

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

# Knowledge Elicitation in Deep Learning Models

Wagner Luiz da Cunha Ceulin



Mestrado em Engenharia e Ciência de Dados

Supervisor: Rui Carlos Camacho de Sousa Ferreira da Silva

July, 2023



# **Knowledge Elicitation in Deep Learning Models**

**Wagner Luiz da Cunha Ceulin**

Mestrado em Engenharia e Ciência de Dados

July, 2023





# Abstract

Though a buzzword in modern problem-solving across various domains, deep learning presents a significant challenge - interpretability. This thesis journeys through a landscape of knowledge elicitation in deep learning models, shedding light on feature visualization, saliency maps, and model distillation techniques.

These techniques were applied to two deep learning architectures: Convolutional Neural Networks (CNN) and a black box package model (Google Vision). Our investigation provided valuable insights into their effectiveness in eliciting and interpreting the encoded knowledge. While they demonstrated potential, limitations were also observed, suggesting room for further development in this field.

This work does not just highlight the need for more transparent, more explainable deep learning models, it gives a gentle nudge to developing innovative techniques to extract knowledge. It is all about ensuring responsible deployment and emphasizing the importance of transparency and comprehension in machine learning.

In addition to evaluating existing methods, this thesis also explores the potential for combining multiple techniques to enhance the interpretability of deep learning models. A blend of feature visualization, saliency maps, and model distillation techniques was used in a complementary manner to extract and interpret the knowledge from our chosen architectures.

Experimental results highlight the utility of this combined approach, revealing a more comprehensive understanding of the models' decision-making processes. Furthermore, we propose a novel framework for systematic knowledge elicitation in deep learning, which cohesively integrates these methods. This framework showcases the value of a holistic approach toward model interpretability rather than relying on a single method.

Lastly, we discuss the ethical implications of our work. As deep learning models continue to permeate various sectors, from healthcare to finance, ensuring their decisions are explainable and justified becomes increasingly crucial. Our research underscores this importance, laying the groundwork for creating more transparent, accountable AI systems in the future.

**Keywords:** knowledge elicitation, deep learning, explainability, model distillation.



# Resumo

Embora o aprendizado profundo (mais conhecido como *deep learning*) tenha se tornado uma ferramenta popular na solução de problemas modernos em vários domínios, ele apresenta um desafio significativo - a interpretabilidade. Esta tese percorre um cenário de elicitación de conhecimento em modelos de deep learning, lançando luz sobre a visualização de características, mapas de saliência e técnicas de destilação.

Estas técnicas foram aplicadas a duas arquiteturas: redes neurais convolucionais (CNN) e um modelo de pacote (Google Vision). A nossa investigação forneceu informações valiosas sobre a sua eficácia na elicitación e interpretação do conhecimento codificado. Embora tenham demonstrado potencial, também foram observadas limitações, sugerindo espaço para mais desenvolvimento neste campo.

Este trabalho não só realça a necessidade de modelos de deep learning mais transparentes e explicáveis, como também impulsiona o desenvolvimento de técnicas para extrair conhecimento. Trata-se de garantir uma implementação responsável e enfatizar a importância da transparência e compreensão da aprendizagem computacional.

Além de avaliar os métodos existentes, esta tese explora também o potencial de combinar múltiplas técnicas para melhorar a interpretabilidade dos modelos de deep learning. Uma mistura de visualização de características, mapas de saliência e técnicas de destilação de modelos foi usada de uma maneira complementar para extrair e interpretar o conhecimento das arquiteturas escolhidas.

Os resultados experimentais destacam a utilidade desta abordagem combinada, revelando uma compreensão mais abrangente dos processos de tomada de decisão dos modelos. Além disso, propomos um novo modelo para a elicitación sistemática de conhecimento em deep learning, que integra de forma coesa estes métodos. Este quadro demonstra o valor de uma abordagem holística para a interpretabilidade do modelo, em vez de se basear num único método.

Por fim, discutimos as implicações éticas do nosso trabalho. À medida que os modelos de deep learning continuam a permear vários setores, desde a saúde até às finanças, garantir que as suas decisões são explicáveis e justificadas torna-se cada vez mais crucial. A nossa investigação sublinha esta importância, preparando o terreno para a criação de sistemas de inteligência artificial mais transparentes e responsáveis no futuro.

**Keywords:** elicitación de conhecimento, aprendizado profundo, explicabilidade, destilação de modelos.



# Acknowledgements

I dedicate this work to my parents, Maria Lúcia and †Welliton Ceulin, who taught me that knowledge opens all doors and sacrificed some of their dreams to provide me with the best study conditions during my youth. I also dedicate this to Thalita Schinniger, my life partner, for all her patience, support, and understanding throughout my master's studies. I am also grateful to my godparents, †Marlene and †Jurandir Ceulin, to my mentors and the people of Aruanda.

My gratitude extends to my brother and partner, Wellington Ceulin, my good friend and partner Otto Ericksson (and his wife, Rafaella Abbott), Eliza Marcia, Raissa Melo, and the entire team at Grupo Ceulin (AWANTT Prime - Portugal). They supported me and provided the space I needed to fully dedicate myself to my studies.

I appreciate all the MECD/FEUP professors, especially Professors João Moreira, Carlos Soares, and Rui Camacho, for their guidance.

To all my MECD/FEUP friends, especially Gabriel Carvalhal, André Afonso, Nuno Vasconcelos, Danilo Brandão, and Heitor Lira, thank you for your friendship, support, and occasional supervision. I also want to thank to my friends Fernando Hora, Juliana Torri, Domingos Paiva, Fernanda Scoralick, Juliana Vogel, and Jordan Azevedo, who helped gather images for our dataset.

To my sister, Wânia Lúcia Ceulin, whose resilience is a life inspiration, and to my lifelong friends Yoshinori Hishinuma, Leonardo Rosa, Wallace Damião, Rodrigo Souto, Patrick Amaral, Antônio Gomes, Núbia Correia, Victor Hugo Ceulin, and Victor Dolavale - your presence, whether near or far, in the past or present, direct or indirect, challenged me, inspired me, and led me to this point. I am grateful for your presence in my life.

Wagner Ceulin



*“The real risk with AI isn’t malice,  
but competence. A superintelligent AI  
will excel at accomplishing its goals.  
If those goals aren’t aligned with ours,  
we’re in trouble.”*

- Stephen Hawking





# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Knowledge Elicitation . . . . .	2
1.2	Objectives . . . . .	3
1.3	Overview . . . . .	3
<b>2</b>	<b>State-Of-The-Art in Deep Learning</b>	<b>5</b>
2.1	Relevant Deep Learning Publications & Literature Review . . . . .	6
2.1.1	Computational Methods for Deep Learning Theoretic, Practice and Applications . . . . .	6
2.1.2	Improving the Robustness and Encoding Complexity of Behavioural Clones	7
2.1.3	A Comparison of Different Compound Representations for Drug Sensitivity Prediction . . . . .	9
2.1.4	Deep Residual Learning for Image Recognition . . . . .	10
<b>3</b>	<b>Methodology</b>	<b>12</b>
3.1	Research Proposal / Objectives . . . . .	12
3.2	Mapped Tasks . . . . .	13
3.3	Expected Results . . . . .	14
3.3.1	Goals and Outputs . . . . .	14
<b>4</b>	<b>Results</b>	<b>16</b>
4.1	Results of Experiment #1: Skin Cancer Classification . . . . .	16
4.1.1	Presentation of Data . . . . .	18
4.1.2	Performance of the MLP model . . . . .	19
4.1.3	Insights from Knowledge Elicitation . . . . .	23
4.1.4	Performance Evaluation . . . . .	28
4.1.5	Additional Pre-Trained Models . . . . .	31
4.1.6	Summary of Findings . . . . .	35
4.2	Results of Experiment #2: Handwriting Detection . . . . .	36
4.2.1	Presentation of Data . . . . .	38
4.2.2	Performance of the MLP model . . . . .	39
4.2.3	Insights from Knowledge Elicitation . . . . .	48
4.2.4	Performance Evaluation . . . . .	49
4.2.5	Summary of Findings . . . . .	51
4.3	Overall Summary of Findings . . . . .	55

<b>5</b>	<b>Conclusions and Future Work</b>	<b>56</b>
5.1	Interpretation of Results . . . . .	56
5.2	Limitations . . . . .	57
5.3	Recommendations for Future Research . . . . .	58
5.4	Final Remarks . . . . .	58
<b>A</b>	<b>Skin Cancer Integrated Gradients and Grad-CAM</b>	<b>59</b>
A.1	Integrated Gradients and Grad-CAM Results Comparison . . . . .	59
<b>B</b>	<b>Skin Cancer Classification Code</b>	<b>62</b>
B.1	CNN Initial Code (base) . . . . .	62
B.2	Grad-CAM & Integrated Gradients code . . . . .	65
B.3	CNN Updated Model - Version A . . . . .	66
B.4	CNN Updated Model - Version B . . . . .	68
B.5	Pre-trained VGG-16 CNN . . . . .	69
B.6	Pre-trained Inception-ResNet-v2 CNN . . . . .	69
<b>C</b>	<b>Handwritten Dataset Images &amp; Detected Texts</b>	<b>71</b>
C.1	Images and Texts . . . . .	71
<b>D</b>	<b>Handwritten Text Detection Code</b>	<b>80</b>
D.1	Google Vision API Initial Code (base) - Without Preprocessing . . . . .	80
D.1.1	Using ONLY Google Vision API . . . . .	81
D.1.2	Function Used to Filter Portuguese Characters . . . . .	82
D.1.3	Using Google Vision API and PySpellChecker . . . . .	82
D.1.4	Using Google Vision API and PyEnchant . . . . .	82
D.2	Preprocessing Code - Option A . . . . .	83
D.3	Preprocessing Code - Option B . . . . .	84
D.4	Preprocessing Code - Option C . . . . .	85
D.5	Preprocessing Code - Option D . . . . .	86
D.6	Preprocessing Code - Option E . . . . .	87
D.7	Preprocessing Code - Option F . . . . .	88
D.8	PyTesseract OCR Code - Preprocessing Images . . . . .	89
	<b>References</b>	<b>92</b>

# List of Figures

4.1	Benign skin moles examples from dataset. . . . .	18
4.2	Malignant skin moles examples from dataset. . . . .	18
4.3	Model architecture - 1 <sup>st</sup> trial. . . . .	21
4.4	Prediction results for demonstration images. . . . .	22
4.5	Grad-CAM - 1 <sup>st</sup> trial. . . . .	24
4.6	Integrated Gradients (seismic colormap) - 1 <sup>st</sup> trial. . . . .	24
4.7	Model architecture - 2 <sup>nd</sup> trial. Common for Versions A and B. . . . .	27
4.8	Model architecture - 2 <sup>nd</sup> trial. L2 regularization (higlited) removed for VB. . . . .	27
4.9	Prediction results for demonstration images. . . . .	30
4.10	Grad-CAM - 2 <sup>nd</sup> trial - VB. . . . .	31
4.11	Integrated Gradients (seismic colormap) - 2 <sup>nd</sup> trial - VB. . . . .	32
4.12	Grad-CAM - 3 <sup>rd</sup> trial - VGG-16. . . . .	33
4.13	Integrated Gradients (seismic colormap) - 3 <sup>rd</sup> trial - VGG-16. . . . .	33
4.14	Grad-CAM - 4 <sup>th</sup> trial - Inception-ResNet-v2. . . . .	34
4.15	Integrated Gradients (seismic colormap) - 4 <sup>th</sup> trial - Inception-ResNet-v2. . . . .	35
4.16	Model architecture - 3 <sup>rd</sup> trial - VGG-16. . . . .	36
4.17	Model architecture - 4 <sup>th</sup> trial - Inception-ResNet-v2. . . . .	36
4.18	Models references and methods - 1 <sup>st</sup> experiment. . . . .	37
4.19	IAM dataset example. . . . .	38
4.20	GNHK dataset example. . . . .	38
4.21	MNIST dataset examples. . . . .	38
4.22	Examples of our customized dataset. . . . .	39
4.23	Some texts extracted from pictures using PyTesseract OCR after preprocessing. . . . .	40
4.24	Examples of word recognition using offline HTR. . . . .	41
4.25	Examples of word recognition using Google Vision API demo. . . . .	42
4.26	Examples of full text recognition using Google Vision API demo (1 of 2). . . . .	43
4.27	Examples of full text recognition using Google Vision API demo (2 of 2). . . . .	44
4.28	Texts extracted from images 27 to 32 using Google Vision API. . . . .	45
4.29	Dataset images 27 to 32. . . . .	46
4.30	Cosine Similarity formula and visualization. . . . .	47
4.31	Comparison of original and preprocessed images. . . . .	50
4.32	Models references and methods - 2 <sup>nd</sup> experiment. . . . .	53
4.33	Texts extracted from images 27 to 32 using models 0M2b, AM3b and CM6. . . . .	54
A.1	Integrated Gradients results comparison for selected images. . . . .	60
A.2	Grad-CAM results comparison for selected images. . . . .	61
C.1	Dataset images and texts 1 to 3. . . . .	71

C.2	Dataset images and texts 4 to 11. . . . .	72
C.3	Dataset images and texts 12 to 17. . . . .	73
C.4	Dataset images and texts 18 to 23. . . . .	74
C.5	Dataset images and texts 24 to 30. . . . .	75
C.6	Dataset images and texts 31 to 35. . . . .	76
C.7	Dataset images and texts 36 to 39. . . . .	77
C.8	Dataset images and texts 40 to 44. . . . .	78
C.9	Dataset images and texts 45 to 50. . . . .	79

# List of Tables

4.1	Confusion Matrix - 1 <sup>st</sup> trial . . . . .	22
4.2	Confusion Matrix - 2 <sup>nd</sup> trial - VA. . . . .	29
4.3	Confusion Matrix - 2 <sup>nd</sup> trial - VB. . . . .	29
4.4	Confusion Matrix - 3 <sup>rd</sup> trial – VGG-16. . . . .	32
4.5	Confusion Matrix - 4 <sup>th</sup> trial - Inception-ResNet-v2. . . . .	34
4.6	Summary of model performances. . . . .	35
4.7	Metrics with their respective advantages and disadvantages. . . . .	48
4.8	Quality levels based on Cosine Similarity score. . . . .	49
4.9	Summary of image preprocessing methods adopted. . . . .	51
4.10	Cosine Similarity results for different preprocessing and text extraction methods. . . . .	52



# Abbreviations

ACC	Accuracy (performance evaluation metric)
Adam	Adaptive Moment Estimation
AI	Artificial Intelligence
ANN	Artificial Neural Network
AUC	Area Under the Curve (performance evaluation metric)
BN	Batch Normalization
BP	Back Propagation
CNN	Convolutional Neural Network
DL	Deep Learning
DNN	Deep Neural Network
DO	Dropout - A regularization technique for reducing overfitting in NNs
F1	F1 Score (performance evaluation metric)
FC	Fully Connected (layer in a neural network)
GAN	Generative Adversarial Networks
K-fold	K-fold Cross Validation (model evaluation technique)
LSTM	Long-Short Term Memory
MAE	Mean Absolute Error
ML	Machine Learning
MLP	Multi-layer Perceptron (model for handwriting recognition)
MSE	Mean Squared Error
NN	Neural Network
PCA	Principal Component Analysis (dimensionality reduction technique)
ReLU	Rectified Linear Unit
RMSE	Root Mean Squared Error (performance evaluation metric)
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent (optimization algorithm)





# Chapter 1

## Introduction

Deep learning is a subfield of machine learning that is concerned with the development of algorithms and models that are inspired by the structure and function of the brain, specifically the neural networks. It is based on artificial neural networks (ANNs) with representation learning, and it can be used for a wide range of applications such as image classification, speech recognition, natural language processing, and many more.

According to the textbook "Deep Learning" (Goodfellow et al., 2016), deep learning is "a family of algorithms that are used to model high-level abstractions in data. These algorithms are characterized by the use of multiple layers of non-linear processing units for feature extraction and transformation". This allows the model to learn increasingly complex features of the data at each layer, leading to a more powerful and expressive model. It is particularly well-suited for tasks that involve large amounts of data, such as image classification, speech recognition, and natural language processing.

Deep learning, also called deep neural networks, is originated from modelling biological vision and brain information processing. Deep learning is one part of the contents of machine learning or machine intelligence. Deep learning is closely related to mathematics, especially optimization, graphical theory, numerical analysis, functional analysis, probability theory, mathematical statistics, information theory, etc. These subjects could provide the analysis for a neural network model. Usually, when we measure a neural network or evaluate an algorithm, we take into consideration of its robustness, stability, and convergence in numerical analysis (Yan, 2021, chap. 1.2 Deep Learning).

Deep learning has seen rapid advancements in recent years, with deep neural networks achieving state-of-the-art results, however, these models are often considered to be "black boxes" due

to their complex internal workings, making it difficult to understand how they arrive at their predictions. Knowledge elicitation in deep learning models aims to address this issue by providing methods for interpreting and understanding the internal workings of these models.

The concept of "black boxes" in deep learning refers to the difficulty in understanding and interpreting the internal workings of deep learning models. This is due to several factors such as the complexity of the models, the non-linearity of the models, the abstraction of the learned representations and the lack of transparency of the predictions. The internal representations learned by neural networks are often difficult to interpret, and the decisions made by these models are often difficult to understand or explain. According to the textbook "Deep Learning" (Goodfellow et al., 2016), "the internal representations learned by neural networks are often difficult to interpret, and the decisions made by these models are often difficult to understand or explain".

However, researchers have been working on ways to make deep learning models more interpretable. They have developed techniques such as visualization, sensitivity analysis, and concept activation vectors (TCAV) to help understand the internal workings of deep learning models. These techniques can help to make deep learning models more transparent, and to provide insights into how the models are making predictions.

## 1.1 Knowledge Elicitation

Knowledge elicitation is a general term that refers to the process of extracting knowledge from a model or expert. In the context of deep learning, knowledge elicitation refers to the process of extracting meaningful insights and knowledge from a trained deep learning model. This can include understanding how the model is making predictions, what features it is using to make those predictions, and how sensitive the model is to changes in those features. Knowledge elicitation can be useful for a variety of purposes, including improving model performance, explaining model decisions to stakeholders, and gaining a better understanding of the underlying data and patterns.

In the field of medicine, for instance, knowledge elicitation in deep learning models can be useful for a variety of purposes. For example, knowledge elicitation can help to identify the key factors that are driving a model's predictions, which can provide valuable insights for medical researchers and clinicians. It can also help to identify potential biases or limitations in the data or model, which can be important for ensuring that the model is making accurate and fair predictions. In addition, knowledge elicitation can be useful for explaining model decisions to patients and other stakeholders, which can help to improve trust and understanding of the use of deep learning in medicine.

In the field of finance, deep learning models are increasingly being used to predict stock prices, detect fraudulent transactions, and identify patterns in financial data. However, these models are often considered to be "black boxes" due to their complex internal workings, making it difficult to understand how they arrive at their predictions. Knowledge elicitation in deep learning models can be useful for a variety of purposes such as understanding the factors that are driving a model's

predictions, identifying potential biases or limitations in the data or model, explaining model decisions to stakeholders, and identifying and understanding the patterns and features that the model is using to make predictions. This can help to improve the transparency and interpretability of deep learning models in the field of finance and ensure that the models are making accurate and fair predictions while building trust and understanding among stakeholders. (Domingos and Domingos, 2012, vol. 55, no. 10, pp. 78–87)

## 1.2 Objectives

This thesis aims to elucidate the mechanisms and methods employed to scrutinize neuronal behavior in deep learning models and examine their contribution to model predictions. Our research evaluates and contrasts various techniques and approaches, pinpointing the fundamental elements that sway the behavior of neurons in these models. Furthermore, the study will delve into the interpretability of decisions made by the neurons, presenting fresh methodologies that enhance both their interpretability and transparency.

We will utilize two deep learning models and carefully chosen datasets to illustrate the practical application of these proposed methods. This will facilitate a more lucid understanding of their real-world relevance and usability. Moreover, this study will provide thoughtful recommendations for forthcoming research in the realm of knowledge elicitation within deep learning models.

Additionally, this research aspires to augment the field of knowledge elicitation in deep learning models by offering an in-depth exploration of neurons' internal mechanisms. We aim to shed light on their predictive capabilities and provide a framework for their interpretation. We will also suggest novel techniques that could bolster the interpretability and transparency of these models. By demonstrating these proposed methods using a specific model and dataset, we intend to illustrate their practical application, thereby emphasizing the potential influence of our research in the domain of deep learning.

## 1.3 Overview

This thesis proposal is divided into five chapters: Introduction, State Of The Art, Methodology, Results, and Conclusions & Future Work. This Introduction chapter provides an overview of the research problem, explain its importance, and give an overview of the research methods and structure of the thesis. Chapter 2, State Of The Art, provides a comprehensive overview of the current state and potential of deep learning and its applications and also highlights the recent advancements in deep learning techniques, including Capsule Networks and Transfer Learning, and their potential for real-world problems. It also provides a literature review of existing work on knowledge elicitation in deep learning models, including different techniques and approaches that have been used to understand the internal workings of these models. Chapter 3, Methodology, will describe the specific research methods and techniques that will be used to investigate the neuron in a deep learning model. Chapter 4, Results, will present the findings of the research, including

any insights gained about the behavior of the neuron investigated and will interpret the results of the research, discussing the implications of the findings for understanding the internal workings of deep learning models and for future research in this area. Finally, Chapter 5, Conclusions & Future Work, will summarize the key findings of the thesis and provide suggestions for future research. The thesis will be based on the current state of the art and will aim to contribute to the field of knowledge elicitation in deep learning models.

## Chapter 2

# State-Of-The-Art in Deep Learning

The field of deep learning is rapidly evolving and new developments and advancements are constantly being made. Currently, the state-of-the-art in deep learning includes a wide range of techniques and architectures, such as:

- Convolutional Neural Networks (CNN) are currently the state-of-the-art for image classification tasks, they have been used to achieve very high accuracy on a wide range of datasets, such as the ImageNet dataset.
- Recurrent Neural Networks (RNN) and their variants such as Long Short-Term Memory (LSTM) networks and Gated Recurrent Unit (GRU) networks are currently the state-of-the-art for tasks involving sequential data, such as speech recognition, natural language processing and time-series forecasting.
- Transformer-based models such as BERT and GPT-3 are currently the state-of-the-art in natural language processing tasks, they have been used to achieve very high accuracy on a wide range of tasks, such as question answering and language translation.

Additionally, the field of deep reinforcement learning is also gaining traction and shows promising results, for example, AlphaGo and AlphaZero which were based on deep reinforcement learning were able to beat human world champions in the games of Go and Chess respectively.

Moreover, Generative models such as Variational Autoencoders (VAE) and Generative Adversarial Networks (GAN) have been used to generate new and realistic images, videos, and text, which could be used in various applications such as image synthesis, style transfer, and data augmentation.

## 2.1 Relevant Deep Learning Publications & Literature Review

### 2.1.1 Computational Methods for Deep Learning Theoretic, Practice and Applications

Yan (2021) offers an in-depth exploration of various subjects, including deep learning platforms, CNN and RNN, autoencoders and GAN, reinforcement learning, Capsule Networks (CapsNet) and Manifold learning, Boltzmann machines, and transfer learning and ensemble learning. Each section offers an introduction, raises questions, and highlights awarded work in deep learning.

Deep learning, an artificial neural network-inspired subfield of machine learning, is experiencing rapid growth. Deep learning algorithms utilize multiple layers of artificial neural networks to learn and improve automatically from vast amounts of data. This approach has driven significant advancements in various applications, including computer vision, natural language processing, and speech recognition. Today, deep learning is a core technology in the industry, with heavy investments from leading companies like Google, Facebook, and Microsoft.

**Convolutional Neural Networks (CNN):** A preferred deep learning architecture for tasks like image classification, object detection, and segmentation, consist of multiple convolutional and pooling layers that extract and reduce spatial dimensions of image features. The final predictions are made by fully connected layers that follow. Convolutional layers are designed to learn spatial hierarchies of features, and pooling layers decrease the resolution of the feature maps, making the network more robust to small translations and deformations in the input images.

**Recurrent Neural Networks (RNN) and Transformers:** Popular deep learning architectures for natural language processing tasks like language translation and text classification, have unique approaches. RNNs process sequential input data by passing hidden states from one time step to the next, allowing the network to capture long-term dependencies in the data. Transformers, in contrast, use self-attention mechanisms to weight the contribution of different input elements to the output, enabling them to process sequences of variable length effectively.

**Generative Adversarial Networks (GAN):** A deep learning architecture for generative modeling, aim to generate new data samples similar to a given set of training samples. Comprising two networks - a generator and a discriminator, GAN are trained together in a zero-sum game framework. While the generator tries to create samples that can fool the discriminator, the discriminator strives to distinguish between real and generated samples. GAN find application in a broad range of tasks, such as image synthesis, style transfer, and text generation.

**Reinforcement Learning:** Reinforcement learning is a type of machine learning where an agent learns to make decisions in an environment by taking actions and receiving rewards. Reinforcement learning has been applied in a variety of domains, including game playing, robotics, and recommendation systems. In deep reinforcement learning, neural networks are used as function approximators to represent the value or policy functions, allowing the agent to learn from high-dimensional sensory inputs and complex environments. Deep reinforcement learning has shown promising results in challenging domains such as Atari games and Go, and has the potential to be

applied to real-world problems such as autonomous driving and decision making in autonomous systems. Applications of Deep Learning

**Advancements in Deep Learning:** In recent years, there have been many advancements in deep learning techniques, such as GAN, Capsule Networks (CapsNets), and Reinforcement Learning. GAN have been used for image synthesis and style transfer. CapsNets have shown promising results in image classification, especially in handling viewpoint variations and occlusions. Reinforcement learning has been applied in game playing and decision making in autonomous systems. These advancements in deep learning have led to improved performance and expanded the scope of applications for this field.

The book provides a comprehensive overview of various deep learning architectures and their applications. It underscores significant breakthroughs deep learning has made in different fields like computer vision, natural language processing, and game playing, and its potential for real-world problem-solving. This book serves as an invaluable resource for those interested in deep learning and its applications.

### 2.1.2 Improving the Robustness and Encoding Complexity of Behavioural Clones

In the recent literature, the technique of behavioural cloning has been highlighted as a method for reverse engineering human control skills with the aim to improve the understandability and robustness of the generated models. This technique has been utilized in various applications, such as training student pilots and predicting rotor loads and performance in helicopter simulators. Early research into synthetic controllers identified significant complexity and a lack of robustness, prompting researchers to propose the use of more powerful representation schemes such as multivariate decision and regression trees and clausal theories to enhance the results and increase the robustness and comprehensibility of the controllers. Approaches such as Reinforcement Learning and Neural Networks, despite their potential, have been discounted due to their lack of symbolic and comprehensible descriptions, a critical aspect in the applications under consideration (Suc and Bratko, 1997).<sup>1</sup>

Suc and Bratko's (1997) methodology for reverse engineering human control skills comprises six stages, starting with characterizing the system being controlled as a set of state and control variables and concluding with the assembly of an artificial controller. To enhance the original methodology, they added two modifications: each step corresponds to a basic control maneuver, and a wrapper was used to assist the controller construction process by modifying the ML and model parameters. The researchers also introduced the concept of achievable goals and homeostatic objectives to define fundamental control operations.

The artificial controllers in the experiments realized by [Camacho and Brazdil \(2001\)](#) consisted of a high-level module, designed to replicate cognitive human control skills like planning and monitoring, and a low-level module, used to model the reactive components of human control

---

<sup>1</sup>Suc, D. and Bratko, I. (1997). Skill reconstruction as induction of lq controllers with sub-goals. In Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence - IJCAI-97, volume 2, pages 914-920, Nagoya, Japan.

skills. In this setup, the high-level module sequences the work plan's phases and switches the low-level modules as per each stage. The low-level module, the only module induced from behavioural traces, is implemented as a set of decision trees. To produce robust controllers, the authors stress the need for data diversity and introduce the Incremental Correction (IC) model, an adaptive model less complex than previous ones and a more plausible model of human control behavior.

Experimental evaluation of the models was conducted on a simulation of an F-16 aircraft, using a public domain flight simulator. Data from 90 levelled left turn missions were split into two: one half for model building and the other for determining its predictive accuracy. The "wrapper" mentioned refers to a method that creates combinations of parameter values and assesses the performance of the controller, however, it does not entirely account for interactions between controls at deployment time. This is an area the authors plan to improve on in future studies.

The control task chosen for the experimental evaluation was the control of a simulation of an F-16 aircraft performing a levelled left turn. The experiments were carried out using a public domain flight simulator. The data from these missions were split into two halves: one for constructing the model and the other to estimate the predictive accuracy of the constructed model. The performance of the controller was evaluated using error measures such as Error Rate (ER) and Root Mean Square Error (RMSE), tree size, number of successful missions, flight smoothness, and flight time percentage spent outside the security envelope.

The IC model was used to control a simulation of an F-16 aircraft performing a levelled left turn. The experiments were conducted using the ACM 2.4 flight simulator. The characteristics of bank angle, bank angle derivative, bank angle acceleration, pitch, pitch rate, altitude deviation, climb rate, climb rate derivative, and climb rate acceleration were used in the development of the controller's ailerons and elevators trees. The performance is evaluated by measuring the flight time percentage spent outside the envelope.

The IC model was tested using two machine learning algorithms, C4.5 and CART. The CART algorithm outperformed C4.5 in terms of smoothness and error rate. The controllers induced were able to successfully complete all 90 missions, demonstrating a Mean Absolute Deviation (MAD) value very close to human performance. In some measurements, such as climb rate and bank angle, the induced controllers surpassed human performance. Furthermore, the size of the decision trees in the IC model was significantly smaller than in previous models, indicating an improvement in model simplicity and understandability.

