problema do vanishing gradient utiliza conexões residuais.

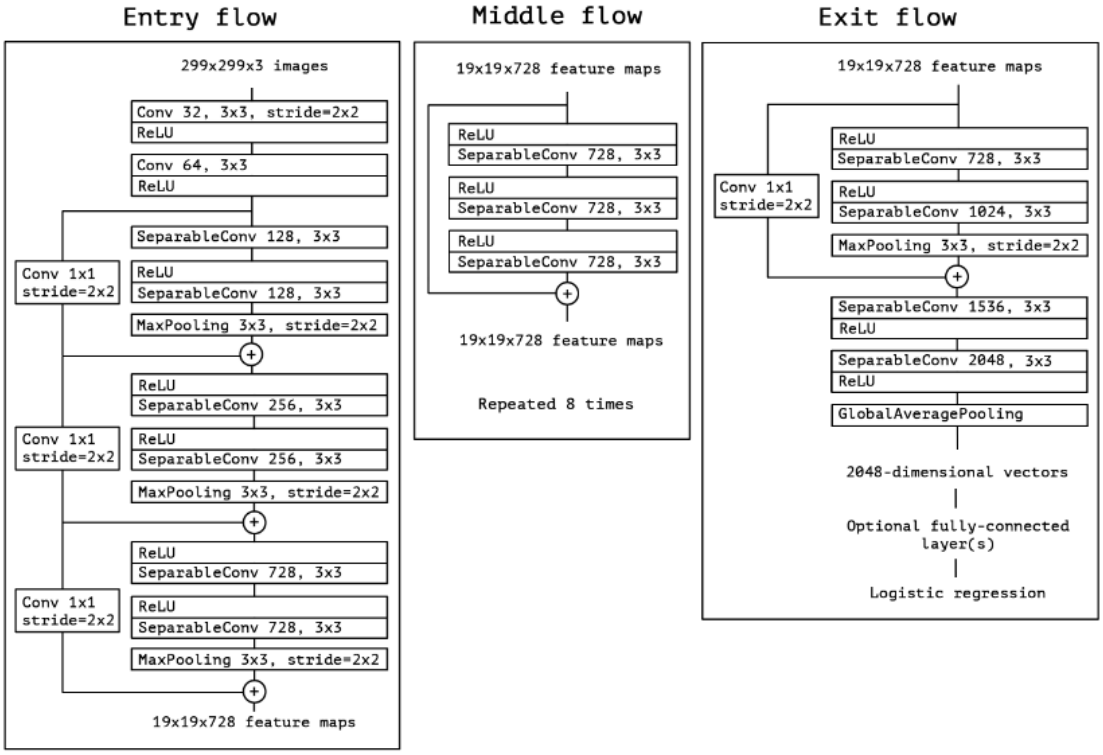

Fig. 16: Apresentação da arquitetura Xception [10].

Na figura 16 é visível a arquitetura do modelo, onde os dados de entrada passam primeiro por duas camadas de convolução, e depois por três blocos de convoluções separáveis em profundidade, onde cada bloco aumenta o número de canais e reduz a dimensão. Depois, no fluxo intermédio, consiste em 8 blocos repetidos sequencialmente, onde cada bloco possui três convoluções separáveis em profundidade mantendo as dimensões e número de canais. Por ultimo, é utilizado um bloco que aumenta o número de canais e são aplicadas duas convoluções separáveis em profundidade, terminando com uma *GlobalAveragePooling* e a camada de classificação.

Este modelo superou a arquitetura InceptionV3 em classificação de imagens no ImageNet, apresentando 79,0% de top-1 exatidão e 94,5% de top-5 exatidão contra o InceptionV3 que consegue 77,9% de top-1 exatidão e 93,7% de top-5 exatidão [10].

Este modelo supera a arquitetura InceptionV3, provando que convoluções separáveis em profundidade podem ser uma alternativa superior às tradicionais convoluções utilizadas.

3.4.9 NasNet

Os modelos NasNet representam uma família de arquiteturas de redes neuronais convolucionais desenvolvidas pelo Google Research [11]. Estes modelos foram pro-

jetados com base em um processo de pesquisa automatizada (NAS - *Neural Architecture Search*), que utiliza algoritmos de aprendizagem por reforço para otimizar configurações de arquitetura [11].

É abordada a criação de modelos de classificação de imagens através da aprendizagem automática das arquiteturas de redes neuronais, propondo uma abordagem em duas etapas:

A primeira etapa consiste na pesquisa de blocos da arquitetura em conjuntos de dados menores, e nela é realizada uma busca pelo melhor bloco arquitetural, denominado "célula", utilizando um conjunto de dados menor, como o CIFAR-10 [40]. Esta célula representa uma camada convolucional otimizada.

A segunda etapa é a transferência para conjuntos de dados maiores, onde a célula otimizada é transferida para um conjunto de dados maior, como o ImageNet [41]. A arquitetura final, chamada NASNet, é construída empilhando múltiplas cópias desta célula, cada uma com seus próprios parâmetros. As arquiteturas estão representadas na figura 17.

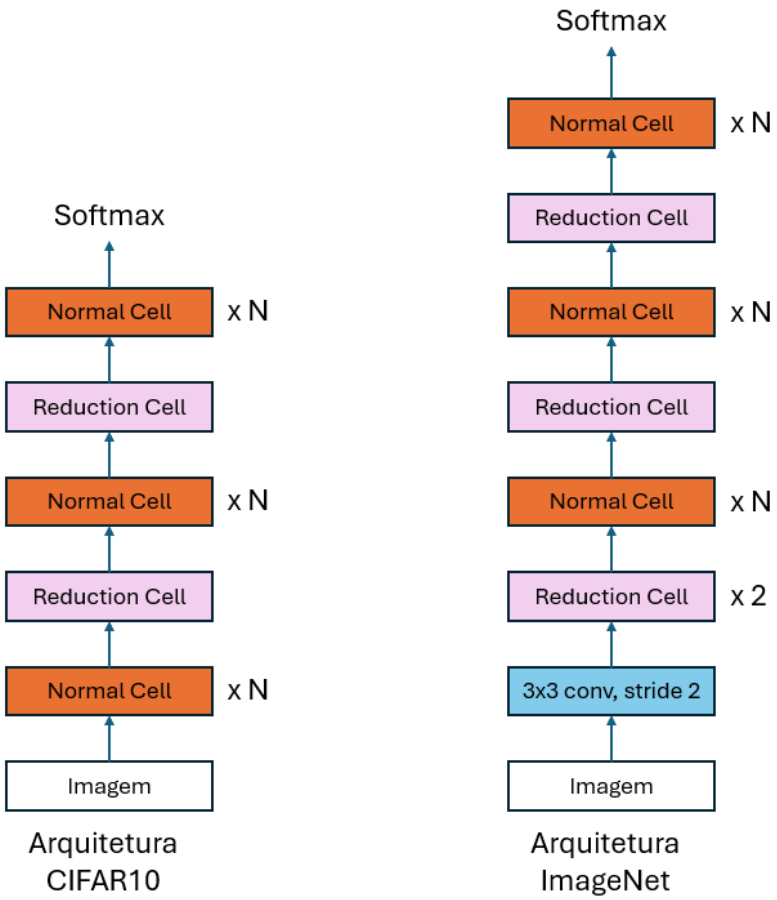

Fig. 17: Apresentação da arquitetura NASNet adaptada de [11].

É proposto o "*NASNet search space*", que facilita a transferência das arquiteturas entre diferentes conjuntos de dados. Além disso, os autores introduzem uma técnica de regularização chamada ScheduledDropPath, que elimina ligações utilizando uma probabilidade crescente a medida que é realizado o treino, em vez de utilizar uma probabilidade fixa, melhorando a capacidade de generalização dos modelos NASNet.

Os resultado obtidos neste modelo foram os seguintes:

- CIFAR-10: O NASNet alcançou uma taxa de erro de 2,4%, estabelecendo um novo estado da arte para este conjunto de dados.
- ImageNet: O NASNet atingiu uma precisão de 82,7% no top-1 e 96,2% no top-5, superando arquiteturas desenvolvidas manualmente e reduzindo em 28% a necessidade computacional em comparação com o modelo anterior mais eficiente [11].

Na Figura 18 está representado o processo de busca automatizada de células convolucionais [11].

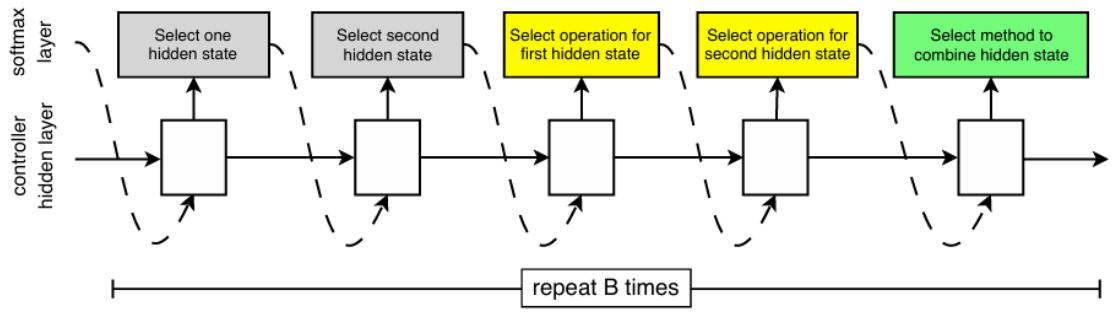

Fig. 18: Processo de busca automatizada de células convolucionais [11].

Na figura 18 pode-se verificar que a cada iteração é selecionada uma camada oculta (hidden layer) como entrada, depois seleciona outra camada oculta para ser combinada, e de seguida são escolhidas duas operações para aplicar aos estados ocultos selecionados. Depois é selecionado o método para combinar os estados ocultos (bloco verde), por exemplo uma soma ou uma concatenação. Este processo é repetido B vezes para criar uma célula convolucional completa. Na figura 19 está um exemplo de célula arquitetural gerada pelo processo descrito, este resultado é uma nova camada oculta.

A NASNetLarge e a NASNetMobile são duas variantes da arquitetura NASNet, projetadas para atender diferentes necessidades de desempenho e eficiência computa-

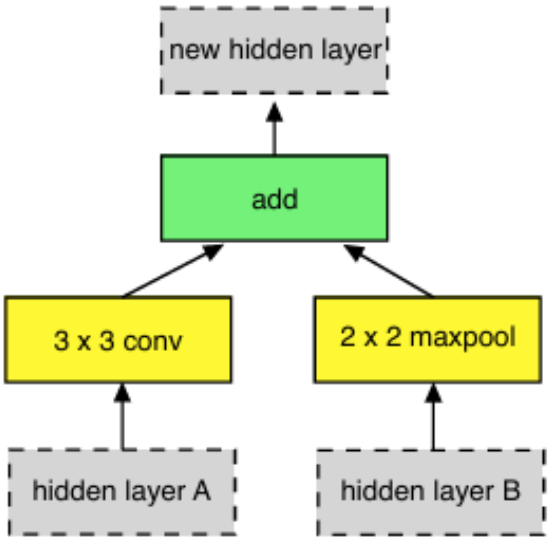

Fig. 19: Exemplo de célula arquitetural gerada [11].

cional. Ambas utilizam a metodologia descrita na NasNet, para criar arquiteturas otimizadas de redes neuronais convolucionais.

A NasNetLarge é uma versão maior da NasNet, que tem como objetivo maximizar a precisão, sem grandes limitações de recursos computacionais, enquanto a NasNetMobile é uma versão compacta e eficiente da NasNet, projetada para dispositivos móveis e aplicações com recursos limitados.

3.4.10 EfficientNet

EfficientNet é uma família de CNNs cuja principal inovação é o método de escalonamento, que ajusta três dimensões da rede: a profundidade (número de camadas), a largura (número de filtros por camada) e a resolução das imagens de entrada [12].

Na figura 20 estão representados de forma gráfica os diferentes escalonamentos da rede: largura ou canais, profundidade ou mais camadas à rede, e a resolução da imagem.

O escalonamento composto (*Compound Scaling*), aplica uma fórmula para aumentar o tamanho do modelo em todas as dimensões simultaneamente:

$$depth \times width \times resolution. \tag{7}$$

A fórmula considera um fator de escala chamado $\phi(phi)$, para dimensionar uniformemente largura, profundidade e resolução da rede.

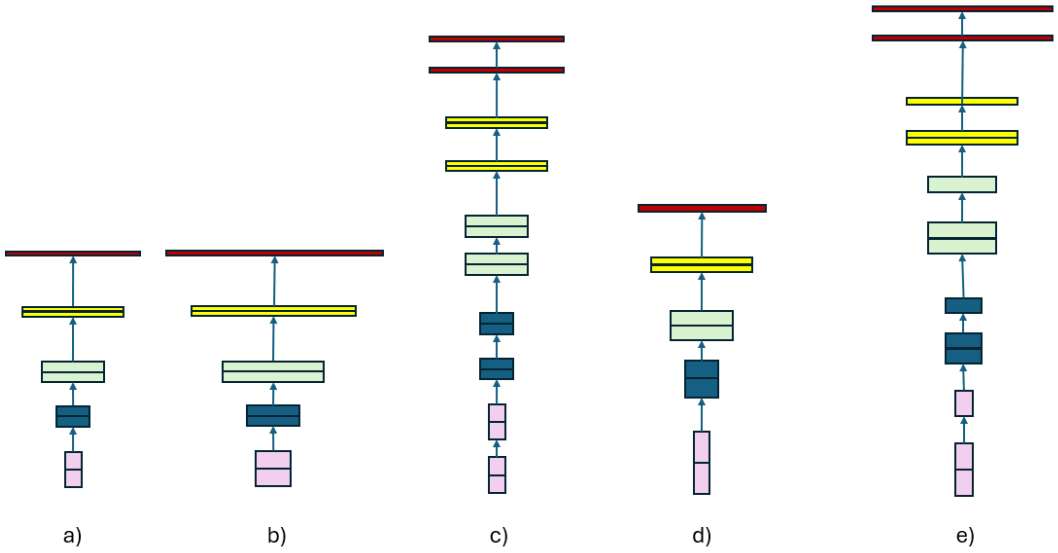

Fig. 20: Escalonamento da EfficientNet, adaptado de [12]. a) Representação básica. b) Escalonamento em largura. c) Escalonamento em Profundidade. d) Escalonamento em resolução da imagem. e) Escalonamento em Largura, profundidade e resolução.

A partir da rede base EfficientNet-B0, ver figura 21, foram obtidas versões escalonadas utilizando o escalonamento composto, dando origem as EfficientNet-B1 a B7. A medida que o índice B_i aumenta, a resolução das imagens de entrada, a profundidade e a largura da rede, também aumentam.

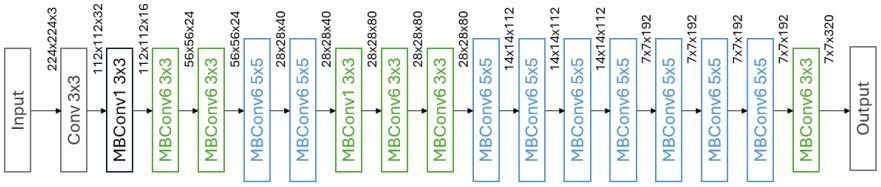

Fig. 21: Apresentação da arquitetura EfficientNetB0 adaptado de [13].

Nos resultados do trabalho pode ser verificado que a EfficientNet-B7, tem uma precisão 84.3% no ImageNet, superando modelos como ResNet e Inception. A EfficientNet utiliza blocos MBConv(*Mobile Inverted Bottleneck Convolutional Blocks*) como bloco base na sua arquitetura

Blocos MBConv

Compostos por várias etapas de transformação, primeiro expande os canais de entrada, para depois realizar uma convolução separável em profundidade (*depthwise*

separable convolution) e por ultimo reduzir os canais com uma convolução pontual (1×1). Se a entrada e a saída tiverem as mesmas dimensões, é utilizada uma conexão residual (skip connection), facilitando o aprendizagem e melhorando a convergência.

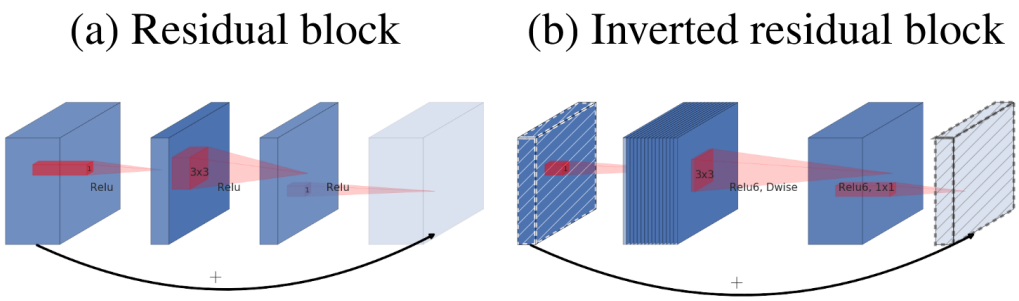

Fig. 22: Diferença entre bloco Residual e Bloco Residual Invertido [14].

3.4.11 ConvNext

A ConvNext é uma arquitetura de rede neuronal convolucional desenvolvida por Zhuang Liu no trabalho "A ConvNet for the 2020s" [42], apresentando uma evolução das arquiteturas CNN anteriores, como ResNet e EfficientNet, incorporando avanços recentes em técnicas de redes neuronais. Introduzem modificações inspiradas nos avanços dos *Transformers* [15], resultando numa família de modelos que conseguem uma precisão de 87,8%.

Os mecanismos de atenção permitem que um modelo preste atenção a diferentes partes de uma entrada, dando mais importância a alguns elementos do que outros. Os modelos *Transformers* utilizam mecanismos de atenção (*Self-Attention*) para capturar relações globais entre os pixeis das imagens.

O funcionamento do mecanismo de atenção, utiliza os seguinte elementos no calculo:

- *Query* ou Consulta (Q): é a informação que se procura
- *Key* ou Chave (K): A chave utilizada para comparar e determinar a relevância de diferentes partes da entrada
- *Value* ou Valor (V): é a informação real associada a chave, quando é determinado que a chave é relevante para a consulta, o valor associado a essa chave é o que será utilizado no cálculo final.

O mecanismo compara a *Query* com todas as *Keys* para determinar quais elementos são mais relevantes, e o resultado da atenção é a soma ponderada dos valores (V).

Existem variantes de mecanismos de atenção, ver figura 23:

- *Self-attention*: onde cada elemento presta atenção a todos os outros elementos
- *Multi-head attention*: que permite o modelo aprender múltiplos padrões de atenção simultaneamente.

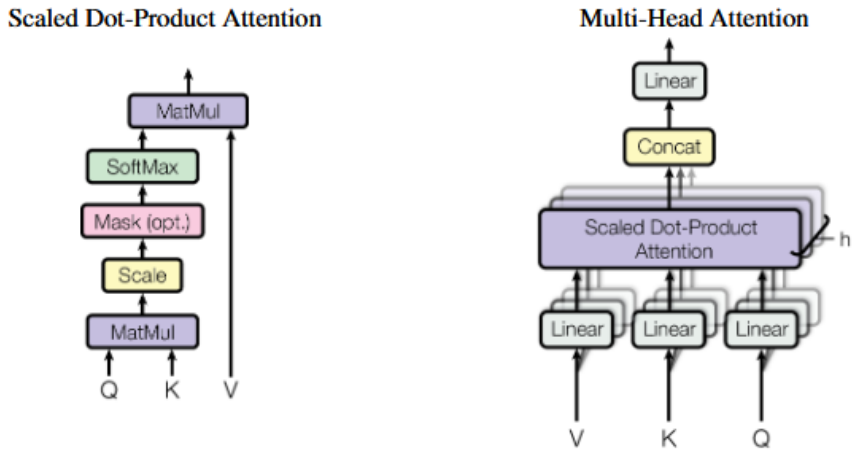

Fig. 23: Mecanismos de atenção de [15].

O modelo ConvNeXt utiliza uma arquitetura ResNet, introduzindo princípios de *design* de *Transformers* que podem ser aplicados a redes convolucionais. O ConvNeXt aprimora as redes neuronais convolucionais com técnicas que permitem obter um desempenho semelhante aplicando convolução. O modelo utiliza blocos convolucionais atualizados, com convoluções de profundidade separável e normalização em grupo, para melhorar a eficiência e o desempenho.

As convoluções foram aumentadas para tamanhos maiores de 7×7, para capturar melhor as caraterísticas, ver Figura 24. Adicionalmente, foi substituida a *Batch Normalization* por *Layer Normalization* e a função de ativação utilizada foi GELU (*Gaussian Error Linear Unit*).

Os blocos convolucionais foram redesenhados para ser mais próximo dos transformadores, utilizando convoluções de Profundidade Separável.

Estas convoluções dividem a operação em duas etapas:

- Convolução de Profundidade (*Depthwise Convolution*), onde é aplicada uma convolução separada em cada canal da entrada e não permite interação entre canais nesta etapa.
- Convolução Pontual (*Pointwise Convolution*): utiliza filtros de dimensão 1×1 para combinar as informações dos diferentes canais gerados na convolução anterior.

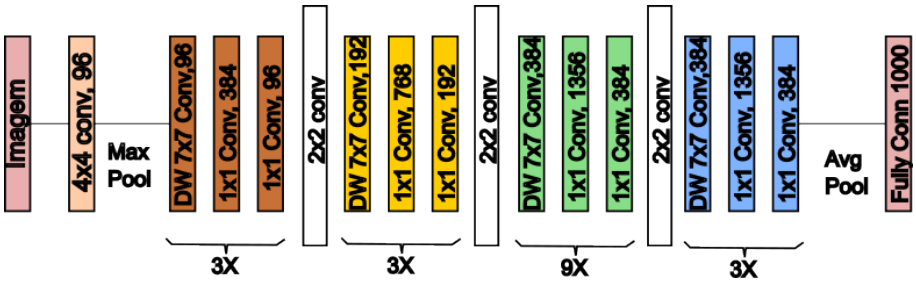

Fig. 24: Apresentação da arquitetura ConvNext [16].

3.4.12 Transferência de conhecimento (*Transfer Learning*)

A transferência de conhecimento ou *Transfer Learning*, refere-se a utilizar o conhecimento de uma tarefa para melhorar o desempenho de outra. Esta abordagem reduz o tempo de treino e alivia a necessidade de grandes quantidades de dados rotulados ou dados treinados [43].

A utilização de *Transfer Learning* permite poupar tempo e recursos, utilizando um modelo que já foi treinado num grande conjunto de dados. O treino de um modelo a partir do zero pode ser demorado e dispendioso em termos computacionais.

Outra vantagem é melhorar o desempenho do modelo, já que é transferido o conhecimento do modelo pré-treinado sobre as características dos dados. É útil quando o conjunto de dados, do novo modelo a treinar, não possui dados suficientes [44].

Tipos de *Transfer Learning*

Existem os seguintes tipos de *Transfer Learning* [44]:

- Extração de características: Na extração de características, o modelo pré-treinado é utilizado para extrair características dos dados e estes recursos são utilizados para treinar o novo modelo, é uma boa abordagem quando os dados não são suficientes para a tarefa.
- *Fine-Tunning*: É uma técnica de aprendizagem em que um modelo pré-treinado é posteriormente treinado num novo conjunto de dados, para melhorar o seu

desempenho em outra tarefa específica. O modelo pré-treinado é normalmente treinado num grande conjunto de dados gerais, enquanto o novo conjunto de dados é específico para a tarefa em questão.

- Aprendizagem multitarefa: Na aprendizagem multitarefa, o modelo pré-treinado é treinado em múltiplas tarefas em simultâneo, ajudando a melhorar o desempenho do modelo em cada tarefa.
- Adaptação de domínios: o modelo pré-treinado é adaptado a um novo domínio, isto pode ser feito ajustando os pesos das camadas do modelo ou utilizando a técnica de *Data Augmentation*, que corresponde a geração de novas imagens aplicando transformações as originais.

3.5 Pontos-chave

Ao longo dos anos tem existido evoluções nas Redes neuronais profundas, por forma a melhorar o desempenho em tarefas complexas do ponto de vista computacional, adoptando técnicas que permitem reduzir o tempo de treino e melhoram o desempenho dos modelos, como é o caso da transferência de conhecimento ou *Transfer Learning*, que ao utilizar o conhecimento de uma tarefa melhoram o desempenho de outra.

As CNNs eram limitadas pela capacidade computacional, o que dificultava a aplicação em tarefas complexas como o reconhecimento de imagens. Em redes muito profundas o treino pode levar a uma diminuição da precisão, ocasionado pelo problema do desvanecimento do gradiente mas com a introdução de *skip connections* da Resnet, permitiu que os gradientes transitem para outras camadas sem desaparecer no treino.

Com a introdução de mais camadas convolucionais, camadas totalmente conectadas e a introdução de *skip connections* melhorou capacidade de aprendizagem. Adicionalmente, as técnicas como regularização (*dropout*), normalização de lotes (*batch normalization*) e *transfer learning* aumentaram desempenho e a precisão dos modelos.

No caso das Redes ConvNeXt, utilizam uma arquitetura ResNet e introduzem princípios de *design* de *Transformers* que podem ser aplicados a redes convolucionais, superando as CNNs tradicionais em tarefas como classificação de imagens. Entre os modelos com maior interesse em desenvolver neste trabalho encontra-se as EfficientNet com o escalonamento composto, conseguindo 84,3% de precisão e a rede ConvNeXt que introduz princípios de *Transformers* aplicados na rede, obtendo uma precisão de 87,8%.

4 Arquitetura do sistema

Neste capitulo são explicadas duas possíveis opções a ser utilizadas na implementação da arquitetura, indicando vantagens e desvantagens de cada uma.

Para estabelecer a arquitetura a utilizar no trabalho foram tomadas em conta duas opções

- Aplicação para o dispositivo Móvel com Modelo em *tflite*
- Aplicação para o dispositivo Móvel com comunicação com um servidor, utilizando o Modelo com maior desempenho sem conversão

4.1 *TFLite* no dispositivo móvel

O TFlite (ou TensorFlow *Lite*) é um conjunto de ferramentas para ML que permite executar modelos em dispositivos móveis, atualmente denominado LiteRT [45].

