BACHELOR'S DEGREE THESIS

Degree in Statistics

Title: Advances in Diagnostic Imaging: Integrating Explainable AI to Optimize Convolutional Networks

Author: Xiuchao Guo

Advisor: Esteban Vegas Lozano

Department: Genetic, Microbiologic and Statistic, University of Barcelona

Academic year: June 2024

UNIVERSITAT DE BARCELONA

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat de Matemàtiques i Estadística

Abstract

Convolutional neural networks (CNNs) are fundamental in deep learning, especially in computer vision tasks. They stand out for their ability to extract spatial features from data. However, their complexity has generated the need for explainability in artificial intelligence (XAI), which seeks to interpret and understand their predictions. This work is carried out with the purpose of knowing the applicability of convolutional networks in the classification of medical images, specifically, endoscopy images already previously collected, and through fine-tunning we will explore architectures that present better performance. Afterwards, we implement the AI explainability techniques, together with the language model we will assess the process of automating the creation of the medical report through the graphic representations created.

Key Words: Convolutional Neural Networks (CNNs), Computer Vision, Explainability, Interpretations, Informed Decision-Making, Fine-tunning.

Resum

Les xarxes neuronals convolucionals (CNNs) són fonamentals en l'aprenentatge profund, especialment en tasques de visió per computadora. Destaquen per la seva capacitat d'extreure característiques espacials de les dades. No obstant això, la seva complexitat ha generat la necessitat d'explicabilitat en intel·ligència artificial (XAI), que busca interpretar i entendre les seves prediccions. Aquest treball es realitza amb la finalitat de conèixer l'aplicabilitat de les xarxes convolucionals en la classificació de les imatges mèdiques, en concret, imatges d'endoscòpia ja prèviament recollides, i mitjançant fine-tuning explorarem les arquitectures que presenten millor rendiment. A posteriori, realitzarem la implementació de les tècniques d'explicabilitat en IA, juntament amb el model de llenguatge, i valorarem el procés d'automatització de la creació d'informes mèdics mitjançant les representacions gràfiques creades.

Paraules Claus: Xarxes Neuronals Convolucionals (CNNs), Visió per Computadora, Explicabilitat, Interpretacions, Presa de Decisions Informada, Ajust Fi.

1 AMS Classification

- **68Txx Artificial Intelligence**
- **92B20 Neural networks, artificial life and related topics**
- **44A35 Convolution**
- **62P10 Applications to biology and medical sciences**
- **92C50 Medical applications (general)**
- **92C55 Biomedical imaging and signal processing [See also 44A12, 65R10, 94A08, 94A12]**

2 Acknowledgements

I would like to express my most sincere thanks to all the people who have contributed to the completion of this Final Degree Project.

First of all, I would like to thank my tutor, Professor Esteban, for his constant guidance and support throughout the whole process. His experience and advice have been fundamental for the completion of this work.

I would also like to thank myself for not giving up on myself and for carrying on in the most difficult moments.

Finally, I would like to thank my family for their unconditional love and support, which has given me the necessary strength to achieve this goal.

To all of you, thank you very much.

Contents

3 Introduction

The proliferation of large volumes of visual data in various disciplines has driven the need for automated and efficient methods for image classification and analysis. In the specific domain of medicine, the ability to accurately interpret medical images is critical for the timely diagnosis and treatment of disease. Deep learning, particularly using convolutional neural networks (CNNs), has emerged as a promising technique, demonstrating a remarkable ability to extract relevant image features and perform classifications with high levels of accuracy.

This project focuses on the exploration of CNNs for medical image . Given the complexity of medical images, CNNs offer a particularly apt approach to address the subtleties and variabilities present in these data, facilitating the identification of both important and unimportant patterns. In addition, the ability of CNNs to learn feature hierarchies directly from the data, without the need for manual feature extraction, makes them a powerful tool for automating medical image analysis.

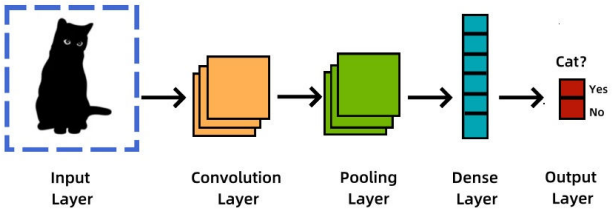


Figure 1: Simplification of CNN [2]

However, despite the demonstrated success of CNNs in image classification, interpretation of deep learning models remains a significant challenge. The ”black box” nature during the learning and final decision process may limit their acceptance in critical applications, especially in medical diagnostics, where understanding the reasoning behind a prediction is essential. Therefore, this work also addresses the application of explainable artificial intelligence (XAI) techniques to improve the transparency and understandability of deep learning models. By integrating XAI into the medical image classification workflow, we aim not only to improve the accuracy of predictions, but also to provide interpretable insights that can support clinical decision making.

And finally, we implement the classified images in a language model to generate an interim medical report, which can help the patient to orientate himself/herself to the disease in question. This interim report provides a generic description of the observations found in the medical images, as well as preliminary recommendations and possible next steps before consultation with a specialist.

The integration of language models in this process makes it possible to translate visual and technical data into an understandable and accessible language for patients. In this way, patients can receive clear and concise information about their health status, enabling them to make informed decisions and better prepare for future medical consultations. And at the same time providing greater understanding and reducing the anxiety associated with waiting for formal diagnoses.

4 Data Description

In this section, the data to be used in this project will be defined. From the available dataset, we will select a part to be used for training. In total, we will work with 8 classes of diseases, each with 500 images. In the following, a brief description of the data will be presented.

4.1 Kvasir Datasets

The next step in our work is to introduce the Kvasir dataset[9], which will be fundamental for the implementation and evaluation of our deep learning models. The Kvasir dataset is a collection of medical images specifically designed for research in the field of gastrointestinal endoscopy. This dataset has been compiled and curated by experts from the Research Group in Image Processing at the University of Tromsø in Norway.

The Kvasir dataset includes a wide variety of endoscopic images that cover different conditions and anatomical structures of the gastrointestinal tract. Among these images are representations of polyps, ulcers, Barrett's esophagus, and other pathologies as well as normal findings. The diversity and quality of the images in this dataset make it a valuable tool for the development and evaluation of artificial intelligence algorithms in the medical field.

Using the Kvasir dataset in our work will enable the construction of accurate and efficient models and the validation of their performance in a realistic clinical context. Additionally, since this dataset is widely recognized and used in the medical research community, our results can be compared and contrasted with previous studies, providing a solid foundation for evaluating our contributions.

4.2 Data Collection

The data for the Kvasir dataset is collected using endoscopic equipment at Vestre Viken Health Trust (VV) in Norway, which consists of four hospitals serving 470,000 people. A significant portion of the training data comes from the Bærum Hospital's large gastroenterology department, with plans to expand the dataset further in the future.

The images are meticulously annotated by one or more medical experts from VV and the Cancer Registry of Norway (CRN). The CRN, part of the South-Eastern Norway Regional Health Authority and organized under Oslo University Hospital Trust, conducts research to advance knowledge about cancer. It oversees national cancer screening programs aimed at preventing cancer deaths by early detection of cancers or pre-cancerous lesions.

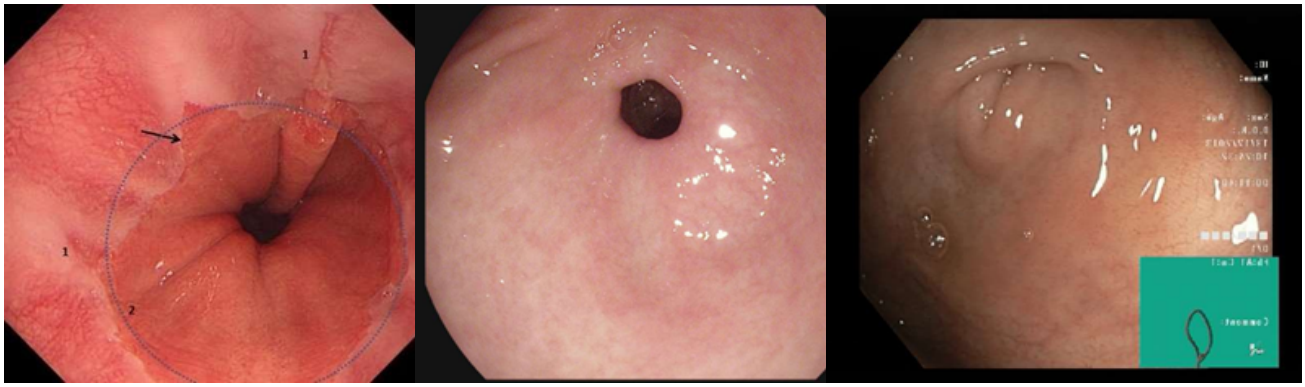
4.3 Datasets Details

The Kvasir dataset is a comprehensive collection of endoscopic images, meticulously annotated and verified by experienced medical doctors (endoscopists). It includes several classes of images that depict anatomical landmarks, pathological findings, and endoscopic procedures within the gastrointestinal (GI) tract, with hundreds of images available for each class.

Anatomical landmarks in the dataset include the Z-line, pylorus, and cecum. Pathological findings cover conditions like esophagitis, polyps, and ulcerative colitis. Additionally, the dataset contains images related to lesion removal procedures, such as ”dyed and lifted polyp” and ”dyed resection margins.”

The images in the Kvasir dataset vary in resolution and are organized into separate folders named according to their content.