It was also tested using an Inductive Logic Programming (ILP) system called IndLog, which encoded the controller components as clausal theories. The results were similar to those achieved with the CART algorithm, although the controller was unable to successfully complete all 90 missions due to instabilities in the control caused by the linear equations. The researchers are considering the implementation of a limit on maximum and minimum control values as a potential solution to this issue.

With its use of goals and attributes that measure deviations from defined goals, the IC model represents a significant advancement in the field. This model enables the use of more diverse training data, leading to more robust controllers. The performance of the IC model controllers in



flight simulation was found to be superior to that of human subjects. Additionally, improvements in model comprehensibility and reduced tree sizes are significant advancements. According to the study, clausal theories and multivariate trees outperformed univariate trees. To further improve the model, the authors plan to include the flight simulator in the wrapper cycle to account for control interactions at deployment time. This comprehensive review of the IC model and its applications has important implications for the ongoing development of deep learning models.

### 2.1.3 A Comparison of Different Compound Representations for Drug Sensitivity Prediction

DeepMol, a comprehensive chemoinformatics package for drug discovery, supports the entire workflow of machine learning and deep learning tasks. It is built with Python, using well-established libraries such as Tensorflow, Keras, Scikit-learn, and DeepChem for constructing custom or pre-defined machine learning and deep learning models. DeepMol integrates the RDKit framework for molecular data operations and offers a range of features, including data loading, standardization, feature computation, feature selection, data splitting, handling of unbalanced datasets, and unsupervised data exploration. It provides a universal platform for building, training, optimizing, and evaluating ML and DL models using different ML frameworks.

In a comprehensive benchmarking study, realized by [Baptista et al. \(2022\)](#), the performance of 12 deep learning algorithms was tested on five drug response datasets, using area under the receiver operating characteristic curve (ROC-AUC) for classification and root mean squared error (RMSE) for regression problems. Results varied depending on the dataset, but some patterns emerged. For instance, ECFP4 fingerprints, LayeredFP, and TextCNN performed well in the NCI 1 classification task, and TextCNN achieved the lowest RMSE in the PC-3 dataset. Interestingly, end-to-end deep learning models performed comparably or even better than pre-computed feature-based models. However, most models demonstrated a propensity to overfit, suggesting the need for mechanisms like early stopping to control overfitting.

The study demonstrated that DeepMol is a robust tool for comparing the performance of various compound representation methods and deep learning algorithms in predicting cancer drug sensitivity. Notably, end-to-end deep learning models outperformed traditional fingerprint-based models, even with small datasets. The study also suggested that less popular molecular fingerprints could be promising alternatives. Despite the current success, the DeepMol framework continues to evolve, with new models and features to be added in the future.

Simultaneously, multilayer perceptrons (MLP), a type of artificial neural network, have shown promise in the field of handwritten recognition, a research area active for decades. Studies have demonstrated the effectiveness of MLP in this task, with recognition accuracy rates reaching over 90%. Despite the rise of deep learning techniques like CNN and RNN, MLP remains a valuable and effective approach in the field of handwritten recognition.

The chemoinformatics field has also found value in deep learning techniques, including MLP,

for drug discovery and predicting pharmacological properties. These methods offer various advantages, such as improved predictive accuracy, complex data structure handling, automation of mundane tasks, handling missing data, and improved generalization. Consequently, these advantages have made deep learning an essential tool for chemoinformatics researchers, enabling greater precision and efficiency in drug development and pharmacological property prediction.

Two papers further illustrate the impact of deep learning methods in chemoinformatics. [Lavecchia and Taulé \(2020\)](#) introduced DeepChem, an open-source software platform for deep learning in chemistry and drug discovery. It demonstrated state-of-the-art performance on tasks like molecular property prediction and virtual screening. In [Ragoza and Aliper \(2017\)](#), the authors investigated various deep learning techniques for predicting drug pharmacological properties and drug repurposing. They demonstrated that deep learning models outperformed traditional machine learning techniques in these tasks.

Both studies emphasize the value of deep learning in chemistry and drug development, demonstrating that modern performance on these tasks can be achieved with deep learning methods such as MLP, CNN, and RNN, outperforming more traditional machine learning techniques. The authors also highlight the necessity for platforms and tools that facilitate the creation and application of deep learning models in this area.

### 2.1.4 Deep Residual Learning for Image Recognition

[He et al. \(2015\)](#) introduced a new architecture in CNN, known as ResNet (Residual Network). The authors posited a solution to a common problem faced in training increasingly deep networks: the issue of vanishing/exploding gradients and the resulting degradation problem. The degradation problem refers to the counterintuitive phenomenon where adding more layers to a deep learning model results in higher training error.

ResNet, the proposed architecture, is unique due to its use of shortcut connections or skip connections. These connections allow the gradients to be directly back-propagated to earlier layers, effectively mitigating the problem of vanishing gradients. This architecture permits the training of extremely deep neural networks with hundreds or even thousands of layers, with a compelling accuracy gain in comparison to shallower architectures.

The authors applied their model to the ImageNet dataset, which comprises over a million labeled images. They demonstrated that their 152-layered ResNet performed significantly better than VGG nets and GoogLeNet, which were previous state-of-the-art models, yet with less complexity. The ResNet model achieved a top-5 error rate of 3.57% on the ImageNet test set, outperforming all other models at the time.

Beyond the ImageNet dataset, the authors also tested their model on the COCO object detection dataset. They found that by using a simple adaptation of their model, they were able to surpass previous state-of-the-art models on the mean Average Precision (mAP) metric. These findings were consistent across multiple datasets and tasks, demonstrating the generalizability and robustness of the ResNet model.

The "Deep Residual Learning for Image Recognition" paper marked a significant advancement in the field of computer vision. The introduction of ResNet and the concept of deep residual learning has significantly influenced the architecture of neural networks for image classification and other computer vision tasks. As of today, ResNet is one of the most influential and widely-used models in the field of deep learning for image recognition.

# Chapter 3

## Methodology

This chapter aims to provide a comprehensive understanding of the steps taken to develop and evaluate the MLP model (GAN) for identifying benign or malignant skin cancer in pictures, including data preparation, model architecture, training and testing procedures, and performance evaluation metrics. Additionally, the same methodology is applied to handwriting recognition. This section explores two types of knowledge elicitation: extracting valuable data and knowledge from images and extracting knowledge from black-box models constructed by deep learning.

### 3.1 Research Proposal / Objectives

This study investigates how knowledge elicitation techniques can be applied to improve the interpretability and accuracy of a Python multi-layer perceptron (MLP) model, specifically for handwritten image interpretation and image classification tasks, such as skin cancer detection.

The study is based on two distinct practical experiments, each with its respective methodologies and approaches: (1) extracting information from deep learning predictions for application in the medical field using skin cancer images provided by ISIC Archive<sup>1</sup>, and (2) analyzing images containing handwritten annotations to convert the content into text files. The motivation behind this solution is to enable the future interpretation of pediatric records books, providing valuable information for predicting and preventing potential health problems.

Overall, this study offers insights into the process of eliciting knowledge from deep learning models and demonstrates how it can be applied to practical problems in handwriting recognition and image classification. By presenting a clear methodology and discussing relevant literature, this chapter is a valuable resource for researchers and practitioners interested in applying deep learning models to their work.

---

<sup>1</sup>The International Skin Imaging Collaboration (ISIC) is an academia and industry partnership designed to facilitate the application of digital skin imaging to help reduce melanoma mortality. See <https://www.isic-archive.com/>

To achieve these objectives, we will first develop separate models in Python: one for skin cancer image analysis and another for handwriting detection. We will then obtain and preprocess datasets for both models. A range of knowledge elicitation techniques will be employed to enhance the interpretability and accuracy of the models, such as conducting sensitivity analysis to identify the most crucial input features, performing feature selection and dimensionality reduction to optimize the models' performance, and visualizing the models' internal layers and activations to gain insights into their prediction mechanisms. We will assess the performance of the models in terms of accuracy, precision, recall, and F1 score, and compare the results with existing models and techniques, whenever possible. Statistical methods will be utilized to analyze the significance of the results.

## 3.2 Mapped Tasks

In view of the objectives of this work, the following tasks were mapped:

- **Develop MLP models for skin cancer classification and handwriting detection**
  - Obtain datasets
  - Preprocess datasets (cleaning, normalization, augmentation if needed)
  - Implement MLP models in Python
  - Train and validate MLP models
  - Optimize MLP models (hyperparameter tuning, feature selection)
  - Evaluate potential overfitting and underfitting issues
- **Evaluate the performance of the MLP models**
  - Measure accuracy, precision, recall, and F1 score
  - Compare results with existing models and techniques
  - Perform statistical tests to measure significance of the results
- **Analyze the internal workings of the MLP models**
  - Investigate knowledge elicitation process
  - Examine feature importance and activation patterns
  - Perform sensitivity analysis to identify crucial features
- **Contribute to the field of deep learning**
  - Discuss implications for knowledge elicitation in MLP models
  - Suggest potential future work and improvements based on results
  - Present results at relevant conferences and publish in academic journals

### 3.3 Expected Results

In this research, we anticipate presenting the outcomes of our developed MLP models for skin cancer classification and handwriting detection. The results will be quantified using metrics such as accuracy, precision, recall, and the F1 score. Furthermore, we will delve into an in-depth analysis of the knowledge elicitation process, articulating its impact on enhancing both our models' interpretability and accuracy.

A comparative study will be presented, juxtaposing our results with those of pre-existing models and techniques, thereby providing a benchmark for our performance. These comparisons will not only serve as a validation of our approach but also shed light on areas of potential improvement.

The implications of our findings will be thoroughly discussed, especially focusing on how they could be practically applied to real-world problems. We will highlight the role of our MLP models and knowledge elicitation techniques in advancing the field of deep learning, specifically for image classification and handwriting detection tasks.

Moreover, we will propose potential future work and improvements based on our results. This would include recommendations on how our models could be further refined, as well as suggestions for new areas of application that could benefit from our approach.

#### 3.3.1 Goals and Outputs

The primary objectives and projected outcomes of this project are:

- **Enhancing the interpretability of the MLP models:** Through the application of knowledge elicitation techniques, our goal is to better understand the decision-making process of the MLP models. This includes identifying and addressing potential biases or limitations in the models, thus improving their transparency and trustworthiness. The output in this case would be a comprehensive report detailing the inner workings of the models.
- **Identifying and rectifying model errors:** We aim to leverage knowledge elicitation to pinpoint and correct anomalies or inaccuracies in the model's predictions. The outcome will be an optimized MLP model with improved predictive performance.
- **Boosting model performance:** Knowledge elicitation techniques will be utilized to identify the features that have the most influence on the model's predictions. This information will be used to refine the feature selection process, thereby enhancing the model's performance. The output will be a performance evaluation report illustrating the improvement in accuracy, precision, recall, and F1 score.
- **Promoting effective human-machine collaboration:** By gaining insights into the model's decision-making process, we aim to foster more effective collaboration between human operators and the MLP models. This will allow users to interact more intuitively with the

models and provide the necessary inputs to achieve the desired outcomes. The output here would be guidelines for interaction between the models and human operators.

In conclusion, this study aims to shed light on the process of eliciting knowledge from deep learning models, specifically MLP models, and how it can be practically applied in the fields of skin cancer classification and handwriting recognition. By establishing a clear methodology and discussing relevant literature, we hope to offer a valuable resource for researchers and practitioners interested in incorporating deep learning models into their work. We will also provide insights and recommendations for future research in the area of knowledge elicitation techniques for MLP models in Python, underlining the significance of this study in advancing the field of deep learning and its potential applications in solving real-world problems.

# Chapter 4

## Results

In this chapter, we delve into the heart of this research project, presenting and dissecting the results obtained from our two key experiments. These experiments were strategically designed to explore the efficacy and interpretability of Multi-Layer Perceptron (MLP) models when applied to different, but equally complex, image-based tasks: skin cancer classification and handwriting detection.

The purpose of segregating the results of the two experiments is twofold. First, it allows us to distinctly understand the nuances and peculiarities associated with each task and how the MLP models respond to these. Second, it enables us to discern common patterns and trends across the experiments, thus revealing the strengths and potential limitations of MLP models in handling diverse image interpretation challenges.

In each section dedicated to the individual experiments, we will first present the raw data obtained from the respective task. Subsequently, we will delve into the performance metrics of our MLP model, detailing the levels of accuracy, precision, recall, and F1 score achieved. Importantly, we will also discuss the insights gained through the process of knowledge elicitation, a crucial aspect of our research designed to enhance the interpretability of the MLP models.

By the end of this chapter, we aim to provide a comprehensive understanding of how our MLP models performed in these tasks and how the process of knowledge elicitation aided in improving the models' interpretability and performance. The findings presented here will form the basis for the discussion and conclusions drawn in the next chapter.

### 4.1 Results of Experiment #1: Skin Cancer Classification

The accurate diagnosis of skin cancer represents a significant challenge in healthcare. While many pathologists concur that overdiagnosis is pervasive, changing diagnostic practices proves difficult (Cassidy et al., 2022). In addition, despite the potential of Artificial Intelligence (AI) in detecting skin cancer, the current application of AI in primary care is hindered due to insufficient evidence



in settings where skin cancer prevalence is low. Despite these challenges, AI technology exhibits considerable promise in the detection of skin cancer from images within specialized clinical environments where diagnostic precision is high (Ali et al., 2021). This work aims to contribute to this burgeoning field by demonstrating how Deep Learning (DL) techniques can enhance the performance of skin cancer classification.

The International Skin Imaging Collaboration (ISIC) represents a cooperative venture between academia and industry aimed at promoting digital skin imaging to reduce melanoma<sup>1</sup> mortality rates. The detection and treatment of melanoma in its earliest stages can lead to high cure rates. Digital images of skin lesions can be used to support melanoma diagnosis through teledermatology, clinical decision support, and automated diagnosis, while also educating both professionals and the public on melanoma detection.

ISIC strives to accomplish its objectives by developing and promoting digital skin imaging standards and integrating dermatology with computational science to enhance diagnostics. The lack of standards for dermatologic imaging currently undermines the quality and usefulness of skin lesion imaging. In response, ISIC is developing proposed standards to enhance digital skin image quality, privacy, and interoperability. ISIC is also creating resources for dermatology and informatics, which includes a large and growing open-source public access archive of skin images. This serves as a public resource for images for teaching, research, and for the development and testing of diagnostic AI algorithms.

To date, hundreds of publications have been published in the medical and scientific literature directly resulting from ISIC projects and challenges (ISIC Grand Challenges), which have been held annually since 2016 and have grown in size, complexity, and impact over time. The 2020 Challenge had over 3,300 participants worldwide, resulting in hundreds of new reports in the computer science literature. The ISIC archive contains the most extensive publicly available collection of quality-controlled dermatoscopic images of skin lesions. Currently, the ISIC archive contains over 13,000 dermatoscopic images collected from leading clinical centers worldwide and acquired from various equipment within each center.<sup>2</sup>

---

<sup>1</sup>Melanoma is a type of skin cancer that develops from the pigment-producing cells known as melanocytes. Melanomas can develop anywhere on the skin, but they are more likely to start on the trunk (chest and back) in men and on the legs in women. The face and neck are also common sites. While it is significantly less common than other types of skin cancer, melanoma is more dangerous due to its ability to spread to other parts of the body if not caught early. Factors such as UV radiation from excessive sun exposure or tanning beds, family history, and certain genetic factors can increase the risk of melanoma (source: American Cancer Society <https://www.cancer.org/cancer/types/melanoma-skin-cancer/about/what-is-melanoma.html>).

<sup>2</sup><https://www.isic-archive.com/>

### 4.1.1 Presentation of Data

The dataset used for this work is sourced from the SIIM-ISIC 2020 Challenge archive<sup>3</sup>. This repository represents the most extensive publicly available collection of quality-controlled skin lesion images and is a valuable resource for training and validating our MLP model. The full dataset contains more than 44,000 images (Rotemberg et al., 2021).

This light-version dataset was kindly provided by Mr. Claudio Fanconi, M.Sc.<sup>4</sup>, and is also available on the Kaggle website<sup>5</sup>. It consists of 3,297 selected dermatoscopic images of skin moles (Figures 4.1 and 4.2 show some of the images extracted from the dataset). The dataset is balanced, with 1,800 images labeled benign skin moles and 1,497 labeled malignant. In order to validate our model, we have randomly selected ten images, five benign and five malignant, for conducting prediction tests.

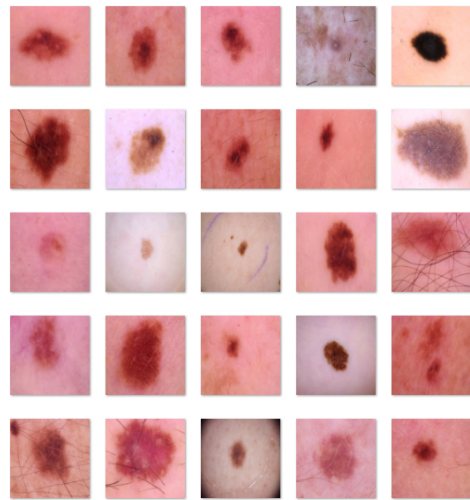


Figure 4.1: Benign skin moles examples from dataset.

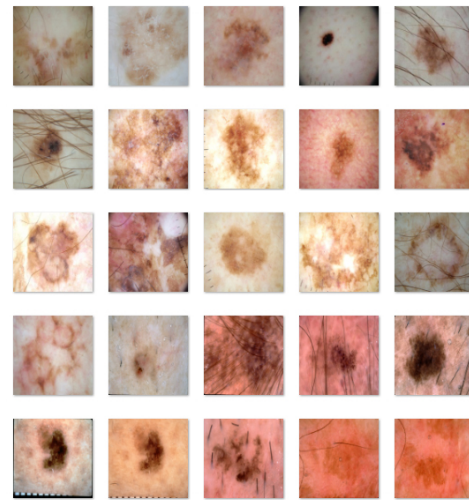


Figure 4.2: Malignant skin moles examples from dataset.

These images, collected from leading clinical centers worldwide using various equipment, provide a representative and clinically relevant sample. The images exhibit a range of characteristics in terms of color, contrast, and noise, posing a realistic challenge for our model. All images underwent a rigorous quality control process as part of ISIC's standards, ensuring privacy and quality assurance. Most images have associated clinical metadata, and a subset of the images has been annotated and marked up by skin cancer experts, emphasizing dermatoscopic features that

<sup>3</sup>International Skin Imaging Collaboration. SIIM-ISIC 2020 Challenge Dataset. International Skin Imaging Collaboration <https://doi.org/10.34970/2020-ds01> (2020). Creative Commons Attribution-Non Commercial 4.0 International License. The dataset was generated by the International Skin Imaging Collaboration (ISIC) and images are from the following sources: Hospital Clínic de Barcelona, Medical University of Vienna, Memorial Sloan Kettering Cancer Center, Melanoma Institute Australia, The University of Queensland, and the University of Athens Medical School. <https://challenge2020.isic-archive.com/>

<sup>4</sup><https://www.linkedin.com/in/claudio-fanconi/>

<sup>5</sup><https://www.kaggle.com/datasets/fanconic/skin-cancer-malignant-vs-benign>

distinguish different skin lesions. This additional information provides valuable context for our analysis and model training.

## 4.1.2 Performance of the MLP model

### 4.1.2.1 Data Preparation and Preprocessing

The first step in our analysis was the preparation and preprocessing of the skin mole image data. This data was organized in directories according to the categories of benign and malignant. The base directory for our data was set and the subdirectories 'benign' and 'malignant' were created within this base directory.

We divided our dataset into training and testing subsets, with a ratio of 80% for training and 20% for testing. To this end, we generated new directories: 'TRAIN' and 'TEST'. Each of these directories also had 'benign' and 'malignant' subdirectories.

Before proceeding with the data split, we implemented a check to ensure the quality and readability of the images. This involved a function, *check\_image()*, which attempted to open each image. If an image was problematic and could not be opened, the function printed an error message and the file was skipped in the data split process.

Once our image data was confirmed to be sound, we proceeded to split the data. If the 'TRAIN' and 'TEST' directories already existed with data, we used them directly. Otherwise, we created and populated these directories by splitting the data from 'benign' and 'malignant' directories. We shuffled the files before splitting to ensure a random distribution in the training and testing sets.

Once our data was divided, we counted the number of images in each category and computed the steps per epoch<sup>6</sup> and validation steps<sup>7</sup> needed for our model training process based on a batch size of 64.

In the final step of data preparation, we used an ImageDataGenerator<sup>8</sup> to perform data augmentation and normalization<sup>9</sup>. This included rescaling pixel values, and randomly applying transformations such as rotations, shifts, zooms, and flips to the training images. The purpose of this was to artificially expand our dataset and make our model more robust to different orientations and positions of skin moles in real-world scenarios. This data was then loaded and preprocessed in batches of 32 images, ready to be input into our MLP model.

This concludes the data preparation and preprocessing phase of our analysis. The next section will discuss the architecture of our MLP model and how it was trained on this data.

---

<sup>6</sup>An epoch in the context of training a MLP or any other neural network refers to one complete pass through the entire training dataset. During an epoch, the model's weights are updated to minimize the loss function (source: Goodfellow et al., Deep Learning, MIT Press, 2016).

<sup>7</sup>A validation step in machine learning refers to the process of evaluating the performance of a trained model on a separate dataset, called the validation set, which was not used during the training phase (source: James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An introduction to statistical learning).

<sup>8</sup>The ImageDataGenerator class in Keras is a data preprocessing tool that generates batches of tensor image data with real-time data augmentation (source: Chollet, F. (2017). Deep learning with Python).

<sup>9</sup>Data augmentation in machine learning is a technique used to increase the diversity of your training set by applying random (but realistic) transformations. On the other hand, data normalization is a preprocessing step used to change the values of numeric columns in the dataset to a common scale, without distorting differences in the ranges of values or losing information (source: Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning).

#### 4.1.2.2 MLP Model Architecture and Training

The architecture of our MLP model was designed to be suitable for the classification of skin mole images. The model is a Convolutional Neural Network (CNN), a type of MLP especially effective for image classification tasks. The model was implemented using TensorFlow's Keras API<sup>10</sup>.

The model architecture comprises several layers. It starts with a sequence of four sets of Convolutional (Conv2D) and MaxPooling2D layers. Convolutional layers are the major building blocks of CNN. These layers apply a specified number of convolution filters to the image. For our model, we started with 32 filters for the first Conv2D layer and then increased this number with depth, going to 64 and then 128 filters. Each filter transforms a part of the image (defined by the kernel size) into a set of abstracted features, performing complex feature extraction. After each Conv2D layer, we applied a MaxPooling2D layer, which reduces the dimensionality of the image, speeding up the computation and helping the model to generalize.

Each Conv2D layer used a 3x3 kernel and a Rectified Linear Unit (ReLU) activation function. ReLU<sup>11</sup> was chosen as the activation function for its efficiency and for mitigating the vanishing gradient problem, which can be an issue in deep neural networks.

Following the Conv2D and MaxPooling2D layers, we applied a Flatten layer to transform the 2D matrix data into a 1D vector. This vector was then passed through a Dropout layer with a rate of 0.5, which helps to prevent overfitting<sup>12</sup> by randomly setting a fraction of input units to 0 at each update during training time.

Finally, we included two Dense layers<sup>13</sup> at the end of our model. The first Dense layer contained 512 units and used a ReLU activation function. The final Dense layer contained a single unit and a sigmoid activation function. This output layer provides a probability that the skin mole image represents a malignant case. The complete architecture of the model can be seen in the Figure 4.3.

---

<sup>10</sup>The Keras API in TensorFlow is a high-level neural networks API that runs on top of TensorFlow, and it's designed for building and training deep learning models (source: Chollet, F. (2017). *Deep learning with Python*).

<sup>11</sup>The Rectified Linear Unit (ReLU) is a commonly used activation function in neural networks and deep learning models. The function returns 0 if the input is less than 0, and it returns the input itself if it's equal to or more than 0. Mathematically, it can be represented as  $f(x) = \max(0, x)$ . ReLU is often used in the hidden layers of a neural network because it introduces non-linearity into the model without requiring computationally expensive operations like exponentials. It also helps alleviate the vanishing gradient problem, which is when the gradients are backpropagated through the network and become increasingly small and have little effect (source: Nwankpa, C., Ijomah, W., Gachagan, A., & Marshall, S. (2018). *Activation Functions: Comparison of trends in Practice and Research for Deep Learning*).

<sup>12</sup>Overfitting and underfitting are common issues that occur during the training of machine learning models. Overfitting happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the model's performance on new data. It essentially means the model is too complex and is fitting too closely to the specific data in the training set, rather than generalizing from it. Underfitting, on the other hand, occurs when a model cannot accurately capture the underlying structure of the data. It might be too simple to capture both the trend and the noise, resulting in poor performance on both the training and test sets. Both overfitting and underfitting lead to poor predictions on new, unseen data (source: Hawkins, D. M. (2004). *The problem of overfitting*. *Journal of chemical information and computer sciences*, 44(1), 1-12).

<sup>13</sup>A Dense layer, also known as a fully connected layer, is a layer in a neural network where each input node is connected to each output node. It works by computing the dot product of the input and the weights, followed by adding a bias value (source: Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*).

```

Model: "sequential"
-----
Layer (type)                Output Shape                Param #
-----
conv2d (Conv2D)              (None, 148, 148, 32)      896
max_pooling2d (MaxPooling2D) (None, 74, 74, 32)        0
conv2d_1 (Conv2D)            (None, 72, 72, 64)        18496
max_pooling2d_1 (MaxPooling2D) (None, 36, 36, 64)        0
conv2d_2 (Conv2D)            (None, 34, 34, 128)       73856
max_pooling2d_2 (MaxPooling2D) (None, 17, 17, 128)       0
conv2d_3 (Conv2D)            (None, 15, 15, 128)       147584
max_pooling2d_3 (MaxPooling2D) (None, 7, 7, 128)        0
flatten (Flatten)            (None, 6272)               0
dropout (Dropout)            (None, 6272)               0
dense (Dense)                 (None, 512)                3211776
dense_1 (Dense)               (None, 1)                   513
-----
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
-----

```

Figure 4.3: Model architecture - 1<sup>st</sup> trial.

To train the model, we used the Adam optimizer<sup>14</sup>, a popular choice for its efficiency and low memory requirements. The loss function used was binary cross-entropy, suitable for our binary classification problem. We measured the performance of our model using accuracy as the metric.

We also used EarlyStopping and ReduceLROnPlateau callbacks during training. EarlyStopping stopped the training when the validation loss didn't improve for a set number of epochs, helping to prevent overfitting. ReduceLROnPlateau reduced the learning rate when the validation loss stopped improving, allowing the model to reach a more optimal solution.

Our model was trained for a total of 30 epochs, although actual training could be stopped earlier by our EarlyStopping callback. The training process was timed, and the total processing time was recorded.

In the next section, we will discuss the evaluation of our model and its performance on the test dataset.

<sup>14</sup>The Adam (Adaptive Moment Estimation) optimizer is a method for efficient stochastic optimization that's been specifically designed for DL. Adam computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients like Adadelta and RMSprop, Adam also keeps an exponentially decaying average of past gradients, similar to momentum. This can make it more suitable for problems that are large in terms of data and/or parameters (source: Kingma, D.P., & Ba, J. (2014). Adam: A Method for Stochastic Optimization).



### 4.1.2.3 Model Evaluation and Performance

In this section, we evaluated the performance of our trained model on the test dataset. The model's performance was assessed based on its accuracy, precision, recall, and F1 score<sup>15</sup>.

First, we loaded the saved model and the test dataset. We then made predictions on this test dataset using the trained model. The model outputted a probability for each image, and we set a threshold of 0.5 to classify each image as benign or malignant.

The prediction and evaluation process took a total processing time of approximately 167.43 seconds. Using this approach, the model was tested on ten demonstration images (five of each kind), correctly classifying seven and misclassifying three, as can be seen in the Figure 4.4.



Figure 4.4: Prediction results for demonstration images.

We calculated the confusion matrix<sup>16</sup> from our predictions and the actual labels of the test dataset. The results obtained are shown in the Table 4.1.

Table 4.1: Confusion Matrix - 1<sup>st</sup> trial

		Predicted	
		Negative	Positive
Actual	Negative	229	130
	Positive	21	278

From these values, we calculated the following metrics:

- Accuracy: 77.05

<sup>15</sup>Precision is the proportion of true positive predictions over the total positive predictions. Recall is the proportion of true positive predictions over the total actual positive cases. F1 score is the harmonic mean of precision and recall, providing a balance between the two metrics (source: James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An introduction to statistical learning).

<sup>16</sup>A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known (source: Kohavi, R., & Provost, F. (1998). Glossary of terms. Machine Learning, 30(2-3), 271-274).

- Benign (Precision: 0.92, Recall: 0.64, F1-score: 0.75)
- Malignant (Precision: 0.68, Recall: 0.93, F1-score: 0.79)

Despite the model's overall accuracy, the lower recall for benign cases suggests that the model may be underperforming in identifying these instances. This area might be a focus for future improvements.

Next, we discuss the results obtained in our evaluation, highlighting the strengths and weaknesses of our model and suggesting possible areas for improvement, including exploring different architectures, hyperparameter adjustments and using different data augmentation techniques, among others.

Based on our observations, three other tests were carried out, with significant changes in the architecture and also with the use of pre-trained models.

### 4.1.3 Insights from Knowledge Elicitation

Knowledge elicitation played an instrumental role in our model refinement process. The insights gleaned from the Grad-CAM and Integrated Gradients visualizations revealed the salient features that our initial model was focusing on during its decision-making process. These insights guided us to modify our model architecture to improve its interpretability and performance.