As vantagens de utilizar um modelo convertido para o formato *tflite* são as seguintes:

- Funciona offline
- Menor latência nas predições
- Maior privacidade, os dados são processados no dispositivo
- Sem custos de infraestrutura de servidor

As desvantagens identificadas foram as seguintes:

- Modelos mais simples devido às limitações de hardware
- Maior uso de recursos do dispositivo (bateria/espaço e processamento)
- Atualizações do modelo implica a atualização da aplicação
- Dimensão da aplicação, dependendo do modelo com maior desempenho pode implicar um maior tamanho da modelo *tflite*

4.2 Arquitetura Cliente-Servidor

Nesta arquitetura, o cliente envia a imagem ao servidor, que a recebe, processa e aplica o modelo de classificação selecionado, retornando o resultado ao cliente. A Figura 25 exemplifica de forma simples o descrito.

Fig. 25: Arquitetura Cliente Servidor.

As vantagens identificadas utilizando a arquitetura com um servidor são as seguintes:

- Maior poder de processamento, O servidor pode executar modelos mais complexos e precisos sem as limitações de hardware dos dispositivos móveis
- Atualizações do modelo são simplificadas, o modelo pode ser atualizado no servidor sem necessidade de atualizar a aplicação.
- Menor consumo de recursos no dispositivo móvel - O processamento do modelo acontece no servidor.

As desvantagens identificadas nesta arquitetura são as seguintes:

- Requer acesso à Internet para processar as imagens e classificar as lesões
- Maior latência nas predições
- Custos de infraestrutura e manutenção do servidor
- Privacidade dos dados enviados ao servidor e respostas ao cliente

4.3 Arquitetura implementada

Tendo em conta as vantagens e desvantagens das duas arquiteturas propostas, a que melhor se adapta a realidade do trabalho, é a arquitetura Cliente-Servidor. Ao converter o modelo para *tflite*, a dimensão do ficheiro é muito grande e pouco prático para implementar na aplicação, implicando que futuras atualizações do modelo obriguem à atualização da aplicação.

Na imagem 26 pode ser visualizada a solução proposta para o processo de classificação multiplataforma. A imagem é obtida num *smartphone*, e é enviada para um servidor que processa a imagem aplicando o modelo de classificação selecionado e retornando o resultado ao dispositivo móvel.

A visualização do resultado é representada num gráfico de barras indicando a probabilidade para cada uma das classes.

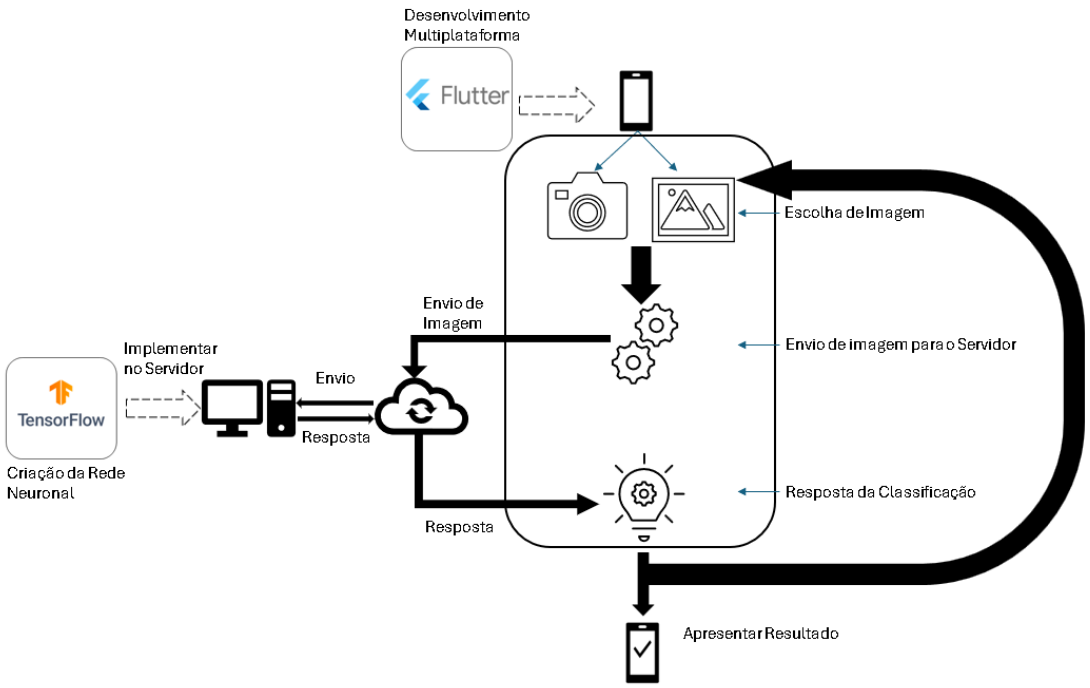

Fig. 26: Solução proposta para o processo de classificação multiplataforma.

4.4 Requisitos Funcionais

Os requisitos funcionais identificados para o desenvolvimento da aplicação são os seguintes:

- RF01: A aplicação deve ser capaz de capturar imagens através da câmara do dispositivo.
- RF02: Deve haver uma funcionalidade para carregar imagens da galeria do dispositivo.
- RF03: A aplicação deve ser capaz de classificar as imagens de lesões de pele.
- RF04: A aplicação deve permitir catalogar uma lesão para monitorização regular.

4.5 Requisitos Não Funcionais

Os requisitos não funcionais identificados, são os seguintes:

- RNF01 - Desempenho: A aplicação deve responder em um tempo máximo de 5 segundos ao pedido de classificação do utilizador.
- Requisito de qualidade do modelo: O modelo de classificação deve apresentar mais de 80% de exatidão.

- RNF02 - Segurança: A aplicação deve garantir a segurança dos dados do utilizador (imagens e resultados de classificação).
- RNF03 - Privacidade: A aplicação deve cumprir com Regulamento Geral sobre a Proteção de Dados (RGPD)
- RNF04 - Usabilidade: A aplicação deve ser fácil de usar, com uma interface intuitiva e instruções claras de captura ou carregamento imagens para classificação.
- RNF05 - Portabilidade: A aplicação deve ser compatível com vários sistemas operativos, como Android e iOS.
- RNF06 - Manutenção: A aplicação deve ser fácil de modificar e atualizar para melhorar funcionalidades e corrigir bugs

4.6 Regulamento Geral sobre a Proteção de Dados

A aplicação não guarda nenhuma das imagens processadas, e não possui dados dos utilizadores. Como prova de conceito a aplicação limita o acesso ao uso da classificação por um *token* predefinido, não associado a nenhum utilizador.

Em termos de dados pessoais a aplicação não guarda nenhum dado pessoal do utilizador, e a imagem submetida não é guardada no servidor, e processada no pedido e retorna o resultado da classificação.

4.7 Segurança da arquitetura

Para garantir a segurança na transmissão dos dados, entre o cliente e o servidor, foi utilizada encriptação entre o dispositivo móvel e o servidor. A encriptação utilizada foi o algoritmo AES (*Advanced Encryption Standard*), que é uma especificação para a criptografia de dados estabelecida pelo Instituto Nacional de Padrões e Tecnologia dos E.U.A. (NIST) em 2001 [46], devido a ser um padrão e com disponibilidade de implementações, ferramentas e bibliotecas de suporte foi utilizado o AES-256 com uma chave de 2048 bit. Adicionalmente o servidor está configurado com um certificado SSL para ligações seguras entre os equipamentos.

O pedido efetuado ao servidor apresentara resposta caso o *Token* enviado ao servidor corresponda com o estabelecido no mesmo, caso não exista correspondência com o *Token*, não será processado o pedido.

Em resumo, existe uma autenticação e dois níveis de encriptação, um primeiro nível com a utilização de um certificado SSL *(Secure Sockets Layer)*, para responder a pedidos https, e um segundo nível de encriptação ao encriptar a imagem para enviar os dados entre o servidor e o cliente.

4.8 Análise de requisitos de segurança

Os requisitos de segurança que a comunicação entre o servidor e aplicação deve garantir, são os seguintes:

- Garantia de segurança de dados
- Privacidade dos dados
- Restrição de acesso

As medidas adotadas na solução, para garantir os pontos indicados anteriormente, são as seguintes:

- Criação do certificado para o servidor
- Encriptação da imagem
- Autenticação via *Token*

4.9 Ambiente de desenvolvimento

A ferramenta de desenvolvimento escolhida foi o *VS Code* por ser um *IDE (Integrated Development Environment)* de código aberto, gratuito e multiplataforma para realizar o desenvolvimento.

A *Framework* ou conjunto de bibliotecas utilizada é o *Flutter*, por ser de código aberto e permite criar aplicações que executam em diferentes sistemas operativos (Android, iOS, Windows) e em ambiente *Web* [47] [48].

4.10 Criação do certificado para o servidor

Para realizar uma ligação segura com o servidor e utilizar o protocolo *SSL*, foi necessário a geração de um certificado. Para a geração de um certificado AES-256 com 2048 bytes foram utilizados os seguintes comandos [49]:

- Para gerar uma chave privada

```
openssl genrsa -aes256 -out server.key 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
.................................+++++
.............+++++
e is 65537 (0x010001)
Enter pass phrase for server.key:
Verifying - Enter pass phrase for server.key:
```

- Gerar o CSR (Certificate Signing Request), é um ficheiro que contém a chave pública, o local e o URL da empresa.

```
openssl req -new -key server.key -out server.csr
Enter pass phrase for server.key:
You are about to be asked to enter information that will be
incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name
or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:PT
State or Province Name (full name) [Some-State]:MADEIRA
Locality Name (eg, city) []:FUNCHAL
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

- Remover a palavra passe do certificado para utilizar no servidor

```
cp server.key server.key.org;
openssl rsa -in server.key.org -out server.key;
Enter pass phrase for server.key.org:
writing RSA key
```

- Geração de um certificado autoassinado (*Self Signed*), é um certificado que não é assinado por uma autoridade de certificação, mas sim pelo desenvolvedor ou empresa responsável.

```
openssl x509 -req -days 365 -in server.csr -signkey server.key -out
server.crt
Signature ok
subject=C = PT, ST = MADEIRA, L = FUNCHAL, O = Internet Widgits Pty
Ltd
Getting Private key
```

4.10.1 Encriptação da imagem

Para garantir a privacidade dos dados ao transitar para o servidor, é realizada uma codificação em *base64* [50] antes e depois de encriptar com *AES-256* para enviar a imagem ao servidor, depois é descodificada e desencriptada no servidor para classificação:

```
var uri = Uri.parse(uriText);
  var request = http.MultipartRequest('POST', uri);
  if (encryption) {
    // Ler a imagem como bytes
    final bytes = await image.readAsBytes();
    final prefs = await SharedPreferences.getInstance();
    //Pass Definition
    final key = encrypt.Key.fromUtf8(
        'secretpasswordxsecretkey2025xxxx'); //chars for AES-256
    final iv = encrypt.IV.fromUtf8('Sixteen byte IV.'); // Random IV

    // Encrypt the image
    final encrypter =
        encrypt.Encrypter(encrypt.AES(key, mode: encrypt.AESMode.cbc));
    final encryptedData = encrypter.encrypt(base64Encode(bytes), iv: iv
);
    String savedToken = prefs.getString('saved_token') ?? 'Error';

    if (savedToken == 'Error') {
      ScaffoldMessenger.of(context).showSnackBar(
          const SnackBar(content: Text("Deve configurar o Token")));
      setState(() {
        _loading = false;
        _response = false;
      });
    } else {
      // Send encrypted data to the server
      try {
        final response = await http.post(
          Uri.parse(uriText),
          headers: {
            'Content-Type': 'application/json',
            'Authorization': savedToken, // Add token to header
          },
          body: jsonEncode({'data': encryptedData.base64, 'iv': iv.
base64}),
        );
```

A resposta é obtida em base64 e descodificada para apresentar o resultado da classificação:

```
if (response.statusCode == 200) {
          // Decrypt the response
```

```
            final responseBody = jsonDecode(response.body);
            final encryptedResponseData = responseBody['data'];
            final responseIv = encrypt.IV.fromBase64(responseBody['iv']);
            final decryptedResponse =
                encrypter.decrypt64(encryptedResponseData, iv: responseIv
  );

            var decodedResponse = json.decode(decryptedResponse);
            setState(() {
              _results = decodedResponse['predictions'];
              _response = true;
              testresult = _results.toString();
              _loading = true;
            });
          } else {
            ScaffoldMessenger.of(context).showSnackBar(
                const SnackBar(content: Text("Erro ao processar o pedido
  ")));
            setState(() {
              _loading = false;
              _response = false;
            });
          }
        } on SocketException catch (e) {
          print('Erro de conexao: ${e.message}');
          ScaffoldMessenger.of(context).showSnackBar(
            const SnackBar(
                content: Text('Erro de conexao: verifique o token ou o
  acesso a Internet')),
          );
        } on HttpException catch (e) {
          print('Erro HTTP: ${e.message}');
          ScaffoldMessenger.of(context).showSnackBar(
            SnackBar(content: Text('Erro no servidor: ${e.message}')),
          );
        } catch (e) {
          print('Erro desconhecido: $e');
          ScaffoldMessenger.of(context).showSnackBar(
            SnackBar(content: Text('Ocorreu um erro inesperado')),
          );
        }
```

No servidor a imagem é desencriptada para depois ser submetida ao modelo de classificação:

```
    @app.route('/process_image', methods=['GET', 'POST'])
def process_image():
    try:

        token = request.headers.get('Authorization')
```

```
        if not token or token != VALID_TOKEN:
            response = jsonify({'error': 'Invalid token'}).get_data(as_text
    =True)
            response_cipher = AES.new(AES_KEY, AES.MODE_CBC)
            response_iv = response_cipher.iv
            encrypted_response = base64.b64encode(response_cipher.encrypt(
    pad(response.encode('utf-8'), AES.block_size)))

            return jsonify({
                'data': encrypted_response.decode('utf-8'),
                'iv': base64.b64encode(response_iv).decode('utf-8')
            })
            #return jsonify({'error': 'Invalid token'})

        logger.debug("Recebeu Imagem")
        data = request.json['data']
        encrypted_data = base64.b64decode(data)
        iv = base64.b64decode(request.json['iv'])
        # Decrypt the data
        cipher = AES.new(AES_KEY, AES.MODE_CBC, iv)
        decrypted_data = unpad(cipher.decrypt(encrypted_data), AES.
    block_size).decode('utf-8')

        # Handle the image file
        file = io.BytesIO(base64.b64decode(decrypted_data))
        image = Image.open(file)

        logger.debug("Desencriptou Imagem")
        image = image.resize((384, 384))
        image_array = np.array(image)
        image_array = np.expand_dims(image_array, axis=0)

        predictions = model.predict(image_array)

        class_names = ['akiec', 'bcc', 'bkl', 'df', 'mel', 'nv', 'vasc']

        results = [
            {'label': class_names[i], 'probability': float(prob)}
            for i, prob in enumerate(predictions[0])
        ]

        response = jsonify({'predictions': results}).get_data(as_text=True)
        response_cipher = AES.new(AES_KEY, AES.MODE_CBC)
        response_iv = response_cipher.iv
        encrypted_response = base64.b64encode(response_cipher.encrypt(pad(
    response.encode('utf-8'), AES.block_size)))

        return jsonify({
            'data': encrypted_response.decode('utf-8'),
            'iv': base64.b64encode(response_iv).decode('utf-8')
```

```
        })
        #return jsonify({'predictions': results})