4.3.1 Anatomical Landmarks

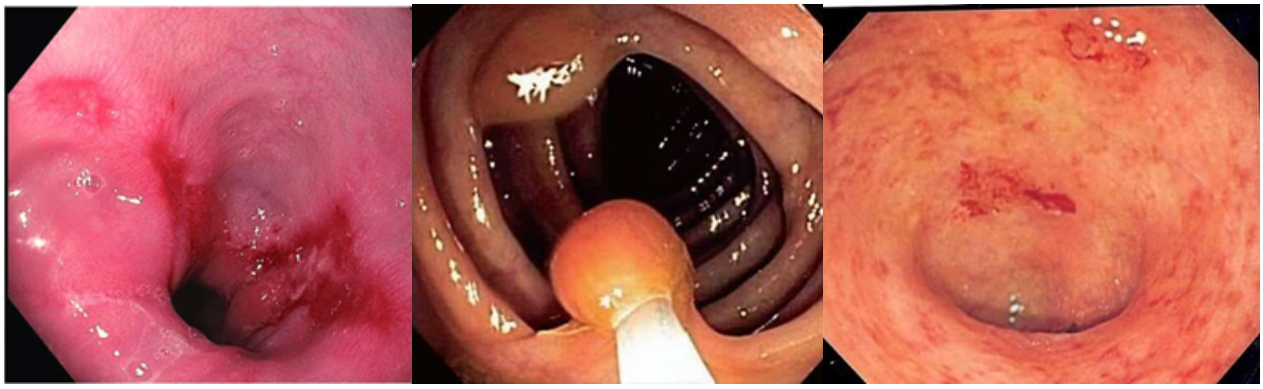


(a) Z-Line (b) Pylorus (c) Cecum

Figure 2: Anatomical Landmarks

- **Z-line**: The Z-line marks the transition between the esophagus and the stomach, visible as a border where the white esophageal mucosa meets the red gastric mucosa. It is crucial for diagnosing diseases like gastroesophageal reflux and serves as a reference point for describing esophageal pathologies.
- **Pylorus**: The pylorus is the opening from the stomach to the duodenum, regulated by muscles controlling food movement. Identifying it is essential for gastroscopy, which inspects both sides for issues like ulcerations or stenosis. The image to the left shows a normal pylorus as a smooth, dark circle surrounded by pink mucosa.
- **Cecum**: The cecum is the start of the large bowel, and reaching it confirms a complete colonoscopy, a key quality indicator. It's identified by features like the appendiceal orifice. The image to the left shows this orifice as a crescent-shaped slit, with a green inset showing the scope's position.

4.3.2 Phatological Findings

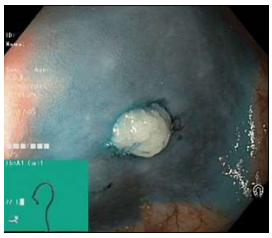


(a) Esophagitis (b) Polypos (c) Ulcerative Colitis

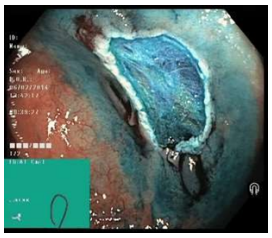
Figure 3: Anatomical Landmarks

- **Esophagitis**: Esophagitis is esophageal inflammation visible as mucosal breaks near the Z-line. The image to the left shows red projections into the white lining. Severity is graded by break length and circumference, often due to reflux, vomiting, or hernia.
- **Polypos**: Polyps are bowel lesions seen as mucosal outgrowths, varying in shape and distinguishable by color and surface pattern. While most are harmless, some can become cancerous, making detection and removal essential for preventing colorectal cancer.
- **Ulcerative Colitis**: Ulcerative colitis is a chronic inflammatory disease of the large bowel that significantly impacts quality of life. Diagnosis is primarily based on colonoscopic findings, with inflammation ranging from mild (swollen, red mucosa) to severe (prominent ulcerations).

4.3.3 Polyp Removal



(a) Dyed and Lifted Polyps



(b) Dyed Resection Margins

Figure 4: Polyp Removal

- **Dyed and Lifted Polyps**: The image to the left shows a polyp lifted by injecting saline and indigocarmine, with light blue margins clearly visible against darker mucosa. Automatic reporting can assess the success of the lifting and identify nonlifted areas that may indicate malignancy.
- **Dyed Resection Margins**: The resection margins are crucial for determining if a polyp is completely removed, as residual tissue can lead to continued growth or malignancy. Figure shows the resection site after polyp removal. Automatic recognition of these sites is valuable for reporting systems and assessing the completeness of polyp removal.

4.4 Data Description

After the detailed description of the data usage, we have identified the categories that will be used in our project based on the previous statements. The dataset is organized into 8 folders, each representing a distinct class. These folders are named according to their respective categories and contain 500 images each. This structure allows for straightforward and efficient utilization of the dataset in our project.

5 Methodology

In this chapter, we will outline the different methodologies [10] employed to achieve the goal of this project: applying convolutional neural networks (CNNs) to analyse and classify our data, seeking optimal predictive performance. We will start by introducing the key concepts of CNNs and their suitability for image analysis, focusing on their architecture and functionality.

Category
Dyed-lifted-polyps
Dyed-resection-margins
Esophagitis
Normal-cecum
Normal-pylorus
Normal-z-line
Polyps
Ulcerative-colitis

Table 1: Categories of Files

We will then detail the preprocessing steps applied to the Kvasir dataset, such as data augmentation and normalisation, to improve model performance. We will also explain the Artificial Intelligence explainability techniques, which brings the improvement at the time of decision making. And finally, we will explain a little bit about the language model, which will be the decisive tool that helps us to build the final pathology report.

5.1 Deep Learning

Before we start talking about CNN's, we first need to know what Deep Learning is. I guess many people have heard of artificial intelligence (AI), machine learning (ML) and deep learning (DL). But they are not clear about the difference between them. Then, Deep learning is a subfield within machine learning, which in turn is a subfield within artificial intelligence.

Artificial intelligence is the creation of machines that perform functions that require intelligence when performed by humans. Machine learning uses algorithms that learn without being explicitly programmed. Deep learning is the subset of machine learning that uses artificial neural networks that mimic how the brain works.

And Deep learning is a machine learning method that mimics the structure and function of the neural networks of the human brain. It learns and extracts features of data through multi-layered neural networks and uses these features to make predictions and decisions.Deep learning has been very successful in the fields of computer vision, natural language processing, and speech

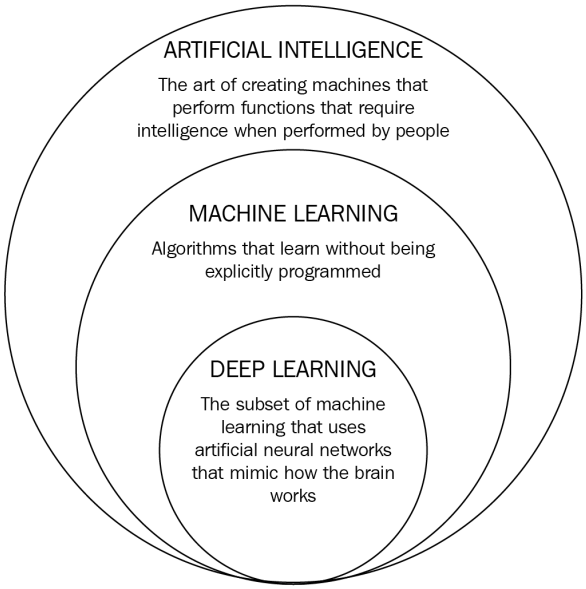


Figure 5: The relationship between AI, ML, and DL [11]

recognition, and it is able to deal with large-scale and complex data and learn more accurate patterns and regularities from it.

Some common deep learning algorithms:

- **Convolutional Neural Networks (CNN)**: Mainly used for processing and analysing data such as images and videos.
- **Recurrent Neural Networks (RNN)**: Mainly used for processing and analysing sequential data, such as speech recognition and natural language processing.
- **Generative Adversarial Networks (GAN)**: mainly used to generate new data samples, such as images and audio, etc.
- **Autoencoder**: Mainly used for data degradation and feature extraction, can be used in the field of image processing, recommender systems, etc.
- **Deep Belief NetworKs (DBN)**: Mainly used for unsupervised learning, can be used in the image and speech data such as feature extraction.
- **Residual Neural Networks (ResNet)**: Mainly used to generate new data samples, such as images and audio, etc.
- **Long Shor-Term Memory (LSTM)**: A special kind of recurrent neural network, which can better deal with long sequential data.

- **Attention Mechanism**: By giving different importance to different parts of the input to the model, to improve the model's effect on the processing of input data.

Advantages of Deep Learning:

- **High accuracy**: Deep learning models have surpassed the accuracy of human experts in many areas.
- **Handles large-scale and complex data**: Deep learning models can handle large-scale data, which improves the efficiency and accuracy of processing data.
- **Automatic feature extraction**: Deep learning models can automatically learn and extract features from data, eliminating the need for manual feature engineering.
- **Perform end-to-end learning**: Deep learning models can learn directly from raw data, eliminating the need for manual preprocessing.

That said, the main algorithm we employ in this work consists of the convolutional neural networks outlined above. These networks are particularly well-suited for tasks in the field of image processing due to their ability to automatically and adaptively learn spatial hierarchies of features. By leveraging these networks, we aim to improve the accuracy and efficiency of image analysis, enabling us to tackle complex problems such as object detection, image segmentation, and pattern recognition.

5.2 Convolutional Neural Network (CNN)

5.2.1 Composition of Convolutional Neural Networks

Convolutional Neural Networks (CNNs) have become one of the hottest topics in the field of artificial intelligence in the last 80-90 years and have played an irreplaceable role in various "image recognition" competitions. They have achieved results far superior to those of traditional digital image processing techniques, and these achievements are gradually being applied to various industries.

It's consisted of one or more convolutional layers, pooling layers, and fully connected layers. Compared to other deep learning architectures, CNNs can provide better results in areas such as image recognition. This model can also be trained using the backpropagation algorithm. In contrast to other shallow or deep neural networks, CNNs require fewer parameters to be considered, making them an attractive deep learning architecture.