#### 4.1.3.1 Model Interpretability Methods

Understanding the decision-making intricacies of our model is especially vital in the sensitive domain of skin cancer detection. We employed two interpretability methods to demystify these decisions: Grad-CAM and Integrated Gradients. Both techniques produce visualizations that pinpoint areas in the input images that profoundly influence the model's predictions.

Grad-CAM taps into the gradients of any target concept, channeling them into the final convolutional layer to create a coarse localization map. This map distinctly illuminates pivotal regions in the image crucial for its predictions. In our experiment, regions highlighted in yellow indicate areas the model focuses on most intently. Within skin cancer, these color-coded visual cues play a two-fold role: They validate the model's concentration on dermatologically significant features, bolstering confidence in its predictions, and underscore regions where potentially misleading features might have unduly influenced its conclusions. We applied Grad-CAM to each image in our prediction dataset<sup>17</sup>, subsequently presenting the color-annotated results. This color-enhanced approach facilitates a deeper insight into the model's behavior. It fosters more precise feedback loops with dermatological experts<sup>18</sup>. Figure 4.5 shows a handpicked selection of these color-highlighted visualizations. Integrated Gradients, on the other hand, is a technique attributing the prediction of

---

<sup>17</sup>These images were not involved in the training or testing stages of the model.

<sup>18</sup>A comprehensive exposition on the functionality and interpretation of Grad-CAM, including the significance of color-coded focal points, can be found in the article "Understand your Algorithm with Grad-CAM" by Daniel Reiff, published in Towards Data Science on Jul 21, 2021. Accessible at: <https://towardsdatascience.com/understand-your-algorithm-with-grad-cam-d3b62fce353>.

a neural network to its features. It computes the integral of the gradients of the network's output with respect to the input along a straight-line path from the input to a baseline. We visualized the attributions for each prediction image, demonstrating the contribution of each pixel to the prediction and some results can be seen in the Figure 4.6.

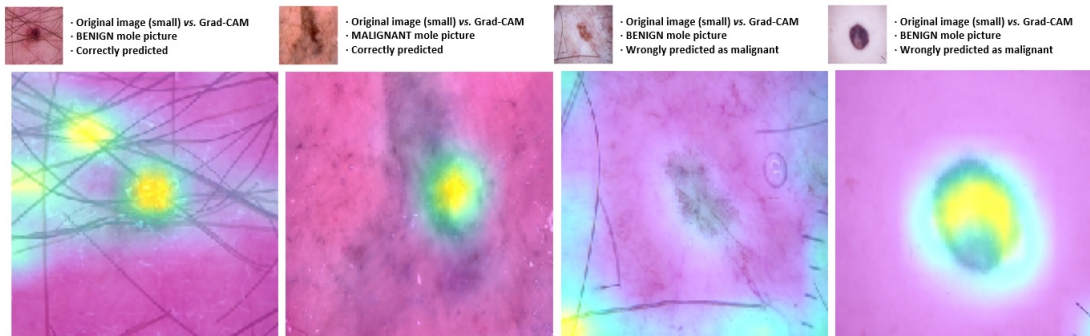


Figure 4.5: Grad-CAM - 1<sup>st</sup> trial.

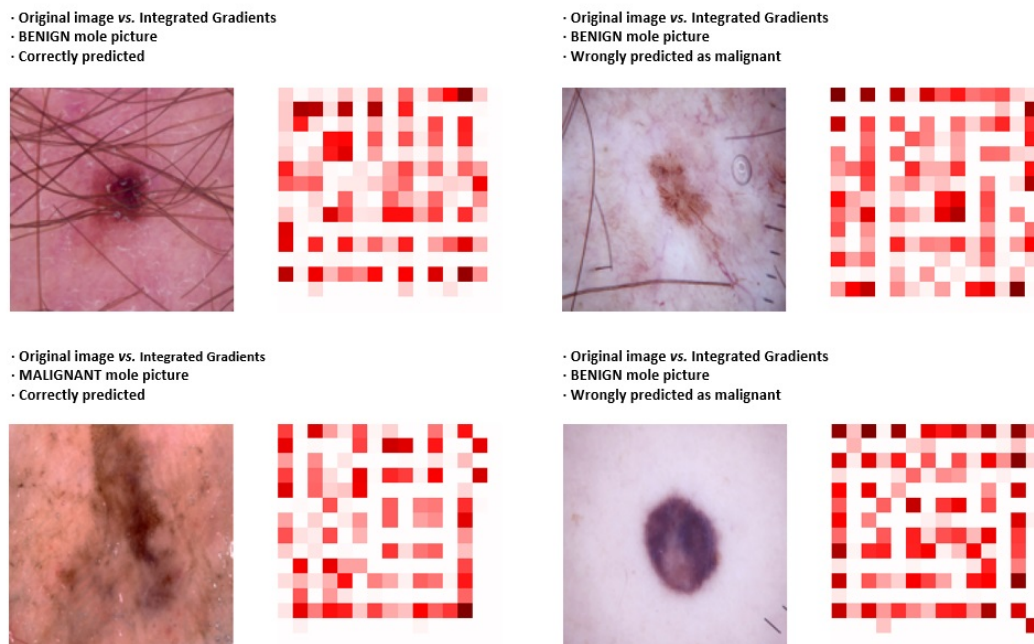


Figure 4.6: Integrated Gradients (seismic colormap) - 1<sup>st</sup> trial.

The Integrated Gradients model's attributions are visualized using a matplotlib seismic colormap<sup>19</sup>. In the context of Integrated Gradients, the attributions signify the contribution of each pixel towards the prediction of the model. In this case, different shades of red indicate varying

<sup>19</sup>The seismic colormap in Matplotlib is a diverging colormap, often used to display data that deviates around a mean value. It provides a clear visual distinction between positive and negative values (source: Matplotlib Documentation at <https://matplotlib.org/stable/tutorials/colors/colormaps.html>).



degrees of positive contribution, with brighter reds signifying a stronger contribution. The white areas signify that these regions do not contribute significantly to the prediction.

These visualizations help us understand where our model is "looking" when making a decision, ensuring it focuses on the correct areas. For example, when correctly identifying malignant lesions, the highlighted areas should be on the lesion itself, not on the surrounding skin or unrelated areas.

#### 4.1.3.2 Model Improvement Strategies and Implementation

A DL model's accuracy can be improved by implementing various strategies. To enhance our model's performance, we considered the following:

- **Increasing Model Complexity:** Adding more Conv2D or Dense layers or increasing the filter/node counts in existing layers (avoiding excessive complexity as it could result in overfitting).
- **Hyperparameter Tuning:** Adjusting learning rates, batch sizes, or the number of epochs can help lower loss during model training.
- **Image Augmentation:** Creating artificial training images through transformations like flips, rotations, shifts, shears, and zooms using tools like ImageDataGenerator (this can improve the model's generalization to unseen images).
- **Pretrained Model Usage (Transfer Learning):** Employing models pre-trained on large datasets like ImageNet can expedite learning by leveraging pre-identified feature sets.
- **Regularization:** Implementing techniques like L1 or L2 regularization, or dropout, to prevent overfitting.
- **Increasing Dataset Size:** Larger datasets often result in better learning and can enhance model performance.
- **Cross-Validation:** Using k-fold cross-validation provides a more accurate assessment of model performance.

Improving a model typically requires an iterative process of trying different strategies and identifying the most suitable ones for the problem at hand. In our work, we began with hyperparameter tuning, image augmentation, regularization, and cross-validation. These strategies were incorporated into our model as follows:

- **Image Augmentation and Regularization:** We used ImageDataGenerator for augmentation and incorporated dropout with L2 regularization to manage overfitting. The parameters of the image data generator (rotation range, width shift range, height shift range, shear range, and zoom range) were tuned to suit our specific problem.

- **Hyperparameter Tuning:** We trained the model with a selection of learning rates, choosing the one that yielded the best results.
- **Cross-Validation:** We manually split the data into subsets and used a for-loop to iterate over these subsets for model evaluation. Each fold was representative of the entire dataset, a crucial aspect of cross-validation.

Model improvement is an ongoing process of tuning, experimentation, and learning until the model achieves an acceptable level of accuracy. We also consider using the pre-trained model, as discussed further in Section 4.1.5. On the other hand, the possibility of increasing the dataset was discarded in our experiment.

#### 4.1.3.3 Model Architecture Changes

Our initial model architecture was composed of four sets of Convolutional Neural Network (CNN) layers, each followed by a MaxPooling2D layer. After the data was flattened, a Dropout layer was applied for regularization, followed by a fully connected layer with 512 units, and finally, an output layer with a sigmoid activation function.

The knowledge elicited from the Grad-CAM and Integrated Gradients visualizations was instrumental in refining our model. These insights revealed the salient features that our initial model focused on, guiding us to modify the architecture to improve its interpretability and performance.

The revised model architecture (Version A: VA) maintains the basic structure of the initial model, but the architecture involves multiple convolutional layers, paired with batch normalization and max pooling operations. Conv2D layers with 32, 64, 128, and 256 filters extract hierarchical image features, utilizing the ReLU activation function. Batch normalization follows each convolutional layer, normalizing inputs to optimize learning and reduce required training epochs. MaxPooling2D layers reduce the spatial dimensions, simplifying computational demands and limiting overfitting. Post these layers, the model flattens the 3D output to a 1D vector for the fully connected layers. These dense layers, with 256 and 128 nodes respectively, use the ReLU activation function to learn global patterns in the data. Overfitting is further controlled with L2 regularization and Dropout operations, where approximately half of the inputs are randomly set to zero during training. Finally, the architecture concludes with a dense layer utilizing a sigmoid activation function, ideal for binary classification tasks, providing the output as a class probability.

We also created an additional revised model (Version B: VB) keeping the same architecture but using only Dropout functions randomly zero-out approximately half of the inputs during training to mitigate overfitting. Changes can be seen in Figures 4.7 and 4.8.

The key differences between initial and revised models are the addition of Batch Normalization layers and an adjustment in the fully connected layers. We anticipate improved performance with the revised model due to these modifications, specifically a higher accuracy on the test set.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 148, 148, 32)	896
batch_normalization (Batch Normalization)	(None, 148, 148, 32)	128
max_pooling2d (MaxPooling2D)	(None, 74, 74, 32)	0
conv2d_1 (Conv2D)	(None, 72, 72, 64)	18496
batch_normalization_1 (Batch Normalization)	(None, 72, 72, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 36, 36, 64)	0
conv2d_2 (Conv2D)	(None, 34, 34, 128)	73856
batch_normalization_2 (Batch Normalization)	(None, 34, 34, 128)	512
max_pooling2d_2 (MaxPooling2D)	(None, 17, 17, 128)	0
conv2d_3 (Conv2D)	(None, 15, 15, 256)	295168
batch_normalization_3 (Batch Normalization)	(None, 15, 15, 256)	1024
max_pooling2d_3 (MaxPooling2D)	(None, 7, 7, 256)	0
flatten (Flatten)	(None, 12544)	0
dense (Dense)	(None, 256)	3211520
batch_normalization_4 (Batch Normalization)	(None, 256)	1024
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 128)	32896
batch_normalization_5 (Batch Normalization)	(None, 128)	512
dropout_1 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 1)	129

```

Total params: 3,636,417
Trainable params: 3,634,689
Non-trainable params: 1,728

```

```

model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(256, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(256, activation='relu', kernel_regularizer=l2(0.01)))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(128, activation='relu', kernel_regularizer=l2(0.01)))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

```

Figure 4.8: Model architecture - 2<sup>nd</sup> trial.  
L2 regularization (highlighted) removed for VB.

Figure 4.7: Model architecture - 2<sup>nd</sup> trial.  
Common for Versions A and B.

#### 4.1.3.4 Model Training and Evaluation

Both models were trained using the Adam optimizer and binary cross-entropy loss function. Early stopping and learning rate reduction were implemented in both cases to prevent overfitting and aid convergence<sup>20</sup>. After the modifications, the model was retrained and evaluated, with prediction results visualized for each image in the prediction dataset, similar to the previous process. The core

<sup>20</sup>Aid in convergence refers to techniques or modifications made in machine learning algorithms that help the model's learning process to converge faster or more effectively. Convergence in the context of machine learning means that the algorithm has reached a state where further training does not significantly improve the model's performance, or the model's error on training and validation sets ceases to decrease and stabilizes. Techniques that aid in convergence can vary from selecting appropriate initial values, normalizing input data, selecting appropriate learning rate, or using advanced optimization techniques (like momentum, Adagrad, RMSProp, Adam). These techniques are designed to expedite the learning process and help navigate the parameter space more efficiently (source: Ruder, S. (2016). An overview of gradient descent optimization algorithms).

of the model consists of four Conv2D-BatchNormalization<sup>21</sup>-MaxPooling2D<sup>22</sup> blocks, followed by a Flatten layer, two Dense-BatchNormalization-Dropout layers, and a final Dense layer for output. L2 regularization was applied in the Dense layers, and a dropout rate of 50% was utilized after the Dense layers.

In the VA, the learning rate of the Adam optimizer was tuned with three different values: 1e-3, 1e-4, and 1e-5. Alongside this, we implemented a 5-fold stratified cross-validation, ensuring that each fold preserved the same percentage of samples for each class. This was done to assess the effectiveness of the model on unseen data. The ReduceLROnPlateau callback was deployed, which reduces the learning rate when there's no improvement in validation loss, aiding the model to find a better minimum. The models were trained for a maximum of 30 epochs, with the actual number possibly being less due to the early stopping mechanism. This approach aimed at enhancing the generalization of the model by adjusting the learning rate and using diverse subsets of the data for training.

VB structure is compiled with the Adam optimizer at a learning rate of 1e-3, utilizing binary cross-entropy as the loss function and accuracy as the performance metric. This version, while simplified, retains its effectiveness for image classification tasks and provides a more streamlined architecture that might be more efficient for specific scenarios.

In the next section, we will discuss the performance of the updated model (Versions A and B), compare it with the initial model, and assess how well our expectations were met.

#### 4.1.4 Performance Evaluation

After implementing the above-mentioned strategies, we evaluated our VA model's performance, and the results were compelling. The model reached an accuracy of approximately 85.26% on the test set. The total processing time recorded was 5608 seconds (or 93 minutes), reflecting the computational effort invested in the model's training and evaluation.

The confusion matrix and classification report further shed light on the model's ability to distinguish between the benign and malignant classes. The confusion matrix, that was represented in Table 4.2, indicates that out of 359 benign instances, the model correctly identified 295 and misclassified 64 as malignant. Similarly, out of 299 malignant instances, 266 were correctly identified while 33 were misclassified as benign.

The classification report, providing precision, recall, and f1-score for both classes, was as follows:

---

<sup>21</sup>Batch normalization is a technique used to improve the training of deep neural networks. It works by normalizing the activations of each layer to have a mean of zero and a standard deviation of one. This normalization process is performed for each mini-batch, which helps in mitigating the problem known as internal covariate shift (source: Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint).

<sup>22</sup>Max pooling is a downsampling operation used in CNN to reduce the spatial dimensions of feature maps. It works by sliding a window (often of size 2x2) across the input feature map and selecting the maximum value from within that window. By doing this, max pooling introduces translational invariance, reduces computation for subsequent layers, and helps in preventing overfitting by providing an abstracted form of the representation (source: Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press).

Table 4.2: Confusion Matrix - 2<sup>nd</sup> trial - VA.

		Predicted	
		Negative	Positive
Actual	Negative	295	64
	Positive	33	266

- Accuracy: 85.26
- Benign (Precision: 0.90, Recall: 0.82, F1-score: 0.86)
- Malignant (Precision: 0.81, Recall: 0.89, F1-score: 0.85)

Both classes showed good precision, recall, and f1-scores, indicating balanced performance across classes.

Adjusting our model to the VB, we observed substantial improvements in its performance. The total processing time for the new version decrease to approximately 563 seconds, and it should be viewed in context: although it represents a 337% increase comparing to the original model it represents only 10,03% of VA processing time. Also, the variation in processing times across multiple runs of both models suggests that this increase is not significantly detrimental. In ten separate runs of each model (original and VB), we observed processing times ranging up to a maximum of 954 seconds for the original model and 563 seconds for the revised model.

The confusion matrix for the updated model VB is shown in Table 4.3. It shows us that the model correctly predicted 312 benign and 256 malignant cases. However, it incorrectly classified 47 benign cases as malignant and 43 malignant cases as benign.

Table 4.3: Confusion Matrix - 2<sup>nd</sup> trial - VB.

		Predicted	
		Negative	Positive
Actual	Negative	312	47
	Positive	43	256

Substantial improvement can be seen in the prediction of false negatives (ie, prediction of benign moles) for both Versions A and B.

The classification report, which provides a more detailed view of the model's performance, is as follows:

- Accuracy: 86.32
- Benign (Precision: 0.88, Recall: 0.87, F1-score: 0.87)
- Malignant (Precision: 0.84, Recall: 0.86, F1-score: 0.85)

From the classification report, we can see that the precision for benign cases is 0.88, indicating that out of all cases predicted as benign, 88% were actually benign. For malignant cases, the precision is 0.84, suggesting that out of all cases predicted as malignant, 84% were indeed malignant.

The recall for benign cases is 0.87, showing that out of all actual benign cases, the model correctly identified 87% of them. For malignant cases, the recall is 0.86, signifying that the model correctly identified 86% of all actual malignant cases.

The F1-score, which is the harmonic mean of precision and recall, is 0.87 for benign cases and 0.85 for malignant cases. This demonstrates a good balance between precision and recall for both types of cases.

It's important to note that all experiments were performed using Google Colab PRO's premium GPU and high RAM, providing a robust and consistent environment for comparison. Thus, despite the increased processing time, the enhanced performance and interpretability of the revised model justifies these changes. Using this new approach, the model was tested on ten demonstration images (five of each kind), correctly classifying nine and misclassifying one, as can be seen in the Figure 4.9.

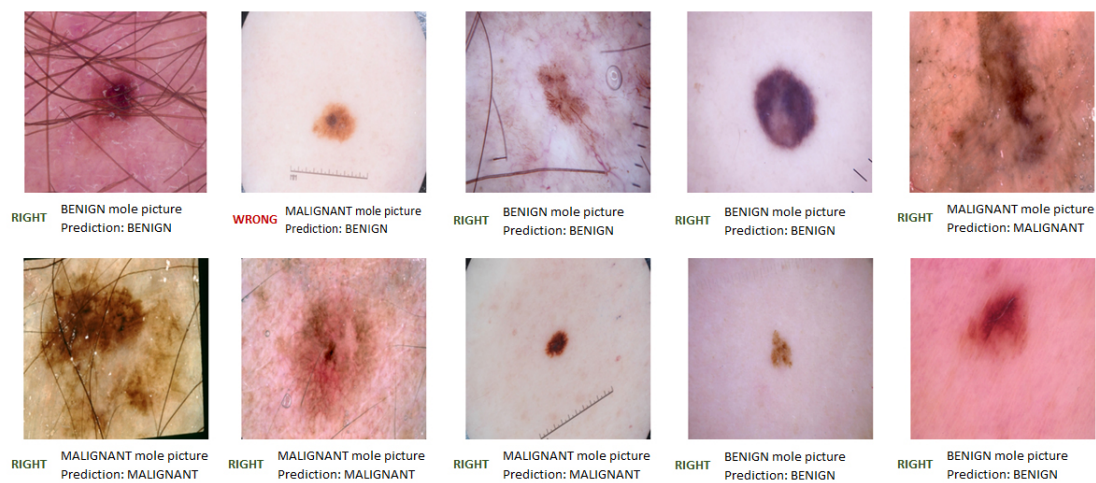
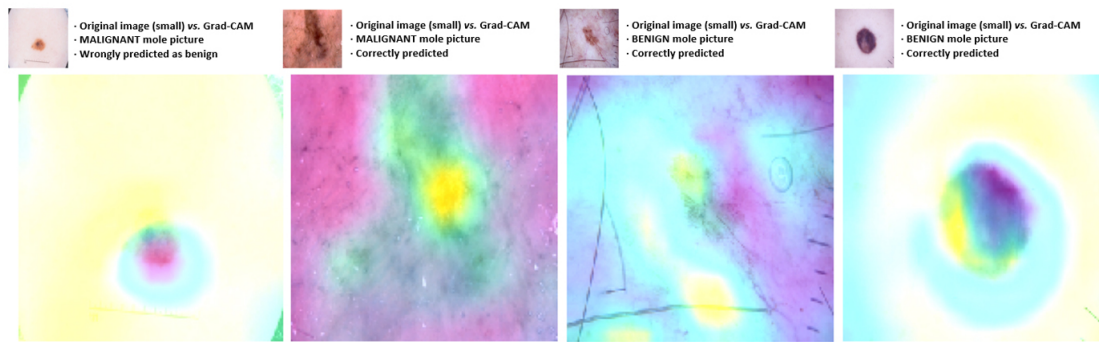


Figure 4.9: Prediction results for demonstration images.

The overall accuracy score of the model VB is 0.8632, or 86.32%, implying that the model correctly predicted the classes for 86.3% of all cases in the dataset.

In conclusion, the refinements we implemented in our model have significantly enhanced both its interpretability and performance. The use of visual explanation techniques, such as Grad-CAM (as depicted in Figure 4.10) and Integrated Gradients (as demonstrated in Figure 4.11), provided valuable insight into the model's decision-making process. They confirmed that our updated model VB now focuses more accurately on the relevant features in the image data. Further validation of our model's improvement comes from the performance metrics, which showed marked enhancements in precision, recall, and accuracy.



Figure 4.10: Grad-CAM - 2<sup>nd</sup> trial - VB.

These results confirmed our expectations and underscore our belief that the adjustments made to the model have been beneficial. As we move forward, we will continue to iterate on and improve this model, using the insights gained from these techniques as a roadmap for future updates.

Meanwhile, we experimented with leveraging pretrained models, specifically VGG-16<sup>23</sup> and Inception-ResNet-v2<sup>24</sup>, to further improve our model's performance. We will present the process and results of these modifications in the subsequent section.

Full Integrated Gradients and Grad-CAM results (for all the ten selected images) can be seen in Appendix A.

#### 4.1.5 Additional Pre-Trained Models

Modern machine learning has introduced a plethora of pre-trained models, each offering unique advantages. Among these, VGG-16 and Inception-ResNet-v2 stand out for their superior performance in image classification tasks. We decided to experiment with these models to explore potential improvements in our original model's performance.

<sup>23</sup>VGG-16 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition". The model achieves 92.7% top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes. VGG-16 refers to a VGG model with 16 layers that include convolutional layers, fully connected layers, and pooling layers. The key aspect of VGG-16 is the use of multiple convolutional layers before each max pooling operation, which was a departure from the architectures of previous models that typically only had a single convolutional layer in these spots. This enabled the model to learn more complex features, and paved the way for future network architectures (source: Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition).

<sup>24</sup>Inception-ResNet-v2 is a convolutional neural network that is 164 layers deep, providing a high level of accuracy for tasks such as image classification. This model is a combination of the Inception architecture and the ResNet architecture, developed by Christian Szegedy et al. from Google Inc. The architecture utilizes the concept of inception modules, where filters of different sizes are applied to the same input, and the results are concatenated, allowing the model to capture information at various scales. The network also uses residual connections, which help to mitigate the vanishing gradient problem during training, particularly in deeper networks. The combination of these two architectures allows Inception-ResNet-v2 to achieve high performance with a lower computational cost compared to similar models (source: Szegedy, C., Ioffe, S., Vanhoucke, V., & Alemi, A. (2017). Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning).

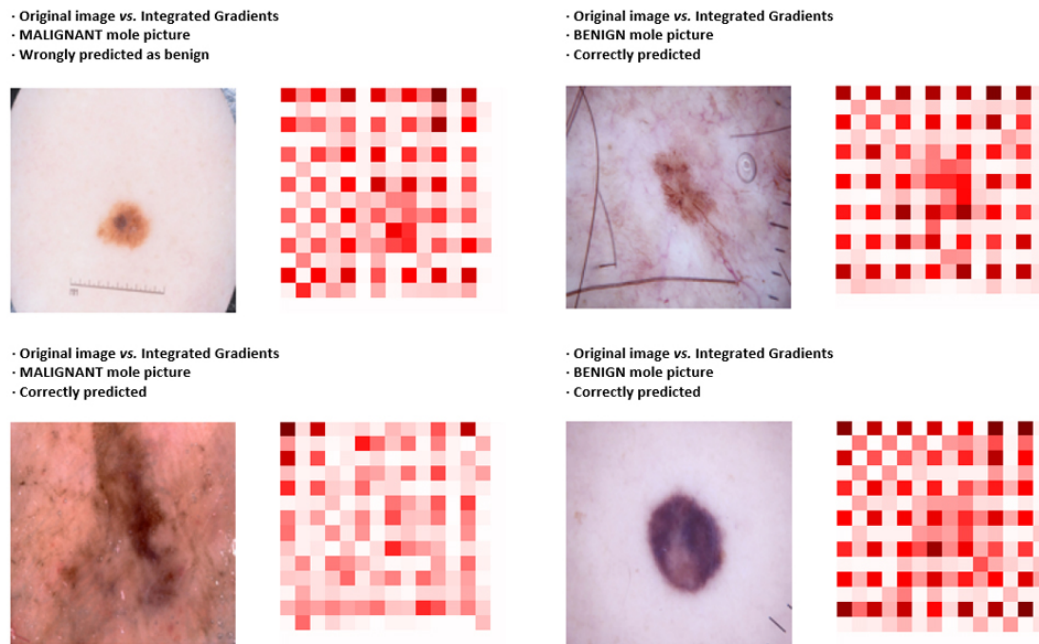


Figure 4.11: Integrated Gradients (seismic colormap) - 2<sup>nd</sup> trial - VB.

#### 4.1.5.1 VGG-16 Pre-Trained Model

We first adapted our model to use VGG-16, a model pre-trained on the ImageNet database. After loading the VGG-16 model and excluding its top layers, we constructed a sequential model (Figure 4.16). This model incorporates a base layer for transfer learning, followed by a Flatten layer, two Dense layers (256 and 128 nodes), featuring ReLU activations and BatchNormalization. It also includes two Dropout layers at a rate of 0.5, and finally, a single node Dense layer with a sigmoid activation function for binary classification.

We froze the VGG-16 base model layers to ensure that their weights were not updated during training. The model was compiled with an Adam optimizer, binary cross-entropy loss function, and accuracy as the metric. Two callbacks, EarlyStopping and ReduceLROnPlateau, optimized the training process.

Training this model took approximately 1475 seconds. Its accuracy score of 0.836 suggests an improvement over the original model but fell short of the modified models' performance (VA and VB). The confusion matrix (Table 4.4) and classification report reflected satisfying results.

Table 4.4: Confusion Matrix - 3<sup>rd</sup> trial – VGG-16.

		Predicted	
		Negative	Positive
Actual	Negative	276	83
	Positive	25	274



- Accuracy: 83.59
- Benign (Precision: 0.92, Recall: 0.77, F1-score: 0.84)
- Malignant (Precision: 0.77, Recall: 0.92, F1-score: 0.84)

Grad-CAM and Integrated Gradients results for some selected images can be seen in Figures 4.12 and 4.13 respectively.

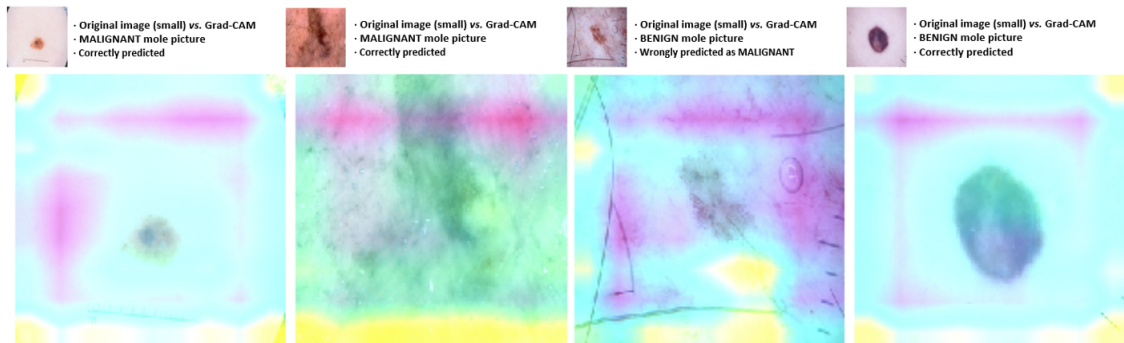


Figure 4.12: Grad-CAM - 3<sup>rd</sup> trial - VGG-16.