    except Exception as e:
        logger.debug(str(e))
```

4.11 Autenticação via *Token*

A aplicação permite indicar o *Token* para estabelecer comunicação com o servidor e classificar a imagem, ver figura 27.

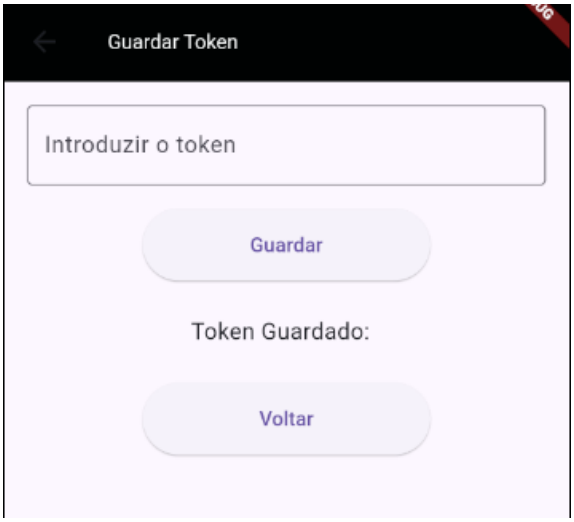

Fig. 27: Ecrã da aplicação da opção de configuração do *Token*.

Para poder realizar o pedido ao servidor o *Token* deve estar configurado, caso contrario o pedido não é realizado.

4.12 Aplicação para Dispositivos móveis

A aplicação, como prova de conceito, disponibiliza duas opções no ecrã inicial, uma primeira opção que permite o acesso a realizar a classificação da imagem que pretende analisar e uma segunda opção que da acesso a configuração do **Token** de acesso, ver figura 28.

Após a seleção da primeira opção "classificar Lesão", é apresentado um segundo ecrã que da acesso a duas opções. Ver Figura 29.

- Câmara: opção que permite realizar a captura da lesão para depois ser submetida ao servidor para classificação.
- Galeria: Esta opção permite selecionar uma imagem já existente no dispositivo móvel e acessível pela galeria de imagens.

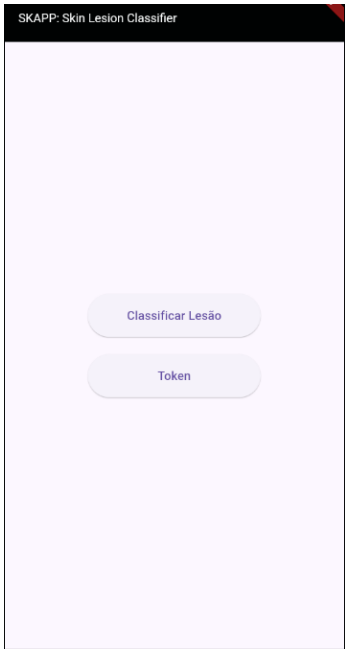

Fig. 28: Ecrã inicial da aplicação para classificação de lesões de pele.

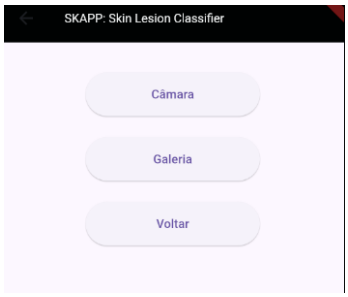

Fig. 29: Ecrã da aplicação para a opção de classificação.

Na figura 30 pode ver-se a opção de seleção da imagem a partir da galeria, que permite selecionar uma imagem para realizar a classificação.

Após a seleção da imagem, é disponibilizado um botão com a opção "Classificar Imagem", que realiza a comunicação com o servidor, encriptando a imagem selecionada. Ver figura 31.

O resultado da classificação é apresentado em um gráfico de barras com a correspondente classificação para cada um dos tipos de lesões, ver figura 32.

4.13 Pontos-chave

A arquitetura proposta é Cliente-Servidor, garantindo maior poder de processamento e futuras atualizações do modelo de forma simplificada. A privacidade no fluxo de dados, entre os equipamentos, é garantida com a utilização de certificado *SSL*, adicionalmente a encriptação dos dados, utilizando o algoritmo de criptografia simétrica

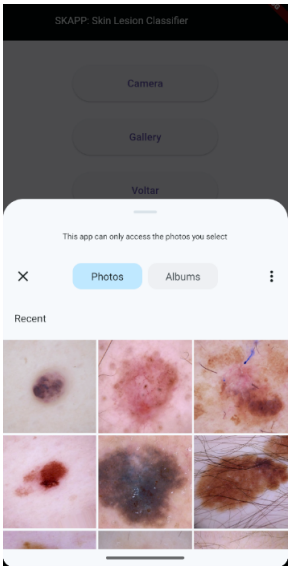

Fig. 30: Ecrã de seleção de imagem da galeria.

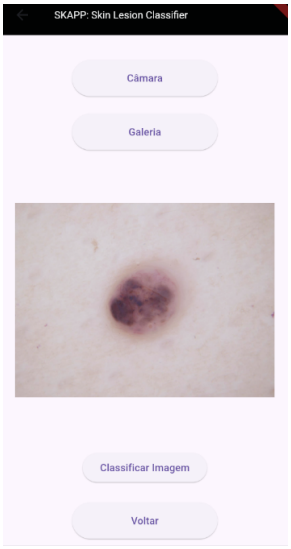

Fig. 31: Ecrã da aplicação para classificar a imagem selecionada.

AES256, garante a privacidade da informação. Ao adicionar o *token* como meio de autenticação, limita o uso a utilizadores legítimos, que podem enviar imagens para o servidor e obter resposta.

O desenvolvimento realizado com a *framework* de código aberto *Flutter* permite criar a aplicação para diferentes sistemas, aproveitando o esforço de desenvolvimento num único projeto para múltiplas plataformas.

Fig. 32: Ecrã da aplicação com o resultado da classificação.

5 Implementação do Modelo

Esta secção tem a descrição das técnicas utilizadas na implementação do modelo da Rede Neuronal Convolucional e a análise dos resultados obtidos ao testar o conjunto de dados de teste, escolhendo o modelo com o melhor desempenho para ser utilizado na solução da aplicação para dispositivos móveis.

5.1 Materiais

O conjunto de dados utilizado para classificar as lesões de pele é o HAM10000 [4], onde as lesões identificadas no repositório de dados, estão classificadas nas categorias indicadas na Figura 33:

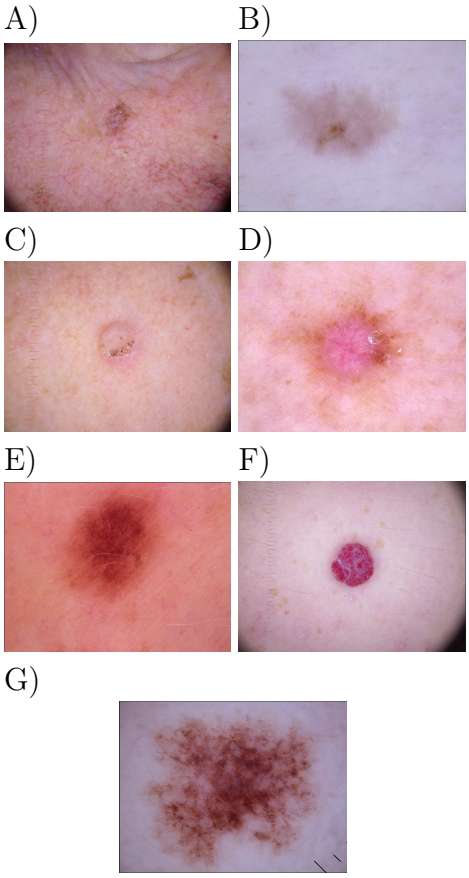

Fig. 33: Tipos de lesões a partir do conjunto de imagens HAM10000 [4]: A) Actinic keratoses, B) Basal cell carcinoma, C) Benign keratosis like, D) Dermatofibroma, E) Melanocytic nevi, F) Vascular Lesion, G) Melanoma.

- Queratoses Actínicas (akiec): Tipos de carcinoma espinocelular, não invasivos, e que podem ser tratados localmente sem cirurgia [51].

- Carcinoma Basocelular (bcc): Um tipo de cancro epitelial de pele que raramente se espalha, mas se não for tratado pode ser letal [51].
- Lesões benignas semelhantes à Queratose (bkl): São lesões benignas similares à Queratose [51].
- Dermatofibroma (df): Lesões cutâneas que são de crescimento benigno ou que resultam de uma resposta inflamatória a pequenos traumas [51].
- Nevos melanocíticos (nv): são neoplasias benignas de melanócitos e aparecem numa variedade de formas e tamanhos; do ponto de vista dermatoscópico, as variantes podem diferir dramaticamente [51].
- Lesões Vasculares (vasc): Estes angiomas podem ser benignos ou malignos.
- Melanoma (mel): O melanoma é um tumor cancerígeno que se desenvolve a partir dos melanócitos e pode assumir muitas formas diferentes. Se detetado precocemente, pode ser tratado com um procedimento cirúrgico básico [51].

A descrição das imagens não especifica se a lesão é maligna ou benigna, o que impossibilita a realização de uma classificação binária com o modelo de dados utilizado.

No conjunto de dados existe uma grande diferença na quantidade das imagens para cada lesão. Na tabela I é apresentada a quantidade de imagens para cada lesão no conjunto de dados HAM10000 [4]. É necessário balancear a quantidade de amostras a utilizar para gerar uma modelo que classifique corretamente as lesões.

Tabela I: Quantidade de imagens por Lesões do conjunto HAM10000 [4].

Tipo de Lesão	**Quantidade**
Actinic keratosis	327
Basal cell carcinoma	514
Benign keratosis like	1099
Dermatofibroma	115
Melanoma	1113
Vascular Lesion	142
Melanocytic nevi	6705
Total	10015

Para a separação dos dados de treino, validação e testes do modelo, foi utilizada a função "train_test_split", que faz parte da biblioteca *sklearn (scikit-learn)*. O conjunto de treino recebe 70% das amostras e os restantes 30% são divididos para

o conjunto de validação e de teste em partes iguais. Devido ao limitado número de elementos de algumas classes, é realizado um balanceamento no conjunto de dados.

Para balancear os dados de treino é necessário realizar transformações nas imagens, aumentando o número de amostras. Do vasto número de possíveis transformações, optou-se por usar só as que são mais provável de ocorrer nas câmaras dos telemóveis, especificamente, a alteração de brilho, a rotação em 45º, o Zoom, e o espelhar imagem horizontalmente e verticalmente, conforme indicado na Figura 34, este processo é tipicamente denominado *data augmentation* [52].

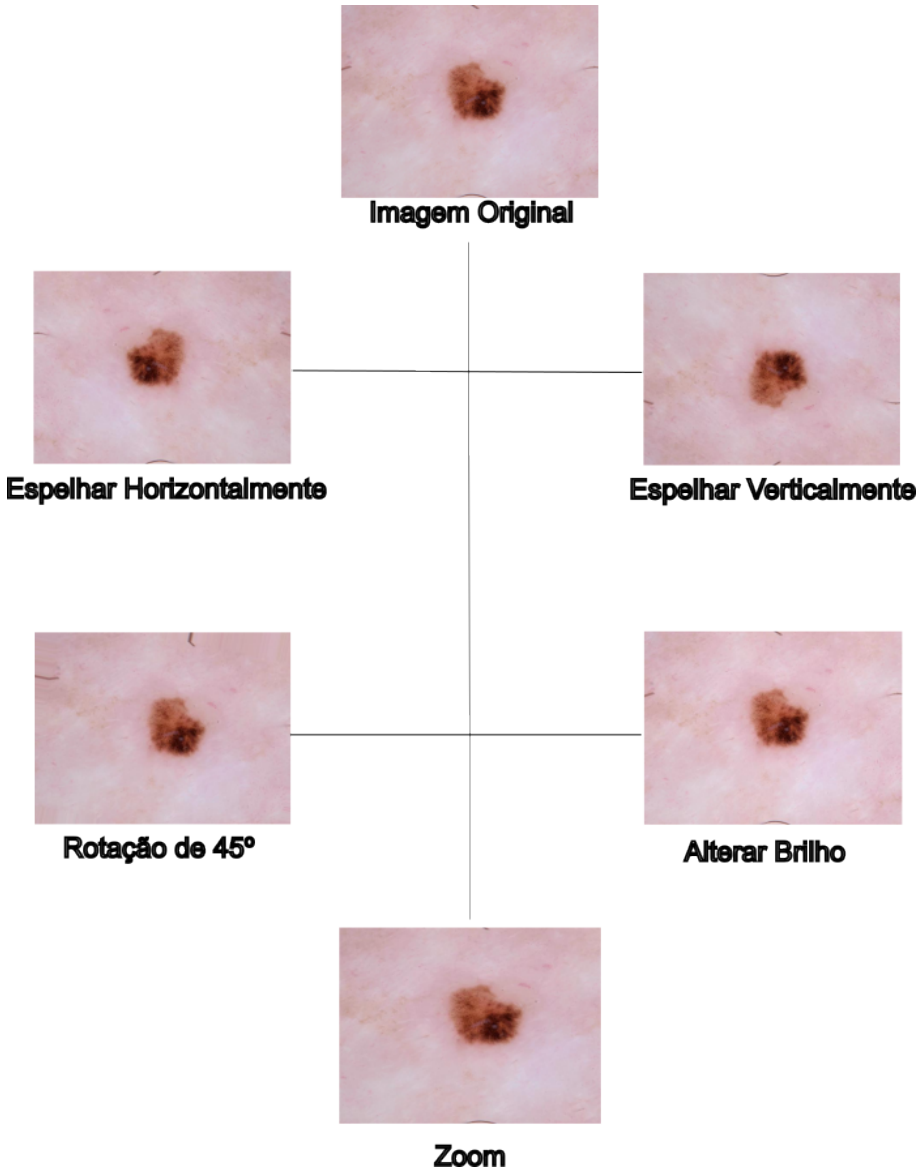

Fig. 34: Data augmentation, transformações de imagem do HAM10000 [4].

Após realizar a *data augmentation* o total de imagens disponíveis estão indicados na tabela II. No conjunto de dados original, ver tabela I, a lesão "Nevos melanocíticos" possui imagens suficientes para realizar o treino, por esse motivo não foram realizadas transformações a esta classe.

Tabela II: Quantidade de imagens para Treino.

Lesão	Quantidade
akiec	1374
bcc	2160
bkl	4614
df	486
mel	4674
vasc	594
nv	4693

O conjunto de dados de teste e validação tem um conjunto inferior de imagens, mas não foram realizadas transformações sobre as mesmas para garantir que as métricas de desempenho a obter são corretas. O total de imagens deste conjunto de dados está indicado nas tabelas III e IV.

Tabela III: Quantidade de imagens para Validação.

Lesão	Quantidade
akiec	49
bcc	77
bkl	165
df	17
mel	167
nv	1006
vasc	21

Tabela IV: Quantidade de imagens para Teste.

Lesão	Quantidade
akiec	49
bcc	77
bkl	165
df	17
mel	167
nv	1006
vasc	22

5.2 Bibliotecas e Hyperparametros

As principais bibliotecas utilizadas foram "Tensorflow 2.15.0", "numpy, versão 1.26.4" e "pandas, versão 2.2.2", entre outras, como "sklearn", "tensorboard", "matplotlib" e "pycm".

Os hiperparâmetros são configurações ou parâmetros externos que não são aprendidos diretamente pelos algoritmos de ML durante o treino, mas são definidos antes do processo de aprendizagem. Eles controlam a aprendizagem do modelo e a sua arquitetura.

Para o treino do modelo foram utilizados os seguintes parâmetros:

- GAUSSIAN NOISE=0.05
- BATCH SIZE=64
- DROPOUT=0.25
- EPOCHS=200

O "GaussianNoise" é utilizado para adicionar ruído aos valores de entrada ou entre camadas ocultas, adicionando ao modelo de rede neuronal uma camada GaussianNoise. Adicionar ruído ao modelo pode ter um efeito de regularização e reduzir o *Overfitting*, quando o modelo não consegue fazer previsões ou conclusões precisas com outros dados que não sejam os utilizados no treino [53].

O "Batch Size" é o número de amostras de dados processadas em cada iteração do algoritmo de treino.

O Dropout é uma técnica que consiste na eliminação de neurónios numa rede neuronal, ver Figura 35. Na rede treinada foi definido que 25% dos neurónios são eliminados temporariamente de forma aleatória, em cada iteração. Todas as conexões do nó eliminado são removidas temporariamente, criando uma nova arquitetura

Tabela V: Bibliotecas Utilizadas.

Biblioteca	Versão	Justificação
Tensorflow	(2.15.0)(2.18.0)	O TensorFlow oferece níveis de abstração para e permite a utilização da API *Keras* de alto nível para criar e treinar modelos
Pandas	(2.2.2)	Pandas é uma biblioteca Python para manipulação e análise de dados, leitura e escrita em diversos formatos (CSV, Excel) e transformação de dados.
Numpy	(1.26.4)	NumPy (Numerical Python) é uma biblioteca fundamental para funções matemáticas, suporte a arrays e manipulação de dados.
Matplotlib	(3.8.0)	Matplotlib é uma biblioteca para criação de gráficos e visualizações de dados
PyCM	(4.0)	PyCM (Python Confusion Matrix) é uma biblioteca Python para calcular e visualizar matrizes de confusão em *machine learning*.
sklearn.metrics	(1.5.2)	É um módulo da biblioteca scikit-learn que fornece funções para avaliar a qualidade de modelos de *machine learning*
Keras	(2.15.0)(3.12.1)	*Keras* é uma biblioteca de rede neuronal que serve como interface de alto nível para construir e treinar redes neuronais.
TF-Keras	(2.14.1)	*TF-Keras* é a implementação de *Keras* integrada no Tensorflow, que substitui o *Keras* original como biblioteca de redes neuronais padrão dentro do ecossistema TensorFlow
Tensorboard	(2.15.2)	Tensorboard é uma ferramenta de visualização do TensorFlow que permite acompanhar métricas do treino de modelos

diferente da original [54]. Esta técnica também ajuda a prevenir o *Overfitting* [17], procurando eliminar ligações menos relevantes.

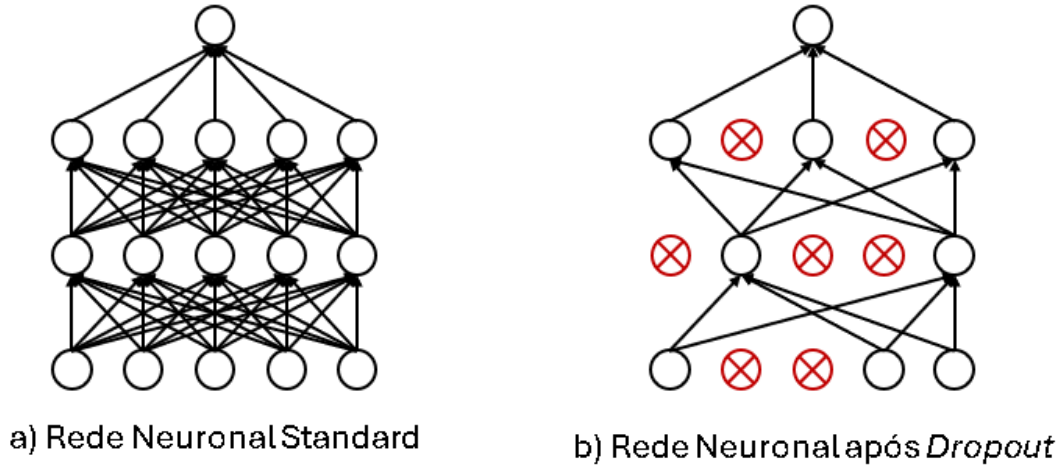

Fig. 35: Dropout [17].

O parametro EPOCHS, representa o número de passagens completas do conjunto de treino por toda a rede neuronal.

5.2.1 Regularização L2

No modelo foi aplicada uma técnica conhecida como regularização L2 ou *Ridge Regression*, de forma reduzir o risco de *overfitting* e garantir uma desempenho mais estável no conjuntos de dados [55].

A regressão de Ridge, também conhecida como regularização L2, é um dos vários tipos de regularização para reduzir os erros causados pela adaptação do modelo ao conjunto dados de treino. A regularização L2 é útil ao desenvolver modelos que tenham um grande número de parâmetros, especialmente se os parâmetros têm pesos altos [56].

A L2 neutraliza coeficientes especialmente altos, reduzindo todos os valores de coeficiente minimizando a adaptação com os dados de treino. Este técnica foi utilizada porque reduz os pesos mas mantém todos os recursos ativos, como todos os dados do conjunto são importantes não é pretendido excluir nenhuma informação, neste trabalho todos os *pixels* das imagens podem contribuir para o desempenho do modelo.

Na tabela VI está um exemplo do cálculo da Regularização L2 para um modelo com 4 pesos

Tabela VI: Exemplo de Regularização L2.

Pesos	$Valor$	$Valor^2$
w_1	0,1	0,01
w_2	-0,2	0,04
w_3	2	4
w_4	-1,5	2,25

O código para aplicar está técnica é utilizado na camada composta por neurónios totalmente conectados ou *Dense Layer*:

layers.Dense(DENSE_ COUNT, activation = 'relu',kernel_ regularizer=l2(0.01)

5.2.2 ReduceLROnPlateau

O *ReduceLROnPlateau* é uma técnica de ajuste dinâmico da taxa de aprendizagem (learning rate), em treinos de redes neuronais, a técnica acompanha uma métrica específica durante o treino, podendo ser utilizadas as métricas "perda" (*loss*) ou "Exatidão" (*accuracy*).

Quando a métrica deixa de melhorar, após um número de épocas definido conhecido como paciência (*Patience*), é reduzida a taxa de aprendizagem (*Learning Rate*) [57].

Configurações utilizadas:

- FACTOR=0.2
- PATIENCE=10
- DELTA= 0.0001
- LR=0.000001

A linha de código utilizada para definir a técnica no modelo é a seguinte:

ReduceLROnPlateau(monitor='val_ sparse_ categorical_ accuracy', factor=0.2, patience=10, verbose=1, mode='auto', min_ delta=0.0001, cooldown=5, min_ lr=0.000001)

5.2.3 EarlyStopping

O EarlyStopping é uma técnica que interrompe o treino de uma rede neuronal quando o desempenho pára de melhorar para uma métrica definida. A função utilizada para definir o número máximo de épocas é *EarlyStopping* da biblioteca *Keras*:

EarlyStopping(monitor="val_ sparse_ categorical_ accuracy", mode="max", patience=20)

5.3 Implementação do Modelo

O conjunto de dados descarregado contém todas as imagens em duas pastas denominadas "HAM10000_images_part_1" e "HAM10000_images_part_2". Também existe um ficheiro csv chamado "HAM10000_metadata.csv" que contém o nome de cada imagem e os dados para cada imagem, como a região da lesão, idade , sexo e tipo de lesão.

Foi criada uma pasta "dataset" contendo subpastas com o nome de cada classe. Com a ajuda do ficheiro "HAM10000_metadata.csv" as imagens foram copiadas para as subpastas da lesão correspondente.

Para a divisão dos dados nos conjuntos de treino, validação e teste foi utilizada a função *train_test_split*, pode ser verificado no seguinte código:

```
#Read original file HAM10000
skin_df = pd.read_csv(mapping_file)


#Create column with path -> mapping
skin_df['image_path'] = skin_df['image_id'].map(all_image_ids.get)
#Drop age, not necessary
skin_df.drop(['age'], axis=1).describe()
#Create column dx_name from acronim
skin_df['dx_name'] = skin_df['dx'].map(dx_name_dict.get)
#create column with id lession / mapping
skin_df['dx_id'] = skin_df['dx'].map(dx_name_id_dict.get).astype(int)



#preparing list to train_test_split 0.7,0.3
selection = skin_df.groupby('image_id').apply(lambda x: x)
selection = selection[['image_id','dx']]
selection= selection.drop_duplicates()

#divide the dataset 0.7, 0.3
raw_t,val_t = train_test_split(selection,
                  test_size = 0.3,
                  stratify = selection['dx'])

#divide the test by 0.5 to valid and test dataset
valid_temp,test_temp= train_test_split(val_t,
                  test_size = 0.5,
                  stratify = val_t['dx'])
```

Código 1: Preparação dos dados do treino

Para realizar o aumento dos dados, foi utilizada a função *ImageDataGenerator* da biblioteca *keras.preprocessing.image*, no código abaixo estão as definições das transformações realizadas as imagens:

```
datagenhf = ImageDataGenerator(horizontal_flip=True)
datagenhf.horizontal_flip = True
datagenhf.flip_horizontal = True
datagenvf = ImageDataGenerator(vertical_flip=True,preprocessing_function=
    lambda x: np.flipud(x))
datagen3 = ImageDataGenerator(rotation_range=45)
datagen3.rotation_range=45
datagen4 = ImageDataGenerator(brightness_range=[0.7,1.2])
datagen4.brightness_range=[0.7,1.2]
datagen5 = ImageDataGenerator(zoom_range=[0.7,1.0])
datagen5.zoom_range=[0.7,1.0]
```

Código 2: Definições das transformações realizadas

No código abaixo está uma porção das transformações aplicadas, o código realiza a transformação do espelhamento horizontal:

```
augmented_imageshf = datagenhf.flow(img_array,batch_size=1,shuffle=False,
    seed=42)[0]
image_flippedhf = augmented_imageshf[0]
image_flippedhf2 = Image.fromarray(image_flippedhf.astype('uint8'))

x=str(df2['image_path'][i]).split('/')
#copy to new directory

newfilename= x[len(x)-1].split('.')[0] + '_hf.jpg'
newpath=newdir+split+df2['dx'][i]+'/'+newfilename

# Save the image
image_flippedhf2.save('/home/jonathan/Documentos'+newpath)
```

Código 3: Transformação de espelhamento horizontal

Após a realização de todas as transformações para aumentar os dados, são definidos os parâmetros do treino do modelo:

```
RESOLUTION=224 #default
IMG_SIZE = (RESOLUTION, RESOLUTION) # [(224, 224), (384, 384), (512, 512),
    (640, 640)]
BATCH_SIZE = 64 # [1, 8, 16, 24]
GAUSSIAN_NOISE = 0.05
DROPOUT = 0.25
DENSE_COUNT = 256
LEARN_RATE = 0.0001
```

```
EPOCHS = 200
CLASSES=['akiec', 'bcc', 'bkl', 'df', 'mel', 'nv', 'vasc']
```

Código 4: Definição de Hyperparametros

Carregamento de imagens:

```
train_ds =tf.keras.utils.image_dataset_from_directory('./dataset/train/',
                                                      labels='inferred',
                                                      label_mode='int',
                                                      class_names=CLASSES,
                                                      color_mode='rgb',
                                                      batch_size=BATCH_SIZE,
                                                      image_size=IMG_SIZE)
valid_ds =tf.keras.utils.image_dataset_from_directory('./dataset/valid/',
                                                      labels='inferred',
                                                      label_mode='int',
                                                      class_names=CLASSES,
                                                      color_mode='rgb',
                                                      batch_size=BATCH_SIZE,
                                                      image_size=IMG_SIZE)
```

Código 5: Carregamento de imagens a partir da pasta

Dependendo do modelo a treinar, são definidas as variáveis da resolução da imagem e do modelo a carregar:

```
RESOLUTION=384
from keras.applications.convnext import ConvNeXtXLarge as Model
base_pretrained_model = Model(input_shape =  (RESOLUTION, RESOLUTION, 3),
                              include_top = False, weights = 'imagenet')
base_pretrained_model.trainable = False
```

Código 6: Carregamento do modelo com transferência de conhecimento

Definição das camadas do modelo a treinar:

```
#Modelo layers:

X = layers.Input((RESOLUTION, RESOLUTION, 3), name='image_rgb_in')
Y = layers.GaussianNoise(GAUSSIAN_NOISE)(X)
Y = base_pretrained_model(Y)
Y = layers.BatchNormalization()(Y)
Y = layers.Flatten()(Y)
Y = layers.Dropout(DROPOUT)(Y)

Y = layers.Dropout(DROPOUT)(layers.Dense(DENSE_COUNT, activation = 'relu',
    kernel_regularizer=l2(0.01))(Y))
Y = layers.Dense(len(CLASSES), activation = 'softmax')(Y)
```

```
Model = models.Model(inputs = X, outputs = Y, name = 'full_model')

optimizer = Adam(learning_rate=LEARN_RATE)
Model.compile(optimizer,
              loss = 'sparse_categorical_crossentropy',
              metrics = ['sparse_categorical_accuracy'])
```

Código 7: Carregamento de imagens a partir da pasta

Definição de callbacks para monitorização durante o treino:

```
checkpoint = ModelCheckpoint(weight_path, monitor='
    val_sparse_categorical_accuracy', verbose=1, save_best_only=True, mode=
    'max')

reduceLROnPlat =  ReduceLROnPlateau(monitor='
    val_sparse_categorical_accuracy', factor=0.2, patience=10, verbose=1,
    mode='auto', min_delta=0.0001, cooldown=5, min_lr=0.000001)
early = EarlyStopping(monitor="val_sparse_categorical_accuracy",
                      mode="max",
                      patience=20)
log_dir = "./logs/" +BASE_MODEL+"_" +datetime.datetime.now().strftime("%Y%m
    %d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(
    log_dir=log_dir,
    histogram_freq=1)