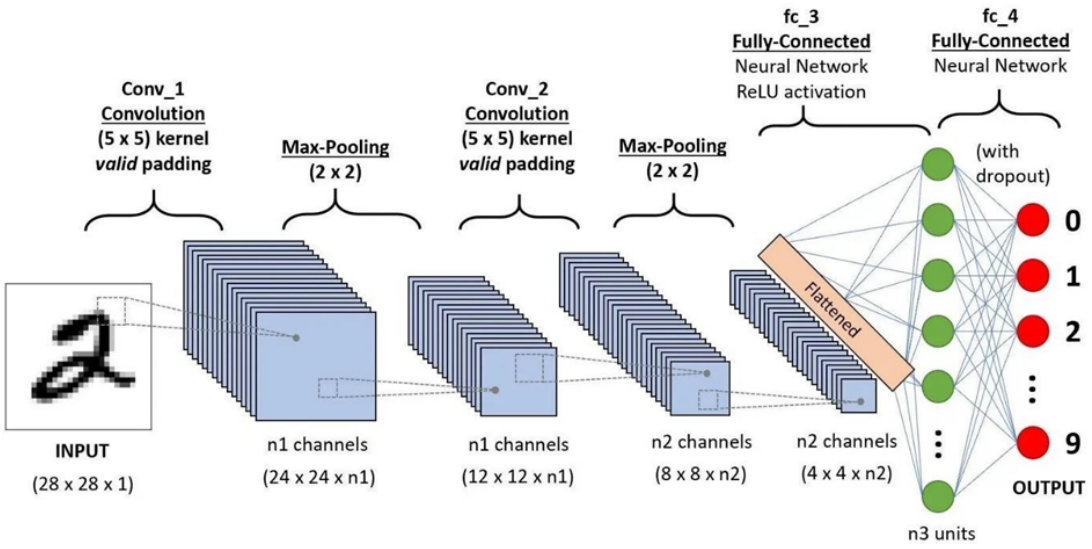


Figure 6: General Structure of CNNs [3]

In the above graph, we see a general representation in the process of running convolutional networks. In addition, in order to improve the generalization ability of the model, the following methods are usually adopted to optimize the training process, which effectively improves the ability of the model to adapt to new data. These methods are:

- **Input layer:** Image data as input
- **Convolutional layers:** The image contains local features that can be extracted.
- **Pooling layer:** Prevent overfitting, reduce the number of parameters.
- **Fully Connected layers:** Sparse connections, each convolution kernel is equivalent to a feature extractor.
- **Output layer:** Connect the output of the convolutional layer to a fully connected layer for the result.

As we can see, the convolutional layer is the most important part of the process, which is why it is named convolutional neural network.

5.2.2 Input Layer

The input layer is relatively simple; its primary job is to take in information such as images. Since Convolutional Neural Networks mainly deal with image-related content, do the images seen by our eyes and processed by the computer look the same? Obviously, they are not the same. For the input image, the first step is to convert it into a correspondence matrix. This three-dimensional matrix is composed of the pixel values of each pixel in the image. We can look at an example, such as the image of the handwritten digit '8' shown below. After the computer reads it, it is stored as a two-dimensional matrix formed by the pixel values.

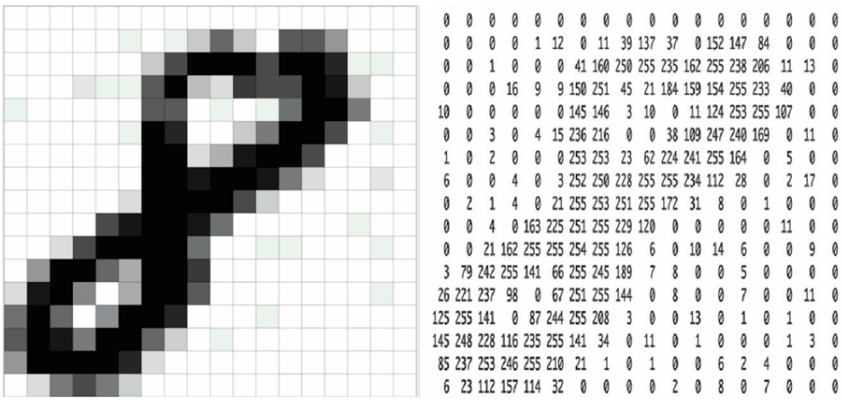


Figure 7: Exemple [6]

The image above is a three-dimensional image that identifies three values: length, width, and depth. Depth here refers to the identification of image colour. The above example is also known as a grayscale image, because each pixel value ranges from 0-255 (from pure black to pure white), indicating the degree of its colour strength, and there are also black and white images, each pixel value is either 0 (indicating pure black) or 255 (indicating pure white).

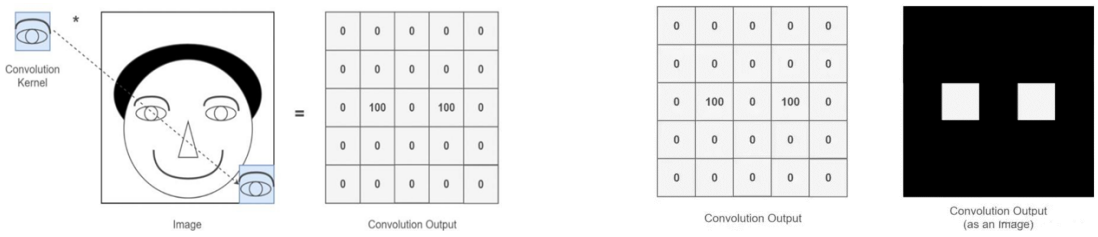
As we now know, the role of the input layer is to convert the image into its corresponding two-dimensional matrix of pixel values, and store this two-dimensional matrix, waiting for the operation of the next layers.

5.2.3 Convolutional Layers

The purpose of convolutional operations is to extract different features from the input. Some convolutional layers may only be able to extract some low-level features such as edges, lines, and corners of the layers, and more layers of the net can iteratively extract more complex features

from the low-level features. So, let's re-enact a simplified example to understand and concretize this one step.

What do we do with the image when it comes in? Assuming we have the 3D matrix of the image and want to extract the features, the convolution operation determines a high value for the region where the feature is present and a low value otherwise. This is done by calculating the value of its product with the convolution kernel. Suppose our input image is now a human head and human eyes are the features we need to extract, then we will use human eyes as the convolution kernel and determine where the eyes are by moving over the image of the human head, the process is shown in Example 1.



(a) Example 1 (b) Example 2

Figure 8: Two images aligned side by side

Through the entire convolution process and get a new two-dimensional matrix, this two-dimensional matrix is also known as the feature, and finally we can get the feature map for the colouring process (just making an analogy, such as the high value for the white, the low value for the black), and finally can be extracted about the characteristics of the human eye, as can see in Example 2.

After reading the description of the above example, the first thing you should know is that the convolution kernel is also a three-dimensional matrix, of course, this three-dimensional matrix is smaller or equal to the two-dimensional matrix of the input image, the convolution kernel by constantly moving on the two-dimensional matrix of the input image, and every time you move it, you will carry out a summation of the product as the value of the position, the process is shown in the following figure.

As you can see, the whole process is a process of dimensionality reduction, through the convolution kernel is constantly moving the calculation, you can extract the most useful features of the image, we usually will be the convolution kernel calculation of the new two- dimensional matrix is called the feature map, for example, above the moving picture, below the moving dark blue squares is the convolution kernel, above the non-moving cyan squares is the feature map.

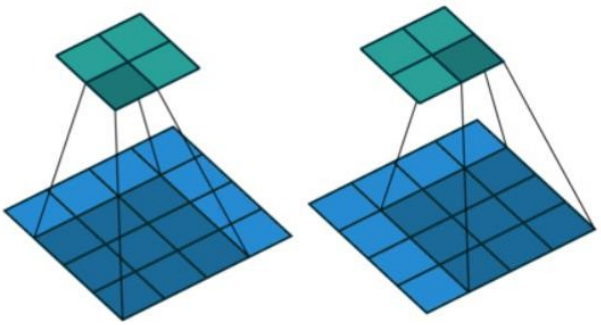
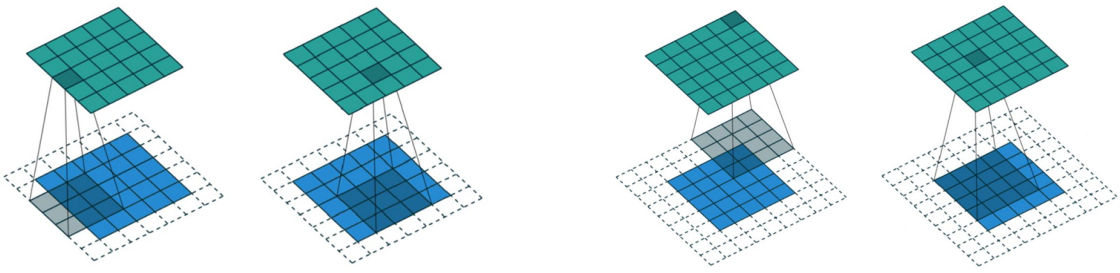


Figure 9: Kernel-convolution motion [4]

Then it should draw attention to the fact that every time the convolution kernel moves the middle position is computed, while the edges of the 2D matrix of the input image are computed only once, will this lead to inaccurate results of the computation? In order to do so, it is necessary to know the following parameters that should be utilized in the convolutional process.

- **Size:** Size of the convolution kernel/filter, typically 1x1, 3x3, or 5x5 (odd numbers).
- **Padding:** Zero padding.
- **Stride:** Step size, usually defaults to 1.

So, after all, let's think carefully. If each calculation, the edge is calculated only once, and the middle is calculated many times, then the feature map will be lost edge features, which will ultimately lead to inaccurate feature extraction, then in order to solve this problem, we can be in the original input image of the two-dimensional matrix around the expansion of one or more circles, so that each position can be fairly calculated, and will not be lost any features, the process can be seen in the following two cases, the expansion of this method to solve the problem of feature loss is also known as Padding. The above is a graphical representation of a convolutional layer, but of course, behind this lies a more rigorous expression of formulas to accurately define the entire process from input to output.