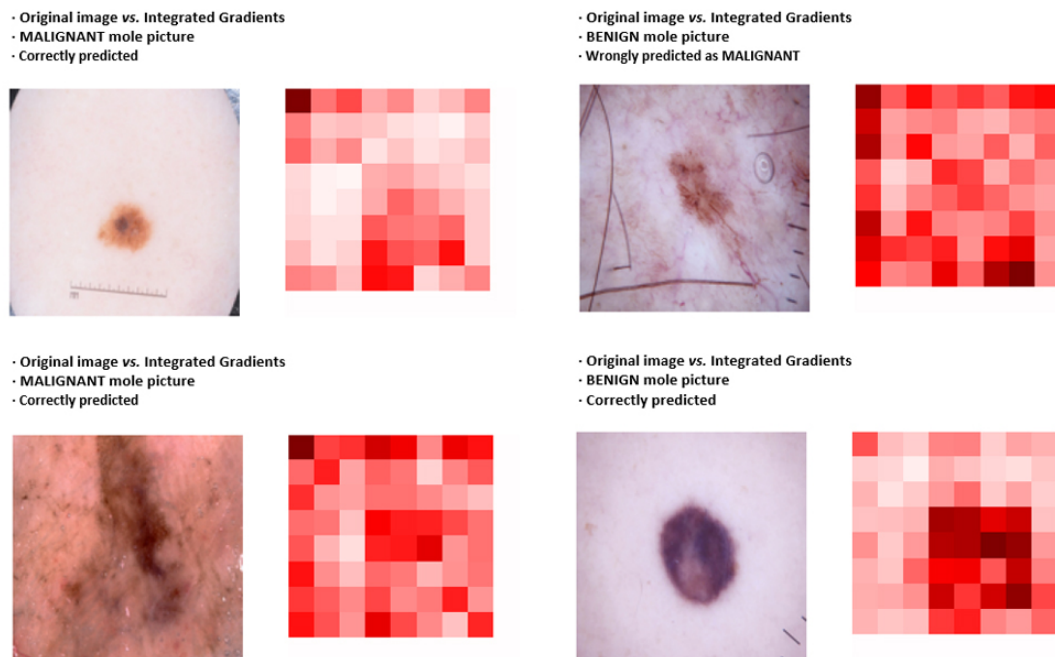


Figure 4.13: Integrated Gradients (seismic colormap) - 3<sup>rd</sup> trial - VGG-16.

#### 4.1.5.2 Inception-ResNet-v2 Pre-Trained Model

Next, we incorporated the Inception-ResNet-v2 model into our framework. Following a similar approach as with the VGG-16 model, we loaded the Inception-ResNet-v2 (excluding its top layers),

created a sequential model comprising base, Flatten, Dense, BatchNormalization, and Dropout layers (Figure 4.17). The model was then compiled and trained using the same specifications as the VGG-16 model.

Training this model took approximately 1152 seconds, slightly less than the VGG-16 model. However, the accuracy dropped slightly to 0.798. Despite the slightly lower accuracy, the precision and recall metrics from the classification report suggest that the model was able to balance the prediction for both classes, as shown in Table 4.5.

Table 4.5: Confusion Matrix - 4<sup>th</sup> trial - Inception-ResNet-v2.

		Predicted	
		Negative	Positive
Actual	Negative	316	43
	Positive	90	209

- Accuracy: 79.78
- Benign (Precision: 0.78, Recall: 0.88, F1-score: 0.83)
- Malignant (Precision: 0.83, Recall: 0.70, F1-score: 0.76)

Grad-CAM and Integrated Gradients results for some selected images can be seen in Figures 4.14 and 4.15 respectively.

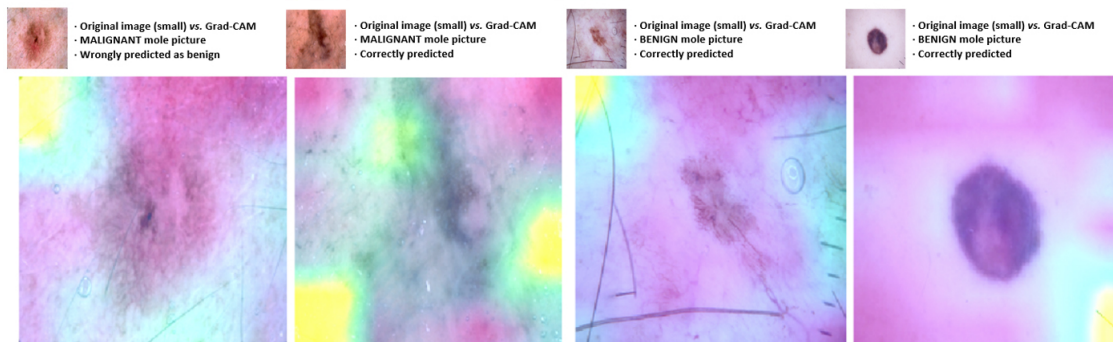


Figure 4.14: Grad-CAM - 4<sup>th</sup> trial - Inception-ResNet-v2.

Considering these observations, both VGG-16 and Inception-ResNet-v2 showed potential as upgrades to the original model. However, due to their higher computational demand, one might prefer our revised model for similar performance with reduced processing time.

This concludes the model evaluation and performance phase of our analysis. In the next section, we will discuss the potential implications of our findings and future directions for this research.

The models references and methods can be seen in Figure 4.18.

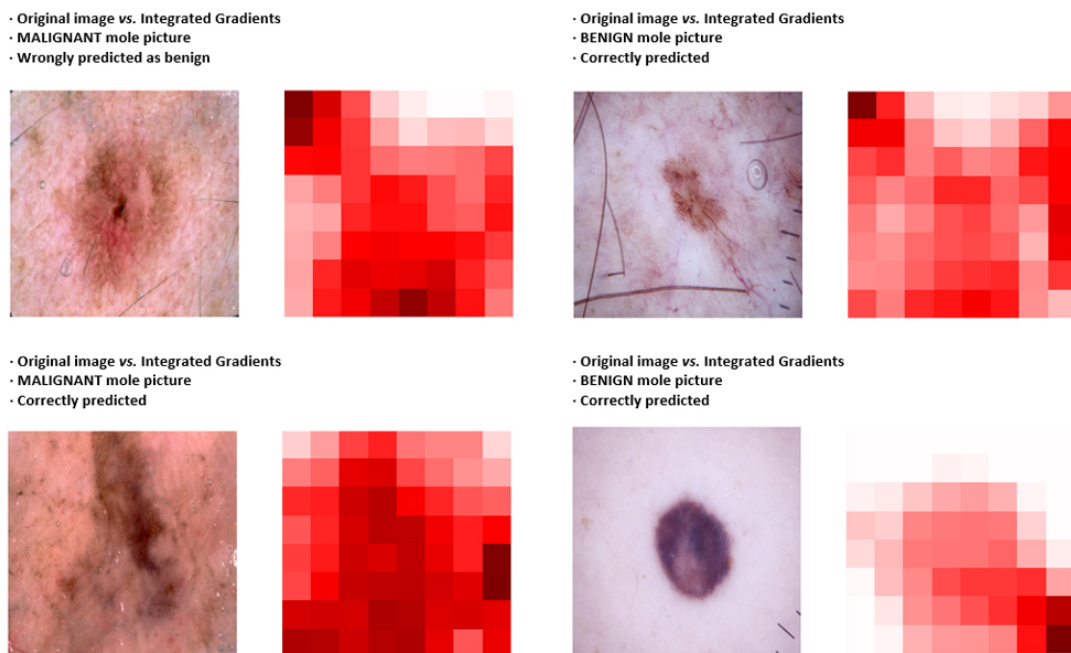


Figure 4.15: Integrated Gradients (seismic colormap) - 4<sup>th</sup> trial - Inception-ResNet-v2.

#### 4.1.6 Summary of Findings

This study presents a potential role of CNN in dermatology, particularly for melanoma detection. Our results demonstrated the effectiveness of our custom CNN architecture, which significantly improved after fine-tuning and implementing transfer learning strategies.

VGG-16 and Inception-ResNet-v2 models showed competitive performance, underscoring the potential of advanced architectures in extracting robust features from dermoscopic images. The trade-off between computational time and accuracy suggests the importance of considering specific task requirements when choosing an architecture.

A summary of model performances, including base model, total training time, accuracy, and F1-Score on test data, is presented in Table 4.6 for easy reference.

Model	Processing Time (Sec.)	Test Accuracy (%)	F1-Score (W-A)
CNN Original Model	0167.43	77.05	0.77
CNN Updated Model VA	5608.20	85.26	0.85
CNN Updated Model VB	0562.85	86.32	0.86
VGG-16 CNN	1474.89	83.59	0.84
Inception-ResNet-v2 CNN	1152.39	79.79	0.80

Table 4.6: Summary of model performances.

Considering these results, the updated model VB, which is a fine-tuned version of the original model with additional data augmentation and regularized dropout, achieved the highest accuracy. However, the primary advantage of using a pre-trained model extends beyond accuracy improvement, as these models can learn complex patterns from large datasets.

Model: "sequential"

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 4, 4, 512)	14714688
flatten (Flatten)	(None, 8192)	0
dense (Dense)	(None, 256)	2097408
batch_normalization (Batch Normalization)	(None, 256)	1024
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 128)	32896
batch_normalization_1 (Batch Normalization)	(None, 128)	512
dropout_1 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 1)	129

=====  
Total params: 16,846,657  
Trainable params: 2,131,201  
Non-trainable params: 14,715,456

Figure 4.16: Model architecture - 3<sup>rd</sup> trial - VGG-16.

Model: "sequential"

Layer (type)	Output Shape	Param #
inception_resnet_v2 (Functional)	(None, 8, 8, 1536)	54336736
flatten (Flatten)	(None, 98304)	0
dense (Dense)	(None, 256)	25166080
batch_normalization_203 (Batch Normalization)	(None, 256)	1024
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 128)	32896
batch_normalization_204 (Batch Normalization)	(None, 128)	512
dropout_1 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 1)	129

=====  
Total params: 79,537,377  
Trainable params: 25,199,873  
Non-trainable params: 54,337,504

Figure 4.17: Model architecture - 4<sup>th</sup> trial - Inception-ResNet-v2.

The VGG-16 model also performed very well. This model is notable for its simplicity and effectiveness, and it was particularly competitive in this task. The relatively high accuracy of this model demonstrates the effectiveness of transfer learning, where a model pre-trained on a large dataset (like ImageNet) can be adapted to a related task with a smaller dataset. However, the choice of the best model can depend on the specific requirements of a project.

As we advance, there are still various aspects to consider for future work. A natural next step is the incorporation of additional pre-trained models, such as EfficientNets or ResNets, to further explore the space of possible solutions. Moreover, expanding our dataset, implementing data augmentation techniques, or tuning the hyperparameters may also lead to improved performance.

The codes (written in Python) used in this experiment are detailed in Appendix B.

## 4.2 Results of Experiment #2: Handwriting Detection

Handwritten recognition represents one of the most challenging tasks within the machine learning and pattern recognition disciplines. With the variability and ambiguity inherent to human handwriting, traditional computational approaches often struggle to maintain high levels of accuracy (Wang et al., 2009). Different individuals, and even the same individual at different times, can produce significantly diverse handwriting styles. This variability poses a considerable obstacle for computational models trying to identify specific characters or text strings from raw handwritten input.

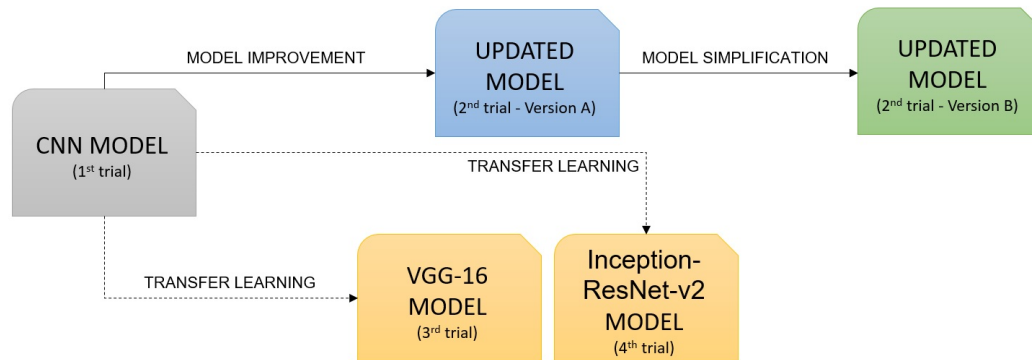


Figure 4.18: Models references and methods - 1<sup>st</sup> experiment.

The problem of handwritten text recognition (HTR)<sup>25</sup> has been a focal point of machine learning research for an extended period, with the widely recognized MNIST dataset<sup>26</sup> for handwritten digits underpinning many initial explorations. HTR holds paramount importance in various applications that necessitate the digitization of documents, spanning sectors like healthcare, insurance, and banking. Despite the existence of software applications that have already incorporated HTR functionalities, perfection in recognizing diverse handwritten styles remains elusive. This has resulted in continued and vigorous research activities within the field<sup>27</sup>.

However, this area has seen substantial progress over recent years despite the inherent difficulties. Much of this advancement can be attributed to the application of MLP models. These artificial neural networks have demonstrated significant promise in tackling the challenges of handwritten recognition. They have been particularly effective in recognizing individual characters, representing one of the task's most demanding aspects (Wang et al., 2009; Jain and Nayak, 2011).

This chapter section delves into the utilization of MLP models in the context of handwriting recognition, bringing into sharp focus the trials encountered, methodologies employed, and the wealth of insights unearthed from the experiment. Furthermore, it extends the exploration to examine the functionality of the Google Vision API<sup>28</sup>.

<sup>25</sup>Handwritten Text Recognition (HTR) is a field of research in pattern recognition, artificial intelligence and machine learning, which aims to develop techniques and systems capable of recognizing and interpreting handwritten text from a variety of sources such as historical documents, personal notes, or addresses on mail. HTR involves several stages including preprocessing, feature extraction, and recognition (source: Plamondon, R., & Srihari, S. N. (2000). Online and off-line handwriting recognition: a comprehensive survey.).

<sup>26</sup>The MNIST (Modified National Institute of Standards and Technology) dataset is a large database of handwritten digits that is commonly used for training and testing in the field of machine learning. The dataset contains 60,000 training images and 10,000 testing images, each of which is a 28x28 pixel grayscale image of a digit between 0 and 9. (source: Deng, L. (2012). The mnist database of handwritten digit images for machine learning research. IEEE Signal Processing Magazine, 29(6), 141–142.).

<sup>27</sup>University of Toronto Machine Intelligence Team (UTMIST) Project by Justin Tran, Fernando Assad, Kiara Chong, Armaan Lalani, and Eeshan Narula. The article can be accessed at: <https://utorontomist.medium.com/handwriting-recognition-using-deep-learning-14ec078872b0>

<sup>28</sup>Google Cloud Vision API is a part of Google Cloud's AI Hub, a suite of machine learning tools developed by Google (source: <https://cloud.google.com/vision/docs>).



### 4.2.1 Presentation of Data

Our search for a suitable dataset composed of Portuguese handwritten texts posed a formidable challenge. However, viable alternatives in English were identified and utilized. The IAM Handwriting dataset<sup>29</sup> served as an invaluable resource, guiding our data preparation efforts. Concurrently, the MNIST<sup>26</sup> and GNHK<sup>30</sup> datasets offered additional insight. For examples of these datasets, please refer to Figures 4.19<sup>29</sup>, 4.20<sup>30</sup>, and 4.21<sup>26</sup>.

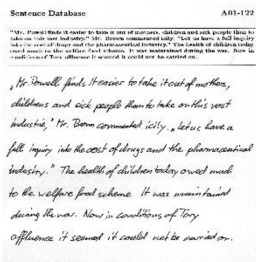


Figure 4.19: IAM dataset example.

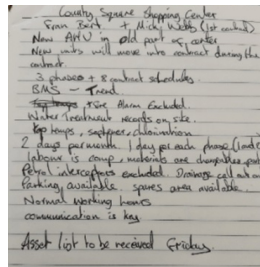


Figure 4.20: GNHK dataset example.

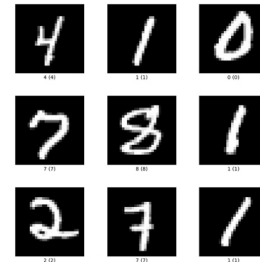


Figure 4.21: MNIST dataset examples.

Recognizing the constraints imposed by processing power, we concluded that a massive image collection was not required. Instead, we adopted a strategic approach. Colleagues from the FEUP's Master in Science and Data Engineering program, as well as a few friends, were engaged to provide samples of their handwritten text. In addition, we conducted a rigorous online search to obtain additional handwritten texts, enriching our dataset. The meticulous gathering process yielded a carefully curated set of fifty images, deemed suitable for the experiment's scope. The Figure 4.22 shows five examples of our dataset<sup>31</sup> with the respective texts<sup>32</sup> extracted to the .txt file.

To establish a rigorous baseline for our metrics, the text in these images was manually transcribed and saved in .txt files<sup>33</sup>. This painstaking groundwork gave us a reliable benchmark, setting the stage for ensuing analyses. This process encapsulates our pragmatic approach to data acquisition for the handwritten text recognition endeavor.

<sup>29</sup>The IAM Handwriting Database (IAM-HDB) is a comprehensive, high-quality dataset of handwritten English text used frequently in training and testing models for handwriting recognition. Produced by the University of Bern in Switzerland, the dataset contains forms of unconstrained handwritten text, which are segmented into lines, words, and individual characters (source: Marti, U. V., & Bunke, H. (2002). The IAM-database: an English sentence database for offline handwriting recognition).

<sup>30</sup>GoodNotes Handwriting Kollection (GNHK) dataset includes unconstrained camera-captured images of English handwritten text sourced from different regions around the world (Source: Lee, Alex W. C.; Chung, Jonathan; Lee, Marco (2021). GNHK: A Dataset for English Handwriting in the Wild. Conference: International Conference of Document Analysis and Recognition (ICDAR). <https://www.goodnotes.com/gnhk>).

<sup>31</sup>Figure 4.22 sources: FILE 30: Gabriel Lucca at [https://pt.wikipedia.org/wiki/É\\_verdade\\_esse\\_bilete](https://pt.wikipedia.org/wiki/É_verdade_esse_bilete). FILES 06, 18, 27 and 44: friends contribution.

<sup>32</sup>Any errors in spelling, punctuation, etc. have been retained.

<sup>33</sup>A .txt file, often called a text file, is a type of computer file structured as a sequence of lines of electronic text containing plain text (source: Microsoft Support).

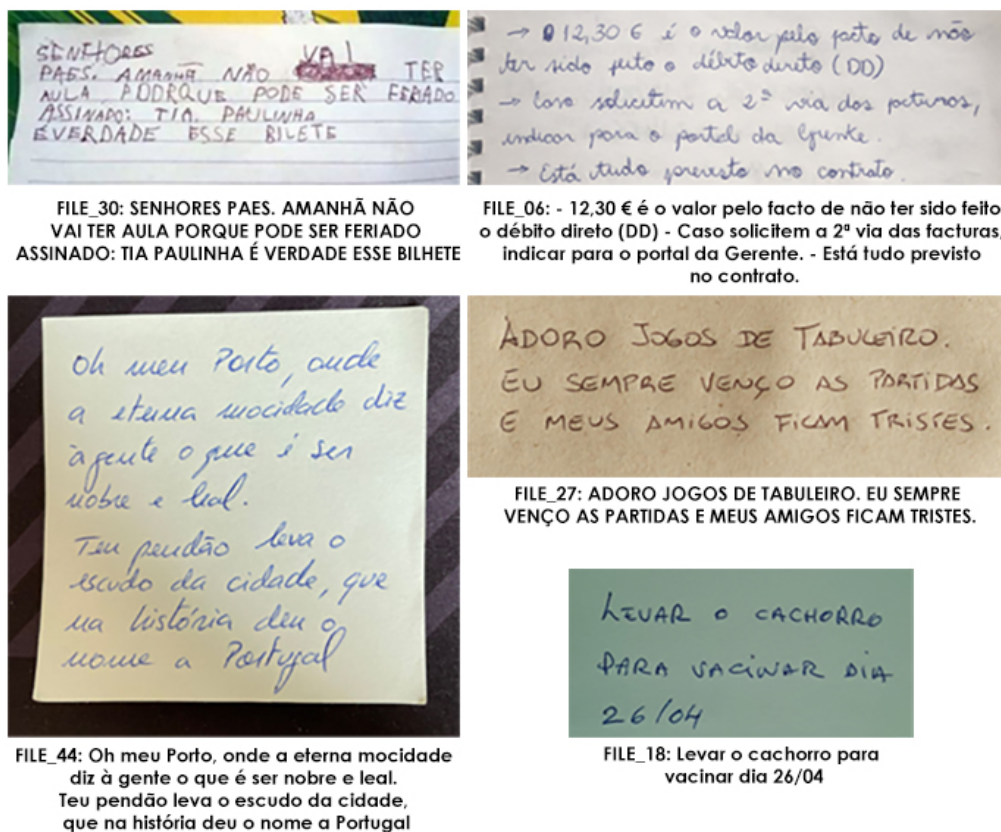


Figure 4.22: Examples of our customized dataset.

### 4.2.2 Performance of the MLP model

Our research began with the application of PyTesseract<sup>34</sup> OCR<sup>35</sup> for the recognition of handwritten text. As predicted, the results fell short of satisfactory. PyTesseract OCR, while robust for recognizing printed text, faced substantial challenges with the variability and fluidity inherent in handwriting. Elements such as diverse handwriting styles, variable letter formation and spacing, along with the nuances of individual cursive styles, often overextended the capabilities of PyTesseract OCR's detection mechanisms.

To extract more data from the images, we attempted to correct skewness and preprocess the images. While this yielded some improvements, the results were limited to isolated words, letters, numbers, and punctuation as can be seen in Figure 4.23.

Subsequently, we aimed to construct an offline Handwritten Text Recognition (HTR) system

<sup>34</sup>PyTesseract is a Python library that uses Optical Character Recognition (OCR) to convert images into text. It is a wrapper for Google's Tesseract-OCR Engine, which is considered one of the most accurate open-source OCR engines currently available (source: <https://github.com/madmaze/pytesseract>).

<sup>35</sup>Optical Character Recognition (OCR) is a technology used to convert different types of documents, such as scanned paper documents, PDF files, or images captured by a digital camera, into editable and searchable data. OCR works by analyzing the shapes and patterns of the characters in the scanned image or document, then converting them into text characters that a computer can understand and manipulate (source: <https://www.ibm.com/cloud/blog/optical-character-recognition>).

```

----- Recognized Text 6 -----
Re o NR E RANDE CAIR ERRADOS CREDO CURE GR ROUND

----- Recognized Text 7 -----
SH Da VS E, e sndigias RR a ea evo SPC ESA aa eo js E PR REA ea e Pa a a us Pop UE: AUT ar GERIR UDN ERR RT CARAS ES PA RES DO NE RIA,

----- Recognized Text 8 -----
e CAN O RR ae E RAND SUDO GO A A ES gm Be SERA SPT io RR SERIE RE E o RO gs | OCO cOTE DRDS da YO ACC SS ME e E CR rar AG AESA Eu AA as

----- Recognized Text 9 -----
REAR O RD RRSRES RGE E E SS e E ONA E a Rc

----- Recognized Text 10 -----
ago RS EN E E NE RE SE e RC ERRA NAO ci Ea a RO SEAT O pra = 4 DE RIR a E a No GR NE OITO VS Dee RR do Ago D a E RAS RREO RRSR VETA RED

----- Recognized Text 11 -----
S E - 8 1 Eron ; E E ER Jada E E |

```

Figure 4.23: Some texts extracted from pictures using PyTesseract OCR after preprocessing.

to transcribe the text contained within our image set. We built a Neural Network trained on word-images from the IAM dataset. Our model architecture, inspired by Harald Scheidl's 2018 article on Towards Data Science<sup>36</sup>, comprised several layers:

- CNN layers for feature extraction, processing input images and passing the learned features to the RNN layers.
- RNN layers for sequence processing, accepting features from the CNN layers and extracting sequential information from the data.
- CTC loss and decoder: The Connectionist Temporal Classification (CTC) loss is utilized for training the RNN layers, while the CTC decoder converts the RNN output into a text sequence.

The aim was to extend the scope of our experiment, initially designed to read a single word, to include the entire image. Several sources inspired this endeavor, including Scheidl's article as well as 'Handwriting Recognition Using Deep Learning' (2022) by the University of Toronto Machine Intelligence Team (UTMIST)<sup>37</sup>, and 'How to Easily do Handwriting Recognition using Machine Learning' (2022) by Anil Chandra Naidu Matcha on the Nanonets website<sup>38</sup>.

Despite our efforts, we faced significant challenges in constructing a model of such complexity. As seen in Figure 4.24, even with simple models, accurately recognizing randomly selected words in either Portuguese or English was problematic<sup>39</sup>. We concluded that training an MLP to detect handwritten text is a task fraught with complexity. The considerable computational requirements of processing the IAM dataset and the time cost due to convolutional layers necessary for handling two-dimensional input data made this method both data and time-intensive.

As such, we shifted our focus to explore pre-built solutions, namely, Google Vision. This pre-trained machine learning model by Google has been fine-tuned to manage a broad spectrum

<sup>36</sup>Scheidl, H. (2018). Build a Handwritten Text Recognition System using TensorFlow. Towards Data Science. <https://towardsdatascience.com/build-a-handwritten-text-recognition-system-using-tensorflow-2326a3487cd5>

<sup>37</sup>Tran, J., Assad, F., Chong, K., Lalani, A., & Narula, E. (2022). Handwriting Recognition Using Deep Learning. Medium. <https://utorontomist.medium.com/handwriting-recognition-using-deep-learning-14ec078872b0>

<sup>38</sup>Matcha, A.C.N. (2022). How to easily do Handwriting Recognition using Machine Learning. Nanonets. <https://nanonets.com/blog/handwritten-character-recognition/>

<sup>39</sup>The model, created by Harald Scheidl, can be tested at: [https://githubharald.github.io/text\\_reader.html](https://githubharald.github.io/text_reader.html)



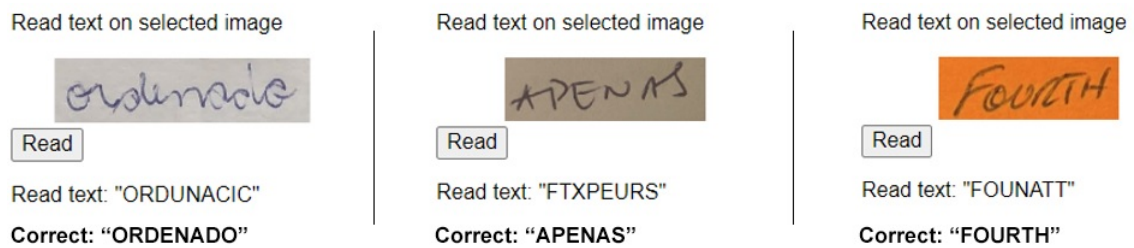


Figure 4.24: Examples of word recognition using offline HTR.

of tasks, including handwriting recognition. The tool has been trained on an extensive amount of data, far surpassing what an individual team could collect and process independently.

The appealing features of Google Vision extend to its ability to recognize text in over a thousand languages and various scripts. Its OCR technology can detect handwriting in images and documents with high accuracy, even against complex backgrounds. This blend of versatility, broad applicability, and superior performance render Google Vision a viable solution for handwriting recognition tasks.

Consequently, our focus pivoted to understanding and utilizing Google Vision and its API, evaluating its results, and exploring opportunities for improvement.

#### 4.2.2.1 Google Vision API

The Google Cloud Vision Application Programming Interface<sup>40</sup> (API) constitutes a robust tool embedded within the Google Cloud Platform, offering machine learning capabilities geared explicitly toward image and video interpretation. This API exhibits an extensive range of visual cognition abilities that encompass object detection, facial recognition, and the identification of significant landmarks or brand logos. It also employs OCR to analyze and comprehend text found within visual data (What is Google Cloud Vision?, 2022)<sup>41</sup>. More sophisticated features offered by the tool include detecting explicit content, sentiment analysis from facial expressions, and even identifying notable product insignias. As a tool, it presents substantial utility for developers who aim to comprehend and scrutinize the vast scope of visual data accessible via the internet.

However, like numerous machine learning frameworks, the Google Cloud Vision API may be characterized as a "black box"<sup>42</sup>. This term refers to systems wherein the internal mechanisms facilitating the formulation of results are not straightforward or easily comprehensible, given that they are established on intricate DL models instead of explicit programming. Despite its complex underpinnings, a high-level understanding of the API's functionality is attainable: the API initiates its process by performing a pre-processing procedure on the provided image, subsequently partitioning it into various sections. These divided sections undergo an analysis through a DL model

<sup>40</sup>Source: Cloud Vision documentation, 2023. <https://cloud.google.com/vision/docs>

<sup>41</sup>Source: What is Google Cloud Vision? (2022, March 21). <https://www.resourcespace.com/blog/what-is-google-vision>

<sup>42</sup>See Chapter 1 - Introduction.

trained on a significant volume of data. This model applies its learning to generate inferences about new data. The resultant insights are consequently relayed back to the user (Cheng, 2022).

In our preliminary exploration of Google Vision, we applied the samples from Figure 4.24 to the demo API <sup>43</sup>, the output of which is documented in Figure 4.25. We observed a notable enhancement in the results. Further, we conducted an identical test using the full-sized images, which allowed us to emphasize the methodology behind the detection process, as depicted in Figures 4.26 and 4.27. The first two samples form a part of the dataset employed within this study. The inclusion of an English handwritten text example serves to illustrate the tool's ability to recognize text across languages, even though it misidentified some characters from the Cyrillic alphabet in a Portuguese text. This cross-lingual capability is indicative of the tool's versatility and wide-ranging applicability.



Figure 4.25: Examples of word recognition using Google Vision API demo.

Interfacing with the Google Cloud Vision API, from a user's perspective, remains quite intuitive, notwithstanding the complexity of its operations. These steps provide a simplified understanding of how the Google Cloud Vision API processes and analyzes an image.

- **Image Upload:** The user uploads an image to the Vision API through an HTTP request. This image can be a URL or direct file data.
- **Pre-processing:** The API performs initial pre-processing tasks on the image, including normalizing the image size, adjusting the color scale, and more. These tasks make the image easier for the model to analyze.
- **Image Segmentation:** The processed image is then segmented into different regions. Each of these regions represents a different part of the image that will be analyzed individually.

<sup>43</sup>The Google Vision API can be tested at <https://cloud.google.com/vision/docs/drag-and-drop>



Figure 4.26: Examples of full text recognition using Google Vision API demo (1 of 2).