callbacks_list = [checkpoint, early, reduceLROnPlat,csv_logger,
    tensorboard_callback]
train_ds.batch_size = BATCH_SIZE
fit_results = Model.fit(train_ds,
                      validation_data=valid_ds,
                      epochs = EPOCHS,
                      callbacks = callbacks_list)
```

Código 8: Treino e Monitorização do modelo

5.4 Exemplo de um Modelo

O modelo resultante encontra-se descrito na tabela VII, esta tabela contem as camadas do modelo ConvNeXtXLarge e os respetivos números de parâmetros por camada. As restantes redes encontram-se disponíveis para consulta nos anexos.

Tabela VII: Modelo ConvNeXtXLarge com as camadas e número parâmetros.

Camada	**Saida**	**Param #**
image_rgb_in (InputLayer)	(None, 384, 384, 3)	0
gaussian_noise (GaussianNoise)	(None, 384, 384, 3)	0
convnext_xlarge (Functional)	(None, 12, 12, 2048)	348147968
batch_normalization	(None, 12, 12, 2048)	8192
flatten (Flatten)	(None, 294912)	0
dropout (Dropout)	(None, 294912)	0
dense (Dense)	(None, 256)	75497728
dropout_1 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 7)	1799

5.5 Pontos-chave

O dataset HAM10000 é amplamente utilizado para estudos relacionados a classificação de lesões na pele. Este conjunto de dados é desbalanceado e para corrigir o balanceamento dos dados de treino é necessário realizar transformações nas imagens, nomeadamente alteração de brilho, rotação em 45º, o Zoom, e o espelhamento horizontal e vertical da imagem, aumentando o número de amostras.

Para reduzir ou evitar o *overfitting* foram aplicadas algumas técnicas, nomeadamente o *Dropout*, onde são eliminadas aleatoriamente 25% das ligações do modelo, foi efetuada a regularização L2 que penaliza os pesos da camada *Dense*, incentivando a rede a utilizar pesos menores e o *batch normalization* em que os valores de ativação são normalizados.

Adicionalmente, foi implementado o *early stopping* para monitorizar a métrica de exatidão da validação e interromper o treinamento caso não exista melhoria após 20 épocas.

Foi utilizada a técnica de *Transfer Learning* dos modelos existentes no *Keras* para realizar o treino, congelando as camadas do modelo base, e definindo o ajuste do *Learning Rate* a cada 10 épocas, caso não exista um aumento no desempenho da métrica *val_sparse_categorical_accuracy*.

6 Resultados

Neste capítulo são analisados os resultados obtidos para cada arquitetura implementada, recorrendo às métricas padrão de ML, nomeadamente exatidão macro (ACC Macro), exatidão geral (Overall ACC), F1 macro e o coeficiente de correlação de Matthews (MCC). São descritas as métricas e apresentados os resultados dos modelos para cada uma delas. Depois, são analisados os gráficos de treino e validação para o modelo com melhor desempenho geral e o *Hardware* utilizados.

6.1 Métricas

Para analisar o desempenho do modelo são utilizadas diversas métricas, que informam sobre a precisão do desempenho do modelo para um conjunto de dados. As métricas utilizadas para analisar os modelos foram calculadas a partir dos dados da tabela conhecida como Matriz de Confusão.

Foram analisadas métricas Macro e Micro, onde as métricas Macro calculam a média das métricas individuais de cada classe, dando peso igual para todas as classes, e as Micro agregam os resultados de todas as classes para calcular a métrica.

6.1.1 Matriz de confusão

A Matriz de Confusão (MC) é uma tabela que permite visualizar o desempenho de um modelo, apresentando a relação entre as predições do modelo e os valores reais. Para uma classificação binária (2 classes), a matriz tem o seguinte formato:

Tabela VIII: Matriz de confusão.

	Valor Real	
Predição	Positivo	Negativo
Positivo	VP	FP
Negativo	FN	VN

A representação visual destes conceitos pode ser verificada na figura 36.

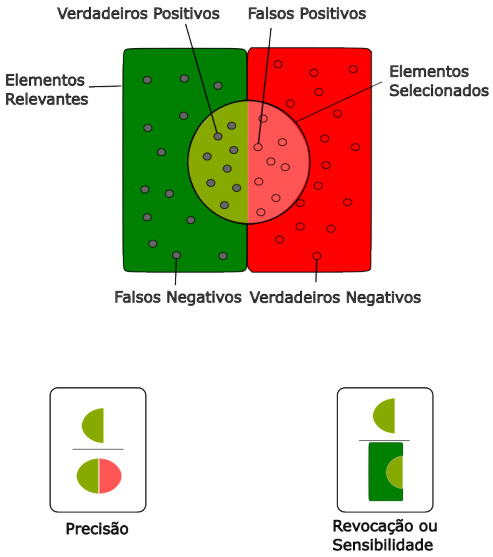

Fig. 36: Precisão e Revocacão [18]

A partir da MC, podem-se calcular várias métricas. As utilizadas para analise de resultados foram as seguintes:

- Exatidão (*accuracy*) macro - ACC macro
- F1 Macro
- Taxa de Falsos Negativos Macro - FNR Macro
- Taxa de Falsos Positivos Macro -FPR Macro
- Taxa de Verdadeiros Negativos Macro -TNR Macro
- Taxa de Verdadeiros Positivos Macro -TPR Macro
- Coeficiente de Correlação de Matthews - Overall MCC
- Exatidão Geral - Overall ACC
- Área sob a curva - AUC

A métrica ACC é um bom indicador quando as classes estão balanceadas, isto é, quando a quantidade de amostras é similar para cada classe. No caso de ter dados desbalanceados o F1-Score e o MCC são métricas indicadas para analise.

Outra métrica é AUC (Area Under the Curve) que é à área sobre uma curva num gráfico, que mede o desempenho do modelo de classificação.

Como foi explicado anteriormente, a matriz de confusão é uma ferramenta que permite avaliar o desempenho do modelo de classificação. São apresentadas as matrizes de confusão para os testes realizados sem Regularização L2 na Figura 37, e com Regularização L2 na Figura 38. Os casos corretamente classificados para cada classe encontram-se na diagonal:

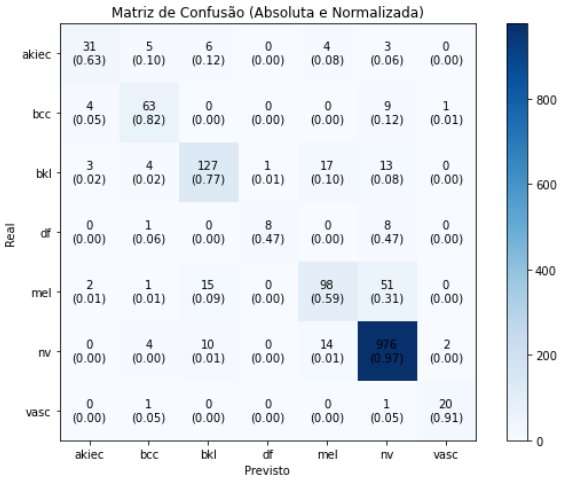

Fig. 37: Resultado da Matriz de confusão com dados de Teste. Sem Regularização L2

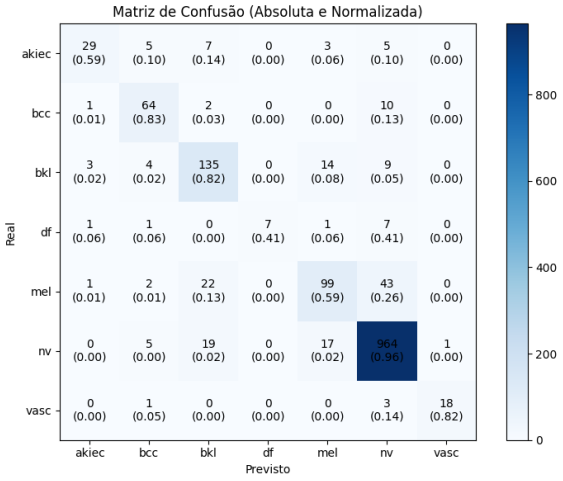

Fig. 38: Resultado da Matriz de confusão com dados de Teste. Com Regularização L2

Ao comparar as duas matrizes pode-se verificar que a classe "nv" e "vasc", no modelo sem L2, apresenta uma taxa de acerto maior mas para a classe "bkl" teve

uma taxa de acerto inferior. A classe "df" teve um desempenho inferior na matriz sem L2, com menos previsões corretas e mais confusões.

O modelo sem regularização L2 apresenta uma melhor capacidade de distinguir a classe "nv" e "vasc", mas apresenta uma leve perda de precisão noutras classes como "bkl" e "df". A regularização L2 ajudou a distribuir melhor as previsões, reduzindo *overfitting* para algumas classes.

Os valores fora da diagonal representam os erros de classificação, por exemplo, para a classe 'akiec', na posição (0, 1) encontra-se o valor 5, o qual indica que 5 imagens dessa classe foram incorretamente classificadas como 'bcc'.

A classe 'nv' teve 964 classificações corretas, o que é significativamente maior que as classificações incorretas, indicando que o modelo consegue classificar com bom resultado essa classe.

A classe 'df' teve 7 classificações corretas, e foi classificada incorretamente como 'nv'.

Para a classe 'vasc' existiram 18 classificações corretas, mas existiram classificações incorretas com a classe 'nv' e 'bcc'.

O modelo tem um bom desempenho para as classe "nv", "bcc" e "vasc", apresentando um desempenho inferior nas clases "df" e "mel". No caso da classe "mel" o modelo está a confundir melanomas com "nv" e "bkl".

A classe "bkl" tem uma taxa de acerto de 82%, mas existe confusão com "mel" e "bcc", uma possível causa é que algumas imagens podem apresentar padrões similares.

Das imagens testadas pode-se verificar a taxa de imagens classificadas corretamente na tabela IX

Tabela IX: Taxa de Classificações Corretas com Regularização L2.

Classe	**Classificações Corretas**	**Total de Amostras**	**Taxa de Acerto (%)**
akiec	29	49	**59.18%**
bcc	64	77	**83.12%**
bkl	135	165	**81.82%**
df	7	17	**41.18%**
mel	99	167	**59.28%**
nv	964	1006	**95.83%**
vasc	18	22	**81.82%**

6.1.2 ACC Macro

A exatidão Macro (ACC Macro) é a média das exatidões calculadas independentemente para cada classe, indicando à exatidão geral do modelo considerando todas as classes.

$$ACCMacro = (ACCclasse_1 + ACCclasse_2 + ... + ACCclasse_n)/n. \quad (8)$$

Onde:

n = número de classes

Para cada classe:

$$ACCclasse_i = \frac{(VP_i + VN_i)}{(VP_i + VN_i + FP_i + FN_i)}. \quad (9)$$

Na tabela X estão descritos os valores obtidos para esta métrica para cada um dos modelos testados. O melhor desempenho foi alcançado pelo ConvNeXtXLarge, com um ACC Macro de 0.96464 (96,46%), com regularização L2 e de 0.96578(96,57%) sem a regularização, sendo o modelo mais preciso entre todos os modelos testados.

Os modelos com pior desempenho foram InceptionResNetV2 com 87,75% com L2 e 87,26% sem L2, e o InceptionV3 perto do 90% com e sem regularização.

As famílias EfficientNet e ConvNeXt apresentaram consistência com a maioria dos modelos. Todos obtiveram bom desempenho na métrica ACC Macro.

Tabela X: ACC Macro dos modelos.

Modelo	**ACC Macro L2**	**ACC Macro Sem L2**
ConvNeXtBase	0.95742	0.95571
ConvNeXtLarge	0.96236	0.96179
ConvNeXtSmall	0.94753	0.94753
ConvNeXtTiny	0.94886	0.94658
ConvNeXtXLarge	0.96464	0.96578
DenseNet121	0.94031	0.93822
DenseNet169	0.93023	0.9367
DenseNet201	0.93898	0.93822
EfficientNetB0	0.95533	0.95419
EfficientNetB1	0.95552	0.95305
EfficientNetB2	0.95191	0.94905
EfficientNetB3	0.95038	0.94981
EfficientNetB4	0.94829	0.94487
EfficientNetB5	0.94563	0.94848
EfficientNetB6	0.94335	0.94411
EfficientNetB7	0.9443	0.93442
EfficientNetV2_B0	0.95229	0.95343
EfficientNetV2_B1	0.95229	0.95229
EfficientNetV2_B2	0.95666	0.95495
EfficientNetV2_B3	0.95229	0.95058
EfficientNetV2_L	0.9443	0.94221
EfficientNetV2_M	0.9424	0.93955
EfficientNetV2_S	0.9462	0.94677
InceptionResNetV2	0.87758	0.87264
InceptionV3	0.90058	0.90229
MobileNet	0.91065	0.91294
MobileNetV2	0.90685	0.91008
NASNetLarge	0.91161	0.89887
NASNetMobile	0.90761	0.90362
RESNET101	0.94487	0.94145
RESNET101V2	0.91256	0.92016
RESNET152	0.9443	0.93974
RESNET152V2	0.90533	0.89868
RESNET50	0.94753	0.94183
RESNET52	0.91579	0.9213
VGG16	0.93537	0.93309
VGG19	0.93119	0.93309
Xception	0.91313	0.9078

6.1.3 F1 Macro

O F1 Score é uma métrica de avaliação que combina duas métricas importantes, a Precisão (*Precision*) e a Sensibilidade ou Revocação (*Recall*), através de sua média harmónica [18].

A precisão mede a proporção de predições positivas corretas, e é calculada conforme é apresentado na seguinte fórmula:

$$Precisao = \frac{VP}{(VP + FP)}. \tag{10}$$

A revocação mede a proporção de casos positivos reais que foram identificados corretamente, e é calculada conforme indicado na seguinte fórmula:

$$Revocacao = \frac{VP}{(VP + FN)}. \tag{11}$$

Para calcular o F1 Score é utilizada a precisão e a revocação conforme indicado na seguinte fórmula:

$$F1 = 2 \times \frac{(Precisao \times Revocacao)}{(Precisao + Revocacao)}. \tag{12}$$

O F1 Macro é a média dos F1-scores calculados separadamente para cada classe, penalizando o modelo se existe baixo desempenho em alguma das classes, dando uma visão mais equilibrada do desempenho.

Ao analisar os resultado do F1 Macro da tabela XI, conclui-se que o modelo ConvNeXtXLarge é o melhor modelo, com um F1 Macro de 76,15% com regularização L2 e 77,03% sem L2. A família ConvNeXt além da melhor exatidão, também apresenta o melhor equilíbrio entre precisão e revocação.

Comparando o ACC Macro com o F1 Macro, é visível uma quebra no resultado da métrica F1 Macro, indicando alguma dificuldade com classes desbalanceadas ou minoritárias.

Tabela XI: Resultado da métrica F1 Macro.

Modelo	**F1 Macro L2**	**F1 Macro Sem L2**
ConvNeXtBase	0.726	0.71801
ConvNeXtLarge	0.74302	0.73854
ConvNeXtSmall	0.66321	0.66598
ConvNeXtTiny	0.67731	0.64671
ConvNeXtXLarge	0.76153	0.77038
DenseNet121	0.65487	0.64688
DenseNet169	0.58909	0.57189
DenseNet201	0.63302	0.61423
EfficientNetB0	0.72748	0.70874
EfficientNetB1	0.70189	0.69157
EfficientNetB2	0.69407	0.66344
EfficientNetB3	0.66976	0.65105
EfficientNetB4	0.65981	0.61469
EfficientNetB5	0.65679	0.68638
EfficientNetB6	0.61938	0.60787
EfficientNetB7	0.59743	0.55453
EfficientNetV2_B0	0.68594	0.68652
EfficientNetV2_B1	0.69759	0.67754
EfficientNetV2_B2	0.72285	0.72966
EfficientNetV2_B3	0.69592	0.67329
EfficientNetV2_L	0.63298	0.62318
EfficientNetV2_M	0.6413	0.63767
EfficientNetV2_S	0.67127	0.65115
InceptionResNetV2	0.25886	0.2834
InceptionV3	0.30434	0.40425
MobileNet	0.44031	0.4482
MobileNetV2	0.3916	0.42039
NASNetLarge	0.21379	0.33581
NASNetMobile	0.39608	0.39799
RESNET101	0.66338	0.61249
RESNET101V2	0.42049	0.46438
RESNET152	0.63179	0.57399
RESNET152V2	0.36629	0.40085
RESNET50	0.64825	0.61175
RESNET52	0.43954	0.53802
VGG16	0.58059	0.5731
VGG19	0.58272	0.56235
Xception	0.44297	0.42154

6.1.4 FNR (False Negative Rate) - Taxa de Falsos Negativos

Indica a proporção de positivos classificados incorretamente como negativos, representada da seguinte forma:

$$FNR = \frac{FN}{(FN + VP)}. \tag{13}$$

A FNR Macro é a média das taxas de falsos negativos de cada classe

$$FNRMacro = \frac{(FNRclasse_1 + FNRclasse_2 + ... + FNRclasse_n)}{n}. \tag{14}$$

Onde para cada classe:

$$FNRclasse_i = \frac{FNclasse_i}{(FNclasse_i + VPclasse_i)}. \tag{15}$$

A FNR Micro é a taxa global de falsos negativos, considerando todas as classes juntas:

$$FNRMicro = \frac{FN_{Total}}{(FN_{Total} + VP_{Total})}. \tag{16}$$

FN_{Total} = Soma dos FN de todas as classes

VP_{Total} = Soma dos VP de todas as classes

Nas tabelas XIII e XII encontram-se os resultados para esta métrica com e sem Regularização L2, sendo possível verificar que o modelo ConvNeXtXLarge apresenta o menor número de falsos negativos. O modelo sem regularização L2 apresentou melhor valor, 26,32% enquanto com regularização apresentou 28,24% , indicando que o modelo comete menos erros ao classificar imagens que pertencem à classe positiva.

6.1.5 TPR (True Positive Rate) - Taxa de Verdadeiros Positivos

O TPR (True Positive Rate) ou Taxa de Verdadeiros Positivos, conhecido como Sensibilidade ou Recall, indica a proporção de positivos classificados corretamente. É calculado da seguinte forma:

$$TPR = \frac{VP}{(VP + FN)}. \tag{17}$$

O TPR Macro é a média das taxas de verdadeiros positivos de cada classe

$$TPR_Macro = \frac{(TPRclasse_1 + TPRclasse_2 + ... + TPRclasse_n)}{n}. \quad (18)$$

Onde para cada classe:

$$TPRclasse_i = \frac{VPclasse_i}{(VPclasse_i + FNclasse_i)}. \quad (19)$$

O TPR Micro é a taxa global de verdadeiros positivos

$$TPRMicro = \frac{VP_{Total}}{(VP_{Total} + FN_{Total})}. \quad (20)$$

Onde: VP_{Total} = Soma dos VP de todas as classes FN_{Total} = Soma dos FN de todas as classes

Na tabelas XIII e XII, é possível verificar que o modelo ConvNeXtXLarge, novamente apresenta o melhor resultado para esta métrica, no modelo com regularização apresentou 71,76% e no modelos sem L2 73,67%, indicando que o modelo está identificando corretamente mais imagens verdadeiras positivas.

6.1.6 FPR (False Positive Rate) - Taxa de Falsos Positivos

Indica a proporção de negativos classificados incorretamente como positivos, representada com a seguinte fórmula:

$$FPR = \frac{FP}{(FP + VN)}. \quad (21)$$

A FPR Macro é a média das taxas de falsos positivos de cada classe

$$FPRMacro = \frac{(FPRclasse_1 + FPRclasse_2 + ... + FPRclasse_n)}{n}. \quad (22)$$

Onde para cada classe:

$$FPRclasse_i = \frac{FPclasse_i}{(FPclasse_i + VNclasse_i)}. \quad (23)$$

A FPR Micro é Taxa global de falsos positivos

$$FPR_Micro = \frac{FP_{Total}}{(FP_{Total} + VN_{Total})}. \quad (24)$$

FP_{Total} = Soma dos FP de todas as classes

VN_{Total} = Soma dos VN de todas as classes

É possível verificar nas tabelas XIII e XII, que o modelo ConvNeXtXLarge apresentou o melhor resultado, no caso do modelo com regularização o valor foi de 3,36% e sem L2 foi de 3,43% de falsos positivos.

6.1.7 TNR (True Negative Rate) - Taxa de Verdadeiros Negativos

O TNR (True Negative Rate) ou Taxa de Verdadeiros Negativos, também conhecido como Especificidade, indica a proporção de casos negativos que foram corretamente classificados como negativos. É calculado da seguinte forma:

$$TNR = \frac{VN}{(VN + FP)}. \quad (25)$$

O TNR Macro é a média das taxas de verdadeiros negativos de cada classe

$$TNRMacro = \frac{(TNRclasse_1 + TNRclasse_2 + ... + TNRclasse_n)}{n}. \quad (26)$$

Onde para cada classe:

$$TNRclasse_i = \frac{VNclasse_i}{(}VNclasse_i + FPclasse_i). \quad (27)$$

O TNR Micro é a taxa global de verdadeiros negativos

$$TNR_Micro = \frac{VN_{Total}}{(VN_{Total} + FP_{Total})}. \quad (28)$$

Onde: VN_{Total} = Soma dos VN de todas as classes FP_{Total} = Soma dos FP de todas as classes

Nas tabelas XIII e XII encontram-se as métricas Macro, e é possível verificar que o modelo ConvNeXtXLarge apresentou o melhor desempenho para TNR, no modelo com L2 foi de 96,64% e sem L2 foi de 96,56%.

Tabela XII: Resultado das métricas FNR, TNR, FPR, TPR sem L2.

Modelo	**FNR Macro**	**FPR Macro**	**TNR Macro**	**TPR Macro**
ConvNeXtBase	0.3071	0.04226	0.95774	0.6929
ConvNeXtLarge	0.29909	0.03567	0.96433	0.70091
ConvNeXtSmall	0.38619	0.05104	0.94896	0.61381
ConvNeXtTiny	0.39085	0.05391	0.94609	0.60915
ConvNeXtXLarge	0.26325	0.03436	0.96564	0.73675
DenseNet121	0.36426	0.05136	0.94864	0.63574
DenseNet169	0.45265	0.05569	0.94431	0.54735
DenseNet201	0.40741	0.05506	0.94494	0.59259
EfficientNetB0	0.31403	0.04129	0.95871	0.68597
EfficientNetB1	0.32407	0.03867	0.96133	0.67593
EfficientNetB2	0.35902	0.04649	0.95351	0.64098
EfficientNetB3	0.36522	0.04223	0.95777	0.63478
EfficientNetB4	0.40594	0.05111	0.94889	0.59406
EfficientNetB5	0.34765	0.04505	0.95495	0.65235
EfficientNetB6	0.42068	0.05449	0.94551	0.57932
EfficientNetB7	0.45028	0.05977	0.94023	0.54972
EfficientNetV2_B0	0.34693	0.04173	0.95827	0.65307
EfficientNetV2_B1	0.3568	0.04616	0.95384	0.6432
EfficientNetV2_B2	0.29964	0.04095	0.95905	0.70036
EfficientNetV2_B3	0.35759	0.04749	0.95251	0.64241
EfficientNetV2_L	0.40641	0.05413	0.94587	0.59359
EfficientNetV2_M	0.40454	0.05876	0.94124	0.59546
EfficientNetV2_S	0.38863	0.0511	0.9489	0.61137
InceptionResNetV2	0.68024	0.08673	0.91327	0.31976
InceptionV3	0.57593	0.06954	0.93046	0.42407
MobileNet	0.57277	0.07235	0.92765	0.42723
MobileNetV2	0.60392	0.07912	0.92088	0.39608
NASNetLarge	0.63076	0.07418	0.92582	0.36924
NASNetMobile	0.59845	0.07813	0.92187	0.40155
RESNET101	0.40587	0.04943	0.95057	0.59413
RESNET101V2	0.52772	0.06032	0.93968	0.47228
RESNET152	0.41275	0.04989	0.95011	0.58725
RESNET152V2	0.56413	0.06501	0.93499	0.43587
RESNET50	0.40672	0.05251	0.94749	0.59328
RESNET52	0.4539	0.05684	0.94316	0.5461
VGG16	0.43876	0.05321	0.94679	0.56124
VGG19	0.42439	0.04922	0.95078	0.57561
Xception	0.57347	0.06703	0.93297	0.42653

Tabela XIII: Resultado das métricas FNR, TNR, FPR, TPR com L2.

Modelo	**FNR Macro**	**FPR Macro**	**TNR Macro**	**TPR Macro**
ConvNeXtBase	0.31447	0.04024	0.95976	0.68553
ConvNeXtLarge	0.29694	0.03521	0.96479	0.70306
ConvNeXtSmall	0.38217	0.05066	0.94934	0.61783
ConvNeXtTiny	0.38192	0.05286	0.94714	0.61808
ConvNeXtXLarge	0.2824	0.0336	0.9664	0.7176
DenseNet121	0.37323	0.05116	0.94884	0.62677
DenseNet169	0.42185	0.05674	0.94326	0.57815
DenseNet201	0.38868	0.0518	0.9482	0.61132
EfficientNetB0	0.30094	0.04014	0.95986	0.69906
EfficientNetB1	0.3079	0.03565	0.96435	0.6921
EfficientNetB2	0.32424	0.04432	0.95568	0.67576
EfficientNetB3	0.34004	0.04208	0.95792	0.65996
EfficientNetB4	0.36097	0.04689	0.95311	0.63903
EfficientNetB5	0.36474	0.04896	0.95104	0.63526
EfficientNetB6	0.40952	0.05303	0.94697	0.59048
EfficientNetB7	0.42253	0.05223	0.94777	0.57747
EfficientNetV2_B0	0.33834	0.04163	0.95837	0.66166
EfficientNetV2_B1	0.34024	0.04599	0.95401	0.65976
EfficientNetV2_B2	0.30908	0.03981	0.96019	0.69092
EfficientNetV2_B3	0.33393	0.04386	0.95614	0.66607
EfficientNetV2_L	0.41762	0.0545	0.9455	0.58238
EfficientNetV2_M	0.39297	0.05512	0.94488	0.60703
EfficientNetV2_S	0.36265	0.05065	0.94935	0.63735
InceptionResNetV2	0.71242	0.09356	0.90644	0.28758
InceptionV3	0.66724	0.07645	0.92355	0.33276
MobileNet	0.57416	0.07264	0.92736	0.42584
MobileNetV2	0.63394	0.08496	0.91504	0.36606
NASNetLarge	0.75729	0.08685	0.91315	0.24271
NASNetMobile	0.61164	0.07727	0.92273	0.38836
RESNET101	0.35314	0.047	0.953	0.64686
RESNET101V2	0.56382	0.06393	0.93607	0.43618
RESNET152	0.3843	0.04554	0.95446	0.6157
RESNET152V2	0.58085	0.06508	0.93492	0.41915
RESNET50	0.3779	0.04594	0.95406	0.6221
RESNET52	0.55047	0.06474	0.93526	0.44953
VGG16	0.42227	0.04916	0.95084	0.57773
VGG19	0.41478	0.04974	0.95026	0.58522
Xception	0.55787	0.06549	0.93451	0.44213

6.1.8 Overall MCC

O MCC é uma métrica que mede a qualidade geral da classificação, sendo importante para avaliação de modelos de classificação. O resultado do coeficiente varia entre -1 e 1, onde 1 indica previsão perfeita, 0 aleatória e -1 uma previsão incorreta.

$$MCC = \frac{(VP \times VN - FP \times FN)}{\sqrt{((VP + FP)(VP + FN)(VN + FP)(VN + FN))}}. \tag{29}$$

Verificando a função, pode-se ver que são tomados em conta todos os valores da Matriz de confusão, sendo assim uma das métricas mais completas para realizar avaliação de modelos de classificação.

Na tabela XIV, a familia dos modelos ConvNeXt foram as que apresentaram melhor resultado. O modelo ConvNeXtXLarge apresentou o melhor desempenho, com 75,81% com regularização L2 e 76,37% sem L2.

Tabela XIV: Resultados da métrica Overall MCC.

Modelo	**Overall MCC Com L2**	**Overall MCC Sem L2**
ConvNeXtBase	0.70791	0.69572
ConvNeXtLarge	0.74239	0.73866
ConvNeXtSmall	0.63467	0.63387
ConvNeXtTiny	0.63936	0.62565
ConvNeXtXLarge	0.75819	0.76379
DenseNet121	0.60431	0.59533
DenseNet169	0.55124	0.57025
DenseNet201	0.59708	0.5847
EfficientNetB0	0.69736	0.69013
EfficientNetB1	0.70735	0.68801
EfficientNetB2	0.67283	0.65433
EfficientNetB3	0.66665	0.664
EfficientNetB4	0.65031	0.62384
EfficientNetB5	0.63176	0.65439
EfficientNetB6	0.61042	0.61046
EfficientNetB7	0.61723	0.5545
EfficientNetV2_B0	0.67972	0.68399
EfficientNetV2_B1	0.67153	0.67188
EfficientNetV2_B2	0.70503	0.69452
EfficientNetV2_B3	0.67461	0.65851
EfficientNetV2_L	0.61146	0.60282
EfficientNetV2_M	0.60185	0.57935
EfficientNetV2_S	0.63028	0.63115
InceptionResNetV2	0.28694	0.31593
InceptionV3	0.38653	0.42551
MobileNet	0.42384	0.43074
MobileNetV2	0.36837	0.39967
NASNetLarge	0.36595	0.39926
NASNetMobile	0.39823	0.39314
RESNET101	0.63466	0.61167
RESNET101V2	0.47683	0.50693
RESNET152	0.636	0.60303
RESNET152V2	0.45315	0.44872