(a) Padding = 1 (b) Padding = 2

Figure 10: Padding takes the value of x, expanding a circle [4]

Suppose we have the following parameters:

H: Height W: Width D: Depth

Input size (image): $H_1 \times W_1 \times D_1$

Convolution operation with 4 hyperparameters:

K: Filter size F: Receptive field size S: Step P: N. of paddings

Output size: $H_2 \times W_2 \times D_2$

$$H_2 = \frac{H_1 - F + 2P}{S} + 1 \qquad W_2 = \frac{W_1 - F + 2P}{S} + 1 \qquad D_2 = K$$

5.2.4 Multi-channel convolution

When the input has more than one channel (for example, the image can have three channels of RGB), the convolution kernel needs to have the same number of channels, each convolution kernel channel is convolved with the corresponding channel of the input layer, and the convolution result of each channel is summed up by bit to get the final Feature Map.

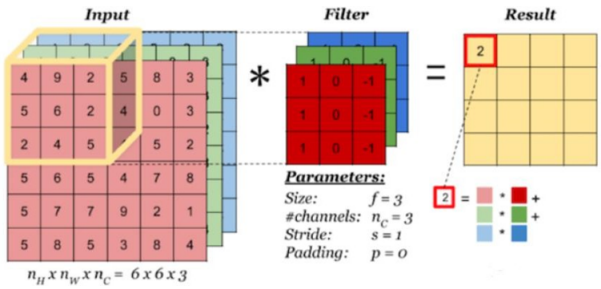


Figure 11: Convolution operation where a 3x3 filter with 3 channels [15]

5.2.5 Multiple Convolutional Kernels (Multiple Filters)

When there are multiple convolutional kernels, many different features can be learned, which corresponds to a Feature Map with multiple channels, e.g., there are two filters in the above figure, so there are two channels in the output. how many convolutional kernels can also be

interpreted as how many neurons. So, the convolutional layer acts as a feature extraction but does not reduce the number of features in the image, the fully connected layer at the end still faces many parameters, so a pooling layer is needed for the reduction of the number of features.

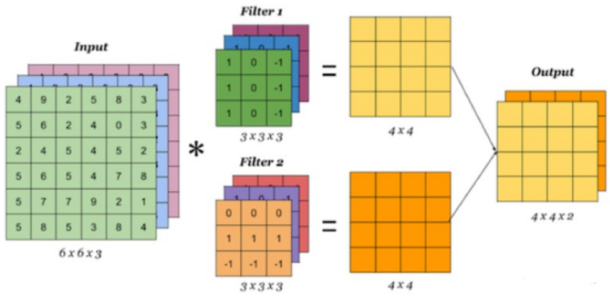


Figure 12: Convolution operation with multiple filters [7]

5.2.6 Activation Function

Each neuron node in a neural network accepts the output value of the neuron in the previous layer as the input value of this neuron and passes the input value to the next layer, where the neuron node in the input layer passes the value of the input attribute directly to the next layer (the hidden layer or the output layer). In a multi-layer neural network, there is a functional relationship between the output of the upper layer nodes and the input of the lower layer nodes, which is called the activation function (also known as the excitation function).

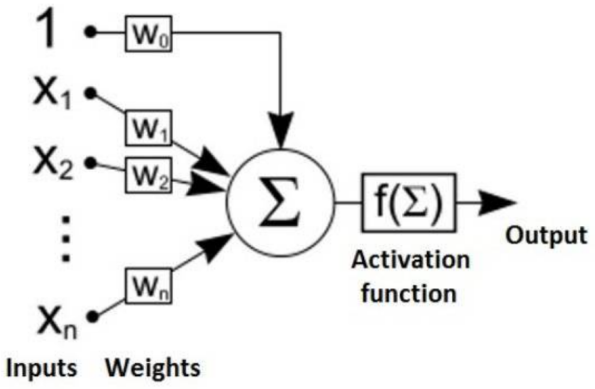
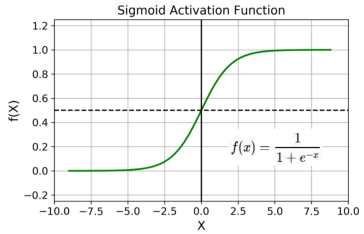


Figure 13: Activation function

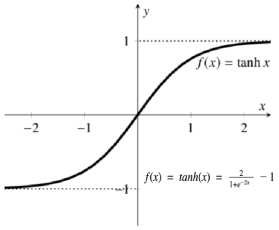
5.2.7 Why needs activation function

If you do not use the excitation function, in this case you each layer of the node's input is a linear function of the output of the upper layer, it is easy to verify that, no matter how many layers of your neural network, the output is a linear combination of the inputs, and the effect of the absence of a hidden layer is comparable to that of the most primitive perceptual machine, then the network's approximation ability is quite limited. Because of the above reasons, we decided to introduce a nonlinear function as the excitation function, so that the deep neural network expression is more powerful (no longer a linear combination of inputs, but can be approximated by almost any function).

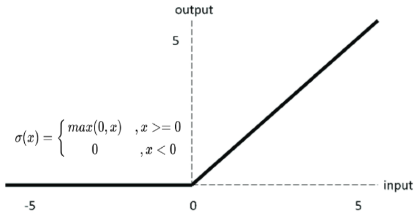
Let's take a brief look at a few of the most commonly used activation functions in the field of deep learning as follows



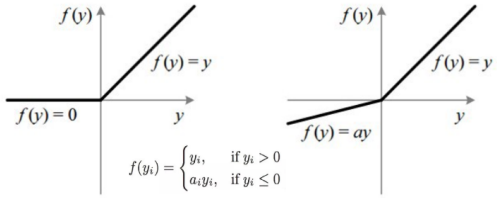
(a) Sigmoid



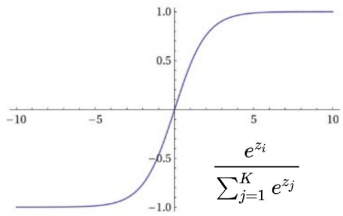
(b) Tanh



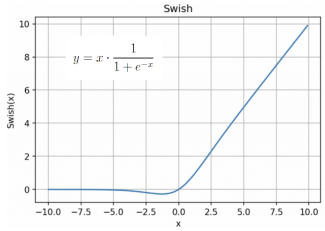
(c) ReLU



(d) Leaky ReLU



(e) Softmax



(f) Swish

Figure 14: Common types of activation function [8]

Knowing this, the application of each type of activation function varies greatly depending on the situation. Each has its own characteristics, leading to certain advantages and disadvantages. For example, activation functions such as ReLU (Rectified Linear Unit) tend to be very effective

in deep neural networks due to their ability to handle the fading gradient problem and allow for more efficient training. On the other hand, sigmoid and tanh functions, while useful in certain applications, can suffer from saturation at their extremes, which can slow down learning.

The appropriate use of activation functions will depend on the specific context of the task being addressed. In some cases, a trigger function may provide better performance in terms of convergence and model accuracy, while in other cases it may not be the optimal choice. It is essential to consider the type of problem, the neural network architecture and the available data when selecting the most appropriate activation function.

5.2.8 Pooling Layer

There are as many feature maps as there are convolution kernels, in reality the situation is certainly more complex, there will also be more convolution kernels, then there will be more feature maps, when there are very many feature maps, it means that we get very many features, but obviously there are a lot of features that we do not vote for, and these redundant features usually give us the following two problems:

· Overfitting · Over dimensionality

To solve this problem, we can use the clustering layer, that is, when we carry out the convolution operation, and then the resulting feature map for feature extraction, the most representative features extracted, can play a role in reducing overfitting and reduce the role of dimensionality, the process is as follows:

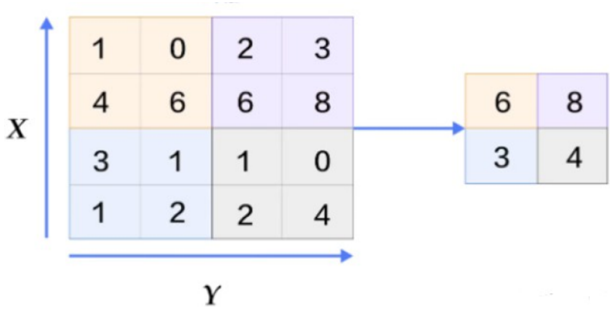


Figure 15: Extraction of key features [12]

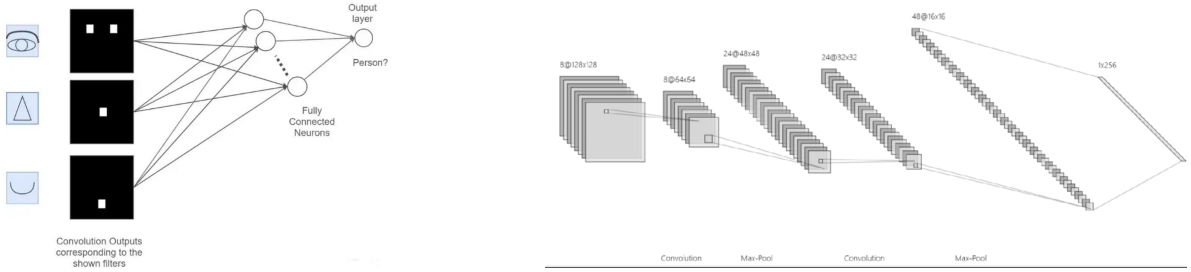
The pooling layer is mainly used for subsampling the feature maps learned by the convolutional layer, which consists of two main types:

- Max Pooling, which takes the maximum value in the window as the output.
- Average Pooling, which takes the average of all the values in the window as the output.

To: Reduce the input dimension of subsequent network layers, reduce model size and increase computational speed. And improve feature robustness and avoid overfitting.