- **Feature Extraction:** In each of these segments, the API then performs feature extraction. This involves identifying and analyzing various characteristics like color, texture, and shapes. The combination of these features forms a unique signature that the model can use to identify and categorize elements in the image.
- **Inference:** Using the extracted features, the deep learning model, which has been trained on large amounts of data, makes predictions about what the image represents. This could include identifying objects, faces, logos, and more.
- **Response:** Finally, the API sends back a response containing the results of its analysis. This can include labels for identified objects, text recognized through OCR, sentiment analysis on faces, and more.

The implementation and utilization of the Google Cloud Vision API necessitate not only adherence to technical prerequisites but also compliance with administrative conditions. A primary requirement for the API's deployment is the acquisition of a valid API key, a unique identifier that authorizes and tracks API usage. This key facilitates interaction with the API and is integral to maintaining the security and integrity of the data being processed and the underlying machine-learning models.

Furthermore, it is essential to note that the Google Cloud Vision API is a paid service<sup>44</sup>. Costs associated with the service are determined by the volume and type of requests made to the API. Google Cloud provides a detailed pricing guide informing users about the costs incurred per thousand units of analysis for varying types of visual data. Potential users are advised to review this pricing guide in detail to ascertain the potential costs associated with their intended API usage.

<sup>44</sup>Pricing for using the Google Evion API can be found at <https://cloud.google.com/vision/pricing/>

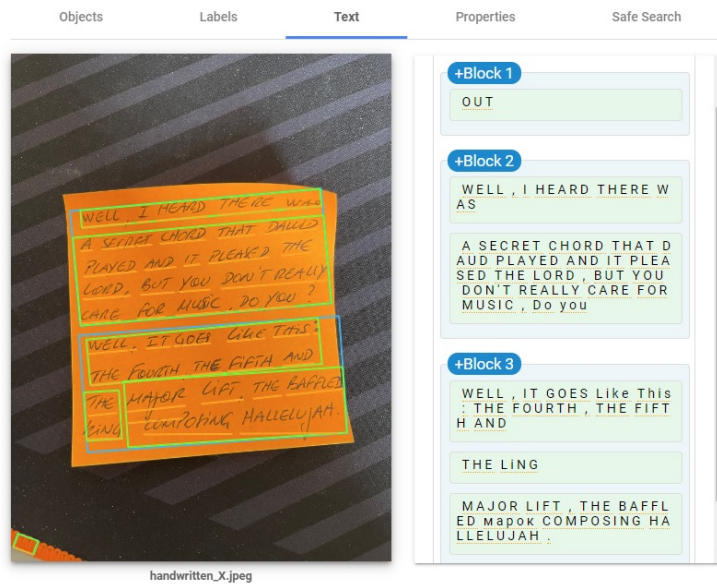


Figure 4.27: Examples of full text recognition using Google Vision API demo (2 of 2).

The provision of a budgeting tool by Google Cloud also assists users in controlling costs by setting custom budgets and alerts to prevent inadvertent excessive usage.

The combination of these conditions elucidates the necessity of understanding the Google Cloud Vision API's technical capabilities and recognizing the administrative obligations accompanying its use. By adhering to these requirements, users can effectively leverage the power of this machine-learning tool while maintaining control over cost and usage.

It is noteworthy to mention the free API credits provided by Google as part of their initiative to encourage the exploration and understanding of their Google Cloud Vision API. These credits were utilized for the execution of the experiments and analyses integral to this thesis. The provision of these free credits, available to anyone eager to explore the API's capabilities, greatly facilitated our exploration and testing of its features. It also significantly contributed to the insights and findings detailed in this research. We appreciate this initiative as it promotes accessibility and provides an open avenue for both academics and enthusiasts to delve into the practical implications of artificial intelligence and machine learning technologies.

#### 4.2.2.2 Implementing Google API Vision

Following our decision to utilize Google Vision, we evaluated its performance. To do this, we requested an API from Google Cloud and devised Python code to execute OCR on our compilation of fifty images. This code, tailored to recognize Portuguese text, was instrumental in documenting the resulting text for subsequent analyses.

The first step in our process involved the initialization of essential libraries. For file and directory operations and handling regular expressions, we leveraged the capabilities of the `os` and `re` libraries. We employed the tools offered by the `google.cloud` to facilitate image recognition and analysis. `vision_v1` library.

To maintain the integrity of our analysis, we ensured that only valid image files (sporting 'jpg', 'jpeg', and 'png' extensions) from a designated directory were processed. Subsequently, a function was employed to read and prime an image file for analysis. OCR was performed on the loaded image through a dedicated function. This function used the "document\_text\_detection" method from the Google Cloud Vision API, taking the image and a language hint (in this instance, 'pt' for Portuguese) as inputs. A filter was subsequently applied to cleanse the input text, purging any non-Portuguese characters.

Finally, in the main segment of the code, functions were invoked sequentially to execute OCR on the entire batch of images. The recognized text from each image was displayed and archived in a text file. All line breaks were omitted.

Before launching the code, it was crucial to establish authentication for the Google Cloud Vision API and specify the paths to the image and text output directories. These components had to be integrated to ensure the seamless functioning of the OCR process.

```

--- IMAGE 27 ---
LABEL: ADORO JOGOS DE TABULEIRO. EU SEMPRE VENÇO AS PARTIDAS E MEUS AMIGOS FICAM TRISTES.
Trial 2.: ADORO JOGOS DE TABULEIRO. EU SEMPRE VENÇO AS PARTIDAS E MEUS AMIGOS FICAM TRISTES.

--- IMAGE 28 ---
LABEL: Beijjos! me liga
Trial 2.: Beijjos me liga B 1

--- IMAGE 29 ---
LABEL: Querida "fada do dente" me dá o meu dinheiro! ATT: Julia DENTE - (EMBAIXO DA FITA)
Trial 2.: Querido fado- do dente me dá o meu dinheiro ATT: Julia DENTE (EMBAIXO DA FITA)

--- IMAGE 30 ---
LABEL: SENHORES PAES. AMANHÃ NÃO VAI TER AULA PORQUE PODE SER FERIADO ASSINADO: TIA PAULINHA É VERDADE ESSE BILHETE
Trial 2.: SENIORES PAES. AMANHÃ NÃO TER AULA ADDRQUE PODE SER FERIADO ASSINADO: TIA. PAULINHA EVERDADE ESSE BILETE

--- IMAGE 31 ---
LABEL: Não falei contigo com medo que os montes e vales que me achas, caissem a teus pés. Saudade é o ar que vou sugando e aceitando, como fruto de verão dos jardins do teu beijo
Trial 2.: Não falei contigo miedo. com que os montes e vales que me achas, caissem a teur pés. Saudade é o an que vou e aceitando, como sigando fruto de verão dos jaudius do teu beija 0000000000

--- IMAGE 32 ---
LABEL: Heróis do mar, nobre povo, nação valente, imortal. Levantai hoje de novo o esplendor de Portugal.
Trial 2.: Herois do mar, nobre povo, valente, imortal. Mação levantai ho hoje esplendor de Podzysel. de novo

```

Figure 4.28: Texts extracted from images 27 to 32 using Google Vision API.

This implementation exemplifies a powerful, scalable method to execute text recognition on a substantial dataset of images, thus enabling a comprehensive analysis of the contained content. Figure 4.28 presents the results obtained in juxtaposition with the image labels for nine items in our dataset (images 27 to 32). Figure 4.29 showcases the images that were loaded.





Figure 4.29: Dataset images 27 to 32.

#### 4.2.2.3 Model Evaluation and Performance

In the domain of Natural Language Processing tasks, traditional metrics like Accuracy, Precision, Recall, and F1 Score often fall short in providing a comprehensive assessment of our system's performance. The inherent complexities of human language, such as nuanced meanings, contextual interpretations, and ambiguous expressions, limit the effectiveness of these metrics. Consequently, they may not accurately represent the true capabilities of our system, especially when it comes to text recognition. While accuracy, recall, and F1 score measure the model's ability to classify or predict outcomes based on input data, they are unsuitable for comparing text strings as they do not account for partial matches. For instance, comparing the words "ball" and "balls" would yield a complete mismatch with these metrics, despite the substantial similarity between the two words. To better evaluate the performance of our system when dealing with text-based data, we employ the Cosine Similarity measure within the context of TF-IDF<sup>45</sup> vectors. This measure calculates how similar the recognized text is to the image label (Chamblee, 2022). The Cosine Similarity computes the cosine of the angle between two vectors in a high-dimensional space, with each dimension representing a term from the text (Figure 4.30<sup>46</sup>) (Korstanje, 2020).

<sup>45</sup>Term Frequency-Inverse Document Frequency (TF-IDF) is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance of the word increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus. This helps to adjust for the fact that some words appear more frequently in general. TF-IDF can be successfully used for stop-words filtering in various subject fields including text summarization and classification (source: Rajaraman, A., & Ullman, J. D. (2011). Data Mining. In Mining of Massive Datasets).

<sup>46</sup>Source: <https://www.engati.com/glossary/cosine-similarity>, edited.

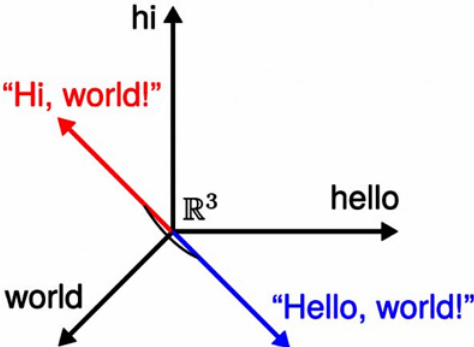
$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$


Figure 4.30: Cosine Similarity formula and visualization.

While other metrics, such as Levenshtein Distance provide value in some instances, we have focused exclusively on Cosine Similarity for several reasons. First, Cosine Similarity excels with longer documents and can capture similarities in the discussed topics. This property aligns well with our task of analyzing long-text handwritten images. Second, the normalization and weighting required by the Levenshtein Distance could introduce additional complexity and potential sources of error into our model evaluation. Lastly, the Cosine Similarity provides a natural way to account for the importance of different words in the text, enhancing our system's ability to distinguish between more and less relevant matches (Thada and Jaglan, 2013).

Table 4.7 summarizes the metrics mapped in this study, where they can be better used (Advantages), and why they were discarded (Disadvantages).

After running our code on a dataset of fifty images, we achieved a Cosine Similarity of 0.7205, where 1.0000 signifies a perfect fit and 0.0000 signifies no recognition at all<sup>47</sup>. In Table 4.8<sup>48</sup> we define the quality range established for this work.

Given the complexities and variability of handwritten text, this initial result is promising. The recognition of handwritten text is a remarkably challenging task, grappling with substantial variations and distortions in individual handwriting styles. Factors such as the quality and diversity of the handwritten data and the complexities of the words or phrases in question inherently influence our performance. While this serves as a solid baseline, we recognize the potential for further improvement. In the upcoming sections, we will explore strategies to enhance these results.

<sup>47</sup>We have tested our model with documents with the exact text and with blank documents and obtained the results of 1.000 and 0.0000, respectively, thus guaranteeing the functioning of the code.

<sup>48</sup>These interpretations of cosine similarity scores are generally accepted but can vary based on the specifics of a given application (Singh et al., 2014).

<b>Metric Selected</b>	<b>Advantages</b>	<b>Disadvantages</b>
Cosine Similarity (a)	Effective with longer documents. Adept at capturing overarching similarities in the topics discussed in the texts.	May not be as effective with very short documents or documents with very different vocabularies.
<b>Discarded Metrics</b>	<b>Advantages</b>	<b>Disadvantages</b>
Levenshtein Distance (b)	Measures minimum number of single-character edits. Sensitive to text length.	May not be optimal for comparisons involving extremely short or long texts.
Jaccard Similarity (c)	Less sensitive to text length. Effective measure when comparing documents with varying word counts.	Cannot capture similarities when similar words are used in different orders.
Dice Similarity (d)	Similar to Jaccard Similarity, but puts more emphasis on the intersection between the sets.	It may over-emphasize the importance of shared words.
Jaro-Winkler Distance (e)	Works well for short texts with small differences.	Might be less appropriate for lengthy or substantially different texts.
Source: (a) (Chamblee, 2022) (b) Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. In Soviet physics doklady (Vol. 10, No. 8, pp. 707-710). (c) (Uniqtech, 2020). (d) (Thada and Jaglan, 2013) (e) Cahyono, S. (2019). Comparison of document similarity measurements in scientific writing using Jaro-Winkler Distance method and Paragraph Vector method. IOP Conference Series: Materials Science and Engineering. 662. 052016. 10.1088/1757-899X/662/5/052016.		

Table 4.7: Metrics with their respective advantages and disadvantages.

### 4.2.3 Insights from Knowledge Elicitation

In our experimentation, we employed many preprocessing techniques to optimize the effectiveness of our OCR using Google Vision API. We explored six options for preprocessing images, each method being tested individually on the images and each employing various techniques ranging from grayscale conversion, noise reduction, skew correction, and morphological operations such as dilation and erosion. The goal of each preprocessing method was to enhance the OCR accuracy by improving the image quality and readability (Patel, 2014).

As can be seen in Figure 4.31, the preprocessing techniques led to noticeable differences in the output images. Specifically, the image preprocessed using Option A appears clearer than that with Options B and C. In Table 4.9, we summarize the preprocessing options and techniques employed for each one.

Once the images were preprocessed, we adopted several text extraction and error correction approaches. The first approach, utilized the Google Vision API for OCR and filtered for Portuguese characters. Subsequently, we extended this method to include a spelling check using the SpellChecker<sup>49</sup> library, which was set to Portuguese. This provided an additional layer of error

<sup>49</sup>The SpellChecker library in Python is a powerful text processing tool that helps to identify and correct spelling errors in strings of text. It is built on the PyEnchant library and uses a Levenshtein Distance algorithm to find permutations within an edit distance of 2 from the original word. It can also use frequency analysis to determine the most



Cosine Similarity Score	Quality Level
0.0000 - 0.2000	Very Poor (The documents share very few or no common terms)
0.2001 - 0.5000	Poor (The documents share few common terms)
0.5001 - 0.7000	Moderate (The documents share a decent amount of common terms)
0.7001 - 0.9000	Good (The documents share many common terms)
0.9001 - 1.0000	Excellent (The documents are very similar or identical)

Table 4.8: Quality levels based on Cosine Similarity score.

correction specific to the characteristics of the Portuguese language. We also introduced another approach using the PyEnchant<sup>50</sup> library to guess and correct missing Portuguese words, supplementing the Google Vision API's OCR and language filtering.

For comparative purposes, we applied the Google Vision API directly to images without any preprocessing, filtering for Portuguese characters, and checked to spell using the SpellChecker library set to Portuguese. This provided us with a baseline against which to evaluate the effectiveness of our preprocessing methods.

Our findings also revealed that the Google Vision API was already set to Portuguese. Therefore, after identifying the most effective preprocessing method (OPTION A), we ran again without any language filtering. This led to valuable insights into the effectiveness of various preprocessing and text extraction strategies for Portuguese documents, which we will describe and evaluate in the following sections.

#### 4.2.4 Performance Evaluation

In this subsection, we present a thorough evaluation of the performance of our implemented methods. The evaluation metric used to assess the results is cosine similarity, a similarity measure between two non-zero vectors, which allows us to determine the degree of likeness between the original and processed text.

A summary of the cosine similarity results for different preprocessing and text extraction methods can be found in Table 4.10. The model references and methods can be seen in Figure 4.32.

Our initial experiments focused on assessing the performance of Google Vision API (GV) with and without spelling correction (using the SpellChecker library), both with language filtering. The results from these experiments served as the baseline for our evaluation. In particular, the

---

likely correct spelling for a misspelled word based on the words around it. The library supports multiple languages and allows the use of custom dictionaries. This is often used in Natural Language Processing tasks for preprocessing text data, such as autocorrect applications, chatbots, or text editors (source: <https://pypi.org/project/pyspellchecker/>).

<sup>50</sup>The PyEnchant library is a spellchecking library for Python, built on top of the Enchant library. Enchant is a free software project developed as part of the AbiWord word processor. It aims to provide a simple but flexible interface to a number of different spell checking libraries. PyEnchant brings this functionality to the Python interpreter, including both a Pythonic, portable API and a C API for extension writers. It supports a range of tasks such as word suggestion for corrections, personal word lists, and a variety of dictionaries in different languages. It's often used for text preprocessing in Natural Language Processing (NLP) tasks (source: <https://pypi.org/project/pyenchant/>).

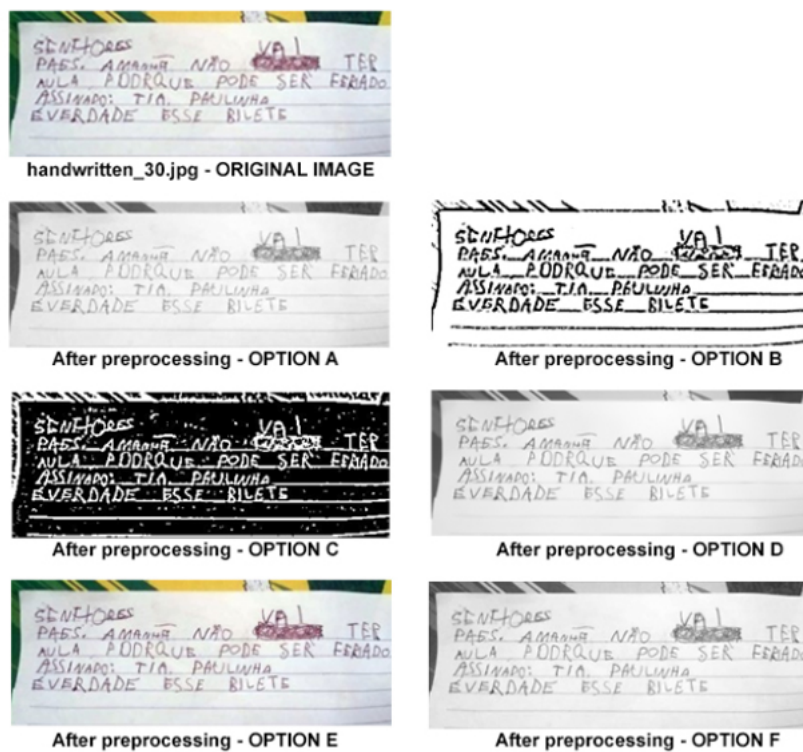


Figure 4.31: Comparison of original and preprocessed images.

cosine similarity results for Google Vision API with filtering (0M2) and Google Vision API with SpellChecker and filtering (0M4) were 0.7205 and 0.7080, respectively.

We then compared these baseline results with those obtained by applying different preprocessing techniques combined with GV and various error correction methods, both with language filtering. For instance, Preprocessing Images (PI) Option A combined with GV and filtering (AM3) yielded a cosine similarity score of 0.7321, an improvement over the baseline. Similarly, when we combined PI Option A, GV, and SpellChecker with filtering (AM5), the score was slightly lower at 0.7298. Interestingly, when we added the PyEnchant library to guess and correct missing Portuguese words to the mix (AM6), the score slightly dropped to 0.7066. Preprocessing options B through F yielded lower cosine similarity scores than option A, indicating that option A led to the best preprocessing outcomes.

Upon discovering that the Google Vision API was already set to Portuguese, we repeated our experiments without any language filtering. We found that the cosine similarity scores remained largely unchanged, and in some cases even slightly improved. For example, PI Option A combined with GV without filtering (AM3b) resulted in a cosine similarity score of 0.7327, which was marginally higher than the score obtained with filtering. This suggests that the language filtering was not a crucial factor in improving the accuracy of the OCR results, and that the preprocessing methods and error correction strategies employed had a more substantial impact on the outcome.

Table 4.9: Summary of image preprocessing methods adopted.

Method	Description
Option A	Convert to grayscale, resize, and apply morphological operations (dilation and erosion)
Option B	Convert to grayscale, apply noise reduction, resize, apply morphological operations, and apply adaptive thresholding for better binarization
Option C	Correct image skewness, convert to grayscale, reduce noise, apply adaptive thresholding, and erode the binary image to de-emphasize text areas
Option D	Convert to grayscale, apply Non-local Means Denoising, and resize the image maintaining the aspect ratio
Option E	Resize the image maintaining the aspect ratio
Option F	Convert to grayscale, resize, and apply morphological operations with a smaller kernel size

#### 4.2.5 Summary of Findings

Our study on using different preprocessing and text extraction methods revealed some interesting insights. Overall, the results, as shown in Table 4.10, indicate that preprocessing images indeed contribute to the performance of OCR using the Google Vision API, albeit to varying degrees depending on the specific techniques employed.

The highest cosine similarity score of 0.7327 was observed for Model AM3b, which employed the preprocessing Option A without any language filtering, suggesting that a combination of grayscale conversion, noise reduction, and morphological operations is particularly effective for enhancing the OCR accuracy and that the language filtering did not significantly contribute to the model's performance.

On the other hand, the preprocessing Option C, which involved additional skew correction, led to lower cosine similarity scores, suggesting that this method may be less beneficial for the document types used in our study.

Using the SpellChecker library generally resulted in slight decreases in cosine similarity scores compared to the respective methods without a spelling check. This could suggest that while this library can help correct individual spelling errors, it might also introduce other alterations to the text, affecting the overall similarity to the original.

In conclusion, our findings suggest that the optimal method for OCR on Portuguese documents, based on our experiments, would involve image preprocessing without language filtering and possibly without using a separate spelling check.

Figure 4.33 presents the results obtained in the models 0M2b (Google Vision baseline, average cosine similarity score of 0.7211), AM3b (best average cosine similarity score: 0.7327), and CM6 (worse average cosine similarity score: 0.5680) in juxtaposition with the image labels for nine items in our dataset (images 27 to 32). Figure 4.29 (already referenced) showcases the images that were loaded.

Model Ref.	Method Description	Average Cosine Similarity
0M2	GV with Filter	0.7205
0M4	GV + SpellChecker with Filter	0.7080
AM3	PI Option A + GV with Filter	0.7321
AM5	PI Option A + GV + SpellChecker with Filter	0.7298
AM6	PI Option A + GV + PyEnchant with Filter	0.7066
BM3	PI Option B + GV with filter	0.6052
BM5	PI Option B + GV + SpellChecker with filter	0.6107
BM6	PI Option B + GV + PyEnchant with filter	0.5741
CM3	PI Option C + GV with filter	0.5684
CM5	PI Option C + GV + SpellChecker with filter	0.5911
CM6	PI Option C + GV + PyEnchant with filter	0.5680
DM3	PI Option D + GV with filter	0.7112
DM5	PI Option D + GV + SpellChecker with filter	0.7211
DM6	PI Option D + GV + PyEnchant with filter	0.6951
EM3	PI Option E + GV with filter	0.7293
EM5	PI Option E + GV + SpellChecker with filter	0.7127
EM6	PI Option E + GV + PyEnchant with filter	0.7025
FM3	PI Option F + GV with filter	0.6977
FM5	PI Option F + GV + SpellChecker with filter	0.7001
FM6	PI Option F + GV + PyEnchant with filter	0.6809
0M2b	GV without filter	0.7211
0M4b	GV + SpellChecker without filter	0.7081
<b>AM3b</b>	<b>PI Option A + GV without Filter</b>	<b>0.7327</b>
AM5b	PI Option A + GV + SpellChecker without Filter	0.7276
AM6b	PI Option A + GV + PyEnchant without Filter	0.7071

Best result highlighted in bold. The Model Labels are explained as follows: GV refers to Google Vision API; 0M2b etc. are model references where 0 means no preprocessed images, A to F represent the preprocessing options, M2 to M6 are the model numbers, and "b" denotes retaken models. There is no M1 assigned, as the model M1 (OCR recognition using PyTesseract) was discarded, see section 4.2.2.

Table 4.10: Cosine Similarity results for different preprocessing and text extraction methods.

The images and the result for the entire dataset (Label and models 0M2b and AM3b) can be seen in Appendix C.

#### 4.2.5.1 Understanding Google Vision "Black Box".

Based on our findings and the common practices in machine learning, it is possible to hypothesize what might be inside the "black box" of Google Vision. Although the precise architecture and methods remain proprietary to Google, we can infer that Google Vision API, as a part of Google Cloud AI, utilizes a combination of complex machine learning models to analyze images, with a heavy reliance on DL technology and CNN for a variety of tasks.

The Google Vision journey likely commences with preprocessing. Images input into the system are probably standardized to a specific format and size. In addition, other preprocessing steps,

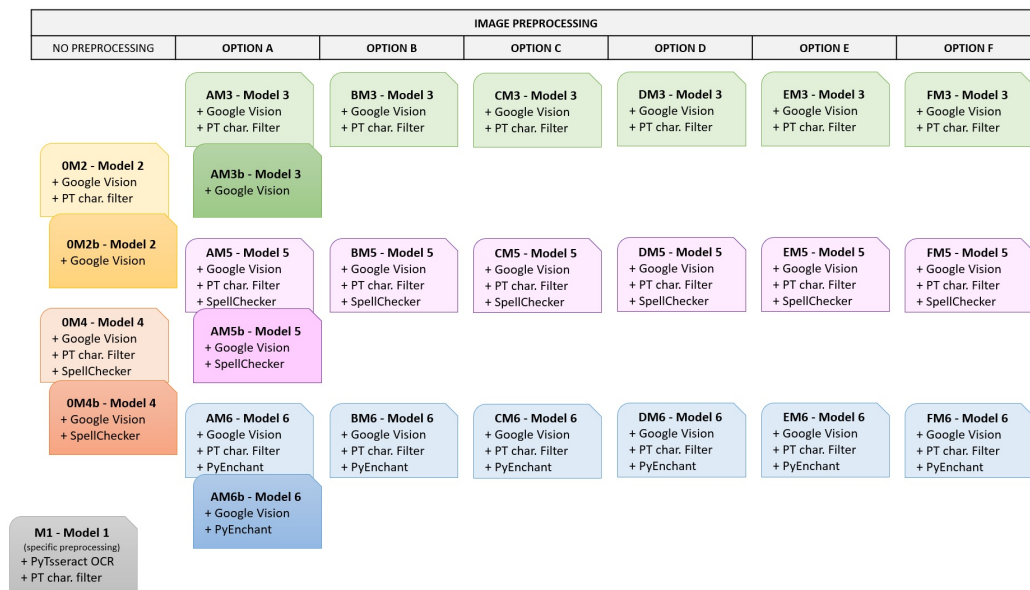


Figure 4.32: Models references and methods - 2<sup>nd</sup> experiment.

such as normalization and augmentation, might be applied to prepare the data for analysis. As evidenced by our second experiment, further preprocessing can yield better results.

Due to their proficiency in image analysis tasks, Convolutional Neural Networks play a vital role. They can extract features from input images through several convolutional and pooling layers. These layers analyze small components of the image to comprehend its local features, then progressively merge this knowledge to understand more global aspects of the image.

OCR algorithms are undoubtedly employed for text detection and image extraction. These involve segmentation to identify and isolate text regions, feature extraction to recognize individual characters, and sequence prediction to comprehend the order of characters and words.

Postprocessing would be the next step in the Google Vision journey. The outputs generated by the neural network are presumably processed further to yield the final results. This process might include decision-making functions, thresholding, and formatting the results to be more user-friendly. As demonstrated in our second experiment, adding extra spell check or guessing words techniques could have been more effective as we got some results improvements. Last but not least, like any machine learning model, the models within Google Vision would be trained on extensive datasets and updated periodically. This iterative process ensures they can improve their performance over time.

Moreover, it is crucial to highlight the security and reliability of using Google Vision. As part of Google’s robust ecosystem, Google Vision is subject to stringent security measures. Data privacy is a top priority, with Google taking comprehensive measures to ensure data security and compliance with global privacy laws. Furthermore, Google’s infrastructure provides high availability and reliability, ensuring the consistent performance of its services.

This hypothesized overview illustrates the potential complexity and sophistication inside Google



Vision's "black box". It is important to note that Google Vision is designed to handle many tasks, extending beyond explicit image recognition and OCR. These may encompass object detection, landmark detection, face detection, image sentiment analysis, and more, all working together to make Google Vision the powerful tool it is today.