RESNET50	0.64815	0.6025
RESNET52	0.47377	0.52301
VGG16	0.58626	0.56449
VGG19	0.57266	0.57725
Xception	0.45657	0.44131

6.1.9 Overall ACC

Overall ACC ou exatidão global de um modelo, é a taxa de classificação corretas do modelo.

A *accuracy* ou exatidão é uma métrica de avaliação de desempenho usada para medir a capacidade do modelo de fazer previsões corretas.

Está é calculada como a razão entre o número de previsões corretas e o número total de previsões feitas:

ACC= Previsões Corretas / Total de Previsões

Por exemplo, se um modelo realizou 85 previsões corretas de um total de 100, o modelo apresenta 85% de exatidão.

Overall ACC é à média da exatidão do modelo sobre todas as classes, considerando todos os dados de teste ou validação.

Na tabela XV estão apresentados os resultados de esta métrica para todos os modelos treinados. O modelo ConvNeXtXLarge apresentou o melhor desempenho com 87,62% utilizando Regularização L2 e 88% sem regularização.

Tabela XV: Resultados da métrica Overall ACC.

Modelo	**Overall ACC Com L2**	**Overall ACC Sem L2**
ConvNeXtBase	0.85096	0.84498
ConvNeXtLarge	0.86826	0.86627
ConvNeXtSmall	0.81637	0.81637
ConvNeXtTiny	0.82102	0.81304
ConvNeXtXLarge	0.87625	0.88024
DenseNet121	0.79108	0.78377
DenseNet169	0.75582	0.77844
DenseNet201	0.78643	0.78377
EfficientNetB0	0.84365	0.83965
EfficientNetB1	0.84431	0.83566
EfficientNetB2	0.83167	0.82169
EfficientNetB3	0.82635	0.82435
EfficientNetB4	0.81903	0.80705
EfficientNetB5	0.80971	0.81969
EfficientNetB6	0.80173	0.80439
EfficientNetB7	0.80506	0.77046
EfficientNetV2_B0	0.833	0.83699
EfficientNetV2_B1	0.833	0.833
EfficientNetV2_B2	0.8483	0.84232
EfficientNetV2_B3	0.833	0.82701
EfficientNetV2_L	0.80506	0.79774
EfficientNetV2_M	0.7984	0.78842
EfficientNetV2_S	0.81171	0.81371
InceptionResNetV2	0.57152	0.55422
InceptionV3	0.65203	0.65802
MobileNet	0.68729	0.69528
MobileNetV2	0.67399	0.6853
NASNetLarge	0.69062	0.64604
NASNetMobile	0.67665	0.66267
RESNET101	0.80705	0.79508
RESNET101V2	0.69395	0.72056
RESNET152	0.80506	0.78909
RESNET152V2	0.66866	0.64538
RESNET50	0.81637	0.79641
RESNET52	0.70526	0.72455
VGG16	0.77379	0.7658
VGG19	0.75915	0.7658
Xception	0.69594	0.67731

6.1.10 AUC

A AUC é um desempenho abrangente que resume a capacidade do classificador em todos os limites de classificação possíveis [58].

A curva ROC (Receiver Operator Characteristic) é uma métrica de avaliação para problemas de classificação binária, é uma curva de probabilidade que apresenta o desempenho de um modelo de classificação, o TPR contra o FPR.

A AUC quantifica a área geral debaixo de toda a curva ROC desde o ponto (0,0) até (1,1). Nesta curva, um maior valor no eixo X indica um número maior de Falsos Positivos do que Verdadeiros Negativos. Um número maior no eixo Y indica um número maior de Verdadeiros Positivos do que Falsos Negativos [59].

A AUC é específica para problemas de classificação binária, mas pode ser utilizada em problemas de classificação multiclasse adaptando o problema. Os métodos heurísticos podem ser utilizados para dividir um problema de classificação multiclasse em vários conjuntos de dados de classificação binária e treinar um modelo de classificação binária para cada classe .

Existem dois métodos heurísticos para utilizar algoritmos de classificação binária para classificação multiclasse, nomeadamente:

- *One-vs-One* (OvO), é um método heurístico para utilizar algoritmos de classificação binária para classificação multiclasse. O OvO divide um conjunto de dados de classificação multiclasse em problemas de classificação binária, mas ao contrário de *One-vs-Rest* (OvR), divide o conjunto de dados num conjunto de dados para cada classe versus cada uma as outras classes [60].

  Por exemplo, um problema de classificação multiclasse com quatro classes: 'vermelho', 'azul' e 'verde', 'amarelo'.

  - Problema de classificação binária 1: vermelho vs. Azul
  - Problema de classificação binária 2: vermelho vs. Verde
  - Problema de classificação binária 3: vermelho vs. Amarelo
  - Problema de classificação binária 4: azul vs. Verde
  - Problema de classificação binária 5: azul vs. Amarelo
  - Problema de classificação binária 6: verde vs. Amarelo

- *One-vs-Rest* (OvR), também conhecido como *One-vs-All* (OvA), realiza a divisão do conjunto de dados multiclasse em vários problemas de classificação binária,

divide o conjunto de dados num conjunto de dados para cada classe versus as restantes.

Por exemplo, dado um problema de classificação multiclasse com as seguintes classes ‘vermelho’, ‘azul’ e ‘verde’. Pode ser dividido em três conjuntos de dados de classificação binária [60]:

- Problema de classificação binária 1: vermelho vs [azul, verde]
- Problema de classificação binária 2: azul vs [vermelho, verde]
- Problema de classificação binária 3: verde vs [vermelho, azul]

Neste trabalho existem 7 classes, para as curvas AUC-ROC e foi utilizado o calculo OvR. De forma a exemplificar, o calculo do ROC da classe ’akiec’ é realizado classificando essa classe contra as restantes, e assim sucessivamente.

Na Figura 39 e 40 estão representadas as curvas para cada classe utilizando classificação OvR.

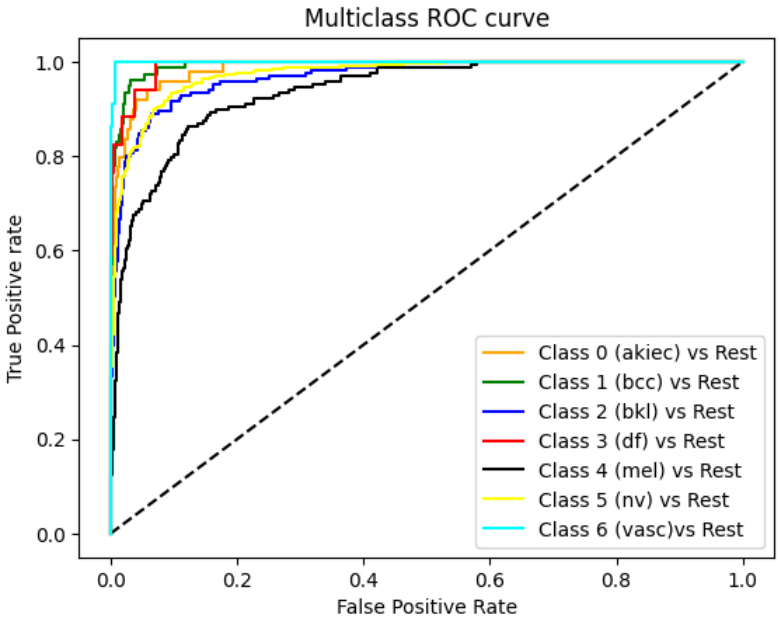

Fig. 39: AUC CURVE por Classe, com Regularizacão L2

Analisando a performance geral do modelo a partir da figuras 39 e 40, é possível concluir que todas as classes apresentam as curvas ROC acima da linha diagonal, que representa uma classificação aleatória, e a maioria das curvas consegue uma boa taxa de verdadeiros positivos e uma baixa taxa de falsos positivos.

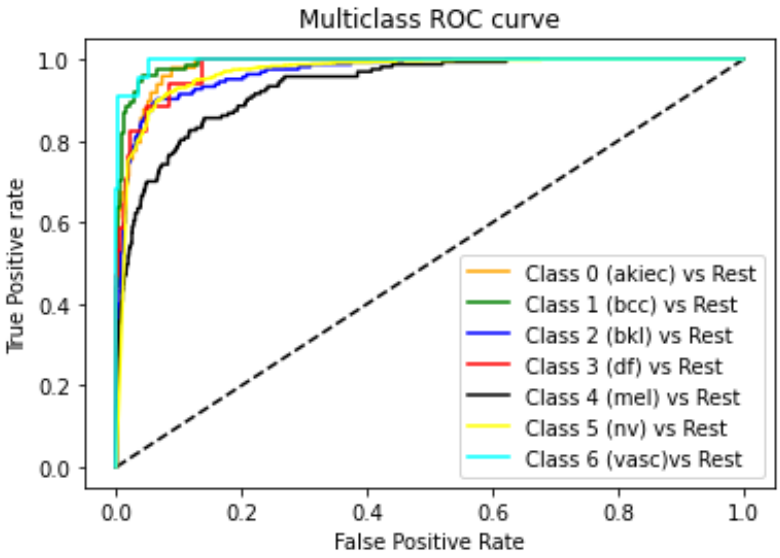

Fig. 40: AUC por Classe sem Regularizacão L2

Ao analisar por classe, é possível verificar que as classes 'bcc' e 'vasc' tem o melhor desempenho, apresentando curvas mais próximas do canto superior esquerdo. A maioria das classes conseguem uma alta taxa de verdadeiros positivos.

Os valores obtidos na métrica AUC para as diferentes classes do modelo ConvNeXtXLarge estão na tabela XVI, com um *Overall ROC AUC Score* de 0,9712.

Tabela XVI: Resultados da métrica AUC.

Modelo	**akiec**	**bcc**	**bkl**	**df**	**mel**	**nv**	**vasc**
ConvNeXtXLarge Sem L2	0.9838	0,9911	0,9691	0,9800	0,9351	0,9697	0,9957
ConvNeXtXLarge Com L2	0.9851	0,9927	0,9679	0,9918	0,9371	0,9735	0,9992

Ao analisar a métrica Overall ROC AUC Score para o modelo sem Regularização L2 apresenta 97,49% de AUC ROC, enquanto o modelo com Regularização L2 apresenta 97,82%, tendo um valor superior em quase todas as classes, exceto na "bkl".

6.1.11 Avaliação de Gráficos de *ACC* e *Loss*

Analisando os gráficos das figuras 41 e 42, onde estão representados os treinos e validações dos modelos com regularização e sem regularização L2, pode ser verificada que a função de perda (*Loss*) ao longo das épocas do treino com L2 diminui consideravelmente quando comparado com o treino sem L2, indicando que o treino sem L2 apresentava algum *overfitting*, com dificuldade de reconhecer as classes. Tendo

em conta este fator e a métrica AUC ROC, o modelo que tem melhor desempenho na classificação é o que tem a regularização L2.

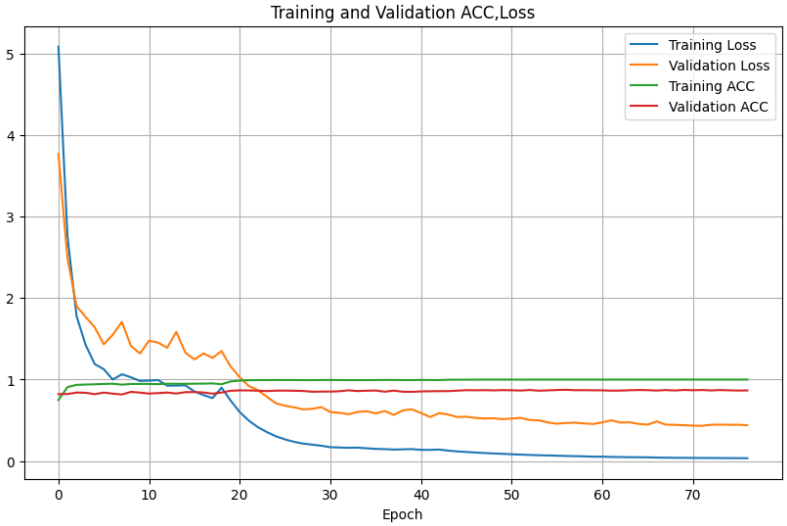

Fig. 41: Treino e Validação - ACC, Loss com Regularização L2

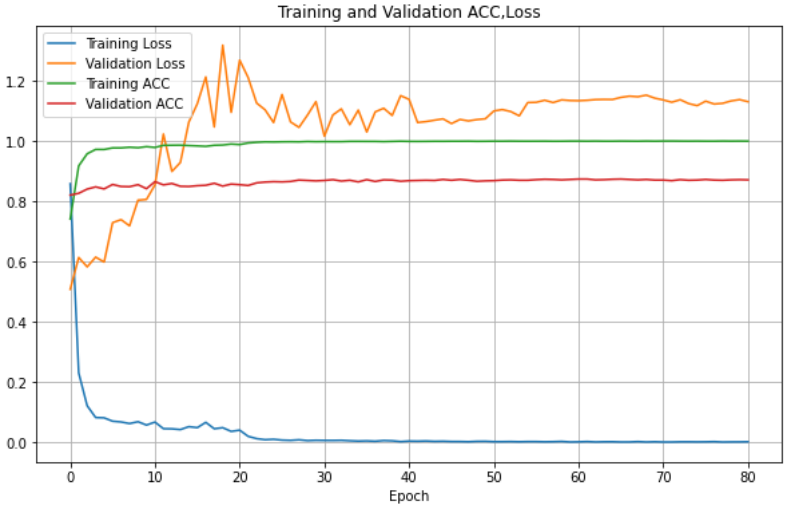

Fig. 42: Treino e Validação - ACC, Loss sem Regularização L2

Nos dois treinos a exatidão perto do 0.8 e aumenta rapidamente nas primeiras épocas, mantendo-se constante a partir da época 20.

Nas figuras 43 e 44 estão refletidos de forma isolada os dados de treino para as métricas *ACC* e *loss*.

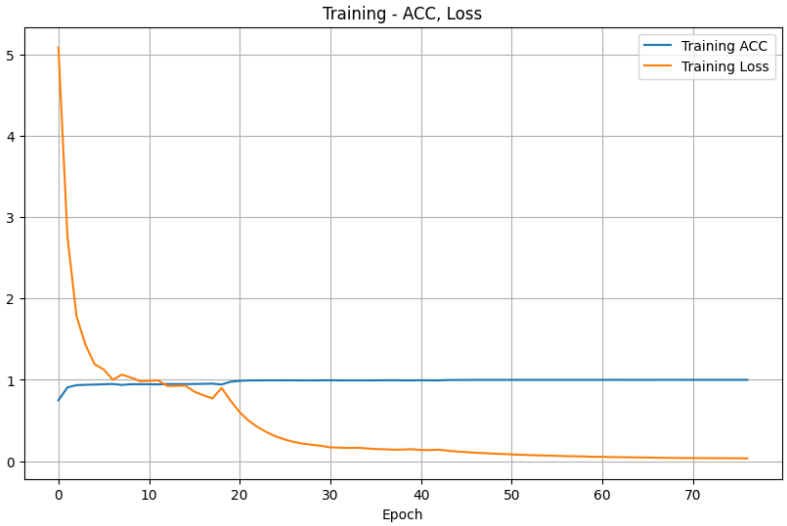

Fig. 43: Dados Treino com Regularização L2

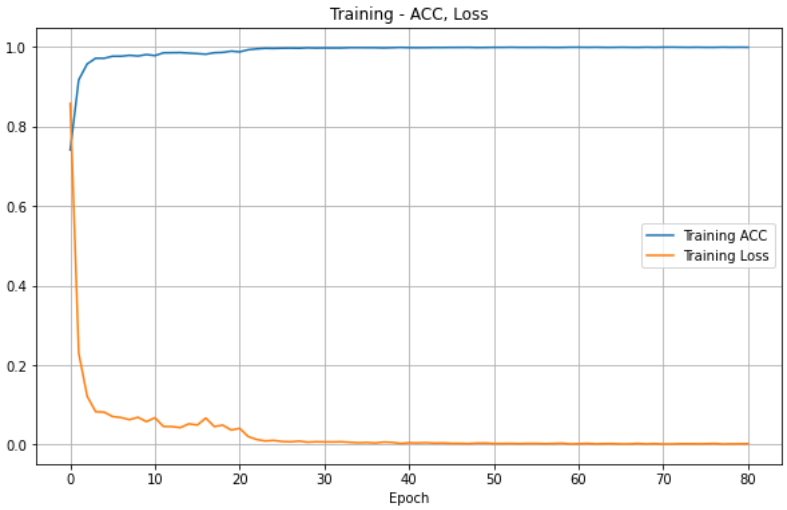

Fig. 44: Dados Treino sem Regularização L2

Nas figuras 45 e 46 estão refletidos os dados da exatidão para o treino e validação do modelo, sendo possível comparar o desempenho da métrica ACC. A linha vermelha representa o valor máximo da métrica *val_sparse_categorical_accuracy*, na época em que foi atingido. No caso do modelo com L2 o valor máximo foi encontrado mais rapidamente do que no modelo sem regularização. No caso do treino com L2 o valor máximo foi encontrado na época 56 e no modelo sem regularização L2 foi na época 60.

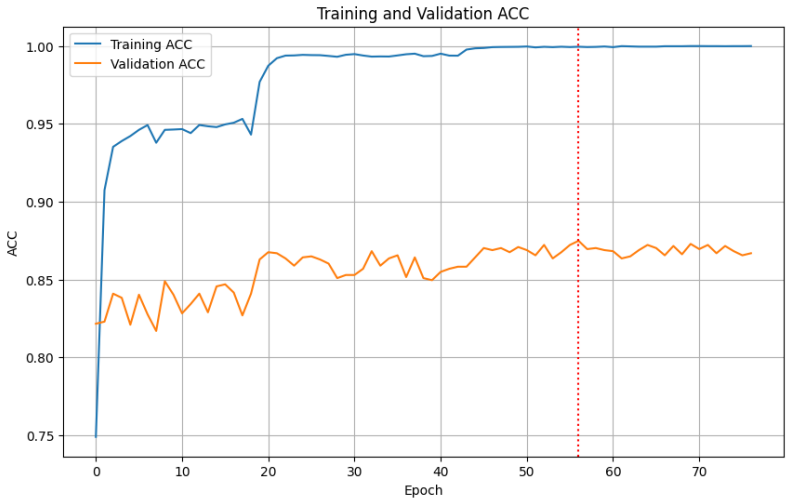

Fig. 45: Treino e Validação - ACC com Regularização L2

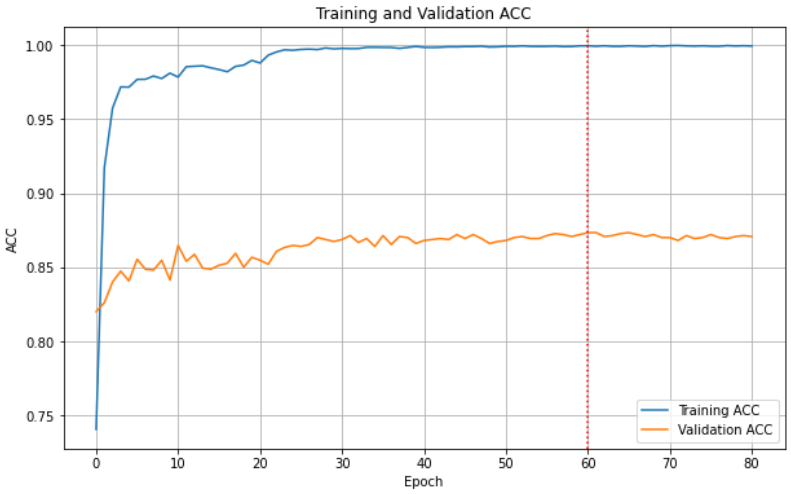

Fig. 46: Treino e Validação - ACC sem Regularização L2

Na figura 47 estão refletidos os dados de validação para as métricas *ACC* e *loss* do modelo com regularização L2. Na época 56 é registado o valor mais alto da

métrica "val_sparse_categorical_ accuracy" com 87,48%, finalizando o treino após as 20 épocas definidas no *EarlyStopping*.

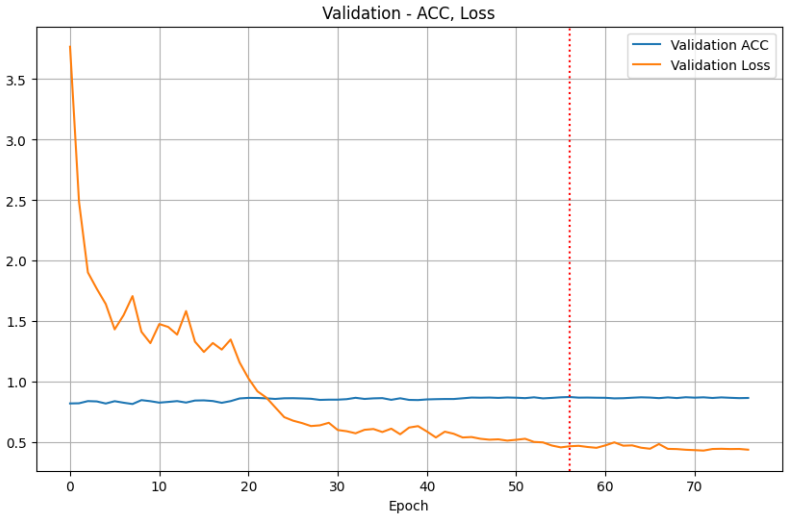

Fig. 47: Validação - ACC, Loss com Regularização L2

Na figura 48 estão refletidos os dados de validação para as métricas *ACC* e *loss* do modelo sem regularização L2, na época 60 é registado o valor mais alto da métrica "val_sparse_categorical_accuracy" com 87,35%.

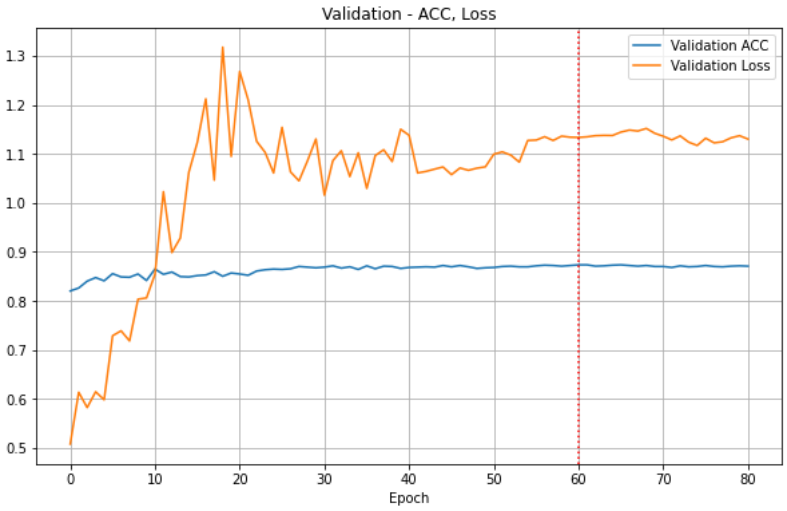

Fig. 48: Validação - ACC, Loss sem Regularização L2

6.2 Descrição de Hardware utilizado

O hardware utilizado para o treino da maioria dos modelos foi um *Personal Computer* (PC), com o sistema operativo Ubuntu 24.04.1 LTS instalado, com as caraterísticas indicadas na tabela XVII

Tabela XVII: Caraterísticas do PC de treino.

Componente	**Modelo/Tamanho**
Placa Gráfica	RTX 4070 ti 12GB GDDR6
Processador	AMD 5800X3D
Memoria RAM	64GB

Alguns modelos podem requerem maior quantidade de memoria para realizar o processamento, devido à dimensão das imagens utilizadas ou pela complexidade e tamanho da rede, como é abordado na secção seguinte.

6.2.1 Limitações de Hardware

Alguns modelos exigem maior capacidade em termos de memoria de GPU e memoria RAM, não sendo possível executá-los num PC. Neste trabalho optou-se pela utilização da plataforma *Google Colab* [61], que é um serviço que fornece acesso a recursos de computação, incluindo GPU (*Graphic Processor Unit*) e TPU (*Tensor Processor Unit*) [62].

Os modelos executados na plataforma *Colab* foram os seguintes:

- ConvNeXtLarge
- ConvNeXtXLarge
- EfficientNetB6
- EfficientNetB7
- EfficientNetV2_M
- EfficientNetV2_L
- NASNetLarge

As caraterísticas do ambiente Colab utilizado foram as seguintes:

Tabela XVIII: Caraterísticas do Ambiente Colab de treino.

Componente	**Modelo**
GPU	A100 de 40GB RAM
Disco	236Gb aprox.
Memoria RAM	83GB aprox.

6.3 Validação dos requisitos

Nesta secção, apresenta-se o estado de implementação de cada um dos requisitos definidos para a arquitetura.

A tabela XIX descreve os requisitos funcionais estabelecidos, bem como o respetivo estado de cumprimento no âmbito da implementação.

Tabela XIX: Resultados Requisitos Funcionais.

Requisito	**Tipo**	**Implementado**	**Observações**
RF01	A aplicação deve ser capaz de capturar imagens através da câmara do dispositivo	Sim	Testado com sucesso
RF02	Funcionalidade para carregar imagens da galeria do dispositivo	Sim	Testado com sucesso
RF03	A aplicação deve ser capaz de classificar as imagens de lesões de pele	Sim	Testado com sucesso
RF04	A aplicação deve permitir catalogar uma lesão para monitorização regula	Não	Trabalho Futuro

O requisito RF04 está definido como trabalho futuro, de forma a desenvolver uma plataforma com a capacidade de gestão de utilizadores e acompanhamento das lesões, com um módulo de gestão de contas. Facilitando o registo e autenticação dos utilizadores, implementando um sistema que permita o acompanhamento das lesões detetadas ao longo do tempo.

O estado de implementação dos requisitos não funcionais, definidos para a arquitetura, encontra-se na tabela XX.

Os requisitos RNF04 e RNF05 estão identificados como trabalho futuro, estes devem ser desenvolvidos e validar a aplicação a nível de usabilidade e de cumprimento do RGPD.

Tabela XX: Resultados Requisitos Não Funcionais.

Requisito	**Tipo**	**Realizado**	**Observações**
RNF01	A aplicação deve responder em um tempo máximo de 5 segundos	Sim	Tempo de resposta: 3-4Seg. Aprox.
RNF02	O modelo de classificação deve apresentar mais de 80% de exatidão	Sim	Modelo apresenta uma exatidão de 87,62%
RNF03	A aplicação deve garantir a segurança dos dados do utilizador (imagens e resultados de classificação)	Sim	Encriptação AES implementada
RNF04	A aplicação deve cumprir com Regulamento Geral sobre a Proteção de Dados	Parcialmente	Não são guardados nenhum tipo de dados dos utilizadores
RNF05	A aplicação deve ser fácil de usar, com uma interface intuitiva e instruções claras de captura ou carregamento imagens para classificação	Parcialmente	Desenvolvido como prova de conceito com uma interface simples. Aplicação não validada. Trabalho futuro.
RNF06	A aplicação deve ser compatível com vários sistemas operativos, como Android e iOS	Parcialmente	Desenvolvido em Flutter para Android, permitindo o desenvolvimento para outras plataformas.
RNF07	A aplicação deve ser fácil de modificar e atualizar para melhorar funcionalidades e corrigir bugs	Parcialmente	Desenvolvido como prova de conceito. A atualização do modelo podem ser realizada facilmente ao substituir no servidor o modelo atualizado. A correção de bugs e atualizações são geridas facilmente por ser de código aberto com repositório github: https://github.com/Jonathan-Vieira25/skin-lession-app.git.

O requisito RNF06 está parcialmente desenvolvido, já que encontra-se implementado para a plataforma Android, utilizando o framework Flutter, sendo possível estender a implementação para as plataformas iOS e Web.

O requisito RNF07, esta indicado como parcialmente desenvolvido por estar em código aberto no repositório https://github.com/Jonathan-Vieira25/skin-lession-app.git, permitindo atualizações e correções de bug. A atualização dos modelos pode ser realizada de forma simples, substituindo o modelo atualizado no servidor.

6.4 Pontos-chave

Os modelos Keras são modelos de aprendizagem profunda disponibilizados juntamente com pesos pré-treinados, estes foram utilizados para treinar a classificação das 7 lesões de pele do conjunto de dados HAM10000.

Este conjunto apresenta um desbalanceamento nas classes e para corrigir este problema é necessário utilizar técnicas de balanceamento, nomeadamente *Data Augmentation*. Esta técnica aumenta o numero de amostras utilizados no treino dos modelos, mas ao apresentar ao modelo exemplos das classes minoritárias no processo de treino pode dar origem a *overffiting*, e para minimiza-lo é necessário aplicar outras técnicas como *Dropout* e Regularização L2.

Dos modelos treinados, os mais recentes apresentaram métricas com resultados superiores. A métrica *Overall ACC* apresentou 87,62% de exatidão com regularização e 88,02% sem regularização L2, mas não foi a única métrica a ser avaliada.

Considerando a existência de desbalanceamento nos dados, foram comparadas as métricas ACC Macro, F1 Macro e Overall MCC. O modelo ConvNeXtXLarge continuou a apresentar o melhor desempenho. Para ACC Macro o valor obtido foi de 96% aproximadamente para ambos modelos (com e sem regularização), o F1 Macro foi 76,15% com L2 e 77,03% sem L2, e o Overall MCC foi 75.81% com L2 e 76,37% sem L2. Ao analisar o curva AUC-ROC do modelo ConvNeXtXLarge com e sem regularização, é possível verificar que o ConvNeXtXLarge com L2 apresenta para todas as classes um melhor desempenho, acima da curva diagonal, com um valor de *Overall ROC AUC Score* de 97,82%.

O treino do modelo com L2 consegue o seu valor máximo da métrica *val_sparse_categorical_accuracy* de 87,48% na epoca 56, sem conseguir melhorar o resultado e finalizando o treino na época 76, enquanto o treino sem L2 consegue o seu valor máximo da métrica de 87,35% na época 60, sem conseguir melhorar o resultado e finalizando o treino na época 80.

Os modelos com maior complexidade e dimensões maiores de imagem implicam um maior consumo de recursos, pelo que o treino nos GPUs de alto desempenho são necessários, pois disponibilizam mais RAM para os GPU e para o processamento em geral.

7 Conclusão

Nos anos mais recentes, tem-se observado um aumento substancial nos cancros de pele e disponibilizar ferramentas abertas que auxiliem na deteção de lesões facilita a deteção do cancro na fase inicial, democratizando o acesso uma ferramenta de triagem tornando o processo acessível à população em geral.

Neste trabalho, foi realizado o treino de diversos modelos de CNNs, utilizando *Transfer Learning*, para identificar o modelo com melhor desempenho na classificação de lesões de pele.

O modelo escolhido é uma peça central na arquitetura Cliente-Servidor proposta como solução para uma aplicação de deteção de lesões de pele, fornecendo uma ferramenta de triagem acessível à população em geral.

Utilizando a biblioteca PyCM de python, foram obtidas as métricas para os diferentes modelos treinados. O modelo que apresentou os melhores resultados em Exatidão macro (ACC Macro), exatidão geral (Overall ACC), F1 macro e o coeficiente de correlação de Matthews (MCC) foi o ConvNeXtXLarge, sendo estes valores os melhores em todas as métricas principais. No desempenho geral dos modelos, o modelo ConvNeXtXLarge é a *CNN* que apresenta melhor desempenho.

Em particular, o melhor modelo produzido alcança uma exatidão superior a 80% na classificação das lesões, que é um dos objetivos principais deste trabalho, e ao ser executado no servidor, mantém o desempenho obtido de 87,62% com Regularização L2. Dessa forma, a solução permite a identificação eficaz das lesões, contribuindo para a saúde pública.

Além disso, essa implementação possibilita escalabilidade e atualizações do modelo de forma transparente para o utilizador, sem comprometer o desempenho nem impor grandes exigências aos dispositivos móveis.

O uso do *Flutter* facilitou a implementação, permitindo a escalabilidade para diferentes plataformas e proporcionando uma experiência amigável ao utilizador no acesso à classificação das lesões.

O desenvolvimento do projeto exigiu tempo e uma cuidadosa planificação, uma vez que o treino dos modelos necessita de uma elevada capacidade de processamento, resultando em tempos de espera elevados. As imagens utilizadas no processamento são utilizadas de forma anónima e não são guardadas no servidor, garantindo a transmissão encriptada da mesma e garantindo que o utilizador não tem a sua informação em risco.

Foi realizada uma avaliação comparativa de 38 arquiteturas diferentes de redes neuronais convolucionais pertencentes a diferentes famílias de redes convolucionais sendo esta analise não identificada em outros trabalhos anteriores.

Este trabalho está limitado à utilização de redes neuronais convencionais, não sendo utilizado nenhum modelo de *Transformers*. Adicionalmente, este trabalho é realizado como prova de conceito de uma arquitetura cliente-servidor que não foi validada por utilizadores, sendo isto estabelecido como trabalhos futuros da solução apresentada.

A arquitetura definida teve como objetivo alcançar um tempo de resposta inferior a cinco segundos, meta que foi atingida através da utilização da GPU do servidor na tarefa de classificação do modelo, permitindo obter tempos de resposta entre três a quatro segundos. A solução implementada registou um desempenho de exatidão de 87,6%, o que permite concluir que os objetivos da investigação foram alcançados, culminando na implementação de uma solução acessível à população e oferecendo uma contribuição relevante para a área científica da saúde.

7.1 Trabalhos Futuros

Alguns trabalhos futuros que podem ser incorporados ou adaptados na solução proposta e incorporar novos conjuntos de dados para melhorar o desempenho do modelo e melhorar a exatidão na classificação das lesões, sendo uma possível melhoria o desenvolvimento de um modelo com *Transformers*.

Uma possibilidade de melhorar este trabalho passa pela avaliação em tempo real, desenvolvendo o sistema de análise em tempo real que permita a deteção de lesões sem a necessidade de capturar imagens e envia-las ao servidor, tornando o processo mais ágil e acessível.

Também seria relevante produzir um modelo específico para distinguir entre lesões benignas e malignas, proporcionando um diagnóstico preliminar mais direcionado.

Por fim, seria importante incluir na plataforma a capacidade de gestão de utilizadores e acompanhamento das lesões, desenvolvendo um módulo de gestão de contas, facilitando o registo e autenticação dos utilizadores e implementando um sistema que permita o acompanhamento das lesões detetadas ao longo do tempo, oferecendo um histórico acessível para o utilizador e possibilitando um melhor acompanhamento.

8 Anexos

8.1 Comparação do Tamanho dos Modelos com Regularização L2

Tabela XXI: Tamanho dos Modelos ConvNeXt.

Parâmetros	Tiny	Small	Base	Large	XLarge
Não Treináveis	106.13 MB	188.66 MB	334.05 MB	748.57 MB	1.30 GB
Do Otimizador	73.53 MB	216.03 MB	288.03 MB	432.04 MB	576.05 MB
Treináveis	36.76 MB	108.01 MB	144.02 MB	216.02 MB	288.02 MB
Totais	216.42 MB	512.70 MB	766.09 MB	1.36 GB	2.14 GB

Tabela XXII: Tamanho dos Modelos EfficientNetB0-B3.

Parâmetros	EfficientNetB0	EfficientNetB1	EfficientNetB2	EfficientNetB3
Não Treinaveis	15.46 MB	25.09 MB	29.65 MB	41.15 MB
Do Otimizador	122.54 MB	160.04 MB	222.79 MB	300.04 MB
Treináveis	61.27 MB	80.02 MB	111.39 MB	150.02 MB
Totais	199.26 MB	265.15 MB	363.83 MB	491.21 MB

Tabela XXIII: Tamanho dos Modelos EfficientNetB4-B7.

Parâmetros	EfficientNetB4	EfficientNetB5	EfficientNetB6	EfficientNetB7
Não Treinaveis	67.43 MB	108.79 MB	156.27 MB	244.53 MB
Do Otimizador	504.04 MB	900.05 MB	1.27 GB	1.76 GB
Treináveis	252.02 MB	450.02 MB	650.28 MB	902.53 MB
Totais	823.50 MB	1.42 GB	2.06 GB	2.88 GB