5.2.9 Fully Connected Layers

Assuming that the above example of the human head, now we have extracted the human eyes, nose and mouth features through the convolution and pooling, if I want to use these features to identify whether the picture is a human head what to do? At this point, we just need to extract all the features of the map to ”flatten”, its dimension to 1xN, this process is the process of full connectivity, that is to say, this step will be all the features of the unfolding and operation, and finally will get a probability value, this probability value is the probability that the input image is a person, this process is as follows The process is as follows:



(a) Example 1 (b) Example 2

Figure 16: Two exemples of full connected layers [13]

It can be seen that, after two convolution and maximum pooling, the final feature map is obtained, at this time the features are obtained after the calculation, so the representation is relatively strong, and finally after the fully connected layer, unfolded into a one-dimensional vector, and then after another calculation, to get the final probability of recognition, which is the whole process of convolutional neural network.

5.3 Output Layer

The output of the convolutional neural network simply takes the one-dimensional vector obtained from the fully connected layer and calculates it to get a probability of the recognized value, of course, this calculation may be linear or non-linear. In deep learning, we need to recognize the results are generally multi-classification, so each position will have a probability value, representing the probability of only don't for the current value, take the largest probability value, is the final recognition results. In the process of training, you can continuously adjust the parameter values to make the recognition results more accurate, to achieve the highest model accuracy.

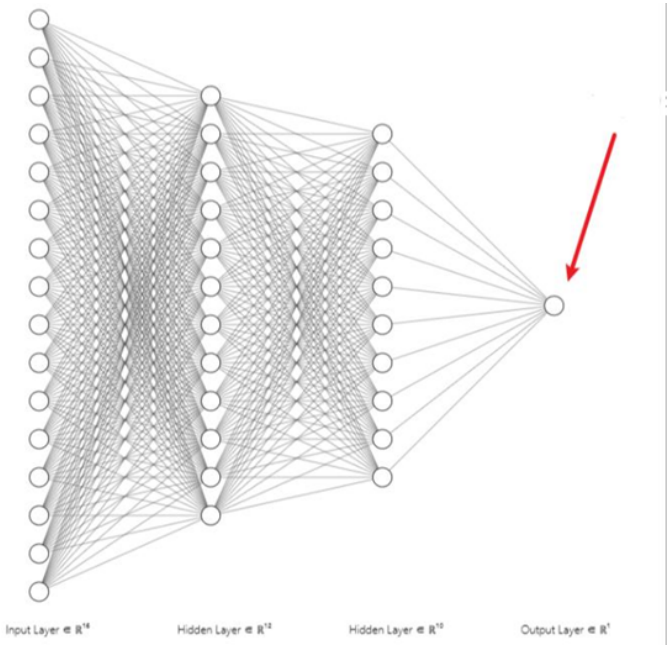


Figure 17: Output process representation

In addition to adjusting parameters, techniques like regularization, dropout, and data augmentation can improve model performance and prevent overfitting. Regularization penalizes large weights, promoting simpler models that generalize better. Dropout randomly deactivates a fraction of neurons during training, preventing reliance on specific units. Data augmentation enhances the training dataset with transformations such as rotation, scaling, and flipping, increasing model robustness to input variations.

5.4 Process Summary

Finally, we are going to see an example of handwritten number recognition, which is one of the basic and fundamental applications in the field of image processing. We will detail step by step each procedure following the explanations we have seen previously.

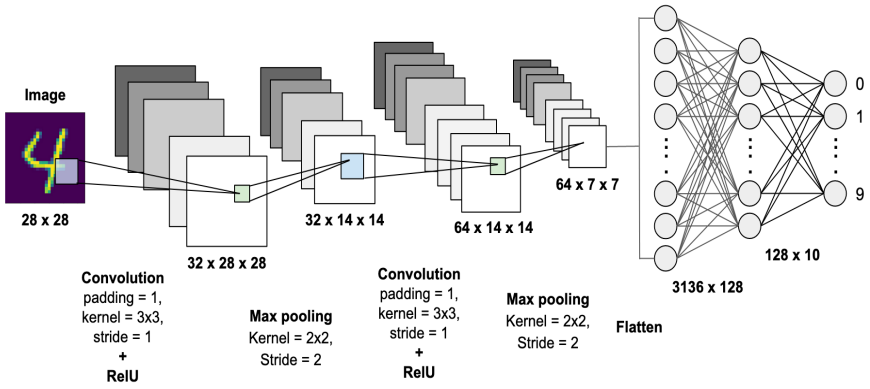


Figure 18: MNIST dataset [14]

1. Convert the handwritten digital image into a pixel matrix.
2. Perform a convolution operation on the pixel matrix with Padding not 0, with the aim of preserving edge features and generating a feature map.
3. Convolution operation is performed on this feature map using x convolution kernels to obtain x feature maps.
4. Pooling operation is performed on each feature map to reduce the data stream while preserving the features, generating x small maps, which look similar to the respective feature maps in the previous layer, but with reduced size.
5. A second convolution operation is performed on the x maps obtained after the pooling operation, generating more feature maps.
6. Pooling operation (down sampling operation) is performed on the feature maps generated by the second convolution operation.
7. The second pooling operation will be the first fully connected features.
8. Perform the second fully connect on the results of the first fully connect.
9. Finally each position (from 0 to 9) has a probability value, which is the probability of recognising the input digit as a digit in the current position, and finally the value of the position with the highest probability is used as the recognition result.

6 Explainable Artificial Intelligence

6.1 Prologue

The rapid development and wide application of Artificial Intelligence (AI) has brought many unprecedented opportunities and challenges. However, as the algorithms of AI become more and more complex and intelligent, people's understanding of the decision-making process and principles of AI system becomes more and more scarce, and this 'black box' phenomenon not only limits people's trust in AI system, but also creates a series of ethical and legal problems. To solve this problem, Explainable Artificial Intelligence (XAI) has emerged.

The goal of Explainable Artificial Intelligence is to improve the intelligibility and comprehensibility of the system, so that people can better understand the decision process, principles and reasoning schedule of the A system. By suggesting the algorithms, data, and features behind the AI system, XAI makes complex decisions more explainable and trustworthy. This not only helps to build trust in AI, but also helps to identify biases and unfairness in AI systems and provide directions for improvement.

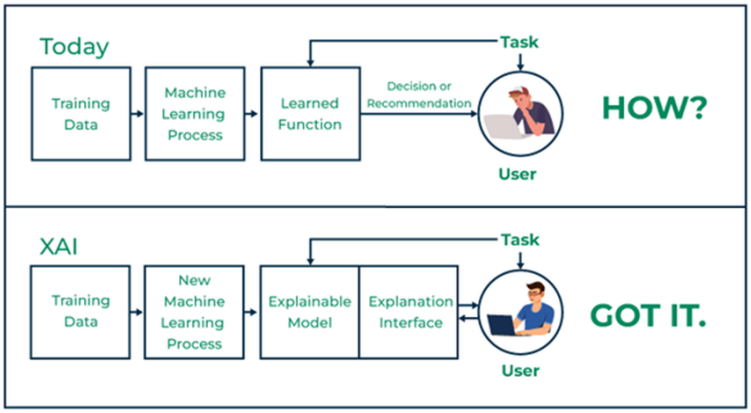


Figure 19: Comparison between traditional machine learning and explainable AI processes. [5]

XAI's research and applications cover multiple human aspects. From the perspective of algorithm improvement, researchers work on designing and developing more interpretable AI algorithms, such as methods based on rules, reasoning and causality, and methods that integrate human expert knowledge.

6.2 Why Interpretable AI is needed

Understanding the correct decision-making mechanism of AI models is an important way to enhance human trust in AI models. Existing research results on AI interpretability reveal that the data-based system decision-making mechanism is still far from the goal of obtaining human trust in at least the following two areas.

- **Theoretical deficiencies of machine learning decision-making mechanisms:**

  Nowadays, machine learning methods usually establish associations between input data and expected results, but due to the limitations and biases of data samples, this kind of association learning inevitably learns a kind of spurious relationship. To discover the true causal relationship, it is necessary to expand the observed phenomena through active intervention experiments and apply Counterfactual Reasoning to remove the spurious relationship.

- **Machine learning application flaws:**

  Limitations and biases in data samples can lead to biases in data-driven AI systems. The 'black box' deep learning network has security risks. From the perspective of the decision-making mechanism, the current analysis of deep learning is still in the opaque groping stage.

6.3 Interpretable AI basic categories

Interpretable AI aims to make artificial intelligence models more transparent and understandable to humans. Here are the basic categories[1]:

- **Model Specific Explainability:**

  Strictly limited to a specific model algorithm explainability, such as decision tree models, Bayesian networks, etc.

- **Model Agnostic Explainability:**

  This type of explanation applies to any type of machine learning model. Typically, a post-analytical approach will be used after the model has been trained, which does not depend on any particular algorithm and does not understand the internal structure and weights.

- **Model Centric Explainability:**

  Most explanatory methods are model-centric because they use stemming to explain how to adjust features and target values, apply various algorithms, and extract specific result sets.

- **Data Centric Explainability:**

  Data plays an important role in model training and prediction, and these methods are mainly used to understand the meaning of the data. Common methods include Data Profiling.

- **Several useful interpretability tools:**

  - **LIME:** The Local Explanatory Model-Sensitivity (LIME) algorithm generates local explanations that help us understand the decision-making process of complex models on individual predictions.
  - **SHAP:** The SHAP method, based on Shapley values, provides a general, interpretable feature importance measure that can explain a wide variety of models, including deep learning.
  - **Visualisation techniques:** Using visualization techniques to show the internal structure and workings of a model, such as activation heat maps for neural networks and structural diagrams for decision tree models, helps to understand the model more intuitively.

- **Case Study:** In the financial sector, interpretable AI is used to interpret and analyse credit scoring models, helping credit institutions to understand and explain the basis of their decisions and improve customer acceptance.

- **Business Integration:** Business knowledge is combined with AI technology to improve the usefulness and interpretability of the model. For example, in the medical field, the diagnosis and treatment plans recommended by AI are interpreted and adjusted in conjunction with the doctor's expertise.

Next, we'll focus on some of the XAI methods we'll be using in this article to make it easier to understand what we'll be doing in the future.