The codes (written in Python) used in this experiment are detailed in Appendix D.

```

--- IMAGE 27 ---
LABEL: ADORO JOGOS DE TABULEIRO. EU SEMPRE VENÇO AS PARTIDAS E MEUS AMIGOS FICAM TRISTES.
AM3b.: ADORO JOGOS DE TABULEIRO. EU SEMPRE VENÇO AS PARTIDAS E MEUS AMIGOS FICAM TRISTES.
OM2b.: ADORO JOGOS DE TABULEIRO. EU SEMPRE VENÇO AS PARTIDAS E MEUS AMIGOS FICAM TRISTES.
CM6.: ADORO JOGOS DE TABULEIRO SEMPRE MICA PAR DAS TRISTE

--- IMAGE 28 ---
LABEL: Beijos! me liga
AM3b.: Beijos me liga B 1
OM2b.: Beijos me liga B 1
CM6.: Beijo BE liga

--- IMAGE 29 ---
LABEL: Querida "fada do dente" me dá o meu dinheiro! ATT: Julia DENTE - (EMBAIXO DA FITA)
AM3b.: Querido: "fado- do dente" me dá o meu dinheiro ATT: Julio- DENTE - (EMBAIXO DA FITA)
OM2b.: Querido "fado- do dente" me dá o meu dinheiro ATT: Julia DENTE + (EMBAIXO DA FITA)
CM6.: Querido fado do dente me dá o meu dinheiro WATT filio DENTE. extremidade DA FIFA

--- IMAGE 30 ---
LABEL: SENHORES PAES. AMANHÃ NÃO VAI TER AULA PORQUE PODE SER FERIADO ASSINADO: TIA PAULINHA É VERDADE ESSE BILHETE
AM3b.: SENITORES PAES. AMANHE NÃO AULA TER ADDRQUE PODE SER FERIADO ASSINADO: TIA. PAULINHA EVERDADE ESSE BILETE
OM2b.: SENIORES PAES. AMANHÃ NÃO TER AULA ADDRQUE PODE SER FERIADO ASSINADO: TIA. PAULINHA EVERDADE ESSE BILETE
CM6.: 1 SENIORES VÁ PESAI AMANHE NÃO AI TER PAULA ARQUEADURA PODE SER FADO ASSASSINO TIA PAU LINHA VERDADE ESSE BAILETE

--- IMAGE 31 ---
LABEL: Não falei contigo com medo que os montes e vales que me achas, caissem a teus pés. Saudade é o ar que vou sugando e aceitando, como fruto de verão dos jardins do teu beijo
AM3b.: Não falei contigo mido To com que os incertes e vales que me achas, caissem a teus pés. Saudade é o an que vou e aceitando, como ido sigando fruto de verão dos jaudius do teu beija
OM2b.: Não falei contigo miedo. com que os montes e vales que me achas, caissem a teur pés. 0000000000 Saudade é o an que vou e aceitando, como sigando fruto de verão dos jaudius do teu beija
CM6.: Não falei contigo com medo que os montes a atias UE valer que assentis a teus pés Saudade é o na que vou suando e aceitando como fruto de verão dos auditas do teu beija CATALOGAL

--- IMAGE 32 ---
LABEL: Heróis do mar, nobre povo, nação valente, imortal. Levantai hoje de novo o esplendor de Portugal.
AM3b.: Herois do mar, nobre povo, valente, imortal. levantai ho O mação hoje esplendor de Podyel. de novo
OM2b.: Herois do mar, nobre povo, valente, imortal. Mação levantai ho hoje esplendor de Podzysel. de novo
CM6.: Heróis do mar as nobre povo valente mortal mação levanta hoje de novo esplendor de Apartotel

```

Figure 4.33: Texts extracted from images 27 to 32 using models 0M2b, AM3b and CM6.

### 4.3 Overall Summary of Findings

Our conducted experiments, aimed at evaluating the practical applications of Machine Learning techniques, ventured into two distinct fields - dermatology and OCR.

In the first experiment, we scrutinized the potential of CNN for melanoma detection using dermatoscopic images. Various CNN architectures were tested, involving both custom and pre-trained models, with a focus on fine-tuning and transfer learning strategies. These efforts bore fruit - our updated custom model VB demonstrated compelling results with superior accuracy. Additionally, the efficacy of the transfer learning approach was underlined by the VGG-16 model, which delivered notable results despite its training on a disparate dataset. The potential of CNN in the healthcare domain, specifically for melanoma detection, was hence accentuated.

The second experiment steered our focus towards the OCR performance on Portuguese documents, utilizing the Google Vision API in conjunction with diverse image preprocessing and text extraction methods. The crucial role of preprocessing methods was highlighted as they substantially enhanced the OCR effectiveness. Model AM3b stood out with the highest cosine similarity score. This model, leveraging grayscale conversion, noise reduction, and morphological operations without language filtering, underscored its utility in bolstering OCR accuracy.

However, the addition of skew correction, as seen in preprocessing Option C, led to a dip in cosine similarity scores, suggesting the efficacy of this method is document-type dependent. Likewise, the deployment of a standalone spelling check introduced subtle yet impactful alterations that potentially affected the overall text similarity.

Indeed, while the applications of these experiments may appear diverse at first glance, both hold considerable potential in the healthcare domain. The melanoma detection model from our first experiment has clear, direct implications for medical imaging and diagnostics. On the other hand, the implications of our OCR work might not be as immediately apparent, yet they are no less profound. Efficient and accurate OCR systems can be instrumental in analyzing large volumes of patient information, transcribing handwritten medical notes, and digitizing printed medical records. Furthermore, such systems could facilitate the automatic interpretation of medical prescriptions, making processes more efficient and reducing the risk of errors. As such, our experiments collectively represent significant advancements toward incorporating machine learning into healthcare, aiming to augment human expertise with computational accuracy and scalability. These investigations underline the flexibility and vast potential of machine learning in driving forward the medical field and contributing to improved patient outcomes.

These findings have broadened our understanding of the capacity of CNN in the medical sphere and the avenues to optimize OCR performance when dealing with Portuguese documents. The promising outcomes highlight the importance of tailoring machine learning techniques to the distinct requirements and nuances inherent to each field. The necessity of further research and experimentation in this domain is thus underscored.

## Chapter 5

# Conclusions and Future Work

The journey that culminates in this final chapter of our research has been informed and directed by a substantial body of work in machine learning and healthcare. Our endeavor to apply CNN and OCR was only possible with the wisdom and insights drawn from the extensive literature in these areas. This carefully curated academic foundation allowed us to conceive and conduct our experiments and interpret our findings within the larger context of the field.

From exploring the use of CNN for melanoma detection to leveraging OCR for digitizing Portuguese documents, our research aimed to push the boundaries of machine learning applications in healthcare. The results, while promising, also highlight the complexities and challenges in these endeavors. The interpretations and limitations of our study, discussed in this chapter, offer meaningful insights for future work in this rapidly evolving domain.

In the spirit of academic contribution, this chapter summarizes our findings and proposes potential directions for future research in this field. Our discussion includes a reflective observation of the impact of our research and some final thoughts, taking into account the collective learning from our experiments and the valuable guidance drawn from our chosen bibliography.

### 5.1 Interpretation of Results

The experiments conducted in this study presented a comprehensive analysis of the potential applications of machine learning, especially CNN and OCR, in - and also for - the healthcare domain. In the first experiment, we explored the power of CNN for the automated detection of melanoma using dermatological images. Our results revealed that our custom CNN model VB, developed and fine-tuned throughout our study, achieved significant accuracy in detecting melanoma. This accuracy, brought about by the power of machine learning, holds the promise of enhancing early detection rates, which could ultimately lead to improved prognosis and treatment outcomes for patients.



In the second experiment, we turned our focus to OCR, another application of machine learning, and investigated its performance on Portuguese documents using the Google Vision API. This broad investigation covered various image preprocessing techniques and text extraction methods. It was evident from our results that preprocessing techniques could dramatically influence the effectiveness of OCR. These methods significantly improved the quality of the extracted text, especially in terms of its similarity with the original content.

Though diverse in their nature and objectives, these experiments shared a common thread – the potential benefits of machine learning applications in healthcare. Whether it was the early detection of a potentially deadly disease or the digitization of medical documents, the benefits were clear and significant. Our research could open new doors for using machine learning, offering value in areas such as digitizing medical records, transcribing patient information, and even automating the interpretation of medical prescriptions.

## 5.2 Limitations

Despite the promising findings in our research, it is crucial to acknowledge that our experiments are full of limitations. Our first experiment employed CNN to detect melanoma from dermatological images. While our custom CNN model exhibited high accuracy, this model was validated on a relatively limited dataset. In a real-world scenario, a more comprehensive range of variables, such as different skin tones, image quality, and other skin conditions, could affect model performance. Hence, to ascertain the robustness of our model, it would be necessary to test it on a more extensive and diverse dataset before its deployment.

In the second experiment, we utilized OCR for text extraction from Portuguese documents, assessing the influence of various preprocessing techniques. While these techniques markedly improved OCR's effectiveness, their performance could differ significantly when applied to different types of documents or even handwriting styles, which we did not test. Additionally, while language filtering and spelling correction algorithms were helpful, they were not flawless. The inaccuracies and potential misinterpretations in the extracted text could have subtly affected our results.

Beyond these experimental constraints, a significant limitation resides in the healthcare field itself and goes much further than this work: the difficulty in sharing medical data worldwide. Global collaboration and data sharing are fundamental to uncovering comprehensive solutions to diseases and advancing the healthcare field as a whole. However, this is often hindered by privacy concerns, varying data standards across regions, and other logistical issues (Panagopoulos et al., 2022). The advancement of machine learning and artificial intelligence in healthcare, which could revolutionize the field, largely depends on overcoming these barriers and fostering global collaboration in data sharing. In the face of these limitations, the potential of AI remains vast, and these challenges underline the importance of continued research and development in this sphere.

### 5.3 Recommendations for Future Research

Moving forward, there are several exciting avenues for future research. More comprehensive datasets could be used in melanoma detection to train the models, including images of varying quality and taken in different conditions. This approach would help ensure the model's effectiveness across various scenarios. Additional pre-trained models could also be tested to find a balance between computational efficiency and accuracy.

Testing on a more diverse set of handwritten documents for the OCR system could provide valuable insights. The system could also be optimized further, for instance, by refining the language filtering and spelling correction process or testing other OCR API apart from Google Vision. The development of a dedicated OCR system specifically for medical documents might also be a worthwhile endeavor.

### 5.4 Final Remarks

Through the scope of this thesis, we have delved deep into the realm of machine learning, exploring its practical applications within the healthcare sector - from utilizing CNN for melanoma detection to deploying OCR as a tool for healthcare information management. It is imperative to note that while these domains each come with their unique set of challenges and stipulations, they converge on the common objective of amplifying human expertise and enhancing patient care. The fruitful results from our explorations underscore the potency of machine learning and emphasize the need for sustained research and development in this field.

Renowned British computer scientist Tim Berners-Lee<sup>1</sup> rightly once said, "We need diversity of thought in the world to face the new challenges." Indeed, our experiments have encompassed a broad range of machine-learning methodologies, paving the way for prospective research and discovery. Today, we find ourselves on the brink of a transformative era in healthcare, a field where technological innovation is gaining a stronger foothold by the day.

As the sphere of machine learning continues to flourish and evolve, the spectrum of its application within healthcare appears boundless. It is with eager anticipation that we foresee these technologies continuing to mold our future, thereby stressing the importance and dire need for relentless exploration in this domain. The promising insights derived from our study and their impact on healthcare's future bear witness to the revolutionary capabilities of machine learning.

---

<sup>1</sup>The inventor of the World Wide Web and one of Time Magazine's '100 Most Important People of the 20th Century', Sir Tim Berners-Lee is a scientist and academic whose visionary and innovative work has transformed almost every aspect of our lives (Source: <https://webfoundation.org/about/sir-tim-berners-lee/>).

## Appendix A

# Skin Cancer Integrated Gradients and Grad-CAM

### A.1 Integrated Gradients and Grad-CAM Results Comparison

The following Figures [A.1](#) and [A.2](#) illustrate the results achieved by applying Integrated Gradients and Grad-CAM methodologies. These results are derived from ten randomly chosen sample images from the dataset and evaluated across three distinct models: the original CNN model, the refined CNN Version B, and the pre-trained VGG-16.

The classification indicators 'B' and 'M' denote 'Benign Skin Moles' and 'Malignant Skin Moles', respectively. The correctness of a prediction is symbolized through color coding: a black indicator represents a correct prediction, whereas a red one signifies an incorrect prediction.

For a thorough understanding of the tools and techniques employed in this evaluation, kindly refer to sections [4.1.3](#) and [4.1.4](#).

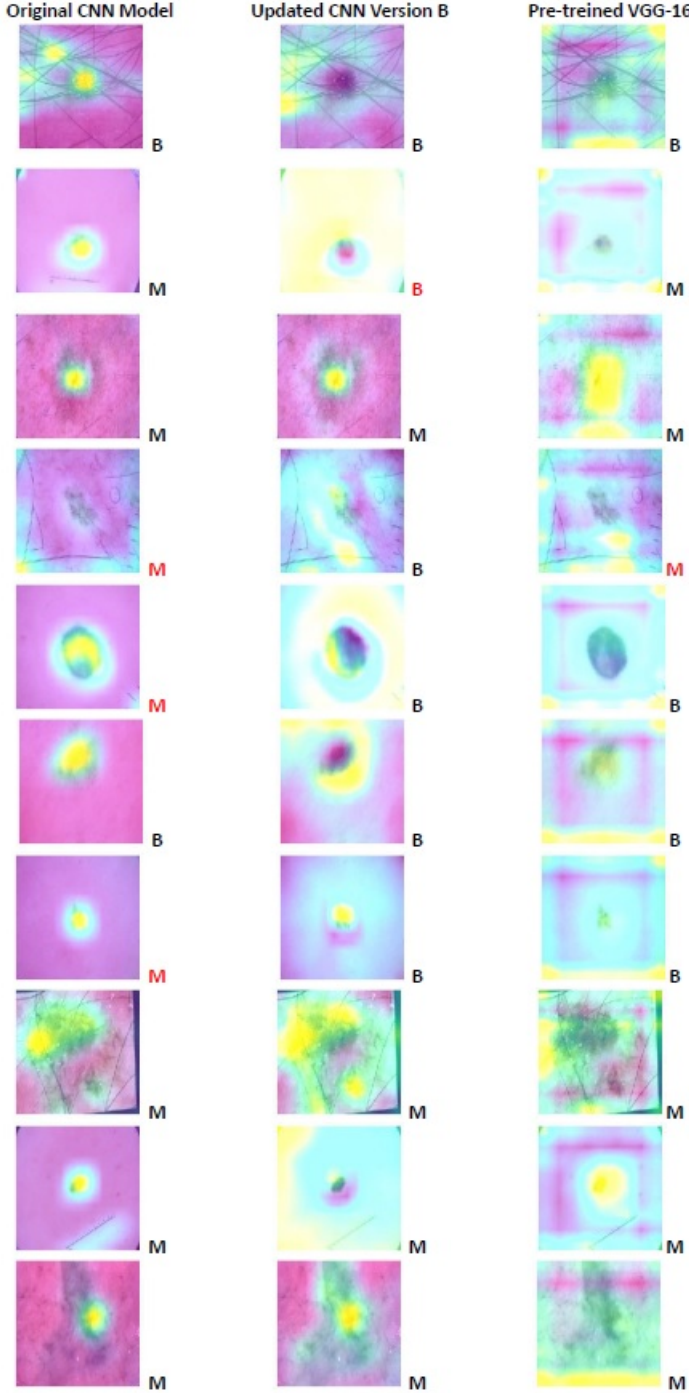


Figure A.1: Integrated Gradients results comparison for selected images.

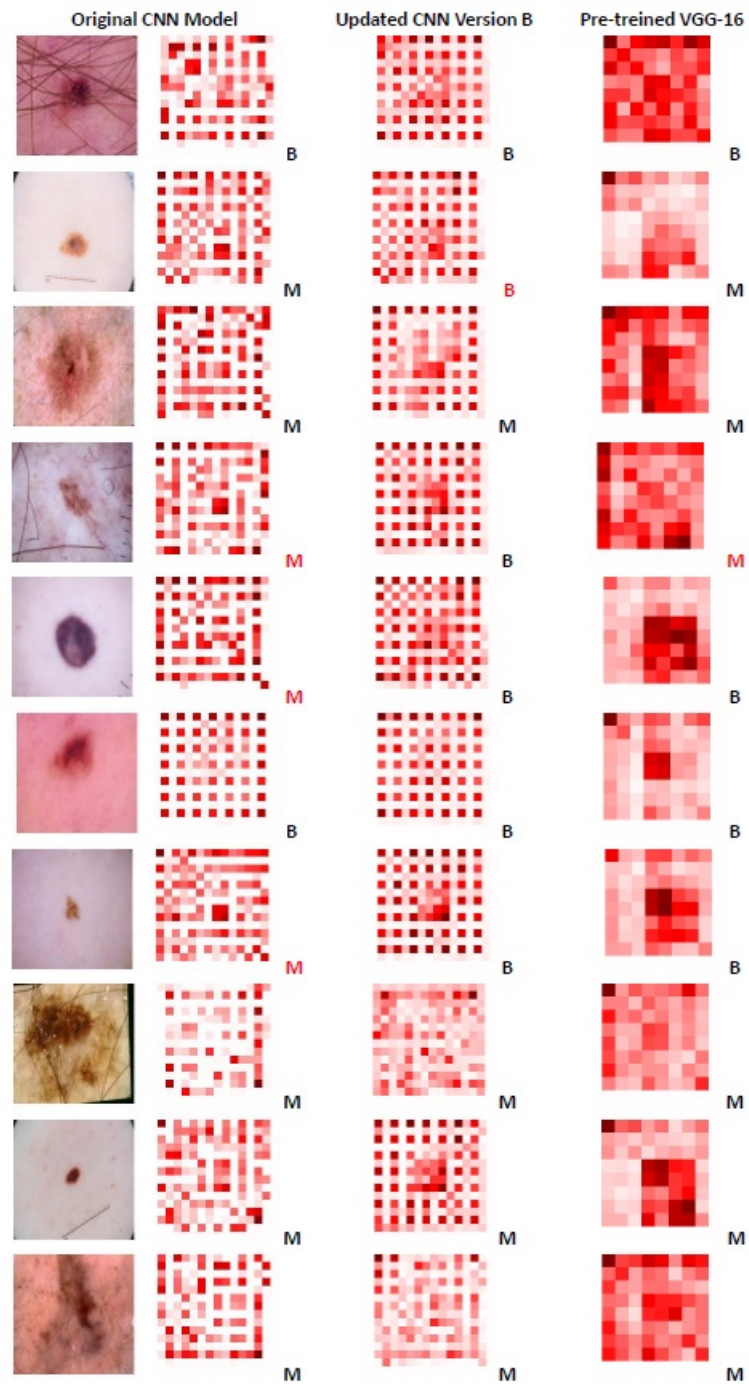


Figure A.2: Grad-CAM results comparison for selected images.

## Appendix B

# Skin Cancer Classification Code

In the following sections, we present the base code used in Experiment 1 (see section 4.1) with its most relevant excerpts and the modifications made during the experiment to obtain improvements.

The programming language used was Python under Google Colab PRO environment.

### B.1 CNN Initial Code (base)

```
1 % Import necessary packages
2 import os
3 import shutil
4 import numpy as np
5 import matplotlib.pyplot as plt
6 from PIL import Image
7 import tensorflow as tf
8 from tensorflow.keras.preprocessing.image import ImageDataGenerator
9 from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D
10 from tensorflow.keras.models import Sequential
11 from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
12 from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
13 import time
14
15 % Set up directories and counts
16 data_dir = '/content/0.3.SKIN_CANCER_Code/data/'
17 benign_dir = os.path.join(data_dir, 'benign')
18 malignant_dir = os.path.join(data_dir, 'malignant')
19 train_dir = os.path.join(data_dir, 'TRAIN')
20 test_dir = os.path.join(data_dir, 'TEST')
21 os.makedirs(train_dir, exist_ok=True)
22 os.makedirs(test_dir, exist_ok=True)
```

```

23 np.random.seed(42)
24 train_ratio = 0.8
25
26 % Define the check_image function
27 def check_image(file_path):
28     try:
29         img = Image.open(file_path)
30         img.verify()
31         return True
32     except Exception as e:
33         print(f"Problematic image: {file_path}")
34         print(f"Error: {e}")
35         return False
36
37 % Define the split_data function
38 def split_data(source_dir, target_train_dir, target_test_dir, train_ratio):
39     files = [f for f in os.listdir(source_dir) if os.path.isfile(os.path.join(
40         source_dir, f))]
41     np.random.shuffle(files)
42     train_files = files[:int(len(files) * train_ratio)]
43     test_files = files[int(len(files) * train_ratio):]
44     os.makedirs(target_train_dir, exist_ok=True)
45     os.makedirs(target_test_dir, exist_ok=True)
46     for file in train_files:
47         file_path = os.path.join(source_dir, file)
48         if not check_image(file_path):
49             print(f"Skipping problematic image: {file_path}")
50             continue
51         shutil.copy(file_path, os.path.join(target_train_dir, file))
52     for file in test_files:
53         file_path = os.path.join(source_dir, file)
54         if not check_image(file_path):
55             print(f"Skipping problematic image: {file_path}")
56             continue
57         shutil.copy(file_path, os.path.join(target_test_dir, file))
58
59 % Split the dataset into training and testing sets
60 split_data(benign_dir, os.path.join(train_dir, 'benign'), os.path.join(test_dir, '
61     benign'), train_ratio)
62 split_data(malignant_dir, os.path.join(train_dir, 'malignant'), os.path.join(
63     test_dir, 'malignant'), train_ratio)
64
65 % Load and preprocess the data with data augmentation
66 class DebugImageDataGenerator(ImageDataGenerator):
67     def __getitem__(self, index):
68         try:
69             return super().__getitem__(index)
70         except Exception as e:
71             print(f"Error occurred for batch index {index}: {e}")

```

```

69         raise
70 train_datagen = DebugImageDataGenerator(rescale=1.0/255, rotation_range=40,
    width_shift_range=0.2, height_shift_range=0.2, shear_range=0.2, zoom_range=0.2,
    horizontal_flip=True, fill_mode='nearest')
71 test_datagen = DebugImageDataGenerator(rescale=1.0/255)
72 train_generator = train_datagen.flow_from_directory(train_dir, target_size=(150,
    150), batch_size=32, class_mode='binary')
73 test_generator = test_datagen.flow_from_directory(test_dir, target_size=(150, 150),
    batch_size=32, class_mode='binary')
74
75 % Modify the CNN model architecture
76 model = tf.keras.models.Sequential([
77     tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)
    ),
78     tf.keras.layers.MaxPooling2D(2, 2),
79     tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
80     tf.keras.layers.MaxPooling2D(2, 2),
81     tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
82     tf.keras.layers.MaxPooling2D(2, 2),
83     tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
84     tf.keras.layers.MaxPooling2D(2, 2),
85     tf.keras.layers.Flatten(),
86     tf.keras.layers.Dropout(0.5),
87     tf.keras.layers.Dense(512, activation='relu'),
88     tf.keras.layers.Dense(1, activation='sigmoid')
89 ])
90
91 % Compile and train the model with early stopping and ReduceLRonPlateau
92 model.compile(optimizer='adam',
93     loss='binary_crossentropy',
94     metrics=['accuracy'])
95 early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)
96 reduce_lr = tf.keras.callbacks.ReduceLRonPlateau(monitor='val_loss', factor=0.1,
    patience=3, min_lr=0.00001)
97 history = model.fit(
98     train_generator,
99     steps_per_epoch=steps_per_epoch,
100    epochs=30,
101    validation_data=test_generator,
102    validation_steps=validation_steps,
103    callbacks=[early_stop, reduce_lr])
104
105 % Evaluate the model and make predictions
106 test_loss, test_acc = model.evaluate(test_generator)
107 print('Test accuracy:', test_acc)

```



## B.2 Grad-CAM & Integrated Gradients code

```

1 % Install necessary packages
2 !pip install tf-explain
3 !pip install tf-explain alibi
4
5 % Import necessary packages
6 import numpy as np
7 import matplotlib.pyplot as plt
8 from tensorflow.keras.preprocessing import image
9 from tf_explain.core.grad_cam import GradCAM
10 import os
11 from PIL import Image
12 from alibi.explainers import IntegratedGradients
13
14 % Define the gradcam_visualization function
15 def gradcam_visualization(image_path, model, layer_name):
16     img = image.load_img(image_path, target_size=(img_width, img_height))
17     img_array = image.img_to_array(img)
18     img_array = np.expand_dims(img_array, axis=0)
19     img_array /= 255.
20     prediction = model.predict(img_array)
21     predicted_class = np.argmax(prediction[0])
22     explainer = GradCAM()
23     grid = explainer.explain((img_array, None), model, class_index=predicted_class,
24                             layer_name=layer_name)
25     plt.imshow(grid)
26     plt.axis('off')
27     plt.show()
28
29 % Specify the layer for the Grad-CAM
30 layer_name = 'conv2d_3'
31 pred_dir = '/content/0.3.SKIN_CANCER_Code/data/pred'
32 image_files = os.listdir(pred_dir)
33
34 % Visualize the Grad-CAM for each image
35 for image_file in image_files:
36     image_path = os.path.join(pred_dir, image_file)
37     gradcam_visualization(image_path, model, layer_name)
38
39 % Define the integrated_gradients_visualization function
40 def integrated_gradients_visualization(image_path, model, layer_name):
41     img = Image.open(image_path)
42     img_resized = img.resize((150, 150))
43     img_array = np.array(img_resized)
44     img_array = np.expand_dims(img_array, axis=0)
45     img_array = img_array / 255.
46     explainer = IntegratedGradients(model, layer=model.get_layer(layer_name))

```

```

46     attributions = explainer.explain(img_array).attributions[0]
47     aggregated_attributions = np.sum(np.abs(attributions), axis=-1)
48     plt.subplot(1, 2, 1)
49     plt.imshow(img_array[0])
50     plt.axis('off')
51     plt.subplot(1, 2, 2)
52     plt.imshow(aggregated_attributions[0], cmap='seismic', vmin=-np.max(np.abs(
        aggregated_attributions[0])), vmax=np.max(np.abs(aggregated_attributions
            [0])))
53     plt.axis('off')
54     plt.show()
55
56 % Directory where your images are stored
57 pred_dir = '/content/0.3.SKIN_CANCER_Code/data/pred'
58
59 % Ensure this layer exists in your model
60 layer_name = 'conv2d_3'
61
62 % List of image files in the directory
63 image_files = os.listdir(pred_dir)
64
65 % Call integrated_gradients_visualization for each image
66 for image_file in image_files:
67     image_path = os.path.join(pred_dir, image_file)
68     integrated_gradients_visualization(image_path, model, layer_name)

```

### B.3 CNN Updated Model - Version A

```

1 # Import necessary packages
2 from keras.models import Sequential
3 from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, BatchNormalization,
    Dropout
4 from keras.regularizers import l2
5 from keras.optimizers import Adam
6 from sklearn.model_selection import StratifiedKFold
7 import tensorflow as tf
8
9 # Define the function to create the model
10 def create_model(lr):
11     model = Sequential()
12     model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)))
13     model.add(BatchNormalization())
14     model.add(MaxPooling2D(pool_size=(2, 2)))
15     model.add(Conv2D(64, (3, 3), activation='relu'))
16     model.add(BatchNormalization())
17     model.add(MaxPooling2D(pool_size=(2, 2)))

```

```

18     model.add(Conv2D(128, (3, 3), activation='relu'))
19     model.add(BatchNormalization())
20     model.add(MaxPooling2D(pool_size=(2, 2)))
21     model.add(Conv2D(256, (3, 3), activation='relu'))
22     model.add(BatchNormalization())
23     model.add(MaxPooling2D(pool_size=(2, 2)))
24     model.add(Flatten())
25     model.add(Dense(256, activation='relu', kernel_regularizer=l2(0.01)))
26     model.add(BatchNormalization())
27     model.add(Dropout(0.5))
28     model.add(Dense(128, activation='relu', kernel_regularizer=l2(0.01)))
29     model.add(BatchNormalization())
30     model.add(Dropout(0.5))
31     model.add(Dense(1, activation='sigmoid'))
32     model.compile(optimizer=Adam(learning_rate=lr), loss='binary_crossentropy',
33                  metrics=['accuracy'])
34
35 # Define Cross-validation and Hyperparameter tuning
36 kf = StratifiedKFold(n_splits=5)
37 for lr in [1e-3, 1e-4, 1e-5]:
38     print("Training model with learning rate: " + str(lr))
39     fold_no = 1
40     for train_index, val_index in kf.split(np.zeros(total_train_images),
41                                           train_generator.labels):
42         model = create_model(lr)
43         print(f'Training for fold {fold_no} ...')
44         early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience
45                                                       =5)
46         reduce_lr = tf.keras.callbacks.ReduceLRonPlateau(monitor='val_loss', factor
47                                                         =0.1, patience=3, min_lr=0.00001)
48         history = model.fit(
49             train_generator,
50             steps_per_epoch=steps_per_epoch,
51             validation_data=validation_generator,
52             validation_steps=validation_steps,
53             epochs=30,
54             callbacks=[early_stop, reduce_lr])
55         fold_no += 1
56
57 # Compile and train the model with early stopping and ReduceLRonPlateau
58 model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
59 early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)
60 reduce_lr = tf.keras.callbacks.ReduceLRonPlateau(monitor='val_loss', factor=0.1,
61                                                  patience=3, min_lr=0.00001)
62 history = model.fit(
63     train_generator,
64     steps_per_epoch=steps_per_epoch,
65     epochs=30,

```

```
62 validation_data=test_generator,  
63 validation_steps=validation_steps,  
64 callbacks=[early_stop, reduce_lr])
```

---

## B.4 CNN Updated Model - Version B

```
1 # Import necessary packages  
2 from keras.models import Sequential  
3 from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, BatchNormalization,  
   Dropout  
4 from keras.optimizers import Adam  
5 import tensorflow as tf  
6  
7 # Define the CNN model architecture  
8 model = Sequential()  
9 model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)))  
10 model.add(BatchNormalization())  
11 model.add(MaxPooling2D(pool_size=(2, 2)))  
12 model.add(Conv2D(64, (3, 3), activation='relu'))  
13 model.add(BatchNormalization())  
14 model.add(MaxPooling2D(pool_size=(2, 2)))  
15 model.add(Conv2D(128, (3, 3), activation='relu'))  
16 model.add(BatchNormalization())  
17 model.add(MaxPooling2D(pool_size=(2, 2)))  
18 model.add(Conv2D(256, (3, 3), activation='relu'))  
19 model.add(BatchNormalization())  
20 model.add(MaxPooling2D(pool_size=(2, 2)))  
21 model.add(Flatten())  
22 model.add(Dense(256, activation='relu'))  
23 model.add(BatchNormalization())  
24 model.add(Dropout(0.5))  
25 model.add(Dense(128, activation='relu'))  
26 model.add(BatchNormalization())  
27 model.add(Dropout(0.5))  
28 model.add(Dense(1, activation='sigmoid'))  
29 # Compile the model  
30 model.compile(optimizer=Adam(learning_rate=1e-3), loss='binary_crossentropy',  
   metrics=['accuracy'])  
31  
32 # Define early stopping and ReduceLROnPlateau  
33 early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)  
34 reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.1,  
   patience=3, min_lr=0.00001)
```

---

## B.5 Pre-trained VGG-16 CNN

```

1 # Import necessary packages
2 from keras.applications.vgg16 import VGG16
3 from keras.models import Sequential
4 from keras.layers import Flatten, Dense, BatchNormalization, Dropout
5 from keras.optimizers import Adam
6 import tensorflow as tf
7
8 # Load VGG16 pre-trained on ImageNet, excluding the top layers
9 base_model = VGG16(weights='imagenet', include_top=False, input_shape=(img_width,
    img_height, 3))
10
11 model = Sequential()
12 model.add(base_model) # Add the base model for transfer learning
13 model.add(Flatten())
14 model.add(Dense(256, activation='relu'))
15 model.add(BatchNormalization())
16 model.add(Dropout(0.5))
17 model.add(Dense(128, activation='relu'))
18 model.add(BatchNormalization())
19 model.add(Dropout(0.5))
20 model.add(Dense(1, activation='sigmoid'))
21
22 # Freeze the layers in the base model so they don't get trained
23 base_model.trainable = False
24
25 # Compile the model
26 model.compile(optimizer=Adam(learning_rate=1e-4), loss='binary_crossentropy',
    metrics=['accuracy'])
27
28 # Define early stopping and ReduceLROnPlateau
29 early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)
30 reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.1,
    patience=3, min_lr=0.00001)

```

## B.6 Pre-trained Inception-ResNet-v2 CNN

```

1 # Import necessary packages
2 from keras.applications.inception_resnet_v2 import InceptionResNetV2
3 from keras.models import Sequential
4 from keras.layers import Flatten, Dense, BatchNormalization, Dropout
5 from keras.optimizers import Adam
6 import tensorflow as tf

```

```
7
8 # Load InceptionResNetV2 pre-trained on ImageNet, excluding the top layers
9 base_model = InceptionResNetV2(weights='imagenet', include_top=False, input_shape=(
    img_width, img_height, 3))
10
11 model = Sequential()
12 model.add(base_model) # Add the base model for transfer learning
13 model.add(Flatten())
14 model.add(Dense(256, activation='relu'))
15 model.add(BatchNormalization())
16 model.add(Dropout(0.5))
17 model.add(Dense(128, activation='relu'))
18 model.add(BatchNormalization())
19 model.add(Dropout(0.5))
20 model.add(Dense(1, activation='sigmoid'))
21
22 # Freeze the layers in the base model so they don't get trained
23 base_model.trainable = False
24
25 # Compile the model
26 model.compile(optimizer=Adam(learning_rate=1e-4), loss='binary_crossentropy',
    metrics=['accuracy'])
27
28 # Define early stopping and ReduceLRonPlateau
29 early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)
30 reduce_lr = tf.keras.callbacks.ReduceLRonPlateau(monitor='val_loss', factor=0.1,
    patience=3, min_lr=0.00001)
```

## Appendix C

# Handwritten Dataset Images & Detected Texts

We present in the next section the images of our dataset, followed by the respective LABEL text and the text detection with models OM2b (using only the Google Vision API) and AM3b (using the Google Vision API after preprocessing the images with option A). Details about the models can be seen in section 4.2.4.