Tabela XXIV: Tamanho dos Modelos EfficientNetV2.

Parâmetros	V2_B0	V2_B1	V2_B2	V2_B3	V2_S	V2_M	V2_L
Não Treinaveis	22.59 MB	26.45 MB	33.46 MB	49.34 MB	77.57 MB	202.76 MB	449.18 MB
Do Otimizador	122.54 MB	160.04 MB	222.79 MB	300.04 MB	250.04 MB	360.04 MB	360.04 MB
Treináveis	61.27 MB	80.02 MB	111.39 MB	150.02 MB	125.02 MB	180.02 MB	180.02 MB
Totais	206.39 MB	266.50 MB	367.64 MB	499.40 MB	452.62 MB	742.82 MB	989.23 MB

Tabela XXV: Tamanho dos Modelos RESNET.

Parâmetros	ResNet101	ResNet101v2	ResNet152	ResNet152V2	ResNet50	ResNet52
Não Treinaveis	162.74 MB	162.62 MB	222.68 MB	222.53 MB	90.00 MB	89.91 MB
Do Otimizador	196.05 MB	196.05 MB	196.05 MB	196.05 MB	196.05 MB	196.05 MB
Treináveis	98.02 MB	98.02 MB	98.02 MB	98.02 MB	98.02 MB	98.02 MB
Totais	456.81 MB	456.69 MB	516.75 MB	516.60 MB	384.07 MB	383.98 MB

Tabela XXVI: Tamanho dos Modelos DenseNet, MobileNet e Xception.

Parâmetros	**DenseNet121**	**DenseNet169**	**DenseNet201**	**MobileNet**	**MobileNetv2**	**Xception**
Não Treinaveis	26.85 MB	48.24 MB	69.91 MB	12.32 MB	8.62 MB	79.60 MB
Do Otimizador	98.03 MB	159.29 MB	183.79 MB	98.03 MB	122.54 MB	400.05 MB
Treináveis	49.02 MB	79.65 MB	91.90 MB	49.02 MB	61.27 MB	200.02 MB
Totais	173.90 MB	287.18 MB	345.60 MB	159.37 MB	192.43 MB	679.67 MB

Tabela XXVII: Tamanho dos Modelos Inception, NasNet e VGG.

Parâmetros	**Inception ResNetV2**	**InceptionV3**	**NASNetMobile**	**NasNetLarge**	**VGG16**	**VGG19**
Não Treinaveis	207.29 MB	83.19 MB	16.30 MB	323.96 MB	56.14 MB	76.39 MB
Do Otimizador	75.04 MB	100.05 MB	101.09 MB	952.95 MB	49.02 MB	49.02 MB
Treináveis	37.52 MB	50.02 MB	50.55 MB	476.48 MB	24.51 MB	24.51 MB
Totais	319.85 MB	233.26 MB	167.94 MB	1.71 GB	129.67 MB	149.93 MB

8.2 Tabela de Camadas e Parâmetros para cada modelo

Tabela XXVIII: Camadas das redes VGG16 e VGG19.

Camada	**VGG16**		**VGG19**	
image rgbb in	Saída: (224, 224, 3)		Saída: (224, 224, 3)	
gaussian noise	Saída: (224, 224, 3)		Saída: (224, 224, 3)	
transfer learning	Saída: (7,7,512)	Param: 14.714.688	Saída: (7,7,512)	Param: 20.024.384
batch normalization	Saída: (7,7,512)	Param: 2.048	Saída: (7,7,512)	Param: 2.048
flatten	Saída: 25088		Saída: 25088	
dropout	Saída: 25088		Saída: 25088	
dense	Saída: 256	Param: 6.422.784	Saída: 256	Param:6.422.784
dropout	Saída: 256		Saída: 256	
dense	Saída: 7	Param: 1.799	Saída: 7	Param: 1.799

Tabela XXIX: Camadas das redes MobileNet e MobileNetV2.

Camada	**MobileNet**		**MobileNetV2**	
image rgbb in	Saída: (224, 224, 3)		Saída: (224, 224, 3)	
gaussian noise	Saída: (224, 224, 3)		Saída: (224, 224, 3)	
transfer learning	Saída: (7,7,1024)	Param: 3.228.864	Saída: (7,7,1280)	Param: 2.257.984
batch normalization	Saída: (7,7,1024)	Param: 4.096	Saída: (7,7,1280)	Param: 5.120
flatten	Saída: 50176		Saída: 50176	
dropout	Saída: 50176		Saída: 50176	
dense	Saída: 256	Param:12.845.312	Saída: 256	Param:16.056.576
dropout	Saída: 256		Saída: 256	
dense	Saída: 7	Param: 1.799	Saída: 7	Param: 1.799

Tabela XXX: Camadas das redes NASNetMobile e NASNetLarge.

Camada	**NASNetMobile**		**NASNetLarge**	
image rgbb in	Saída: (224, 224, 3)		Saída: (331, 331, 3)	
gaussian noise	Saída: (224, 224, 3)		Saída: (331, 331, 3)	
transfer learning	Saída: (7,7,1056)	Param: 4.269.716	Saída: (11, 11, 4032)	Param: 84.916.818
batch normalization	Saída: (7,7,1056)	Param: 4.224	Saída: (11, 11, 4032)	Param: 16.128
flatten	Saída: 51744		Saída: 487872	
dropout	Saída: 51744		Saída: 487872	
dense	Saída: 256	Param: 13.246.720	Saída: 256	Param:124.895.488
dropout	Saída: 256		Saída: 256	
dense	Saída: 7	Param: 1.799	Saída: 7	Param: 1.799

Tabela XXXI: Camadas das redes InceptionV3 e InceptionResNetV2.

Camada	**InceptionV3**		**InceptionResNetV2**	
image rgbb in	Saída: (224, 224, 3)		Saída: (224, 224, 3)	
gaussian noise	Saída: (224, 224, 3)		Saída: (224, 224, 3)	
transfer learning	Saída: (5, 5, 2048)	Param: 21.802.784	Saída: (5, 5, 1536)	Param: 54.336.736
batch normalization	Saída: (5, 5, 2048)	Param: 8.192	Saída: (5, 5, 1536)	Param: 6.144
flatten	Saída: 51200		Saída: 38400	
dropout	Saída: 51200		Saída: 38400	
dense	Saída: 256	Param: 13.107.456	Saída: 256	Param:9.830.656
dropout	Saída: 256		Saída: 256	
dense	Saída: 7	Param: 1.799	Saída: 7	Param: 1.799

Tabela XXXII: Camadas das redes DenseNet121 e DenseNet169.

Camada	**DenseNet121**		**DenseNet169**	
image rgbb in	Saída: (224, 224, 3)		Saída: (224, 224, 3)	
gaussian noise	Saída: (224, 224, 3)		Saída: (224, 224, 3)	
transfer learning	Saída: (7, 7, 1024)	Param: 7.037.504	Saída: (7, 7, 1664)	Param: 12.642.880
batch normalization	Saída: (7, 7, 1024)	Param: 4.096	Saída: (7, 7, 1664)	Param: 6.656
flatten	Saída: 50176		Saída: 50176	
dropout	Saída: 50176		Saída: 50176	
dense	Saída: 256	Param: 12.845.312	Saída: 256	Param: 20.873.472
dropout	Saída: 256		Saída: 256	
dense	Saída: 7	Param: 1.799	Saída: 7	Param: 1.799

Tabela XXXIII: Camadas da rede DenseNet201.

Camada	**DenseNet201**	
image rgbb in	Saída: (224, 224, 3)	
gaussian noise	Saída: (224, 224, 3)	
transfer learning	Saída: (7, 7, 1920)	Param: 18.321.984
batch normalization	Saída: (7, 7, 1920)	Param: 7.680
flatten	Saída: 94080	
dropout	Saída: 94080	
dense	Saída: 256	Param: 24.084.736
dropout	Saída: 256	
dense	Saída: 7	Param: 1.799

Tabela XXXIV: Camadas das redes RESNET50 e RESNET50V2.

Camada	**RESNET50**		**RESNET50V2 (RESNET52)**	
image rgbb in	Saída: (224, 224, 3)		Saída: (224, 224, 3)	
gaussian noise	Saída: (224, 224, 3)		Saída: (224, 224, 3)	
transfer learning	Saída: (7, 7, 2048)	Param: 23.587.712	Saída: (7, 7, 2048)	Param: 23.564.800
batch normalization	Saída: (7, 7, 2048)	Param: 8.192	Saída: (7, 7, 2048)	Param: 8,192
flatten	Saída: 100352		Saída: 100352	
dropout	Saída: 100352		Saída: 100352	
dense	Saída: 256	Param: 25.690.368	Saída: 256	Param: 25.690.368
dropout	Saída: 256		Saída: 256	
dense	Saída: 7	Param: 1.799	Saída: 7	Param: 1.799

Tabela XXXV: Camadas das redes RESNET101 e RESNET101V2.

Camada	**RESNET101**		**RESNET101V2**	
image rgbb in	Saída: (224, 224, 3)		Saída: (224, 224, 3)	
gaussian noise	Saída: (224, 224, 3)		Saída: (224, 224, 3)	
transfer learning	Saída: (7, 7, 2048)	Param: 42.658.176	Saída: (7, 7, 2048)	Param: 42.626.560
batch normalization	Saída: (7, 7, 2048)	Param: 8.192	Saída: (7, 7, 2048)	Param: 8.192
flatten	Saída: 100352		Saída: 100352	
dropout	Saída: 100352		Saída: 100352	
dense	Saída: 256	Param: 25.690.368	Saída: 256	Param: 25.690.368
dropout	Saída: 256		Saída: 256	
dense	Saída: 7	Param: 1.799	Saída: 7	Param: 1.799

Tabela XXXVI: Camadas das redes RESNET152 e RESNET152V2.

Camada	**RESNET152**		**RESNET152V2**	
image rgbb in	Saída: (224, 224, 3)		Saída: (224, 224, 3)	
gaussian noise	Saída: (224, 224, 3)		Saída: (224, 224, 3)	
transfer learning	Saída: (7, 7, 2048)	Param: 58.370.944	Saída: (7, 7, 2048)	Param: 58.331.648
batch normalization	Saída: (7, 7, 2048)	Param: 8.192	Saída: (7, 7, 2048)	Param: 8.192
flatten	Saída: 100352		Saída: 100352	
dropout	Saída: 100352		Saída: 100352	
dense	Saída: 256	Param: 25.690.368	Saída: 256	Param: 25.690.368
dropout	Saída: 256		Saída: 256	
dense	Saída: 7	Param: 1.799	Saída: 7	Param: 1.799

Tabela XXXVII: Camadas das redes ConvNeXtTiny e ConvNeXtSmall.

Camada	**ConvNeXtTiny**		**ConvNeXtSmall**	
image rgbb in	Saída: (224, 224, 3)		Saída: (384, 384, 3)	
gaussian noise	Saída: (224, 224, 3)		Saída: (384, 384, 3)	
transfer learning	Saída: (7, 7, 768)	Param: 27.820.128	Saída: (12, 12, 768)	Param: 49.454.688
batch normalization	Saída: (7, 7, 768)	Param: 3.072	Saída: (12, 12, 768)	Param: 3.072
flatten	Saída: 37632		Saída: 110592	
dropout	Saída: 37632		Saída: 110592	
dense	Saída: 256	Param: 9.634.048	Saída: 256	Param: 28.311.808
dropout	Saída: 256		Saída: 256	
dense	Saída: 7	Param: 1.799	Saída: 7	Param: 1.799

Tabela XXXVIII: Camadas das redes ConvNeXtBase e ConvNeXtLarge.

Camada	**ConvNeXtBase**		**ConvNeXtLarge**	
image rgbb in	Saída: (384, 384, 3)		Saída: (384, 384, 3)	
gaussian noise	Saída: (384, 384, 3)		Saída: (384, 384, 3)	
transfer learning	Saída: (12, 12, 1024)	Param: 87.566.464	Saída: (12, 12, 1536)	Param: 196.230.336
batch normalization	Saída: (12, 12, 1024)	Param: 4.096	Saída: (12, 12, 1536)	Param: 6.144
flatten	Saída: 147456		Saída: 221184	
dropout	Saída: 147456		Saída: 221184	
dense	Saída: 256	Param: 37.748.992	Saída: 256	Param: 56.623.360
dropout	Saída: 256		Saída: 256	
dense	Saída: 7	Param: 1.799	Saída: 7	Param: 1.799

Tabela XXXIX: Camadas das redes EfficientNetB0 e EfficientNetB1.

Camada	**EfficientNetB0**		**EfficientNetB1**	
image rgbb in	Saída: (224, 224, 3)		Saída: (240, 240, 3)	
gaussian noise	Saída: (224, 224, 3)		Saída: (240, 240, 3)	
transfer learning	Saída: (7, 7, 1280)	Param: 4.049.571	Saída: (8, 8, 1280)	Param: 6.575.239
batch normalization	Saída: (7, 7, 1280)	Param: 5.120	Saída: (8, 8, 1280)	Param: 5.120
flatten	Saída: 62720		Saída: 81920	
dropout	Saída: 62720		Saída: 81920	
dense	Saída: 256	Param: 16.056.576	Saída: 256	Param: 20.971.776
dropout	Saída: 256		Saída: 256	
dense	Saída: 7	Param: 1.799	Saída: 7	Param: 1.799

Tabela XL: Camadas das redes EfficientNetB2 e EfficientNetB3.

Camada	**EfficientNetB2**		**EfficientNetB3**	
image rgbb in	Saída: (260, 260, 3)		Saída: (300, 300, 3)	
gaussian noise	Saída: (260, 260, 3)		Saída: (300, 300, 3)	
transfer learning	Saída: (9, 9, 1408)	Param: 7.768.569	Saída: (10, 10, 1536)	Param: 10.783.535
batch normalization	Saída: (9, 9, 1408)	Param: 5.632	Saída: (10, 10, 1536)	Param: 6.144
flatten	Saída: 114048		Saída: 153600	
dropout	Saída: 114048		Saída: 153600	
dense	Saída: 256	Param: 29.196.544	Saída: 256	Param: 39.321.856
dropout	Saída: 256		Saída: 256	
dense	Saída: 7	Param: 1.799	Saída: 7	Param: 1.799

Tabela XLI: Camadas das redes EfficientNetB4 e EfficientNetB5.

Camada	EfficientNetB4		EfficientNetB5	
image rgbb in	Saída: (380, 380, 3)		Saída: (456, 456, 3)	
gaussian noise	Saída: (380, 380, 3)		Saída: (456, 456, 3)	
transfer learning	Saída: (12, 12, 1792)	Param: 17.673.823	Saída: (15, 15, 2048)	Param: 28.513.527
batch normalization	Saída: (12, 12, 1792)	Param: 7.168	Saída: (15, 15, 2048)	Param: 8.192
flatten	Saída: 258048		Saída: 460800	
dropout	Saída: 258048		Saída: 460800	
dense	Saída: 256	Param: 66.060.544	Saída: 256	Param: 117.965.056
dropout	Saída: 256		Saída: 256	
dense	Saída: 7	Param: 1.799	Saída: 7	Param: 1.799

Tabela XLII: Camadas das redes EfficientNetB6 e EfficientNetB7.

Camada	EfficientNetB6		EfficientNetB7	
image rgbb in	Saída: (528, 528, 3)		Saída: (600, 600, 3)	
gaussian noise	Saída: (528, 528, 3)		Saída: (600, 600, 3)	
transfer learning	Saída: (17, 17, 2304)	Param: 40.960.143	Saída: (19, 19, 2560)	Param: 64.097.687
batch normalization	Saída: (17, 17, 2304)	Param: 9.216	Saída: (19, 19, 2560)	Param: 10.240
flatten	Saída: 665856		Saída: 924160	
dropout	Saída: 665856		Saída: 924160	
dense	Saída: 256	Param: 170.459.392	Saída: 256	Param: 236.585.216
dropout	Saída: 256		Saída: 256	
dense	Saída: 7	Param: 1.799	Saída: 7	Param: 1.799

Tabela XLIII: Camadas das redes EfficientNetV2-B0 e EfficientNetV2-B1.

Camada	EfficientNetV2-B0		EfficientNetV2-B1	
image rgbb in	Saída: (224, 224, 3)		Saída: (240, 240, 3)	
gaussian noise	Saída: (224, 224, 3)		Saída: (240, 240, 3)	
transfer learning	Saída: (7, 7, 1280)	Param: 5.919.312	Saída: (8, 8, 1280)	Param: 6.931.124
batch normalization	Saída: (7, 7, 1280)	Param: 5.120	Saída: (8, 8, 1280)	Param: 5.120
flatten	Saída: 62720		Saída: 81920	
dropout	Saída: 62720		Saída: 81920	
dense	Saída: 256	Param: 16.056.576	Saída: 256	Param: 20.971.776
dropout	Saída: 256		Saída: 256	
dense	Saída: 7	Param: 1.799	Saída: 7	Param: 1.799

Tabela XLIV: Camadas das redes EfficientNetV2-B2 e EfficientNetV2-B3.

Camada	EfficientNetV2-B2		EfficientNetV2-B3	
image rgbb in	Saída: (260, 260, 3)		Saída: (300, 300, 3)	
gaussian noise	Saída: (260, 260, 3)		Saída: (300, 300, 3)	
transfer learning	Saída: (9, 9, 1408)	Param: 8.769.374	Saída: (10, 10, 1536)	Param: 12.930.622
batch normalization	Saída: (9, 9, 1408)	Param: 5.632	Saída: (10, 10, 1536)	Param: 6.144
flatten	Saída: 114048		Saída: 153600	
dropout	Saída: 114048		Saída: 153600	
dense	Saída: 256	Param: 29.196.544	Saída: 256	Param: 39.321.856
dropout	Saída: 256		Saída: 256	
dense	Saída: 7	Param: 1.799	Saída: 7	Param: 1.799

Tabela XLV: Camadas das redes EfficientNetV2-S e EfficientNetV2-M.

Camada	EfficientNetV2-S		EfficientNetV2-M	
image rgbb in	Saída: (300, 300, 3)		Saída: (380, 380, 3)	
gaussian noise	Saída: (300, 300, 3)		Saída: (380, 380, 3)	
transfer learning	Saída: (10, 10, 1280)	Param: 20.331.360	Saída: (12, 12, 1280)	Param: 53.150.388
batch normalization	Saída: (10, 10, 1280)	Param: 5.120	Saída: (12, 12, 1280)	Param: 5.120
flatten	Saída: 128000		Saída: 184320	
dropout	Saída: 128000		Saída: 184320	
dense	Saída: 256	Param: 32.768.256	Saída: 256	Param: 47.186.176
dropout	Saída: 256		Saída: 256	
dense	Saída: 7	Param: 1.799	Saída: 7	Param: 1.799

Tabela XLVI: Camadas das redes EfficientNetV2-L e Xception.

Camada	EfficientNetV2-L		Xception	
image rgbb in	Saída: (380, 380, 3)		Saida: (299, 299, 3)	
gaussian noise	Saída: (380, 380, 3)		Saida: (299, 299, 3)	
transfer learning	Saída: (12, 12, 1280)	Param: 117.746.848	Saida:(10, 10, 2048)	Param: 20.861.480
batch normalization	Saída: (12, 12, 1280)	Param: 5.120	Saida:(10, 10, 2048)	Param: 8.192
flatten	Saída: 184320		Saida: 204800	
dropout	Saída: 184320		Saida: 204800	
dense	Saída: 256	Param: 47.186.176	Saida:256	Param: 52.429.056
dropout	Saída: 256		Saida:256	
dense	Saída: 7	Param: 1.799	Saida:7	Param: 1.799

8.3 Código

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Feb  2 18:12:48 2025

@author: jonathan
"""

import matplotlib.pyplot as plt
from keras import models, layers
from keras.optimizers import Adam
from pathlib import Path
import numpy as np
import pandas as pd
#from skimage.util import montage
import tensorflow as tf
from keras.callbacks import CSVLogger
from keras.preprocessing.image import ImageDataGenerator
from PIL import Image
from sklearn.model_selection import train_test_split
import shutil

#for CALLBACK
csv_logger = CSVLogger("Keras_pd_data10000.csv", append=True)
```

```
tf.debugging.set_log_device_placement(False)

root=Path('.')
base_dir = Path('.') / 'input'
image_dir = base_dir / 'ham10000_images'
mapping_file = base_dir / 'HAM10000_metadata.csv'
#Dataset new Dir
newdir='/newdataset2025/'


dx_name_dict = {
    'akiec': 'Actinic keratoses and intraepithelial carcinoma',
    'bcc': 'basal cell carcinoma',
    'bkl': 'benign keratosis-like',
    'df': 'dermatofibroma',
    'mel': 'melanoma',
    'nv': 'melanocytic nevi',
    'vasc': 'vascular lesions'
}

#Find all images on path
all_image_ids = {c_path.stem: c_path for c_path in base_dir.glob('**/*.jpg'
    )}
#Mapping lesion to number
dx_name_id_dict = {name: id for id, name in enumerate(dx_name_dict.keys())}

#Read original file HAM10000
skin_df = pd.read_csv(mapping_file)


#Create column with path -> mapping
skin_df['image_path'] = skin_df['image_id'].map(all_image_ids.get)
#Drop age, not necessary
skin_df.drop(['age'], axis=1).describe()
#Create column dx_name from acronim
skin_df['dx_name'] = skin_df['dx'].map(dx_name_dict.get)
#create column with id lession / mapping
skin_df['dx_id'] = skin_df['dx'].map(dx_name_id_dict.get).astype(int)



#preparing list to train_test_split 0.7,0.3
selection = skin_df.groupby('image_id').apply(lambda x: x)
selection = selection[['image_id','dx']]
selection= selection.drop_duplicates()

raw_t,val_t = train_test_split(selection,
                 test_size = 0.3,
                  # hack to make stratification work
                 stratify = selection['dx'])
```

```
#divide the test by 0.5 to valid and test dataset
valid_temp,test_temp= train_test_split(val_t,
                 test_size = 0.5,
                  # hack to make stratification work
                 stratify = val_t['dx'])

#create dataframe for train, valid and test
raw_train_df = skin_df[skin_df['image_id'].isin(raw_t['image_id'])]
valid_df = skin_df[skin_df['image_id'].isin(valid_temp['image_id'])]
test_df = skin_df[skin_df['image_id'].isin(test_temp['image_id'])]


#save to pickle
#raw_train_df.to_pickle('train_df.pkl')
#valid_df.to_pickle('valid_df.pkl')
#test_df.to_pickle('test_df.pkl')



#create secondary dataframe to use on augmentation
df2=pd.DataFrame(skin_df)
df2.columns =['lesion_id', 'image_id', 'dx', 'dx_type', 'age', 'sex', '
    localization',
       'image_path', 'dx_name', 'dx_id']



#load pickles - last train
#raw_train_df=pd.read_pickle('train_df.pkl')
#valid_df=pd.read_pickle('valid_df.pkl')
#test_df=pd.read_pickle('test_df.pkl')



#preparing imagedatagenerator for each transformation
datagenhf = ImageDataGenerator(horizontal_flip=True)
datagenhf.horizontal_flip = True
datagenhf.flip_horizontal = True
datagenvf2 = ImageDataGenerator(vertical_flip=True,preprocessing_function=
    lambda x: np.flipud(x))
datagenhf.vertical_flip = True
datagenhf.flip_vertical = True
datagen3 = ImageDataGenerator(rotation_range=45)
datagen3.rotation_range=45
datagen4 = ImageDataGenerator(brightness_range=[0.7,1.2])
datagen4.brightness_range=[0.7,1.2]
datagen5 = ImageDataGenerator(zoom_range=[0.7,1.0])
datagen5.zoom_range=[0.7,1.0]
```

```
#var with the transf. to DO
do= 'br'
#var boolean to copy orig to new dir
copyOrig=False

i=1
for i in range(len(df2)):
  print(i)
  if df2['dx'][i]!='nv':

    img = Image.open(df2['image_path'][i])
    split='/none/'
    #verify if image is on train
    result = raw_train_df[raw_train_df['image_path'].apply(lambda x : x ==
   Path(df2['image_path'][i]))]
    if (result['lesion_id'].count()>0):
        split='train/'
    #verify if image is on valid
    result = valid_df[valid_df['image_path'].apply(lambda x : x == Path(df2
   ['image_path'][i]))]
    if (result['lesion_id'].count()>0):
        split='val/'
    #verify if image is on test
    result = test_df[test_df['image_path'].apply(lambda x : x == Path(df2['
   image_path'][i]))]
    if (result['lesion_id'].count()>0):
        split='test/'

    if (split!='/none/'):

        img_array = np.array(img)
        img_array = img_array.reshape((1,) + img_array.shape)
        if (do=='hf'):
            augmented_imageshf = datagenhf.flow(img_array,batch_size=1,
   shuffle=False,seed=42)[0]
            image_flippedhf = augmented_imageshf[0]
            image_flippedhf2 = Image.fromarray(image_flippedhf.astype('
   uint8'))

            x=str(df2['image_path'][i]).split('/')
            #copy to new directory

            newfilename= x[len(x)-1].split('.')[0] + '_hf.jpg'
            newpath=newdir+split+df2['dx'][i]+'/'+newfilename

            # Save the image
            image_flippedhf2.save('/home/jonathan/Documentos'+newpath)
        #Vertical Flip
        if (do=='vf'):
```

```
            augmented_imagesvf = datagenvf2.flow(img_array,batch_size=1,
shuffle=False,seed=42)[0]
            image_flippedvf = augmented_imagesvf[0]
            image_flippedvf2 = Image.fromarray(image_flippedvf.astype('
uint8'))

            x=str(df2['image_path'][i]).split('/')
            newfilename= x[len(x)-1].split('.')[0] + '_vf.jpg'
            newpath=newdir+split+df2['dx'][i]+'/'+newfilename

            # Save the image
            image_flippedvf2.save('/home/jonathan/Documentos'+newpath)

        #rotation
        if (do=='r'):
            augmented_imagesr = datagen3.flow(img_array,batch_size=1,
shuffle=False,seed=42)[0]


            image_flippedr = augmented_imagesr[0]
            image_flippedr2 = Image.fromarray(image_flippedr.astype('uint8'
))

            x=str(df2['image_path'][i]).split('/')
            newfilename= x[len(x)-1].split('.')[0] + '_45.jpg'
            newpath=newdir+split+df2['dx'][i]+'/'+newfilename

            # Save the image
            image_flippedr2.save('/home/jonathan/Documentos'+newpath)
        #BRIGHTNESS
        if (do=='br'):
            augmented_imagesbr = datagen4.flow(img_array,batch_size=1,
shuffle=False,seed=42)[0]

            image_flippedbr = augmented_imagesbr[0]
            image_flippedbr2 = Image.fromarray(image_flippedbr.astype('
uint8'))

            x=str(df2['image_path'][i]).split('/')
            newfilename= x[len(x)-1].split('.')[0] + '_br.jpg'
            newpath=newdir+split+df2['dx'][i]+'/'+newfilename

            # Save the image
            image_flippedbr2.save('/home/jonathan/Documentos'+newpath)

        #
        if (do=='z'):
            augmented_imagesz = datagen5.flow(img_array,batch_size=1,
shuffle=False,seed=42)[0]
            image_flippedz = augmented_imagesz[0]
```

```
            image_flippedz2 = Image.fromarray(image_flippedz.astype('uint8'
    ))

            x=str(df2['image_path'][i]).split('/')
            newfilename= x[len(x)-1].split('.')[0] + '_Z.jpg'
            newpath=newdir+split+df2['dx'][i]+'/'+newfilename

            # Save the image
            image_flippedz2.save('/home/jonathan/Documentos'+newpath)
        if (copyOrig==True):
            shutil.copy(df2['image_path'][i], '/home/jonathan/Documentos'+
    newdir+split+df2['dx'][i]+'/'+x[len(x)-1])



  #if =nv not augmentation
  else:
    x=str(df2['image_path'][i]).split('/')
    split='/none/'

    result = raw_train_df[raw_train_df['image_path'].apply(lambda x : x ==
    Path(df2['image_path'][i]))]
    if (result['lesion_id'].count()>0):
        split='train/'

    result = valid_df[valid_df['image_path'].apply(lambda x : x == Path(df2
    ['image_path'][i]))]
    if (result['lesion_id'].count()>0):
        split='val/'

    result = test_df[test_df['image_path'].apply(lambda x : x == Path(df2['
    image_path'][i]))]
    if (result['lesion_id'].count()>0):
        split='test/'

    if (split!='/none/') and (result.empty!=True):
        newpath=newdir+split+df2['dx'][i]+'/'+x[len(x)-1]
        shutil.copy(df2['image_path'][i], '/home/jonathan/Documentos'+
    newpath)
```