6.4 Shapley Value

The Shapley value is a concept in the theory of cooperative games, introduced by Lloyd Shapley in 1951, for which he won the Nobel Prize in Economics in 2012. For each cooperative game, the total lift of the model generated by the institutions can be formed into an efficient distribution of contributions across the institutions. In contrast to traditional games that consider individuals to be independent of each other and analyze their Nash equilibrium, cooperative games consider the collaborative relationship between each player and analyze scenarios such as the $1 + 1 > 2$ gains that can occur in cooperation. A co-operative game typically contains N players, and a value function v for evaluating the gains from cooperation between different players, and $v(\gamma) = 0$. Shapley values are characterized by a range of desirable properties, including the following:

1. **Symmetry:** The distribution of co-operative profits does not vary according to each person's mark or order in the co-operation.
2. **Effectiveness:** The sum of the profits of the co-operating parties is equal to the profits of the co-operation.
3. **Redundancy:** If a member does not contribute to any of the co-operative unions in which he participates, he should not benefit from the co-operation as a whole.
4. **Independence of irrelevant alternatives:** When there are multiple co-operations, the way in which the benefits of each co-operation are distributed is independent of the results of the other co-operations.

6.5 Definition

Given a cooperative game with a set of players $N = \{1, 2, \ldots, n\}$ and a characteristic function $v : 2^N \to \mathbb{R}$ that maps each coalition $S \subseteq N$ to a real number $v(S)$ representing the total worth of that coalition, the Shapley value $\phi_i(v)$ for a player $i \in N$ is given by:

$$\phi_i(v) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(n - |S| - 1)!}{n!} \left(v(S \cup \{i\}) - v(S)\right)$$

- $|S|$ is the number of players in coalition S.
- n is the total number of players.
- $v(S)$ is the contribution of player i.
- $v(S \cup \{i\}) - v(S)$ is the marginal contribution of player i to the coalition S.

6.6 Kernel-Shap (Shapley additive explanations)

Kernel SHAP is a machine learning model interpretation technique based on the concept of Shapley values from game theory explained above. Kernel SHAP adapts this concept for machine learning models, providing a way to explain model predictions. The Kernel Shap is a combination of LIME (where you can find a detailed explanation in the appendix) and Shapley values.

- LIME is an additive feature attribution method.
- An additive feature attribution method satisfies three properties: Local accuracy, Missingness, and Consistency.
- Shap-values also satisfy these three properties under the constraints of the LIME framework.

Thus, these two approaches are combined in a unified framework. LIME can be interpreted as a framework rather than a specific algorithm because the choices of loss function L, weighting kernel π_x, and regularization term Ω are not specified. Kernel Shap, on the other hand, provides a concrete implementation that satisfies the three properties mentioned above.

Under Definition 1, the specific forms of π_x, L, and Ω that make solutions of Equation 2 consistent with Properties 1 through 3 are:

$$\Omega(g) = 0,$$

$$\pi'_x(z') = \frac{(M-1)}{\binom{M}{|z'|}|z'|(M-|z'|)},$$

$$L(f, g, \pi'_x) = \sum_{z' \in Z} \left[f(h_x^{-1}(z')) - g(z') \right]^2 \pi'_x(z'),$$

where $|z'|$ is the number of non-zero elements in z'.

Kernel-Shap leverages the strengths of both LIME and Shapley values to provide a robust method for explaining machine learning model predictions. By combining these techniques, Kernel-Shap ensures that the explanations are both locally accurate and consistent with the theoretical properties of Shapley values.

6.7 Features of Kernel-Shap

1. **Local Accuracy**: This property ensures that the sum of the feature attributions (Shapley values) is equal to the output of the model for the specific instance being explained. In other words, the explanation accurately reflects the model's behavior for that instance.

$$f(x) = g\left(x'\right) = \phi_0 + \sum_{i=1}^{M} \phi_i x'_i$$

2. **Missingness**: This property dictates that features which are missing (or have no impact) in the model should have a Shapley value of zero. This ensures that only the relevant features are given importance in the explanation.

$$x'_i = 0 \implies \phi_i = 0$$

3. **Consistency**: If a model changes such that a feature contributes more to the prediction (without changing other features), the Shapley value for that feature should not decrease. This ensures that the explanations are consistent with the model's behavior.

7 Grad-CAM

As mentioned at the beginning of this section, one of the essential tools for AI interpretability, in addition to Kernel-Shap, is the use of visualization techniques. Visualization techniques play a crucial role in understanding and interpreting the decisions made by AI models, particularly in the context of Convolutional Neural Networks (CNNs). For CNN implementations, one of the most widely used and effective visualization techniques is known as Grad-CAM (Gradient-weighted Class Activation Mapping).

Grad-CAM provides insights into the inner workings of CNNs by highlighting the regions of an input image that are most influential in determining the model's predictions. This technique utilizes the gradients of the target concept, flowing into the final convolutional layer, to produce a heatmap that highlights the important regions in the image. By overlaying this heatmap on the original image, Grad-CAM allows researchers and practitioners to visually interpret which parts of the image contribute most significantly to the model's decision-making process.

Incorporating Grad-CAM into the analysis of CNN models not only enhances interpretability but also helps in diagnosing potential issues such as model biases, ensuring more robust and reliable AI systems. Through these visualization techniques, stakeholders can gain a better understanding of AI behavior, fostering greater trust and transparency in AI-driven solutions.

To sum up, the purpose of the Grad-CAM method is to see which areas our model focuses on during the training process. If a model always classifies incorrectly and focuses on unimportant features, the Grad-CAM method can be used to see which areas the model is focusing on during training. For example, when we classify cats and dogs, Grad-CAM can show us which areas the model focuses on when making classifications.

- When it classifies as a cat, it focuses on the features on the cat's body.
- When it classifies as a dog, it focuses on the features on the dog's body.

These two images are of a nurse and a doctor. When the model is trained with bias, you can see that the model's attention is focused on the face, which is obviously not where it should be. When the training is normal, you can see that the model's attention is focused on the tools used by the doctor and nurse.

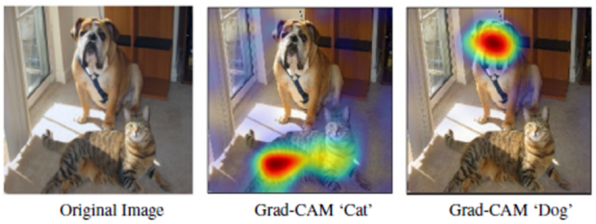


Figure 20: Grad-CAM visual explaination

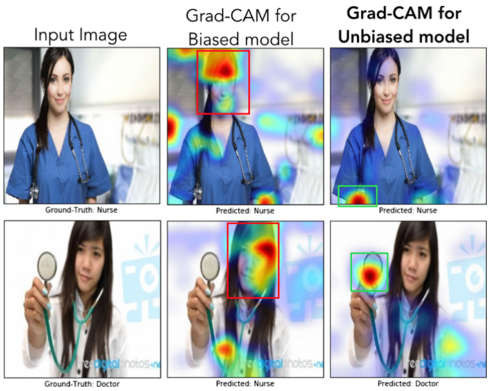


Figure 21: Difference between unimodal and multimodal XAI

By computing the gradients of the class score with respect to the feature maps and then using these gradients to weight the feature maps, Grad-CAM produces a heat map that highlights important regions. This method allows us to understand why the model made a particular decision. The resulting class activation map can be overlaid on the original image to visualize the regions that are most important for the "cat/dog" classification.

8 Language Model

A language model is a tool in the field of natural language processing (NLP) that is capable of understanding, generating and manipulating text in human language. It is an artificial intelligence-based system that trains on large amounts of text to learn the structures, patterns and meanings underlying human language. Its main objective is to predict the probability of a sequence of words. This means that you can anticipate the next word in a sentence given a series of previous words.

As we have said, the final phase of our project is based on the incorporation of a language model (for example: GPT-4) together with the output images processed after the application of explainability technique, in this way, a provisional medical support report is generated.

Task	Dataset	Model			
		Gemini Pro	GPT 3.5 Turbo	GPT 4 Turbo	Mixtral
Knowledge-based QA	MMLU (5-shot)	64.12	67.75	**80.48**	-
	MMLU (CoT)	60.63	70.07	**78.95**	-
Reasoning	BIG-Bench-Hard	65.58	71.02	**83.90**	41.76
Mathematics	GSM8K	69.67	74.60	**92.95**	58.45
	SVAMP	79.90	82.30	**92.50**	73.20
	ASDIV	81.53	86.69	**91.66**	74.95
	MAWPS	95.33	**99.17**	98.50	89.83
Code Generation	HumanEval	52.44	65.85	**73.17**	-
	ODEX	38.27	42.60	**46.01**	-
Machine Translation	FLORES (0-shot)	29.59	37.50	**46.57**	-
	FLORES (5-shot)	29.00	38.08	**48.60**	-
Web Agents	WebArena	7.09	8.75	**15.16**	1.37

Figure 22: Results of benchmarking

Therefore, the applicability of what has been said allows us to create reports in the following aspects:

1. **Text Generation**: Language models can generate detailed and coherent descriptions of the medical images being analyzed, they can describe the characteristics that are observed, such as the presence of certain anomalies, location, etc.

2. **Automatic translation**: Language models can translate automatically generated medical reports into different languages, ensuring that the information is accessible to healthcare professionals and patients who speak different languages.

3. **Sentiment Analysis**: In medical reports, it can be adapted to assess the urgency or severity of observations. This can help prioritize more critical cases.

9 Implementation

9.1 Software Used

The software I use is focused on machine learning, allowing for the efficient development of neural networks. This makes it possible to conduct a large number of experiments with various models.

The software we used is Google Colab, a free platform from Google that allows you to write and run Python code directly in your browser. Google Colab is particularly useful for developing machine learning and deep learning projects, as it provides access to powerful computing resources, including GPUs and TPUs, without the need for complicated configurations.