### C.1 Images and Texts

The following Figures C.1, C.2, C.3, C.4, C.5, C.6, C.7, C.8 and C.9 concatenate the images from our dataset and the texts obtained (Portuguese only). Some images have been cropped to fit the page.

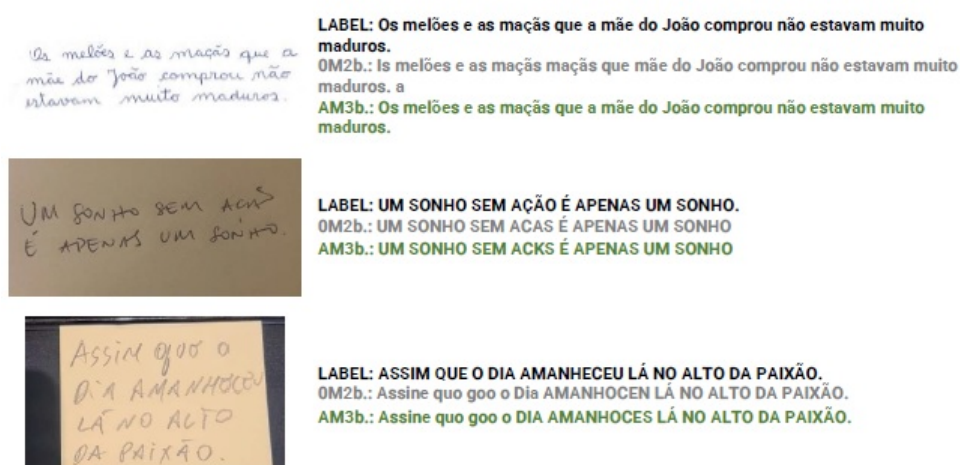
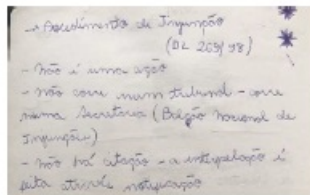


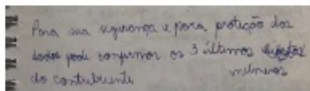
Figure C.1: Dataset images and texts 1 to 3.



**LABEL:** Procedimento de Injunção (DL 269/98) - Não é uma ação - Não corre num tribunal - corre numa secretaria (Balcão Nacional de Injunções) - Não há citação - a interpelação é feita através notificação

**OM2b.:** → Procedimento de Injunção (DL 269/98) não é uma ação não corre num tribunal numa secretaria (Balgão nacional de Injunções) - não há citação feita através notificação 8 - corre - a interpelação i

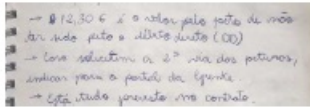
**AM3b.:** → Poedimento de Injunção (DL 269/98) -Não é uma ação não covre num tubundl - corre numa secretaria (Bolgão nacional de i - Injunções) não há citação - a interpelação pita através notificação



**LABEL:** Para sua segurança e para proteção dos dados pode confirmar os 3 últimos do contribuinte números

**OM2b.:** Para dos segurança e para proteção dados pode comprmor os 3 últimos dias do contulunte inúmeros sura

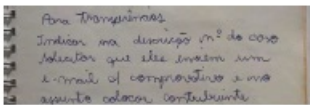
**AM3b.:** Para sua segurança e para proteção dos dados pode comprmor os 3 últimos altos do contribuinte inimeros



**LABEL:** - 12,30 € é o valor pelo facto de não ter sido feito o débito direto (DD) - Caso solicitem a 2ª via das facturas, indicar para o portal da Gerente. - Está tudo previsto no contrato.

**OM2b.:** → 12,30 € é o valor pelo pacto de não ter sido feito o débito direto (DD) - Cose solicitem a 2ª via dos pocturos, indicar para o portal da Grenke → Está tudo previsto no contrato

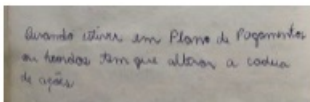
**AM3b.:** → 12,30 € é o valor pelo pacto de não ter sido feito o débito direto (DD) Case solicitem a 2ª via dos pocturos, indicar para o portal da Grenke → Está tudo previsto no contrato



**LABEL:** Para transferências Indicar na descrição nº do caso solicitar que eles enviem um e-mail c/ comprovativo e no assunto colocar contribuinte

**OM2b.:** Para transferências Indicar na descrição in do caso solicitar eles enrem um que e-mail of comprovativo assunto colocar contribuinte e no

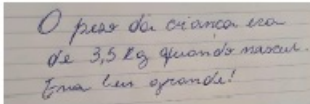
**AM3b.:** Para transerências Indicar na descrição in do caso eles enrem um solicitor que e-mail o comprovativo e no assunto colocar contribuinte



**LABEL:** Quando estiver em Plano de Pagamentos ou Acordos tem que alterar a cadeia de ações

**OM2b.:** Quando estiver em Plano de ou Acordos tem que alteron de ações Pagamentos cadeia

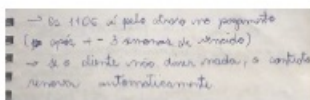
**AM3b.:** Quando estiver em Plano de Pagamentos Acordos tem que alteron ou de ações alteron a cadeia



**LABEL:** O peso da criança era de 3,5 kg quando nasceu. Era bem grande!

**OM2b.:** O pese da criança era ) de 3,5 kg quando nascul Era ben grande!

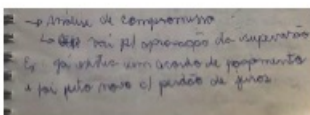
**AM3b.:** O ) peso da criança era de 3,5 kg quando nascul Ena ben grande!



**LABEL:** - Os 110€ é pelo atraso no pagamento ( após + - 3 semanas de vencido) - se o cliente não disser nada, o contrato renova automaticamente

**OM2b.:** Os 110€ pelo atroro no pagamento ( após + - 3 semanas de vencido) → se o cliente não disser nada, o contrato automaticamente renova

**AM3b.:** -> os 110€ i pelo atraso no pagamento u ( após + - 3 semanas de vencido) → se o cliente não disser nada, o contrato automaticamente renova



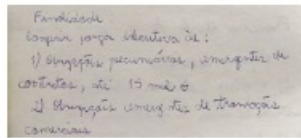
**LABEL:** - Análise de compromisso - vai p/ aprovação da supervisão Ex: Já existia um acordo de pagamento e foi feito novo c/ perdão de juros.

**OM2b.:** 3 Análise de compromisso vai pl aprovação da supervisão Ex: já existic um acordo de pagamento a foi pito novo c/ perdão de juros.

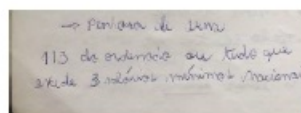
**AM3b.:** 3 Análise de compromisso 4 vai pl aprovação da supervisão Ex: Já existic um acordo de pagamento Ja l foi pito novo c/ perdão de juros.

Figure C.2: Dataset images and texts 4 to 11.

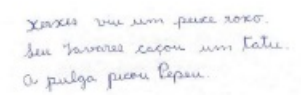




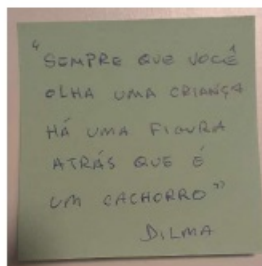
**LABEL: Finalidade: Conferir força executiva às : 1) Obrigações pecuniárias, emergentes de contratos, até 15 mil E 2) Obrigações emergentes de transações comerciais**  
**OM2b.: Fndidade: Conferir força executiva às : 1) Obrigações pecuniárias, emergentes de contratos, até 15 m 6 2) Obriziçais emergentes de tramaçãs comerciais**  
**AM3b.: Fndidade: Conferir força executiva às : 1) obrigações pecuniárias, emergentes de contratos, até 15 mi 6 2) Obriziçais emergentes de tramaçãs comerciais**



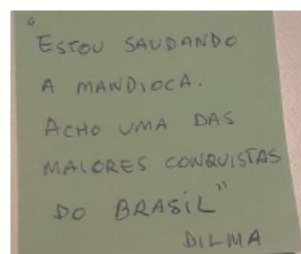
**LABEL: - Penhora de bens 1/3 do ordenado ou tudo que excede 3 salários mínimos nacionais**  
**OM2b.: -> Penhora di vens 113 de ordemde ou tude que avede 3 salários mínimos nacionais**  
**AM3b.: -> Penhora de pens 113 de ordemde ou tude que exede 3 salários mínimos nacionais**



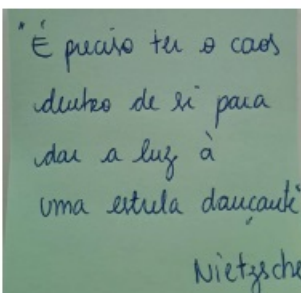
**LABEL: Xerxes viu um peixe roxo. Seu Tavares caçou um tatu. A pulga picou Pepeu.**  
**OM2b.: viu um peixe roxo. Xerxes Seu Tavares caçou um a pulga picon Pepen. tatu.**  
**AM3b.: viu um peixe roxo. Xerxes Seu Tavares caçou um a pulga picon Pepen tatu.**



**LABEL: "SEMPRE QUE VOCÊ OLHA UMA CRIANÇA HÁ UMA FIGURA ATRÁS QUE É UM CACHORRO" DILMA**  
**OM2b.: u SEMPRE Que você OLHA UMA CRIANÇA HÁ ATRÁS QUE um UMA FIGURA É CACHORRO >> DILMA**  
**AM3b.: u SEMPRE Que Que vo você OLHA UMA CRIANÇA HÁ ATRÁS QUE um UMA FIGURA - 10 CACHORRO DILMA >>**

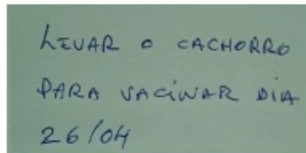


**LABEL: "ESTOU SAUDANDO A MANDIOCA. ACHO UMA DAS MAIORES CONQUISTAS DO BRASIL" DILMA**  
**OM2b.: ù ESTOU SAUDANDO A MANDIOCA. ACHO UMA DAS MAIORES CONQUISTAS DO BRASIL DILMA**  
**AM3b.: ù ESTOU SAUDANDO A MANDIOCA. ACHO UMA DAS MAIORES CONQUISTAS DO BRASIL 21 DILMA**



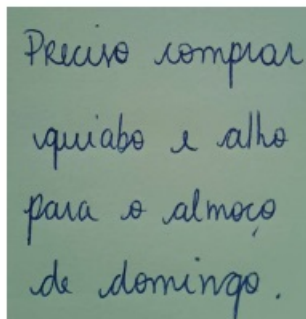
**LABEL: "É preciso ter o caos dentro de si para dar a luz à uma estrela dançante" Nietzsche**  
**OM2b.: É preciso dentro de li para a luz à uma estrula daucante Nietzsche dar a ter o caos**  
**AM3b.: É preciso iso te o caos dentro de si para a luz à uma estrula daucante. Nietzsche dar a**

Figure C.3: Dataset images and texts 12 to 17.



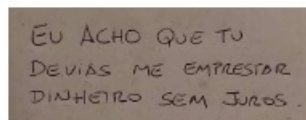
LEVAR O CACHORRO  
PARA VACINAR DIA  
26/04

**LABEL:** Levar o cachorro para vacinar dia 26/04  
**OM2b.:** LEVAR PARA VACINAR DIA 26/04 • CACHORRO  
**AM3b.:** LEVAR PARA VACINAR DIA 26/04 O CACHORRO



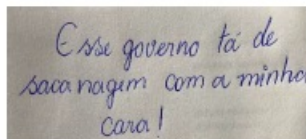
Preciso comprar  
quiabo e alho  
para o almoço  
de domingo.

**LABEL:** Preciso comprar quiabo e alho para o almoço de domingo.  
**OM2b.:** Preciso comprar quiabo e alho para o almoço de domingo.  
**AM3b.:** Preciso comprar quiabo e alho para o almoço de domingo.



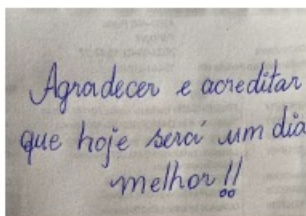
EU ACHO QUE TU  
DEVIAS ME EMPRESTAR  
DINHEIRO SEM JUROS.

**LABEL:** EU ACHO QUE TU DEVIAS ME EMPRESTAR DINHEIRO SEM JUROS .  
**OM2b.:** EU ACHO QUE TU DEVIAS ME EMPRESTAR DINHEIRO SEM JUROS  
**AM3b.:** EU ACHO QUE TU DEVIAS ME EMPRESTOR DINHEIRO SEM JUROS



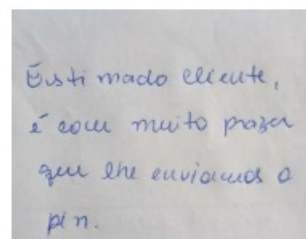
Esse governo tá de  
sacanagem com a minha  
cara!

**LABEL:** Esse governo tá de sacanagem com a minha cara!  
**OM2b.:** Esse ta de governo sacanagem com a minhar Cara !  
**AM3b.:** Esse tá de governo Sacanagem com a minha Cara !



Agradecer e acreditar  
que hoje serci um dia  
melhor!!

**LABEL:** Agradecer e acreditar que hoje será um dia melhor!!  
**OM2b.:** ono 801-Cosh 2007 Agradecer e acreditar que hoje serci um dia melhor!! 00  
**AM3b.:** 15:27 Agradecer e acreditar que hoje serci um dia melhor!!



Estimado cliente,  
é com muito prazer  
que lhe enviamos a  
pin.

**LABEL:** Estimado cliente, é com muito prazer que lhe enviamos o pin.  
**OM2b.:** Estimado cliente, é com muito prazer que lhe enviamos a pin.  
**AM3b.:** Estimado cliente, é com muito prazer i que lhe enviamos a pin.

Figure C.4: Dataset images and texts 18 to 23.

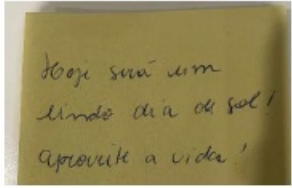
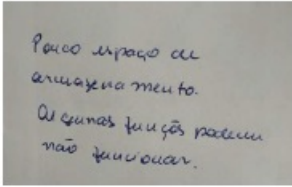
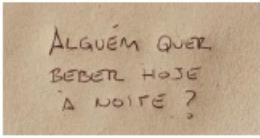
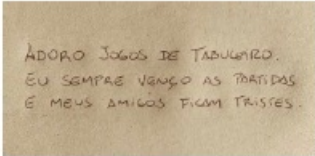
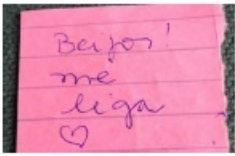
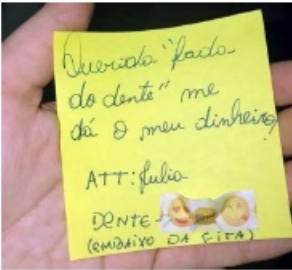
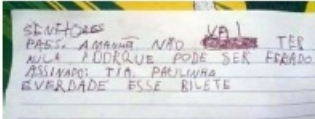
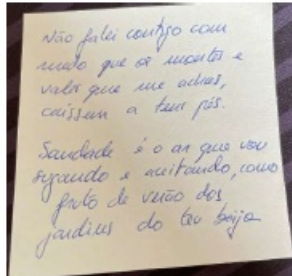
	<p><b>LABEL:</b> Hoje será um lindo dia de sol! aproveite a vida!  <b>OM2b.:</b> Hoje será um lindo dia de sol / aproveite vida! a  <b>AM3b.:</b> Hoje será um lindo dia ou sol / aproveite vida! a</p>
	<p><b>LABEL:</b> Pouco espaço de armazenagem. Alguma funções podem não funcionar.  <b>OM2b.:</b> Pouco espaço de a armazenagem meu to. Algumas funções podem não funcionar.  <b>AM3b.:</b> Pouco espaço de a armazenagem meu to. Algumas funções podem não funcionar.</p>
	<p><b>LABEL:</b> ALGUÉM QUER BEBER HOJE À NOITE ?  <b>OM2b.:</b> ALGUÉM QUER BEBER HOJE A NOITE ?  <b>AM3b.:</b> ALGUÉM QUER BEBER HOJE A NOITE ?</p>
	<p><b>LABEL:</b> ADORO JOGOS DE TABULEIRO. EU SEMPRE VENÇO AS PARTIDAS E MEUS AMIGOS FICAM TRISTES.  <b>OM2b.:</b> ADORO JOGOS DE TABULEIRO. EU SEMPRE VENÇO AS PARTIDAS E MEUS AMIGOS FICAM TRISTES.  <b>AM3b.:</b> ADORO JOGOS DE TABULEIRO. EU SEMPRE VENÇO AS PARTIDAS E MEUS AMIGOS FICAM TRISTES.</p>
	<p><b>LABEL:</b> Beijos! me liga  <b>OM2b.:</b> Beijos me liga B 1  <b>AM3b.:</b> Beijos me liga B 1</p>
	<p><b>LABEL:</b> Querida "fada do dente" me dá o meu dinheiro! ATT: Julia DENTE - (EMBAIXO DA FITA)  <b>OM2b.:</b> Querido "fado- do dente" me dá o meu dinheiro ATT: Julia DENTE + (EMBAIXO DA FITA)  <b>AM3b.:</b> Querido: "fado- do dente" me dá o meu dinheiro ATT: Julio- DENTE - (EMBAIXO DA FITA)</p>
	<p><b>LABEL:</b> SENHORES PAES. AMANHÃ NÃO VAI TER AULA PORQUE PODE SER FERIADO  <b>ASSINADO:</b> TIA PAULINHA É VERDADE ESSE BILHETE  <b>OM2b.:</b> SENIORES PAES. AMANHÃ NÃO TER AULA PORQUE PODE SER FERIADO  <b>ASSINADO:</b> TIA. PAULINHA É VERDADE ESSE BILETE  <b>AM3b.:</b> SENITORES PAES. AMANHE NÃO AULA TER PORQUE PODE SER FERIADO  <b>ASSINADO:</b> TIA. PAULINHA É VERDADE ESSE BILETE</p>

Figure C.5: Dataset images and texts 24 to 30.

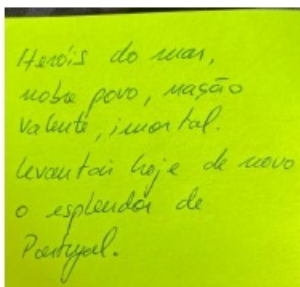




**LABEL:** Não falei contigo com medo que os montes e vales que me achas, caissem a teus pés. Saudade é o ar que vou sugando e aceitando, como fruto de verão dos jardins do teu beija

**0M2b.:** Não falei contigo medo. com que os montes e vales que me achas, caissem a teur pés. 0000000000 Saudade é o an que vou e aceitando, como sigando fruto de verão dos jaudius do teu beija

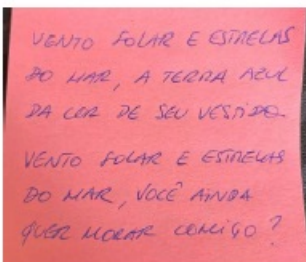
**AM3b.:** Não falei contigo mido To com que os incertes e vales que me achas, caissem a teus pés. Saudade é o an que vou e aceitando, como ido sigando fruto de verão dos jaudius do teu beija



**LABEL:** Heróis do mar, nobre povo, nação valente, imortal. Levantai hoje de novo o esplendor de Portugal.

**0M2b.:** Herois do mar, nobre povo, valente, imortal. Mação levantai ho hoje esplendor de Podzysel. de novo

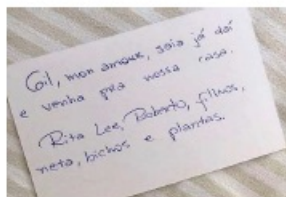
**AM3b.:** Herois do mar, nobre povo, valente, imortal. levantai ho O mação hoje esplendor de Podyel. de novo



**LABEL:** VENTO POLAR E ESTRELAS DO MAR, A TERRA AZUL DA COR DE SEU VESTIDO VENTO POLAR E ESTRELAS DO MAR, VOCÊ AINDA QUER MORAR COMIGO?

**0M2b.:** VENTO FOLAR E ESTRELAS DO MAR, A TERRA AZUL DA COR DE SEU VESTIDO VENTO SOLAR E ESTRELAS DO MAR, VOCÊ AINDA QUER MORAR COMIGO?

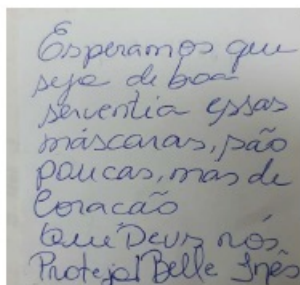
**AM3b.:** VENTO FOLAR E ESTRELAS DO MAR, A TERRA AZUL DA COR DE SEU VESTIDO VENTO SOLAR E ESTRELAS DO MAR, VOCÊ AINDA QUER MORAR COMIGO?



**LABEL:** Gil, mon amour, saia já daí e venha pra nossa casa. Rita Lee, Roberto, filhos, neta, bichos e plantas.

**0M2b.:** Gil, e nossa venha mon amour, saia já daí pra Rita Lee, Roberto, fillos, neta, bichos plantas. casa. e

**AM3b.:** Gil, e nossa mon amour, saia já daí pra Rita Lee, Roberto, filhos, neta, bichos plantas. venha casa. e

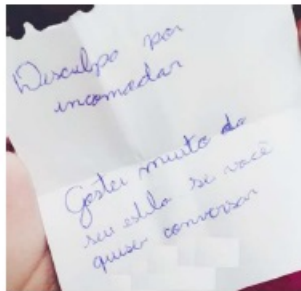


**LABEL:** Esperamos que seja de boa serventia essas máscaras, são poucas, mas de coração Que Deus nós Proteja! Belle Inês

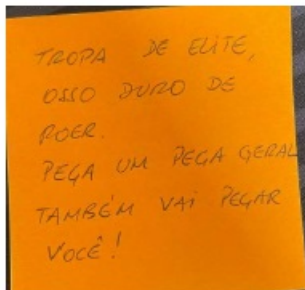
**0M2b.:** Esperamos que seja serventia essas máscaras, pão paucas, mas de Coração Que Deus nos Protejal Belle Ines

**AM3b.:** Esperamos que seja serventia essas máscaras, pão paucas, mas de Coração Que Deus nos Proteja Belle Ines

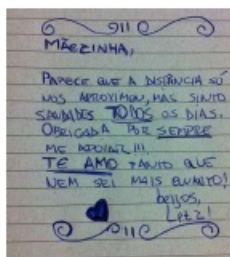
Figure C.6: Dataset images and texts 31 to 35.



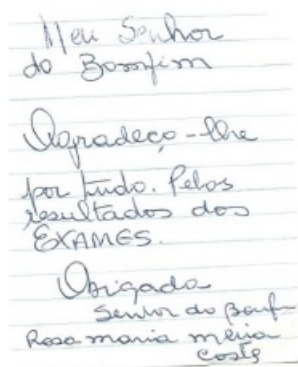
**LABEL:** Desculpe por incomodar Gostei muito do seu estilo se você quiser conversar  
**OM2b.:** por Desculpo incomadar ei muito da Gester seu estilo, se você quise conversar  
**FAZ**  
**AM3b.:** por Descalipo incomadar Goster ei muito da seu estilo se você quise conversar  
**FAZ**



**LABEL:** TROPA DE ELITE, OSSO DURO DE ROER. PEGA UM PEGA GERAL TAMBÉM VAI PEGAR VOCÊ!  
**OM2b.:** TROPA DE ELITE OSSO DURO DE ROER. PEGA UM PEGA GERAL TAMBÉM VAI PEGAR Você!  
**AM3b.:** TROPA DE ELITE OSSO DURO DE ROER. PEGA UM PEGA GERAL TAMBÉM VAI PEGAR Você! **NANDO**

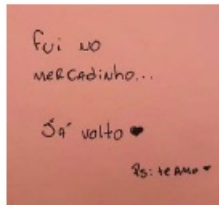


**LABEL:** MÃEZINHA, PARECE QUE A DISTÂNCIA SÓ NOS APROXIMOU, MAS SINTO SAUDADES TODOS OS DIAS. OBRIGADA POR SEMPRE ME APOIAR!!! TE AMO TANTO QUE NEM SEI MAIS QUANTO! beijos, Letz!  
**OM2b.:** 911 0 MÃECINHA, PARECE QUE A DISTANCIA SO NOS APROXIMOU, MAS SINTO SAUMDES TODOS OS DIAS. OBRIGADA POR SEMPRE ME APOIAR !!! TE AMO TANTO QUE MAIS QUANTO! NEM SEI Ple beijos, Letz!  
**AM3b.:** 9110 MÃECINHA, PARECE QUE A DISTANCIA SO NOS APROXIMOU, MAS SINTO SAUMDES TODOS OS DIAS. OBRICADA POR SEMPRE ME APOIAR !!! TE AMO TANTO QUE NEM SEI MAIS QUANTO! Ple beijos, Letz!

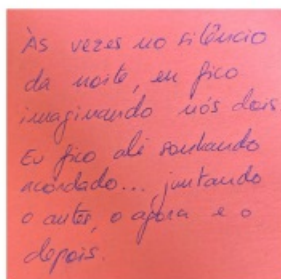


**LABEL:** Meu Senhor do Bomfim Agradeçi-lhe por tudo. Pelos resultados dos EXAMES. Obrigada Senhor do Bouf Rosa maria maia costa  
**OM2b.:** Meu Senhor do Bomfim Agradeço - the por tudo. Pelos resultados dos EXAMES. Obrigado Senhor do Bouf Resa maria meria costs  
**AM3b.:** Neu Senhor do Bomfim Agradeço - the por tudo. Pelos resultados dos EXAMES Obrigada Senhor do Bouf Rosa maria meria costs

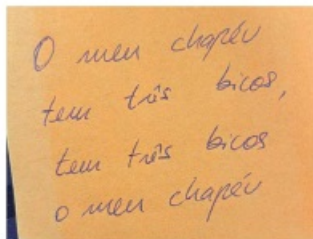
Figure C.7: Dataset images and texts 36 to 39.