Código 9: Criação do conjunto de dados 'Aumentados'

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Dec 29 13:10:45 2024

@author: jonathan
"""
```

```
import datetime
import matplotlib.pyplot as plt
from keras import models, layers
from pathlib import Path
import numpy as np
import pandas as pd
import tensorflow as tf
from keras.callbacks import CSVLogger
from keras.optimizers import Adam
from keras.regularizers import l2

datestr= datetime.datetime.now().strftime('%Y%m%d')

tf.debugging.set_log_device_placement(False)

root=Path('.')


#--------------------------------

BASE_MODEL='RESNET50' # ['VGG16', 'RESNET52', 'InceptionV3', 'Xception', '
    DenseNet169', 'DenseNet121','DenseNet201']
MODELS=['ConvNeXtXLarge']
#MODELS=["MobileNet","MobileNetV2","VGG16","VGG19","RESNET50","RESNET52","
    RESNET101","RESNET101V2","RESNET152","RESNET152V2","InceptionV3","
    Xception",'DenseNet169', 'DenseNet121','DenseNet201',"InceptionResNetV2
    ",\
#                   "NASNetMobile","NASNetLarge","'EfficientNetB0'","
    EfficientNetB1","EfficientNetB2","EfficientNetB3","EfficientNetB4","
    EfficientNetB5","EfficientNetB6","EfficientNetB7",\
#                   "EfficientNetV2B0","EfficientNetV2B1","EfficientNetV2B2
    ","EfficientNetV2B3","EfficientNetV2_S","EfficientNetV2_M","
    EfficientNetV2_L",\
#                   "ConvNeXtTiny","ConvNeXtSmall","ConvNeXtBase","
    ConvNeXtLarge","ConvNeXtXLarge"]

RESOLUTION=224 #default
IMG_SIZE = (RESOLUTION, RESOLUTION) # [(224, 224), (384, 384), (512, 512),
    (640, 640)]
BATCH_SIZE = 64 # [1, 8, 16, 24]
GAUSSIAN_NOISE = 0.05
DROPOUT = 0.25
DENSE_COUNT = 256
LEARN_RATE = 0.0001
EPOCHS = 200
FLATTEN = True
CLASSES=['akiec', 'bcc', 'bkl', 'df', 'mel', 'nv', 'vasc']

TUNNING=True
```

```
if (TUNNING):
    AUTOTUNE = tf.data.AUTOTUNE


for BASE_MODEL in MODELS:
    with open("./dataset/"+BASE_MODEL+'_'+datestr+'_SkinLesion_stats.csv',
    'w') as file:

        if BASE_MODEL == 'MobileNetV2':
            from keras.applications.mobilenet_v2 import MobileNetV2 as
    Model
        elif BASE_MODEL=='MobileNet':
            from keras.applications.mobilenet import MobileNet as Model
        elif BASE_MODEL=='VGG16':
            from keras.applications.vgg16 import VGG16 as Model
        elif BASE_MODEL=='VGG19':
            from keras.applications.vgg19 import VGG19 as Model
        elif BASE_MODEL=='RESNET50':
            from keras.applications.resnet import ResNet50 as Model
        elif BASE_MODEL=='RESNET52':
            from keras.applications.resnet_v2 import ResNet50V2 as Model
        elif BASE_MODEL=='RESNET101':
            from keras.applications.resnet import ResNet101 as Model
        elif BASE_MODEL=='RESNET101V2':
            from keras.applications.resnet_v2 import ResNet101V2 as Model
        elif BASE_MODEL=='RESNET152':
            from keras.applications.resnet import ResNet152 as Model
        elif BASE_MODEL=='RESNET152V2':
            from keras.applications.resnet_v2 import ResNet152V2 as Model
        elif BASE_MODEL=='InceptionV3':
            from keras.applications.inception_v3 import InceptionV3 as
    Model
        elif BASE_MODEL=='Xception':
            from keras.applications.xception import Xception as Model
            RESOLUTION=299
        elif BASE_MODEL=='DenseNet169':
            RESOLUTION=224
            from keras.applications.densenet import DenseNet169 as Model
        elif BASE_MODEL=='DenseNet121':
            RESOLUTION=224
            from keras.applications.densenet import DenseNet121 as Model
        elif BASE_MODEL=='DenseNet201':
            RESOLUTION=224
            from keras.applications.densenet import DenseNet201 as Model
        elif BASE_MODEL=='InceptionResNetV2':
            from keras.applications.inception_resnet_v2 import
    InceptionResNetV2 as Model
        elif BASE_MODEL=='NASNetMobile':
            from keras.applications.nasnet import NASNetMobile as Model
```

```
    elif BASE_MODEL=='NASNetLarge':
        from keras.applications.nasnet import NASNetLarge as Model
        RESOLUTION=331
    elif BASE_MODEL=='EfficientNetB0':
        from keras.applications.efficientnet import EfficientNetB0 as
Model
    elif BASE_MODEL=='EfficientNetB1':
        from keras.applications.efficientnet import EfficientNetB1 as
Model
        RESOLUTION=240
    elif BASE_MODEL=='EfficientNetB2':
        from keras.applications.efficientnet import EfficientNetB2 as
Model
        RESOLUTION=260
    elif BASE_MODEL=='EfficientNetB3':
        from keras.applications.efficientnet import EfficientNetB3 as
Model
        RESOLUTION=300
    elif BASE_MODEL=='EfficientNetB4':
        from keras.applications.efficientnet import EfficientNetB4 as
Model
        RESOLUTION=380
    elif BASE_MODEL=='EfficientNetB5':
        from keras.applications.efficientnet import EfficientNetB5 as
Model
        RESOLUTION=456
    elif BASE_MODEL=='EfficientNetB6':
        from keras.applications.efficientnet import EfficientNetB6 as
Model
        RESOLUTION=528
    elif BASE_MODEL=='EfficientNetB7':
        from keras.applications.efficientnet import EfficientNetB7 as
Model
        RESOLUTION=600
    elif BASE_MODEL=='EfficientNetV2_B0':
        from keras.applications.efficientnet_v2 import EfficientNetV2B0
 as Model
    elif BASE_MODEL=='EfficientNetV2_B1':
        from keras.applications.efficientnet_v2 import EfficientNetV2B1
 as Model
        RESOLUTION=240
    elif BASE_MODEL=='EfficientNetV2_B2':
        from keras.applications.efficientnet_v2 import EfficientNetV2B2
 as Model
        RESOLUTION=260
    elif BASE_MODEL=='EfficientNetV2_B3':
        from keras.applications.efficientnet_v2 import EfficientNetV2B3
 as Model
        RESOLUTION=300
    elif BASE_MODEL=='EfficientNetV2_S':
```

```
        from keras.applications.efficientnet_v2 import EfficientNetV2S
as Model
        RESOLUTION=300
    elif BASE_MODEL=='EfficientNetV2_M':
        from keras.applications.efficientnet_v2 import EfficientNetV2M
as Model
        RESOLUTION=380
    elif BASE_MODEL=='EfficientNetV2_L':
        RESOLUTION=380
        from keras.applications.efficientnet_v2 import EfficientNetV2L
as Model
    elif BASE_MODEL=='ConvNeXtTiny':
        RESOLUTION=224
        from keras.applications.convnext import ConvNeXtTiny as Model
    elif BASE_MODEL=='ConvNeXtSmall':
        RESOLUTION=384
        from keras.applications.convnext import ConvNeXtSmall as Model
    elif BASE_MODEL=='ConvNeXtBase':
        RESOLUTION=384
        from keras.applications.convnext import ConvNeXtBase as Model
    elif BASE_MODEL=='ConvNeXtLarge':
        RESOLUTION=384
        from keras.applications.convnext import ConvNeXtLarge as Model
    elif BASE_MODEL=='ConvNeXtXLarge':
        RESOLUTION=384
        from keras.applications.convnext import ConvNeXtXLarge as Model
    else:
        raise ValueError('Unknown model: {}'.format(BASE_MODEL))

    IMG_SIZE = (RESOLUTION, RESOLUTION)

    csv_logger = CSVLogger("./dataset/"+BASE_MODEL+"_"+datestr+"
_KerasMel_pd_data10000.csv", append=True)


    train_ds =tf.keras.utils.image_dataset_from_directory('./dataset/
train/',
                                                          labels='
inferred',
                                                          label_mode='
int',
                                                          class_names=
CLASSES,
                                                          color_mode='
rgb',
                                                          batch_size=
BATCH_SIZE,
                                                          image_size=
IMG_SIZE)
```

```
    valid_ds =tf.keras.utils.image_dataset_from_directory('./dataset/
valid/',
                                                          labels='inferred',
                                                          label_mode='int',
                                                          class_names=CLASSES,
                                                          color_mode='rgb',
                                                          batch_size=
BATCH_SIZE,
                                                          image_size=IMG_SIZE)

    if (TUNNING):
        train_ds = train_ds.cache().prefetch(buffer_size=AUTOTUNE)
        valid_ds = valid_ds.cache().prefetch(buffer_size=AUTOTUNE)
    #MODEL

    base_pretrained_model = Model(input_shape =  (RESOLUTION,
RESOLUTION, 3),
                                  include_top = False, weights = '
imagenet')
    base_pretrained_model.trainable = False


    #Modelo layers:

    X = layers.Input((RESOLUTION, RESOLUTION, 3), name='image_rgb_in')
    Y = layers.GaussianNoise(GAUSSIAN_NOISE)(X)
    Y = base_pretrained_model(Y)
    Y = layers.BatchNormalization()(Y)
    Y = layers.Flatten()(Y)
    Y = layers.Dropout(DROPOUT)(Y)

    Y = layers.Dropout(DROPOUT)(layers.Dense(DENSE_COUNT, activation =
'relu',kernel_regularizer=l2(0.01))(Y))
    Y = layers.Dense(len(CLASSES), activation = 'softmax')(Y)

    Model = models.Model(inputs = X, outputs = Y, name = 'full_model')

    optimizer = Adam(learning_rate=LEARN_RATE)
    Model.compile(optimizer,
                      loss = 'sparse_categorical_crossentropy',
                      metrics = ['sparse_categorical_accuracy'])

    Model.summary()

    #CALLBACKS

    from keras.callbacks import ModelCheckpoint, EarlyStopping,
ReduceLROnPlateau
    from IPython.display import clear_output
```

```
        weight_path="./dataset/"+BASE_MODEL+"_"+"{}_weights.best.tf".format
    ('skin_cancer_detector')
        checkpoint = ModelCheckpoint(weight_path, monitor='
    val_sparse_categorical_accuracy', verbose=1,
                                     save_best_only=True, mode='max')

        reduceLROnPlat =  ReduceLROnPlateau(monitor='
    val_sparse_categorical_accuracy', factor=0.2, patience=10, verbose=1,
    mode='auto', min_delta=0.0001, cooldown=5, min_lr=0.000001)
        early = EarlyStopping(monitor="val_sparse_categorical_accuracy",
                              mode="max",
                              patience=20)
        log_dir = "./logs/" +BASE_MODEL+"_" +datetime.datetime.now().
    strftime("%Y%m%d-%H%M%S")
        tensorboard_callback = tf.keras.callbacks.TensorBoard(
            log_dir=log_dir,
            histogram_freq=1)
        #Como chamar o log tensorflow: tensorboard --logdir logs

        callbacks_list = [checkpoint, early, reduceLROnPlat,csv_logger,
    tensorboard_callback]

        train_ds.batch_size = BATCH_SIZE

        fit_results = Model.fit(train_ds,
                              validation_data=valid_ds,
                              epochs = EPOCHS,
                              callbacks = callbacks_list)
        clear_output();

        file.write(f"model: {BASE_MODEL}, loss: {fit_results.history['loss
    ']}, val_loss:{fit_results.history['val_loss']}, Training: {fit_results
    .history['sparse_categorical_accuracy']}, validation: {fit_results.
    history['val_sparse_categorical_accuracy']};\n")
```

Código 10: Treino da Rede Neuronal Convolucional

```
import 'package:camera/camera.dart';
import 'package:flutter/material.dart';
import 'package:skinlesionv2/mainpage.dart';
import 'dart:io';

class MyHttpOverrides extends HttpOverrides{
  @override
  HttpClient createHttpClient(SecurityContext? context){
    return super.createHttpClient(context)
```

```
      ..badCertificateCallback = (X509Certificate cert, String host, int
    port)=> true;
  }
}

late List<CameraDescription> cameras;
Future<void> main() async {
  // Ensure that plugin services are initialized so that `availableCameras
    ()`
  // can be called before `runApp()`
  WidgetsFlutterBinding.ensureInitialized();

  // Obtain a list of the available cameras on the device.
  cameras = await availableCameras();

  // Get a specific camera from the list of available cameras.
  final selectedCamera = cameras.first;

  HttpOverrides.global = MyHttpOverrides();
  runApp(
    MaterialApp(
      //theme: ThemeData.dark(),
      home: Mainpage(
        // Pass the appropriate camera to the TakePictureScreen widget.
        selectedCamera: selectedCamera,
      ),
    ),
  );


}
```

Código 11: Classe inicial da aplicação

```
import 'dart:io';

import 'package:camera/camera.dart';
import 'package:flutter/material.dart';
import 'package:image_picker/image_picker.dart';
import 'dart:convert';

import 'package:http/http.dart' as http;
import 'package:encrypt/encrypt.dart' as encrypt;
import 'package:shared_preferences/shared_preferences.dart';
import 'package:skinlesionv2/bargraph.dart';

class TakePictureScreen extends StatefulWidget {
  const TakePictureScreen({
    super.key,
```

```
    required this.camera,
  });

  final CameraDescription camera;

  @override
  TakePictureScreenState createState() => TakePictureScreenState();
}

class TakePictureScreenState extends State<TakePictureScreen> {
  late CameraController _controller;
  bool _loading = false;
  bool _response = false;
  File? _image;
  List<dynamic>? _results;
  bool encryption = true;
  String testresult =
      "[{label: \"1\", probability: 0.1},{label: \"2\", probability: 0.2},{
    label: \"3\", probability: 0.1},{label: \"4\", probability: 0.1},{label
    : \"5\", probability: 0.1},{label: \"6\", probability: 0.1},{label:
    \"7\", probability: 0.3}]";

  @override
  void initState() {
    super.initState();
    // To display the current output from the Camera,
    // create a CameraController.
    _controller = CameraController(
      // Get a specific camera from the list of available cameras.
      widget.camera,
      // Define the resolution to use.
      ResolutionPreset.medium,
    );
  }

  @override
  void dispose() {
    // Dispose of the controller when the widget is disposed.
    _controller.dispose();
    super.dispose();
  }

  _pickImagefromGallery() async {
    try {
      final pickedImage =
          await ImagePicker().pickImage(source: ImageSource.gallery);
      if (pickedImage != null) {
        setState(() {
          _image = File(pickedImage.path);
          _loading = true;
```

```
        _response = false;
      });
    } else {
      ScaffoldMessenger.of(context).showSnackBar(
          const SnackBar(content: Text("Deve selecionar uma imagem")));
    }
  } catch (e) {
    ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(content: Text("Erro ao Processar a imagem")));
  }
}

_pickImagefromCamera() async {
  try {
    final pickedImage =
        await ImagePicker().pickImage(source: ImageSource.camera);
    if (pickedImage != null) {
      setState(() {
        _image = File(pickedImage.path);
        _loading = true;
        _response = false;
      });
    } else {
      ScaffoldMessenger.of(context).showSnackBar(
          const SnackBar(content: Text("Deve selecionar uma imagem")));
    }
  } catch (e) {
    ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(content: Text("Erro ao Processar a imagem")));
  }
}

Future classifyImage(File image) async {
  String uriText = "";
  if (encryption) {
    uriText = 'https://192.168.1.11:5000/process_image';
  } else {
    uriText = 'https://192.168.1.11:5000/classify';
  }
  try {
    var uri = Uri.parse(uriText);
    var request = http.MultipartRequest('POST', uri);
    if (encryption) {
      // Ler a imagem como bytes
      final bytes = await image.readAsBytes();
      final prefs = await SharedPreferences.getInstance();
      //Pass Definition
      final key = encrypt.Key.fromUtf8(
          'secretpasswordxsecretkey2025xxxx'); //chars for AES-256
      final iv = encrypt.IV.fromUtf8('Sixteen byte IV.'); // Random IV
```

```
      // Encrypt the image
      final encrypter =
          encrypt.Encrypter(encrypt.AES(key, mode: encrypt.AESMode.cbc));
      final encryptedData = encrypter.encrypt(base64Encode(bytes), iv: iv
);
      String savedToken = prefs.getString('saved_token') ?? 'Error';

      if (savedToken == 'Error') {
        ScaffoldMessenger.of(context).showSnackBar(
            const SnackBar(content: Text("Deve configurar o Token")));
        setState(() {
          _loading = false;
          _response = false;
        });
      } else {
        // Send encrypted data to the server
        try {
          final response = await http.post(
            Uri.parse(uriText),
            headers: {
              'Content-Type': 'application/json',
              'Authorization': savedToken, // Adicione o token no
cabecalho
            },
            body: jsonEncode({'data': encryptedData.base64, 'iv': iv.
base64}),
          );

          if (response.statusCode == 200) {
            // Decrypt the response
            final responseBody = jsonDecode(response.body);
            final encryptedResponseData = responseBody['data'];
            final responseIv = encrypt.IV.fromBase64(responseBody['iv']);
            final decryptedResponse =
                encrypter.decrypt64(encryptedResponseData, iv: responseIv
);

            var decodedResponse = json.decode(decryptedResponse);
            setState(() {
              _results = decodedResponse['predictions'];
              _response = true;
              testresult = _results.toString();
              _loading = true;
            });
          } else {
            ScaffoldMessenger.of(context).showSnackBar(
                const SnackBar(content: Text("Erro ao processar o pedido"
)));
            setState(() {
```

```
              _loading = false;
              _response = false;
            });
          }
        } on SocketException catch (e) {
          print('Erro de conex o: ${e.message}');
          ScaffoldMessenger.of(context).showSnackBar(
            const SnackBar(
                content: Text('Erro de conex o: verifique o token ou o
acesso a Internet')),
          );
        } on HttpException catch (e) {
          print('Erro HTTP: ${e.message}');
          ScaffoldMessenger.of(context).showSnackBar(
            SnackBar(content: Text('Erro no servidor: ${e.message}')),
          );
        } catch (e) {
          print('Erro desconhecido: $e');
          ScaffoldMessenger.of(context).showSnackBar(
            SnackBar(content: Text('Ocorreu um erro inesperado')),
          );
        }
      }
    } else {
      //encryption=false
      request.files
          .add(await http.MultipartFile.fromPath('image', image.path));

      var response = await request.send();
      if (response.statusCode == 200) {
        var responseData = await response.stream.bytesToString();
        var decodedResponse = json.decode(responseData);
        setState(() {
          _results = decodedResponse['predictions'];
          testresult = _results.toString();
          _loading = true;
          _response = true;
        });
      } else {
        ScaffoldMessenger.of(context).showSnackBar(SnackBar(
            content: Text(
                'Erro ao classificar a imagem. Codigo: ${response.
statusCode}')));
        setState(() {
          _loading = false;
          _response = false;
        });
      }
    }
  } catch (e) {
```