One of the biggest advantages of using Google Colab is access to GPUs (Graphics Processing Units) and TPUs (Tensor Processing Units), which are crucial for the development and training of machine learning and deep learning models,

When it comes to code implementation, the framework for developing the deep learning model is Tensorflow, originally designed with a declarative syntax and the use of static computational graphics. To be specific, the main library used to carry out this project is called keras, where it has been integrated complementarily within Tensorflow. Highlighting its advantage by:

1. **Ease of Use:** Keras is designed to be user-friendly, modular, and extensible. The syntax is simple and clean, allowing developers to build and train models quickly and efficiently without needing to understand the underlying mathematical computations.
2. **Rapid Development:** It enables developers to move from idea to implementation with just a few lines of code, facilitating rapid prototyping and experimentation.

Finally, we employ explanatory techniques to improve the performance and visualisation of the model, and the output of the results will be incorporated into a model language that will play an essential role in the final report.

9.2 Configure the data set

First, we exported the data from Escriptorio, a system where the file is already classified according to the diseases under study. This file contains a set of 4000 images that are evenly distributed, meaning that there are 500 examples corresponding to each disease category. Each of these categories represents a specific disease, and the images have been pre-arranged to facilitate analysis and research. This pre-classification process ensures that each group of images related to a disease has equal representation in the dataset, which is crucial for accurate comparisons and meaningful results in the study of diseases.

Categorie	**Number of Files**
dyed-lifted-polyps	500
dyed-resection-margins	500
esophagitis	500
normal-cecum	500
normal-pylorus	500
normal-z-line	500
polyps	500
ulcerative-colitis	500

Table 2: Distribution of image files across categories

9.2.1 Images and Labels Reading

To create the set of features (X) and labels (y), we start by reading each image from the dataset, where each of the images in the dataset has a height of 576 pixels and a width of 720 pixels (size: 576 x 720). As we process each image, we resize it to a size of 100x100 pixels, which helps speed up the learning process by reducing the amount of data the model needs to process. Each resized image is stored in the list, which represents our feature set. Simultaneously, the class or category to which the image belongs is identified and this information is stored in the list ”y”, which contains the labels corresponding to each image. This procedure ensures that each image is linked to its correct category, allowing the machine learning model to train effectively using the images and their respective labels.

$$X : (4000, 100, 100, 3)$$
$$y : (4000,)$$

In this context, X and y have the following dimensions:

- X: $(4000, 100, 100, 3)$ - This indicates that X is an array of 4000 images, where each image has a size of 100x100 pixels and 3 color channels (e.g., RGB).
- y: $(4000,)$ - This indicates that y is an array of 4000 images of labels, where each image corresponds to the class of one of the labels in X.

9.3 Model building

To create the training, test, and validation sets, we first split the original dataset into 80% for training and 20% for testing using from datasets. Then, we take the 80% assigned to training and split it again, this time into 70% for the final training and 30% for validation. This way, we obtain three sets: one for training the model, another to evaluate its performance during training, and a final set to test its overall performance.

$Xtrain$**:** 2240 images $Xvalid$**:** 960 images $Xtest$**:** 800 images

9.3.1 Encoding for the model

We have to do a OneHot Encoding with the corresponding function, to transform all 3 file groups, train, test and validation, to convert them into vector form.

The reason for using One-Hot encoding on the labels is to convert them into a format suitable for machine learning algorithms, especially neural networks. Instead of having categorical labels (e.g. numbers from 1 to 8), One-Hot encoding transforms each label into a binary vector in which only one position is ”1” (indicating the class) and the others are ”0”. This format prevents the model from interpreting non-existent ordinal relationships between classes and ensures that each class is treated equally.

9.4 Data Augmentation

On the original images, we employ an image data generator using the Keras ImageDataGenerator class to perform data augmentation. This technique is used to increase the diversity of the training dataset, helping to prevent overfitting and improving the model's ability to generalise. In this case, several random transformations are specified to be applied to the images: a random rotation of up to 20 degrees (rotation range), horizontal and vertical shifts of up to 20% of the image size (width shift range and height shift range), a shear of up to 20 degrees (shear range), a random zoom of up to 20% (zoom range), and a horizontal flip (horizontal flip). And additionally, the 'nearest' parameter to fill the pixels that are left empty after transformations. This approach helps to create a more robust and varied dataset from the original images, thus improving model performance.

1. **Improvement of Model Performance**: By providing more varied data, overfitting is reduced, allowing the model to generalize better to unseen data.
2. **Simulation of Real-World Variability**: Real-world images can vary due to multiple factors, such as position and angle.
3. **Increase in Dataset Size**: Especially useful when a small dataset is available, as it significantly increases the number of training examples.

9.5 Pre-trained Model

In this project, we will not train a new model from scratch, but we will apply a common deep learning technique called Fine-Tuning. Fine-tuning is a technique in deep learning that consists of taking a model pre-trained on a large, generic dataset, and tuning its additional parameters using a new dataset specific to a particular task. This approach is especially useful when you have a limited amount of task-specific data, allowing you to leverage the knowledge learned by the model on the original dataset, which is suitable in our case.

During the construction of our model, we tested several architectures noted for their excellent performance in vision tasks. These included the VGG-19, ResNet, EfficientNetb0 and EfficientNetb7 architectures. After extensive testing, we finally decided to carry out the project with the EfficientNet architecture. The most optimal option would be the implementation of

EfficientNetb7, the most advanced model, which stands out for its higher accuracy and generalisation capacity and at the same time, handles the problem of overfitting. However, it requires a higher cost in terms of computational resources. But as Google Colab allows us to train for free, it is not a problem to consider.

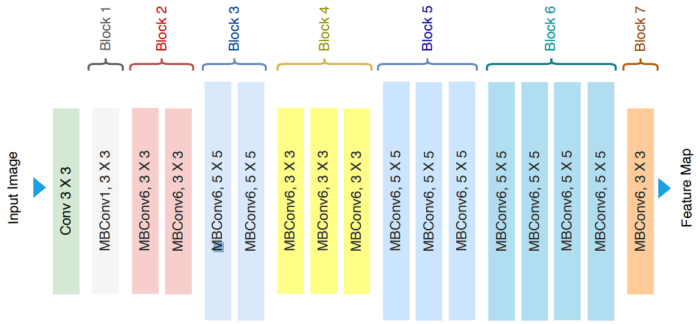


Figure 23: EfficientNet-b7 architecture

It consists of several layer blocks and MBConv (Mobile Inverted Bottleneck Convolution), which are mobile inverted bottle convolutions. It is called inverted bottle because unlike traditional network architecture, it reduces the number of input channels before applying convolution and then expands the channels. Inverted bottle convolutions begin with an expansion of the channels before applying the convolution, and are called mobile, due to their improvement in computational efficiency and performance.

Two types of MBConv layers are shown in the image: In MBConv1, the expansion factor is 1, which means that the number of input channels remains the same during operation, without expansion. In contrast, MBConv6 has an expansion factor of 6, which implies that the number of input channels is multiplied by 6 before convolution. This allows MBConv6 to capture more complex and detailed features. The overall structure follows a tiered design, starting with an initial 3x3 convolution, followed by different MBConv blocks with different filters and kernel sizes (3x3, 5x5). Each block is labeled (Block 1 to Block 7), indicating the different stages of feature processing.

9.6 Configuration of Hyperparametres

In this section we describe the hyperparameter settings used to train our model. We select a batch size of 32 samples. This value determines the amount of data that is processed before the model updates its parameters. We consider this batch size to be the most appropriate one that allows us a balance between training speed and convergence stability. The number of epochs is set to 100

units. Each epoch represents a complete iteration over the training dataset. A larger number of epochs allows the model to learn better, although it may increase the risk of overfitting.

The SGD (Stochastic Gradient Descent) optimiser has been set to a learning rate of 0.001, this hyperparameter controls the magnitude of the adjustments the model makes at each weight update. A well-adjusted learning rate is crucial to ensure fast and stable convergence. And a momentum of 0.9, a technique that helps to speed up training and overcome possible local minima in the loss function. The model is compiled using the configured SGD optimiser. The selected loss function is categorical crossentropy, suitable for multi-class classification problems. In addition, it is specified that the accuracy metric will be monitored during training, allowing the proportion of correct predictions to be evaluated.

The configuration of the callbacks is also a relevant factor in the construction of our model, as a good choice can improve performance and reduce overfitting. We have configured 2 callbacks for testing purposes.

1. Reducción de la tasa de aprendizaje (ReduceLROnPlateau):

```
lrr = ReduceLROnPlateau(monitor="val_acc", factor=0.01, patience=3, min_lr=1e-5)
```

The ReduceLROnPlateau callback is used to reduce the learning rate when the specified metric (in this case, validation accuracy) shows no improvement. This callback monitors the validation accuracy and, if it does not improve after 3 epochs (patience=3), reduces the learning rate by a factor of 0.01 (factor=0.01). The learning rate will not fall below 1e-5 (min lr=1e-5). This technique helps to adjust the learning rate during training, allowing the model to converge more effectively.

2. Early Stopping:

```
early_stop = EarlyStopping(monitor='val_loss', patience=20, restore_best_weights=True)
```

The EarlyStopping callback is used to stop training early if the monitored metric (val loss, in this case, validation loss) stops improving. This callback monitors the validation loss and, if it does not improve after 20 epochs (patience=20), stops training and restores the model weights to the state in which the validation metric was best (restore best weights=True). This technique helps prevent overfitting by ensuring that the model does not train more than necessary.

After several tests, we realise that the implementation of EarllyStop leads to a better performance compared to ReduceLROnPlateau, which decreases the overfitting effect of our model and at the same time, more efficient at runtime, without occupying unnecessary resources.

Finally, we have just built the appropriate model with the fitted hyperparameter values from above.

```
history = model.fit(
    datagen.flow(x_train, y_train, batch_size=batch_size),
    epochs=epochs,
    steps_per_epoch=x_train.shape[0] // batch_size,
    validation_data=(x_val, y_val),
    callbacks=[early_stop],
    verbose=1,
)
```

Note that the data augmentation process has only been applied to the train data, and the validation set we have kept the original images. This is not trivial, it has implications.