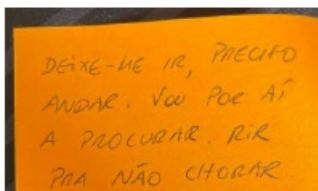
**LABEL:** Fui no mercadinho... Já volto PS: te amo  
**OM2b.:** fui MeR CADinho... NO Ja volto Ps: te AMO  
**AM3b.:** fui NO MERCADinho... Já volto Ps: te AMO -



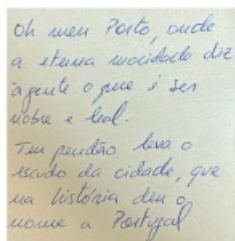
**LABEL:** Às vezes no silêncio da noite, eu fico imaginando nós dois. Eu fico ali sonhando acordado... juntando o antes, o agora e o depois.  
**OM2b.:** As vezes no silêncio da noite, en fico imaginando nós dois ali sonkando Eu fico acordado... juntando e o autes, o apora 0 depois.  
**AM3b.:** Às vezes no silêncio AS da noite, en fico imaginando nós dois ali souhando Eu fico acordado... juntando . auter, o apora depois. e o



**LABEL:** O meu chapéu tem três bicos, tem três bicos o meu chapéu  
**OM2b.:** 0 tem tis tem três meer o men chapéu bicos, bicos chapéi  
**AM3b.:** 0 tem três tem três meer o men chapéu bicos, bicos chapéi



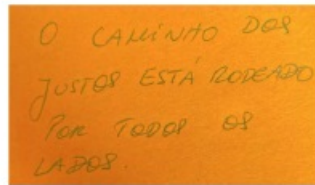
**LABEL:** DEIXE-ME IR, PRECISO ANDAR. VOU POR AÍ A PROCURAR. RIR PRA NÃO CHORAR.  
**OM2b.:** PRECIFO DEIXE-ME IR, ANDAR, VOU POR AT A PROCURAR. RIR PRA NÃO CHORAR  
**AM3b.:** PRECIFO ANDAR. VOU POR AT 2 DEIXE-HE IR, A PROCURAR. RIR PRA NÃO CHORAR



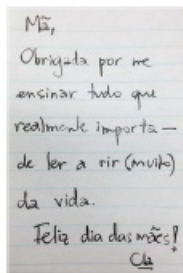
**LABEL:** Oh meu Porto, onde a eterna mocidade diz à gente o que é ser nobre e leal. Teu pendão leva o escudo da cidade, que na história deu o nome a Portugal  
**OM2b.:** oh men Porto, ande a étema mocidade diz è ser agente o que leal. nobre e Teu pendão leva o escudo da cidade, que na história den nome a Postyal IN  
**AM3b.:** oh men Porto, ande a etumma mocidade diz é ser agente o que beal. nobre e Teu pendão leva o escudo da cidade, que na história den nome a Portyal CANZO 2010 Y 70

Figure C.8: Dataset images and texts 40 to 44.

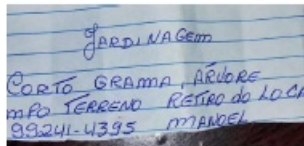




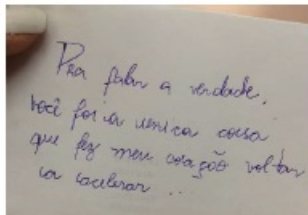
**LABEL:** O CAMINHO DOS JUSTOS ESTÁ RODEADO POR TODOS OS LADOS.  
**OM2b.:** O CAMINHO DOS JUSTOS ESTA RODEADO Jos POR TODO LADOS of  
**AM3b.:** O CAMINHO DOS JUSTOS ESTA RODEADO Jos Por TODOP LADOS of



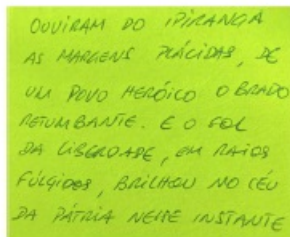
**LABEL:** Mã, Obrigada por me ensinar tudo que realmente importa - de ler a rir (muito) da vida. Feliz dia das mães! Cla  
**OM2b.:** 7 C Mã, Obrigada por me ensinar tudo que realmente importa- de ler a rir (muito) da vida. Feliz dia das mães! Cla  
**AM3b.:** e e e e C C O Mã, Obrigada por me ensinar tudo que realmente importa- de ler a rir (muito) da vida. Feliz dia das mães! Cla



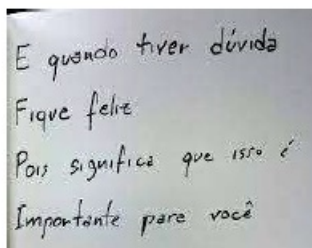
**LABEL:** JARDINAGEM CORTO GRAMA, ÁRVORE MPO TERRENO RETIRO DO LO CA 99241-4395 MANOEL  
**OM2b.:** JARDINAGEM CORTO GRAMA, ARVORE MPO TERRENO RETIRO do LO CE 99241-4395 MANDEL  
**AM3b.:** JARDINAGEM CORTO GRAMA, ARVORE MPO TERRENO RETIRO do LO CR 99241-4395 MANOEL



**LABEL:** Pra falar a verdade, você foi a unica coisa que fez meu coração voltar a acelerar ...  
**OM2b.:** Pra falar a verdade, você for a unica coisa que fez meu coração voltar 16 la cocelerar 902  
**AM3b.:** Pra falar a verdade, Ra você foi ia unica coisa que fez meu coração la lacelerar 902 voltan



**LABEL:** OUVIRAM DO IPIRANGA AS MARGENS PLÁCIDAS, DE UM POVO HERÓICO O BRADO RETUMBANTE. E O SOL DA LIBERDADE, EM RAIOS FÚLGIDOS, BRILHOU NO CÉU DA PÁTRIA NESSE INSTANTE  
**OM2b.:** OUVIRAM DO IPIRANGA AS MARGENS PLÁCIIONS, DE UM POVO HEROICO OBRADO NETUM BANTE. E O SOL DA LIBERDADE, EM RAies FULGIDOS, BRILHOU NO CÉU DA PATRIA NERE INSTANTE  
**AM3b.:** OUVIRAM DO IPIRANGA AS MARGENS PLÁCIDAS, DE UM POVO HEROICO NETUM BANTE. E O SOL DA LIBERDADE, EM RAios FÓLGIDOS, BRILHOU NO CÉU DA PATRIA NERE INSTANTE OBRADO



**LABEL:** E quando tiver dúvida Fique feliz Pois significa que isso é importante para você  
**OM2b.:** E quando tiver dúvida Fique feliz Pois significa que importante para 1550 é você  
**AM3b.:** E quando tiver dúvida Fique felie Pois significa que isso é importante para para você

Figure C.9: Dataset images and texts 45 to 50.

## Appendix D

# Handwritten Text Detection Code

In the following sections, we present the base code used in Experiment 2 (see section 4.2) with its most relevant excerpts and the modifications made during the experiment to obtain improvements.

The programming language used was Python under Google Colab PRO environment.

### D.1 Google Vision API Initial Code (base) - Without Preprocessing

Code start common to all presented solutions. The Google Vision API key has been hidden.

```
1 from google.colab import drive
2 import os
3
4 # Mount the Google Drive
5 drive.mount('/content/drive')
6
7 # Define directory paths
8 BASE_DIR = os.path.join('/content', '0.2.OCR_CodeVF', 'data')
9 IMAGES_PATH = os.path.join(BASE_DIR, 'images')
10 TXT_OUTPUT_PATH = os.path.join(BASE_DIR, 'txt')
11 PREPROCESSED_IMAGES_PATH = os.path.join(IMAGES_PATH, 'preprocessed0')
12
13 def create_required_directories():
14     """Create the required directories if they don't exist."""
15     os.makedirs(IMAGES_PATH, exist_ok=True)
16     os.makedirs(TXT_OUTPUT_PATH, exist_ok=True)
17     os.makedirs(PREPROCESSED_IMAGES_PATH, exist_ok=True)
18
19 def set_credentials():
20     # Set the path to the Google Cloud credentials
```



```

21     credentials_path = os.path.join('/content', '0.2.OCR_CodeVF', 'data', 'lively-
        shelter-000000-x00xxx00x000.json')
22     os.environ['GOOGLE_APPLICATION_CREDENTIALS'] = credentials_path

```

### D.1.1 Using ONLY Google Vision API

```

1  import re
2  from google.cloud import vision_v1
3
4  def load_image(file_name):
5      with open(file_name, 'rb') as image_file:
6          content = image_file.read()
7          return vision_v1.types.Image(content=content)
8
9  def ocr_image(client, image, language_hint):
10     image_context = vision_v1.types.ImageContext(language_hints=[language_hint])
11     response = client.document_text_detection(image=image, image_context=
        image_context)
12     return response.full_text_annotation
13
14 def save_text_to_file(text, image_file_name, output_path):
15     # Create a text file with the recognized text
16     output_file_name = os.path.splitext(image_file_name)[0] + '_model.txt'
17     output_file_path = os.path.join(output_path, output_file_name)
18     with open(output_file_path, mode='w', encoding='utf-8') as file:
19         file.write(text)
20     print(f'Text saved in {output_file_path}')
21
22 def get_image_files(images_path, valid_extensions=('jpg', 'jpeg', 'png')):
23     return [file for file in os.listdir(images_path) if file.lower().endswith(
        valid_extensions)]
24
25 def main(image_files):
26     set_credentials()
27     client = vision_v1.ImageAnnotatorClient()
28     for i, image_file in enumerate(image_files):
29         image_path = os.path.join(IMAGES_PATH, image_file)
30         image = load_image(image_path)
31         document = ocr_image(client, image, 'pt')
32         text = document.text.replace('\n', ' ')
33         print(f'----- Recognized Text {i+1} -----')
34         print(text)
35         output_file_name = f'recognized_text_{i+1}.txt'
36         save_text_to_file(text, image_file, TXT_OUTPUT_PATH)
37
38 if __name__ == "__main__":

```

```
39 image_files = get_image_files(IMAGES_PATH)
40 main(image_files)
```

### D.1.2 Function Used to Filter Portuguese Characters

```
1 def filter_portuguese_characters(text):
2     # Define a pattern to match Portuguese characters, digits, spaces, and some
      punctuation marks
3     pattern = r"[A-Za-z<Portuguese characters>0-9\s.,;:?!()-]+"
4     matches = re.findall(pattern, text)
5     return ' '.join(matches)
```

### D.1.3 Using Google Vision API and PySpellChecker

```
1 from spellchecker import SpellChecker
2
3 def correct_text(text, language='pt'): # Changed or added for this step
4     spell = SpellChecker(language=language)
5     words = text.split()
6     corrected_words = [spell.correction(word) if spell.correction(word) is not None
      else word for word in words]
7     return ' '.join(corrected_words)
```

### D.1.4 Using Google Vision API and PyEnchant

```
1 import enchant
2
3 def correct_text(text):
4     # Create an instance of the Enchant spell checker for the Portuguese language
5     spell_checker = enchant.Dict("pt_PT")
6
7     # Split the text into words
8     words = text.split()
9
10    # Iterate over the words and check for spelling errors
11    corrected_words = []
12    for word in words:
13        if not spell_checker.check(word):
14            suggestions = spell_checker.suggest(word)
15            if suggestions:
16                # Use the first suggestion as the corrected word
```

```

17         corrected_word = suggestions[0]
18         corrected_words.append(corrected_word)
19     else:
20         # If no suggestions are available, use the original word
21         corrected_words.append(word)
22     else:
23         corrected_words.append(word)
24
25     # Join the corrected words into a string
26     corrected_text = ' '.join(corrected_words)
27
28     return corrected_text

```

## D.2 Preprocessing Code - Option A

```

1 # OPTION A - path preprocessed1
2 def preprocess_image(image_path):
3     # Load and convert the image to grayscale
4     image = cv2.imread(image_path)
5     gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
6
7     # Resize and apply morphological operations
8     resized = cv2.resize(gray, None, fx=2, fy=2, interpolation=cv2.INTER_CUBIC)
9     kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))
10    dilated = cv2.dilate(resized, kernel, iterations=1)
11    eroded = cv2.erode(dilated, kernel, iterations=1)
12
13    # Save the preprocessed image
14    preprocessed_image_path = os.path.join(PREPROCESSED_IMAGES_PATH, os.path.
15        basename(image_path))
16    cv2.imwrite(preprocessed_image_path, eroded)
17
18    return preprocessed_image_path

```

The `preprocess_image` function performs several preprocessing steps on an image to prepare it for OCR (Optical Character Recognition). The steps are:

- **Loading and Grayscale Conversion:** The image is loaded and then converted to grayscale. Grayscale simplifies the image, reducing the amount of information the OCR needs to process.
- **Resizing:** The grayscale image is resized, in this case, the size is doubled along both the x and y dimensions. Resizing can help improve the visibility of text in the image, which can lead to better OCR results.

- **Morphological operations:** Dilation and Erosion are applied. Dilation is a process that adds pixels to the boundaries of objects in an image, while erosion removes pixels on object boundaries. The degree to which the image is dilated or eroded depends on the size and shape of the structuring element used to process the image. In this case, a 3x3 square structuring element is used.
- **Save the preprocessed image:** The processed image is then saved and the function returns the path of this preprocessed image. This processed image is then ready for OCR.

### D.3 Preprocessing Code - Option B

```

1 # OPTION B - path preprocessed2
2 def preprocess_image(image_path):
3     # Load and convert the image to grayscale
4     image = cv2.imread(image_path)
5     gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
6
7     # Apply noise reduction techniques
8     blurred = cv2.GaussianBlur(gray, (5, 5), 0)
9     median_blurred = cv2.medianBlur(blurred, 3)
10
11    # Resize and apply morphological operations
12    resized = cv2.resize(median_blurred, None, fx=2, fy=2, interpolation=cv2.
13        INTER_CUBIC)
14    kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))
15    dilated = cv2.dilate(resized, kernel, iterations=1)
16    eroded = cv2.erode(dilated, kernel, iterations=1)
17
18    # Apply adaptive thresholding for better binarization
19    thresholded = cv2.adaptiveThreshold(eroded, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C
20        , cv2.THRESH_BINARY, 11, 2)
21
22    # Save the preprocessed image
23    preprocessed_image_path = os.path.join(PREPROCESSED_IMAGES_PATH, os.path.
24        basename(image_path))
25    cv2.imwrite(preprocessed_image_path, thresholded)
26
27    return preprocessed_image_path

```

The `preprocess_image` function in this case follows a similar procedure to the previous one but adds a couple of additional preprocessing steps:

- **Noise Reduction:** After converting the image to grayscale, Gaussian blur and median blur are applied to the image to reduce noise. Noise can interfere with the OCR process, so reducing it can help improve OCR results.

- Adaptive Thresholding: After the morphological operations, adaptive thresholding is applied. Thresholding is the process of binarizing an image - i.e., converting it to a black and white image.

As before, the processed image is then saved and the function returns the path of this preprocessed image. This processed image is then ready for further processing or OCR.

## D.4 Preprocessing Code - Option C

---

```

1 # OPTION C - path preprocessed3
2 def correct_skew(image, delta=1, limit=10):
3     gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
4     thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)
5         [1]
6
7     coords = np.column_stack(np.where(thresh > 0))
8     angle = cv2.minAreaRect(coords)[-1]
9
10    if angle < -45:
11        angle = -(90 + angle)
12    else:
13        angle = -angle
14
15    if abs(angle) <= limit:
16        (h, w) = image.shape[:2]
17        center = (w // 2, h // 2)
18        rotation_matrix = cv2.getRotationMatrix2D(center, angle, 1.0)
19        rotated = cv2.warpAffine(image, rotation_matrix, (w, h), flags=cv2.
20            INTER_CUBIC, borderMode=cv2.BORDER_REPLICATE)
21
22        return rotated
23    else:
24        return image
25
26 def preprocess_image(image_path):
27     # Load the image and correct skewness
28     image = cv2.imread(image_path)
29     corrected = correct_skew(image)
30
31     # Convert the corrected image to grayscale
32     gray = cv2.cvtColor(corrected, cv2.COLOR_BGR2GRAY)
33
34     # Apply a slight Gaussian blur to reduce noise
35     blurred = cv2.GaussianBlur(gray, (3, 3), 0)
36
37     # Apply adaptive thresholding to create a binary image

```

```

36     thresh = cv2.adaptiveThreshold(blurred, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
37         cv2.THRESH_BINARY_INV, 11, 2)
38
39     # Erode the binary image to de-emphasize text areas
40     kernel = np.ones((1, 1), np.uint8)
41     eroded = cv2.erode(thresh, kernel, iterations=1)
42
43     # Save the preprocessed image
44     preprocessed_image_path = os.path.join(PREPROCESSED_IMAGES_PATH, os.path.
45         basename(image_path))
46     cv2.imwrite(preprocessed_image_path, eroded)
47
48     return preprocessed_image_path

```

This `preprocess_image` function follows a similar process as the previous ones, but includes an additional skew correction step:

- **Skew Correction:** The `correct_skew` function is used to correct the skew of an image. This can be helpful in improving OCR results, especially for documents that were scanned at an angle. The function uses the image's pixel coordinates to compute the rotation angle that would align the document properly. If the computed angle is within a specified limit (10 degrees by default), the image is rotated accordingly.
- **Noise Reduction and Thresholding:** The image is converted to grayscale, Gaussian blur is applied for noise reduction, and then adaptive thresholding is applied to create a binary image.
- **Erosion:** The binary image is eroded to de-emphasize text areas. This is done using a kernel of ones (a 1x1 matrix in this case) and the `erode` function in OpenCV.

After all these steps, the preprocessed image is saved and the path to the preprocessed image is returned.

## D.5 Preprocessing Code - Option D

```

1 # OPTION D - path preprocessed4
2 def preprocess_image(image_path, output_size=(800, 800)):
3     # Load and convert the image to grayscale
4     image = cv2.imread(image_path)
5     gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
6
7     # Apply Non-local Means Denoising
8     denoised = cv2.fastNlMeansDenoising(gray, None, 10, 7, 21)
9
10    # Resize the image while maintaining the aspect ratio

```

```

11     original_height, original_width = denoised.shape
12     aspect_ratio = float(original_width) / float(original_height)
13
14     if original_width >= original_height:
15         new_width = output_size[0]
16         new_height = int(new_width / aspect_ratio)
17     else:
18         new_height = output_size[1]
19         new_width = int(new_height * aspect_ratio)
20
21     resized = cv2.resize(denoised, (new_width, new_height), interpolation=cv2.
22         INTER_CUBIC)
23
24     # Save the preprocessed image
25     preprocessed_image_path = os.path.join(PREPROCESSED_IMAGES_PATH, os.path.
26         basename(image_path))
27     cv2.imwrite(preprocessed_image_path, resized)
28
29     return preprocessed_image_path

```

This version of the `preprocess_image` function is relatively simpler than the previous ones and focuses on two main aspects:

- **Noise Reduction:** After converting the image to grayscale, it applies Non-local Means Denoising, which is an effective noise-reducing algorithm. The parameters to this function (10, 7, 21) are the `h` value (which affects the strength of the filter), the `templateWindowSize` (odd size of the neighborhood window), and the `searchWindowSize` (size in pixels of the window that is used to compute a weighted average for the given pixel), respectively.
- **Image Resizing:** Then the function resizes the image to a certain output size while maintaining the original aspect ratio. If the width of the image is greater than or equal to its height, the width is set to the output width, and the height is adjusted to maintain the aspect ratio. Otherwise, the height is set to the output height, and the width is adjusted.

After these steps, the preprocessed image is saved and the path to it is returned.

## D.6 Preprocessing Code - Option E

```

1 # OPTION E - path preprocessed5
2 def preprocess_image(image_path, output_size=(800, 800)):
3     # Load the image
4     image = cv2.imread(image_path)
5
6     # Resize the image while maintaining the aspect ratio
7     original_height, original_width, _ = image.shape

```

```

8   aspect_ratio = float(original_width) / float(original_height)
9
10  if original_width >= original_height:
11      new_width = output_size[0]
12      new_height = int(new_width / aspect_ratio)
13  else:
14      new_height = output_size[1]
15      new_width = int(new_height * aspect_ratio)
16
17  resized = cv2.resize(image, (new_width, new_height), interpolation=cv2.
18      INTER_AREA)
19
20  # Save the preprocessed image
21  preprocessed_image_path = os.path.join(PREPROCESSED_IMAGES_PATH, os.path.
22      basename(image_path))
23  cv2.imwrite(preprocessed_image_path, resized)
24
25  return preprocessed_image_path

```

This version of the `preprocess_image` function is the simplest of all the previous ones and only focuses on one major preprocessing step: Image Resizing. It first loads the image, then resizes the image to a specified output size while maintaining the original aspect ratio. The resizing is done in a way that the image's original aspect ratio is preserved. The new width and height are calculated based on whether the original width is greater than or equal to the original height or not. After resizing, the image is saved and the path to it is returned.

## D.7 Preprocessing Code - Option F

```

1  # OPTION F - path preprocessed6
2  def preprocess_image(image_path):
3      # Load and convert the image to grayscale
4      image = cv2.imread(image_path)
5      gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
6
7      # Resize and apply morphological operations
8      resized = cv2.resize(gray, None, fx=2, fy=2, interpolation=cv2.INTER_CUBIC)
9      kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (1, 1))
10     dilated = cv2.dilate(resized, kernel, iterations=1)
11     eroded = cv2.erode(dilated, kernel, iterations=1)
12
13     # Save the preprocessed image
14     preprocessed_image_path = os.path.join(PREPROCESSED_IMAGES_PATH, os.path.
15         basename(image_path))
16     cv2.imwrite(preprocessed_image_path, eroded)

```



```
17     return preprocessed_image_path
```

This version of the `preprocess_image` function is very similar to the OPTION A, the only difference is the kernel size for the morphological operations: In this case, a 1x1 square structuring element is used which means no effect.

## D.8 PyTesseract OCR Code - Preprocessing Images

```
1  !pip install pytesseract
2
3  !sudo apt-get install tesseract-ocr
4  !sudo apt-get install libtesseract-dev
5  !sudo apt-get install tesseract-ocr-por
6
7  import pytesseract
8  import cv2
9  import numpy as np
10
11 def create_required_directories():
12     """Create the required directories if they don't exist."""
13     os.makedirs(IMAGES_PATH, exist_ok=True)
14     os.makedirs(TXT_OUTPUT_PATH, exist_ok=True)
15     os.makedirs(PREPROCESSED_IMAGES_PATH, exist_ok=True)
16
17 def correct_skew(image, delta=1, limit=10):
18     gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
19     thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)
20     [1]
21
22     coords = np.column_stack(np.where(thresh > 0))
23     angle = cv2.minAreaRect(coords)[-1]
24
25     if angle < -45:
26         angle = -(90 + angle)
27     else:
28         angle = -angle
29
30     if abs(angle) <= limit:
31         (h, w) = image.shape[:2]
32         center = (w // 2, h // 2)
33         rotation_matrix = cv2.getRotationMatrix2D(center, angle, 1.0)
34         rotated = cv2.warpAffine(image, rotation_matrix, (w, h), flags=cv2.
35             INTER_CUBIC, borderMode=cv2.BORDER_REPLICATE)
36
37     return rotated
38 else:
```

```

37     return image
38
39 def preprocess_image(image_path):
40     # Load the image and correct skewness
41     image = cv2.imread(image_path)
42     corrected = correct_skew(image)
43
44     # Convert the corrected image to grayscale
45     gray = cv2.cvtColor(corrected, cv2.COLOR_BGR2GRAY)
46
47     # Apply a slight Gaussian blur to reduce noise
48     blurred = cv2.GaussianBlur(gray, (3, 3), 0)
49
50     # Apply adaptive thresholding to create a binary image
51     thresh = cv2.adaptiveThreshold(blurred, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
52                                   cv2.THRESH_BINARY_INV, 11, 2)
53
54     # Erode the binary image to de-emphasize text areas
55     kernel = np.ones((1, 1), np.uint8)
56     eroded = cv2.erode(thresh, kernel, iterations=1)
57
58     # Save the preprocessed image
59     preprocessed_image_path = os.path.join(PREPROCESSED_IMAGES_PATH, os.path.
60                                             basename(image_path))
61     cv2.imwrite(preprocessed_image_path, eroded)
62
63     return preprocessed_image_path
64
65 def save_text_to_file(text, image_file_name):
66     """Save the OCR-recognized text to a file."""
67
68     output_file_name = os.path.splitext(image_file_name)[0] + '_model.txt'
69     output_file_path = os.path.join(TXT_OUTPUT_PATH, output_file_name)
70
71     with open(output_file_path, mode='w', encoding='utf-8') as file:
72         file.write(text)
73
74     print(f'Text saved in {output_file_path}')
75
76 def get_image_files(images_path, valid_extensions=('jpg', 'jpeg', 'png')):
77     """Return a list of image files with valid extensions."""
78
79     return [file for file in os.listdir(images_path) if file.lower().endswith(
80             valid_extensions)]
81
82 def main():
83     create_required_directories()
84     image_files = get_image_files(IMAGES_PATH)

```

```
83     for i, image_file in enumerate(image_files):
84         image_path = os.path.join(IMAGES_PATH, image_file)
85         preprocessed_image_path = preprocess_image(image_path)
86
87         # Apply OCR to the image
88         tessdata_path = os.path.join('/', 'usr', 'share', 'tesseract-ocr', '4.00',
            'tessdata')
89         config = f'--tessdata-dir {tessdata_path} --oem 1 --psm 6'
90         document = pytesseract.image_to_string(preprocessed_image_path, lang='por',
            config=config)
91
92         text = document.replace('\n', ' ')
93
94         print(f'\033[91m----- Recognized Text {i+1} -----\033[0m')
95         print(f'\033[1m' + text + '\033[0m')
96
97         save_text_to_file(text, image_file)
98
99 if __name__ == "__main__":
100     main()
```

# References

- Md Ali, Md Sipon Miah, Jahurul Haque, M Mahbubur Rahman, and Md Islam. An enhanced technique of skin cancer classification using deep convolutional neural network with transfer learning models. *Machine Learning with Applications*, 5:100036, 2021. doi: 10.1016/j.mlwa.2021.100036.
- Delora Baptista, João Correia, Bruno Pereira, and Miguel Rocha. A comparison of different compound representations for drug sensitivity prediction. pages 145–154, 2022. doi: 10.1007/978-3-030-86258-9\_15.
- Rui Camacho and Pavel Brazdil. Improving the robustness and encoding complexity of behavioural clones. pages 37–48, 2001. doi: 10.1007/3-540-44795-4\_4.
- B Cassidy, C Kendrick, A Brodzicki, J Jaworek-Korjakowska, and MH Yap. Analysis of the isic image datasets: Usage, benchmarks and recommendations. *Med Image Anal*, 75:102305, 2022. doi: 10.1016/j.media.2021.102305.
- Ben Chamblee. What is cosine similarity? how to compare text and images in python. <https://towardsdatascience.com/what-is-cosine-similarity-how-to-compare-text-and-images-in-python-d2bb6e411ef0>, 2022.
- Tim Cheng. Introduction to google vision ocr. <https://nanonets.com/blog/google-cloud-vision/>, 2022.
- Pedro Domingos and Kevin Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, 2012.
- Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.
- Anil K Jain and Rangachar Nayak. Multi-layer perceptron in handwritten character recognition. pages 383–392, 2011.
- Joos Korstanje. 3 text distances that every data scientist should know. <https://towardsdatascience.com/3-text-distances-that-every-data-scientist-should-know-7fcd850e510>, 2020.
- Adrien Lavecchia and Matthias Taulé. Deepchem: A software platform for deep learning in chemistry and drug discovery. *Journal of Chemical Information and Modeling*, 60(2):393–405, 2020.

- Andreas Panagopoulos, Timo Minssen, Katerina Sideri, Helen Yu, and Marcelo Corrales Compagnucci. Incentivizing the sharing of healthcare data in the ai era. *Computer Law Security Review*, 45:105670, 2022. doi: 10.1016/j.clsr.2022.105670.
- Poorvi Patel. The different image processing techniques to text from natural images priyanka muchhadiya. *Journal of Operating Systems Development Trends*, 1:1–7, 2014.
- Marina Ragoza and Alexey Aliper. Deep learning applications for predicting pharmacological properties of drugs and drug repurposing using transcriptomic data. *Molecular Pharmaceutics*, 14(2):772–788, 2017.
- V Rotemberg, N Kurtansky, B Betz-Stablein, et al. A patient-centric dataset of images and metadata for identifying melanomas using clinical context. *Sci Data*, 8:34, 2021. doi: 10.1038/s41597-021-00815-z.
- Jaswinder Singh, Parvinder Singh, and Yogesh Chaba. A study of similarity functions used in textual information retrieval in wide area networks. *International Journal of Computer Science and Information Technologies*, 5:7880–7884, 2014.
- Vikas Thada and Vivek Jaglan. Comparison of jaccard, dice, cosine similarity coefficient to find best fitness value for web retrieved documents using genetic algorithm. *International Journal of Innovations in Engineering and Technology*, 2:202–205, 2013.
- Uniqtech. Understand jaccard index, jaccard similarity in minutes. <https://medium.com/data-science-bootcamp/understand-jaccard-index-jaccard-similarity-in-minutes-25a703fbf9d7>, 2020.
- Xian-Sheng Wang, Zhi-Hua Chen, and Jian-Yong Li. Handwritten digit recognition using multi-layer perceptron neural network. pages 3570–3574, 2009.
- Weiqi Yan. *Computational Methods for Deep Learning: Theoretic, Practice and Applications*. 01 2021. ISBN 978-3-030-61080-7. doi: 10.1007/978-3-030-61081-4.