```
    ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(content: Text("Erro ao Processar a imagem")));

    setState(() {
      _loading = false;
      _response = false;
    });
  }
}

@override
Widget build(BuildContext context) {
  //var h= MediaQuery.of(context).size.height;
  //var w= MediaQuery.of(context).size.width;

  return Scaffold(
    appBar: AppBar(
      title: const Text('SKAPP: Skin Lesion Classifier'),
      backgroundColor: Colors.black,
      titleTextStyle: const TextStyle(color: Colors.white),
    ),
    body: Container(
        //height:h,
        //width:w,

        padding: const EdgeInsets.all(10),
        child:
            Column(crossAxisAlignment: CrossAxisAlignment.center,
  children: [
          const SizedBox(height: 24),
          SizedBox(
              width: 200, // Largura desejada
              height: 50, // Altura desejada
              //padding: const EdgeInsets.all(10),
              child: ElevatedButton(
                onPressed: () async {
                  await _pickImagefromCamera();
                },
                child: const Text('C mara'),
              )),
          const SizedBox(height: 24),
          SizedBox(
              width: 200, // Largura desejada
              height: 50, // Altura desejada
              //padding: const EdgeInsets.all(10),
              child: ElevatedButton(
                onPressed: () async {
                  await _pickImagefromGallery();
                },
                child: const Text('Galeria'),
```

```
                )),
            _loading
                ? Expanded(
                    child: Padding(
                      padding: const EdgeInsets.all(8.0),
                      child: Image(
                        image: FileImage(_image!),
                      ),
                    ),
                  )
                : Container(),
            _loading
                ? ElevatedButton(
                    onPressed: () async {
                      await classifyImage(_image!);
                    },
                    child: const Text('Classificar Imagem'),
                  )
                : Container(),
            _response
                ? BarGraph(
                    testresult: testresult,
                  )
                  :Container(),
            const SizedBox(height: 24),
            SizedBox(
                width: 200, // Largura desejada
                height: 50,
                child: ElevatedButton(
                  onPressed: () {
                    Navigator.pop(context); // Voltar a tela anterior
                  },
                  child: const Text('Voltar'),
                ))
          ])),
    );
  }
}
```

Código 12: Janela da Aplicação que permite selecionar a imagem e obter o resultado da classificação

```
import 'package:flutter/material.dart';
import 'package:shared_preferences/shared_preferences.dart';

class SaveToken extends StatefulWidget {
  const SaveToken({super.key});

  @override
  _SaveTokenState createState() => _SaveTokenState();
```

```
}

class _SaveTokenState extends State<SaveToken> {
  final TextEditingController _textController = TextEditingController();
  String _savedToken = '';

  @override
  void initState() {
    super.initState();
    _loadSavedString();
  }

  // Carregar string salva
  Future<void> _loadSavedString() async {
    final prefs = await SharedPreferences.getInstance();
    setState(() {
      _savedToken = prefs.getString('saved_token') ?? 'Nenhum valor salvo.'
    ;
    });
  }

  // Salvar string
  Future<void> _saveString(String value) async {
    final prefs = await SharedPreferences.getInstance();
    await prefs.setString('saved_token', value);
    setState(() {
      _savedToken = value;
    });
    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(content: Text('Token guardado com sucesso!')),
    );
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Guardar Token'),backgroundColor: Colors.black,
    titleTextStyle: const TextStyle(color: Colors.white),
      ),
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.center,
          children: [
            TextField(
              controller: _textController,
              decoration: const InputDecoration(
                labelText: 'Introduzir o token',
                border: OutlineInputBorder(),
```

```
            ),
          ),
          const SizedBox(height: 16),
          SizedBox(
            width: 200, // Largura desejada
            height: 50,
            child:
          ElevatedButton(
            onPressed: () {
              final input = _textController.text;
              if (input.isNotEmpty) {
                _saveString(input);
                _textController.clear();

              } else {
                ScaffoldMessenger.of(context).showSnackBar(
                  const SnackBar(content: Text('Introduzir o token antes
   de guardar!')),
                );
              }
            },
            child: const Text('Guardar'),
          )),
          const SizedBox(height: 24),
          Text(
            'Token Guardado: $_savedToken',
            style: const TextStyle(fontSize: 16),
          ),
          const SizedBox(height: 24),
          SizedBox(
            width: 200, // Largura desejada
            height: 50,
            child:ElevatedButton(
            onPressed: () {
              Navigator.pop(context); // Voltar a tela anterior
            },
            child: const Text('Voltar'),
          ))
        ],

      ),
    ),
  );
 }
}
```

Código 13: Janela da aplicação que permite guardar o token de autorização da aplicação

```
import 'package:camera/camera.dart';
import 'package:flutter/material.dart';
import 'package:skinlesionv2/save_token.dart';
import 'package:skinlesionv2/takepicture_screen.dart';


class Mainpage extends StatelessWidget {
  const Mainpage({
    super.key,
    required this.selectedCamera,// required camera,
  });

  final CameraDescription selectedCamera;

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'SKAPP: Skin Lesion Classifier',
      home: Scaffold(
        appBar: AppBar(
          title: const Text('SKAPP: Skin Lesion Classifier'),
    backgroundColor: Colors.black,titleTextStyle: const TextStyle(color:
    Colors.white),
        ),
        body: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: <Widget>[

              SizedBox(
              width: 200, // Largura desejada
              height: 50, // Altura desejada
              child: ElevatedButton(
                 onPressed: () {
                  Navigator.push(
                  context,
                  // Acao do terceiro botao
                  MaterialPageRoute(builder: (context) =>
    TakePictureScreen(
                        // Pass the appropriate camera to the
    TakePictureScreen widget.
                        camera: selectedCamera,
                      ),
                    ),
                  );
                },
                child: const Text('Classificar Lesao'),
              )
              ),
              const SizedBox(height: 20), // Espacamento entre os botoes
```

```
              SizedBox(
              width: 200, // Largura desejada
              height: 50, // Altura desejada
              child:ElevatedButton(

                onPressed: () {
                  Navigator.push(
                  context,
                  // Acao do terceiro botao
                   MaterialPageRoute(builder: (context) => const SaveToken())
                  );
                  print('Token');
                },
                child: const Text('Configurar Token'),
              )
              ),
            ],
          ),
        ),
      ),
    );
  }
}
```

Código 14: Janela principal da aplicação com as opções de configuração e classificação

```
import 'dart:convert';

import 'package:flutter/material.dart';


class BarGraph extends StatelessWidget {
  final String testresult;

  const BarGraph({
    super.key,
    required this.testresult,
  });
  @override
  Widget build(BuildContext context) {
    // Convert to lists for BarGraph
    var testresult2 = testresult.replaceAll("label", "\"label\"");
    testresult2 = testresult2.replaceAll("probability", "\"probability\"");
    testresult2 = testresult2.replaceAll("df", "\"df\"");
    testresult2 = testresult2.replaceAll("nv", "\"nv\"");
    testresult2 = testresult2.replaceAll("akiec", "\"akiec\"");
    testresult2 = testresult2.replaceAll("mel", "\"mel\"");
```

```
    testresult2 = testresult2.replaceAll("bcc", "\"bcc\"");
    testresult2 = testresult2.replaceAll("bkl", "\"bkl\"");
    testresult2 = testresult2.replaceAll("vasc", "\"vasc\"");

    final List<dynamic> jsonData = json.decode(testresult2);

    // Convert to lists for BarGraph
    final labels = jsonData.map((item) => item["label"].toString()).toList
    ();
    final values = jsonData
        .map((item) => (item["probability"] as num).toDouble())
        .toList();
    double maxValue = values.reduce((curr, next) => curr > next ? curr :
    next);

    return Container(
      padding: const EdgeInsets.all(16),
      height: 300,
      child: Column(
        children: [
          const Text(
            'Classifica  o',
            style: TextStyle(
              fontSize: 20,
              fontWeight: FontWeight.bold,
            ),
          ),
          const SizedBox(height: 20),
          Expanded(
            child: Row(
              crossAxisAlignment: CrossAxisAlignment.end,
              children: List.generate(
                values.length,
                (index) => Expanded(
                  child: Padding(
                    padding: const EdgeInsets.symmetric(horizontal: 4),
                    child: Column(
                      mainAxisAlignment: MainAxisAlignment.end,
                      children: [
                        Container(
                          height: (values[index] / maxValue) * 100,
                          decoration: BoxDecoration(
                            color: Colors.blue.shade300,
                            borderRadius: const BorderRadius.vertical(
                              top: Radius.circular(4),
                            ),
                          ),
                        ),
                        const SizedBox(height: 8),
                        Text(
```

```
                          labels[index],
                          style: const TextStyle(fontSize: 12),
                        ),
                        const SizedBox(height: 4),
                        Text(
                          values[index].toStringAsFixed(1),
                          style: const TextStyle(
                            fontSize: 10,
                            color: Colors.grey,
                          ),
                        ),
                      ],
                    ),
                  ),
                ),
              ),
            ),
          ),
        ],
      ),
    );
  }
}
```

Código 15: Classe utilizada para apresentação de resultados da classificação

```
from flask import Flask, request, jsonify
from PIL import Image
import numpy as np
import io
import tensorflow as tf

from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image

import logging
import os
from keras.preprocessing import image
from Crypto.Cipher import AES
import base64
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms,
    modes
from cryptography.hazmat.backends import default_backend
from Crypto.Util.Padding import pad, unpad

logging.basicConfig(level=logging.DEBUG)
logger = logging.getLogger(__name__)

app = Flask(__name__)
```

```
name="ConvNeXtXLarge_skin_cancer_detector.keras"
#  modelo pr -treinado
if os.name == 'nt':
    model = load_model('C:\\Users\\Utilizador\\Documents\\Mestrado\\Thesis
    \\SkinLesion\\ServidorFlask\\'+name)
else:
    model = load_model('/home/jonathan/Documentos/SkinLesion/ServidorFlask/
    '+name)

@app.route('/classify', methods=['GET', 'POST'])
def classify_image():
    if 'image' not in request.files:
        return jsonify({'error': 'No image file provided'}), 400

    file = request.files['image']


    image = Image.open(io.BytesIO(file.read()))
    # Pr -processamento da imagem
    image = image.resize((384, 384))
    image_array = np.array(image)
    image_array = np.expand_dims(image_array, axis=0)

    # previs o
    predictions = model.predict(image_array)

    class_names = ['akiec', 'bcc', 'bkl', 'df', 'mel', 'nv', 'vasc']

    results = [
        {'label': class_names[i], 'probability': float(prob)}
        for i, prob in enumerate(predictions[0])
    ]

    return jsonify({'predictions': results})
    #return predictions[0].toString()

def _preProcessInput(image,model):


  inputTensor = TensorImage(model.getInputTensor(0).type)
  inputTensor.loadImage(image)


  minLength = min(inputTensor.height, inputTensor.width)
  fcropOp = ResizeWithCropOrPadOp(minLength, minLength)


  shapeLength = model.getInputTensor(0).shape[1]
  resizeOp = ResizeOp(shapeLength, shapeLength, ResizeMethod.BILINEAR)
```

```
  normalizeOp = NormalizeOp(127.5, 127.5)


  imageProcessor = ImageProcessorBuilder().add(cropOp).add(resizeOp).add(
    normalizeOp).build()

  imageProcessor.process(inputTensor)


  return inputTensor

# A mesma chave definida no Flutter
AES_KEY = b'secretpasswordxsecretkey2025xxxx'  # Deve ser exatamente igual
       keyString do Flutter
iv = b'Sixteen byte IV.'
VALID_TOKEN = "tokenteste"  # Token esperado

@app.route('/process_image', methods=['GET', 'POST'])
def process_image():
    try:

        token = request.headers.get('Authorization')
        if not token or token != VALID_TOKEN:
            response = jsonify({'error': 'Invalid token'}).get_data(as_text
    =True)
            response_cipher = AES.new(AES_KEY, AES.MODE_CBC)
            response_iv = response_cipher.iv
            encrypted_response = base64.b64encode(response_cipher.encrypt(
    pad(response.encode('utf-8'), AES.block_size)))

            return jsonify({
                'data': encrypted_response.decode('utf-8'),
                'iv': base64.b64encode(response_iv).decode('utf-8')
            })
            #return jsonify({'error': 'Invalid token'})

        logger.debug("Recebeu Imagem")
        data = request.json['data']
        encrypted_data = base64.b64decode(data)
        iv = base64.b64decode(request.json['iv'])
        # Decrypt the data
        cipher = AES.new(AES_KEY, AES.MODE_CBC, iv)
        decrypted_data = unpad(cipher.decrypt(encrypted_data), AES.
    block_size).decode('utf-8')

        # Handle the image file
        file = io.BytesIO(base64.b64decode(decrypted_data))
        image = Image.open(file)
```

```
        logger.debug("Desencriptou Imagem")
        image = image.resize((384, 384))
        image_array = np.array(image)
        image_array = np.expand_dims(image_array, axis=0)

        predictions = model.predict(image_array)

        class_names = ['akiec', 'bcc', 'bkl', 'df', 'mel', 'nv', 'vasc']

        results = [
            {'label': class_names[i], 'probability': float(prob)}
            for i, prob in enumerate(predictions[0])
        ]

        response = jsonify({'predictions': results}).get_data(as_text=True)
        response_cipher = AES.new(AES_KEY, AES.MODE_CBC)
        response_iv = response_cipher.iv
        encrypted_response = base64.b64encode(response_cipher.encrypt(pad(
    response.encode('utf-8'), AES.block_size)))

        return jsonify({
            'data': encrypted_response.decode('utf-8'),
            'iv': base64.b64encode(response_iv).decode('utf-8')
        })
        #return jsonify({'predictions': results})

    except Exception as e:
        logger.debug(str(e))

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, ssl_context=('./server.crt', './
    server.key'))
```

Código 16: Servidor em Python que realiza a classificação

References

[1] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," 2014. [Online]. Available: https://arxiv.org/abs/1409.4842

[2] L. Ali, F. Alnajjar, H. Jassmi, M. Gochoo, W. Khan, and M. Serhani, "Performance evaluation of deep cnn-based crack detection and localization techniques for concrete structures," *Sensors*, vol. 21, p. 1688, 03 2021.

[3] "Understanding ResNet-50 in Depth: Architecture, Skip Connections, and Advantages Over Other Networks," https://wisdomml.in/understanding-resnet-50-in-depth-architecture-skip-connections-and-advantages-over-other-networks/, [Online; accessed 26 dezembro 2023].

[4] "Skin Cancer MNIST: HAM10000, a large collection of multi-source dermatoscopic images of pigmented lesions," https://www.kaggle.com/datasets/kmader/skin-cancer-mnist-ham10000/data, [Online; accessed 24 outubro 2023].

[5] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," 2018. [Online]. Available: https://arxiv.org/abs/1608.06993

[6] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," 2016. [Online]. Available: https://arxiv.org/abs/1602.07261

[7] E. Bendersky, "Depthwise separable convolutions for machine learning," https://eli.thegreenplace.net/2018/depthwise-separable-convolutions-for-machine-learning/, [Online; accessed 12 Dezembro 2024].

[8] A. Pandey, "Depth-wise Convolution and Depth-wise Separable Convolution," https://medium.com/@zurister/depth-wise-convolution-and-depth-wise-separable-convolution-37346565d4ec, [Online; accessed 12 Dezembro 2024].

[9] A. Tragoudaras, P. Stoikos, K. Fanaras, A. Tziouvaras, G. Floros, G. Dimitriou, K. Kolomvatsos, and G. Stamoulis, "Design space exploration of a sparse mobilenetv2 using high-level synthesis and sparse matrix techniques on fpgas," *Sensors*, vol. 22, p. 4318, 06 2022.

[10] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," 2017. [Online]. Available: https://arxiv.org/abs/1610.02357

[11] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," 2018. [Online]. Available: https://arxiv.org/abs/1707.07012

[12] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," 2020. [Online]. Available: https://arxiv.org/abs/1905.11946

[13] R. Al-ahmadi, H. Al-ghamdi, and L. Hsairi, "Classification of diabetic retinopathy by deep learning," *International Journal of Online and Biomedical Engineering (iJOE)*, vol. 20, pp. 74–88, 01 2024.

[14] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," 2019. [Online]. Available: https://arxiv.org/abs/1801.04381

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023. [Online]. Available: https://arxiv.org/abs/1706.03762

[16] A. Erdoğan, "ConvNeXt — Next Generation of Convolutional Networks," https://medium.com/@atakanerdogan305/convnext-next-generation-of-convolutional-networks-325607a08c46, [Online; accessed 20 Novembro 2024].

[17] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[18] "Precision and recall," https://en.wikipedia.org/wiki/Precision_and_recall, [Online; accessed 20 setembro 2024].

[19] "Cancro da pele (não Melanoma)," https://www.cuf.pt/saude-a-z/cancro-da-pele-nao-melanoma, [Online; accessed 24 outubro 2023].

[20] "Melanoma," https://www.ligacontracancro.pt/melanoma/, [Online; accessed 25 outubro 2023].

[21] A. Mahbod, G. Schaefer, C. Wang, R. Ecker, and I. Ellinge, "Skin lesion classification using hybrid deep neural networks," in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 1229–1233.

[22] A. Hekler, J. S. Utikal, A. H. Enk, A. Hauschild, M. Weichenthal, R. C. Maron, C. Berking, S. Haferkamp, J. Klode, D. Schadendorf, B. Schilling, T. Holland-Letz, B. Izar, C. von Kalle, S. Fröhling, T. J. Brinker, L. Schmitt, W. K. Peitsch, F. Hoffmann, J. C. Becker, C. Drusio, P. Jansen, J. Klode, G. Lodde, S. Sammet, D. Schadendorf, W. Sondermann, S. Ugurel, J. Zader, A. Enk, M. Salzmann, S. Schäfer, K. Schäkel, J. Winkler, P. Wölbing, H. Asper, A.-S. Bohne, V. Brown, B. Burba, S. Deffaa, C. Dietrich, M. Dietrich, K. A. Drerup, F. Egberts, A.-S. Erkens, S. Greven, V. Harde, M. Jost, M. Kaeding, K. Kosova, S. Lischner, M. Maagk, A. L. Messinger, M. Metzner, R. Motamedi, A.-C. Rosenthal, U. Seidl, J. Stemmermann, K. Torz, J. G. Velez, J. Haiduk, M. Alter, C. Bär, P. Bergenthal, A. Gerlach, C. Holtorf, A. Karoglan, S. Kindermann, L. Kraas, M. Felcht, M. R. Gaiser, C.-D. Klemke, H. Kurzen, T. Leibing, V. Müller, R. R. Reinhard, J. Utikal, F. Winter, C. Berking, L. Eicher, D. Hartmann, M. Heppt, K. Kilian, S. Krammer, D. Lill, A.-C. Niesert, E. Oppel, E. Sattler, S. Senner, J. Wallmichrath, H. Wolff, A. Gesierich, T. Giner, V. Glutsch, A. Kerstan, D. Presser, P. Schrüfer, P. Schummer, I. Stolze, J. Weber, K. Drexler, S. Haferkamp, M. Mickler, C. T. Stauner, and A. Thiem, "Superior skin cancer classification by the combination of human and artificial intelligence," *European Journal of Cancer*, vol. 120, pp. 114–121, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0959804919304277

[23] A. Al-Rasheed, A. Ksibi, M. Ayadi, A. Alzahrani, and M. Elahi, "An ensemble of transfer learning models for the prediction of skin lesions with conditional generative adversarial networks," *Contrast Media and Molecular Imaging*, vol. 2023, pp. 1–15, 04 2023.

[24] L. Soenksen, T. Kassis, S. Conover, B. Marti-Fuster, J. Birkenfeld, J. Tucker-Schwartz, A. Naseem, R. Stavert, C. Kim, M. Senna, J. Avilés-Izquierdo, J. Collins, R. Barzilay, and M. Gray, "Using deep learning for dermatologist-level detection of suspicious pigmented skin lesions from wide-field images," *Science Translational Medicine*, vol. 13, p. eabb3652, 02 2021.

[25] D. B. Mendes and N. C. da Silva, "Skin lesions classification using convolutional neural networks in clinical images," 2018. [Online]. Available: https://arxiv.org/abs/1812.02316

[26] K. Agarwal and T. Singh, "Classification of skin cancer images using convolutional neural networks," 2022. [Online]. Available: https://arxiv.org/abs/2202.00678

[27] M. S. Akter, H. Shahriar, S. Sneha, and A. Cuzzocrea, "Multi-class skin cancer classification architecture based on deep convolutional neural network," in *2022 IEEE International Conference on Big Data (Big Data)*. IEEE, Dec. 2022, p. 5404–5413. [Online]. Available: http://dx.doi.org/10.1109/BigData55660.2022.10020302

[28] D. A. V. Nunez and Y. Li, "Skin lesion diagnosis using convolutional neural networks," 2023. [Online]. Available: https://arxiv.org/abs/2305.11125

[29] A. Ech-Cherif, M. Misbhauddin, and M. Ech-Cherif, "Deep neural network based mobile dermoscopy application for triaging skin cancer detection," in *2019 2nd International Conference on Computer Applications & Information Security (ICCAIS)*, 2019, pp. 1–6.

[30] I. Oztel, G. Yolcu Oztel, and V. H. Sahin, "Deep learning-based skin diseases classification using smartphones," *Advanced Intelligent Systems*, vol. 5, no. 12, p. 2300211, 2023. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/aisy.202300211

[31] R. Francese, M. Frasca, M. Risi, and G. Tortora, "A mobile augmented reality application for supporting real-time skin lesion analysis based on deep learning," *J. Real-Time Image Process.*, vol. 18, no. 4, p. 1247–1259, Aug. 2021. [Online]. Available: https://doi.org/10.1007/s11554-021-01109-8

[32] N. Hameed, A. Shabut, F. Hameed, S. Cirstea, S. Harriet, and A. Hossain, "Mobile-based skin lesions classification using convolution neural network," *Annals of Emerging Technologies in Computing*, vol. 4, pp. 26–37, 04 2020.

[33] F. Galbusera, G. Casaroli, and T. Bassani, "Artificial intelligence and machine learning in spine research," *JOR SPINE*, vol. 2, no. 1, p. e1044, 2019. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/jsp2.1044

[34] "Supervised and Unsupervised Machine Learning Algorithms," https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/, [Online; accessed 05 novembro 2023].

[35] "Funções de ativação: definição, características, e quando usar cada uma," https://iaexpert.academy/2020/05/25/funcoes-de-ativacao-definicao-caracteristicas-e-quando-usar-cada-uma/, [Online; accessed 10 dezembro 2023].

[36] "How to Choose an Activation Function for Deep Learning," https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/, [Online; accessed 10 dezembro 2023].

[37] S. Ruder, “An overview of gradient descent optimization algorithms,” 2017. [Online]. Available: https://arxiv.org/abs/1609.04747

[38] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[39] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” 2017. [Online]. Available: https://arxiv.org/abs/1704.04861

[40] “ cifar10,” https://www.tensorflow.org/datasets/catalog/cifar10?hl=pt, [Online; accessed 20 Novembro 2024].

[41] “imagenet,” https://www.image-net.org/, [Online; accessed 20 Novembro 2024].

[42] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, “A convnet for the 2020s,” 2022. [Online]. Available: https://arxiv.org/abs/2201.03545

[43] M. Iman, H. R. Arabnia, and K. Rasheed, “A review of deep transfer learning and recent advancements,” *Technologies*, vol. 11, no. 2, 2023. [Online]. Available: https://www.mdpi.com/2227-7080/11/2/40

[44] D. Fagbuyiro, “Guide To Transfer Learning in Deep Learning,” https://medium.com/@davidfagb/guide-to-transfer-learning-in-deep-learning-1f685db1fc94, [Online; accessed 24 Novembro 2024].

[45] “Visão geral do LiteRT,” https://ai.google.dev/edge/litert?hl=pt-br, [Online; accessed 12 Dezembro 2024].

[46] “Advanced Encryption Standard,” https://pt.wikipedia.org/wiki/Advanced_Encryption_Standard, [Online; accessed 10 Dezembro 2024].

[47] “Flutter,” https://flutter.dev/, [Online; accessed 06 novembro 2023].

[48] “Flutter,” https://pt.wikipedia.org/wiki/Flutter, [Online; accessed 06 novembro 2023].

[49] Kracekumar, “Advanced Encryption Standard,” https://kracekumar.com/post/54437887454/ssl-for-flask-local-development/, [Online; accessed 10 Dezembro 2024].

[50] “BASE64,” https://en.wikipedia.org/wiki/Base64, [Online; accessed 12 Dezembro 2024].

[51] B. Shetty, R. Fernandes, A. P. Rodrigues, R. Chengoden, S. Bhattacharya, and K. Lakshmanna, "Skin lesion classification of dermoscopic images using machine learning and convolutional neural network," *Scientific Reports*, vol. 12, no. 1, p. 18134, Oct 2022. [Online]. Available: https://doi.org/10.1038/s41598-022-22644-9

[52] "Aumento de dados," https://www.tensorflow.org/tutorials/images/data_augmentation?hl=pt-br, [Online; accessed 24 outubro 2023].

[53] "O que é overfitting?" https://www.ibm.com/br-pt/topics/overfitting, [Online; accessed 20 Novembro 2024].

[54] H. Yadav, "Dropout in Neural Networks," https://towardsdatascience.com/dropout-in-neural-networks-47a162d621d9, [Online; accessed 20 Novembro 2024].

[55] C. Cortes, M. Mohri, and A. Rostamizadeh, "L2 regularization for learning kernels," 2012. [Online]. Available: https://arxiv.org/abs/1205.2653

[56] J. M. Ph.D. and E. Kavlakoglu, "What is ridge regression? ," https://www.ibm.com/think/topics/ridge-regression, [Online; accessed 12 Dezembro 2024].

[57] "ReduceLROnPlateau," https://keras.io/api/callbacks/reduce_lr_on_plateau/#reducelronplateau-class, [Online; accessed 19 Novembro 2024].

[58] A. J. J. Branco, F. M. R. M. F. Dias, and F. R. S. Mendonça, "Sentiment analysis of restaurant reviews in portuguese: A transfer learning and ensemble approach with edge computing," 2023. [Online]. Available: http://hdl.handle.net/10400.13/5602

[59] A. Bhandari, "Guide to AUC ROC Curve in Machine Learning," https://www.analyticsvidhya.com/blog/2020/06/auc-roc-curve-machine-learning/, [Online; accessed 19 Novembro 2024].

[60] J. Brownlee, "One-vs-Rest and One-vs-One for Multi-Class Classification," https://machinelearningmastery.com/one-vs-rest-and-one-vs-one-for-multi-class-classification/, [Online; accessed 19 Novembro 2024].

[61] "Google Colab ," https://colab.google/, [Online; accessed 24 Novembro 2024].

[62] M. McHugh-Johnson, “Ask a Techspert: What’s the difference between a CPU, GPU and TPU?” https://blog.google/technology/ai/difference-cpu-gpu-tpu-trillium/, [Online; accessed 12 Dezembro 2024].