Applying data augmentation only to the training set has several benefits. Primarily, it helps prevent overfitting, a situation where the model learns the features of the training set too well, thus losing its ability to generalise to unseen data. By presenting different variations of the training images, the model is constantly confronted with new examples and learns to recognise larger and more meaningful patterns.

In contrast, the validation set should not be subjected to data augmentation. The purpose of the validation set is to evaluate the performance of the model on data not seen during training. It provides a realistic estimate of how the model will behave in real-world situations. If data augmentation were applied to the validation set, it would introduce noise and variability that is not present in the real data, which could lead to an inaccurate assessment of model performance. Therefore, keeping the validation data unmodified ensures that the evaluation reflects the model's true ability to generalise and that any observed improvement is due to its ability to learn meaningful features and not to memorise artificial variations.

9.7 Visualisation

After completing both the training and validation phases, the next step is to graphically represent the evaluative factors of our dataset. We will plot the accuracy and loss values for both training and validation sets. This initial visualization provides a clear indication of our model's performance.

Next, we will generate a confusion matrix, which will help us assess the accuracy of our model's predictive ability across different categories. The confusion matrix offers more detailed insights into where our model performs well and where it might be struggling. Additionally, we will closely examine each misclassified image. This process involves filing these images and performing a detailed analysis, either visually or with the assistance of an explainability engine. By doing so, we can identify and understand the specific problems leading to errors. This thorough examination will help us refine our model and improve its accuracy.

At the end of this project, we will draw a small pilot sample to achieve our final goal: creating a provisional medical report. This report will be generated using a language model, specifically ChatGPT, due to its ease of access and high performance. Each test sample will be saved in a Google Drive file, accompanied by its respective Grad-CAM and Kernel-SHAP analyses.

We must also highlight the choice of layers for the heatmap representation. Since it will influence a good display of the output image. It consists of a trivial process, where there is no specific methodology to calculate the most ideal layer. It is about carrying out tests and choosing the most appropriate to adapt to each person's objectives.

In the case of Kernel-Shap, we will use the shap.GradientExplainer function to explain the predictions of a model using gradients.

```
explainer = shap.GradientExplainer(model, x_train[:100], local_smoothing=0)
```

By default, it applies SHAP values to the model output, that is, to the final model predictions. This implies that you are considering all the calculations performed from the input to the output of the model, that is, after passing through all the layers of the model. But the SHAP values it produces are related to the final output of the model.

To choose the right layer to apply Grad-CAM, we can follow some principles and methodologies, to choose the right Layer:

- The first convolutional layers capture low-level features such as edges, textures, and colors.
- Deeper layers capture high-level features such as specific shapes and objects.
- Middle layers balance between capturing low- and high-level features.

```
base_model = EfficientNetB7(weights='imagenet',
                            include_top=False, input_shape=(100, 100, 3))
base_model.summary()
```

If we want to visualize general activations and locate specific areas of interest, the middle and deep layers are better. To capture very detailed and fine features, the first few layers may be more useful, but this may lead to more noise. We'll apply Grad-CAM to multiple layers and compare the results to see which provides the most interpretable and useful visualizations.

9.8 Report generating

It is important to note that current natural language models lack the analytical capabilities needed to interpret medical diagnostic images. For example, when presented with an endoscopy image, ChatGPT cannot distinguish or explain the specific disease depicted. Therefore, we will not only provide the original image, but also include the classified disease and the heat map generated by Kernel-SHAP and Grad-CAM. These heat maps, which consist of colour overlays on the original image, highlight the regions where the pathology is located. The areas of focus, marked by more prominent colours, indicate where the model was concentrated during training.

Therefore, if we previously introduce a query in our natural language program following a series of restrictions, every time we send our file for each sample (patient) to the model, the files will be read and the report will be created with the format that we have previously established. Since it is an automated process, we can feed the model with certain instructions so that it has a basic template for how to generate the report. So when we provide the images, the model will automatically complete each section following the example of the template.

As has already been mentioned from the beginning, this is a pilot test, so the generated report will be adapted in a simplified way without reaching the point of being equal to a professional, the principle is to make it easier for the patient to do. a quick and easy read before obtaining the formal document written by the medical professional.

At the time of model choice, if we opt for performance and ease of use, ChatGPT would be our first choice. But taking into account that currently the free version of Chat does not offer the option to upload images or files in the prompts, then we must look for another alternative to guarantee the reproduction of our work. And the most suitable option taking into account several factors, we opted for the Gemini model, a large multimodal language model developed by Google DeepMind. Whose original version of use allows the user to import images. Convince stands out for its inferior performance compared to ChatGPT, and the inability to auto-create reports.

The most sophisticated way to elaborate would be to directly connect our code with the language model through an API key. But since the API is a personal control tool, it makes it very difficult to reproduce our work. In this case, we will opt for a code separation, where we will divide the procedure into parts, making it automated as much as possible and at the same time, guaranteeing reproducibility.

We extract a small sample of the pilot, from which we apply the Kernl-Shap and Grad-Cam to generate the corresponding heatmap, and save it in a Google Drive file. As we have said before, Gemini has limited capacity, we can only provide one image at a time, therefore, we will combine the two heatmaps into one. The entire previous procedure is carried out through python code, the manual part of the procedure will consist of downloading the combined image and sending to Gemini along with a series of Queries previously described in the code. So we will paste the output provided by the Gemini into the corresponding section of pre-written code and they will generate a decent report on your drive.

9.8.1 Query for the model Prompts

This section will provide you with the template for the prompts of the implementation model. They will serve as a guide when generating the response. The language model will respond step by step following the questions we have provided.

Example of Use

By entering two images (Grad-Cam and Kernel-Shap previously generated) and the patient's personal data (name, age, date), the language model follows the template to generate a structured report. The entry might look something like this:

INPUT: (two images and following information about personal data)

- **Name**: Juan Perez
- **Age**: 45
- **Date**: 28/06/2024
- **Study**: Endoscopy
- **Reason for Study**: Persistent pain in the lower abdomen.

PROMPT: Generate a medical report following this structure:

- **Name**: (Pre-introduced by the user)
- **Age**: (Pre-introduced by the user)
- **Date**: (Pre-introduced by the user)

- **Study**: Describe the type of study being conducted.
- **Reason for Study**: Describe the reasons that have led to this study.
- **Findings**: Detail the observations and results obtained from the study.
- **Impression**: Summarize the overall interpretation of the findings.
- **Recommendations**: List any suggested next steps or actions based on the findings and impression.
- **Comments**: Add any additional remarks or notes that are pertinent to the study.
- **Images/Heatemaps**: Images or heatmaps pre-introduced.

10 Results

10.1 Final model

The model starts with the base layer of EfficientNetB7, a pre-trained architecture that is characterized by its high efficiency and accuracy in image classification tasks. This initial layer takes as output images with a shape of (None, 4, 4, 2560), where 2560 represents the number of filters applied by EfficientNetB7. The choice of this architecture was based on its ability to extract meaningful and complex features from images, due to its pre-training on an extensive ImageNet data set.

The output of the EfficientNetB7 layer, which is the final convolutional layer of this architecture, is flattened by a Flatten layer, transforming the 4x4x2560 matrix into a vector of 40960 elements. This flattening is essential to be able to connect the output of the convolutional network to the downstream dense layers. Subsequently, several dense layers are added to refine the extracted features and adapt them to the specific classes of the Kvasir dataset. The first dense layer has 1024 neurons, followed by a second with 512 neurons and a third with 256 neurons. Progressively reducing the number of neurons in these layers allows the model to learn more compact and relevant representations. To avoid overfitting, a Dropout layer is incorporated after the third dense layer. This layer randomly removes a percentage of the connections during training, which helps to improve the generalisation of the model. The fourth layer has 128 neurons and the last output has 8, corresponding to the 8 classes of the problem.

Layer (type)	**Output Shape**	**Param #**
efficientnetb7 (Functional)	(None, 4, 4, 2560)	65,097,687
flatten (Flatten)	(None, 40960)	0
dense (Dense)	(None, 1024)	41,944,064
dense_1 (Dense)	(None, 512)	524,800
dense_2 (Dense)	(None, 256)	131,328
dropout (Dropout)	(None, 256)	0
dense_3 (Dense)	(None, 128)	32,896
dense_4 (Dense)	(None, 8)	1,032

Total params	106,731,807
Trainable params	106,421,080
Non-trainable params	310727

In terms of parameters, the model has a total of 106,731,807 parameters, of which 106,421,080 are trainable and 310,727 are non-trainable. The untrainable parameters mainly correspond to the pretrained EfficientNetB7 layer, which is kept fixed during training to preserve previously acquired knowledge. These layers include all convolution blocks and associated layers within EfficientNetB7, which make up the vast majority of the model parameters, totaling 65,097,687 parameters. The structure of the model allows taking advantage of both the power of the pretrained network and the adaptation capacity through additional dense layers, which include several 'dense' layers and a 'dropout' layer to improve generalization.

10.2 Results of the Epochs and Use of Early Stopping

During the training of the image classification model of the Kvasir dataset, 100 epochs were used to adjust the model parameters. However, thanks to the implementation of the EarlyStopping callback, the training was stopped early. As you can see, it has stopped at epoch 51, which helps prevent overfitting and save computational time. Over the epochs, a trend can be observed in which training loss and training accuracy show consistent improvements. However, reaching epochs close to 50, the validation metrics (val loss and val accuracy) do not follow a constant improvement and present fluctuations. In this case, although training was planned for 100 epochs, the model was stopped early because no significant improvement in validation loss was observed. This behavior is expected and desired, since it indicates that the model reached its optimal generalization point before completing all epochs.

```
Epoch 1/100
70/70 [========================] - loss: 2.0060 - accuracy: 0.2272
                        - val_loss: 1.7291 - val_accuracy: 0.4437
Epoch .../100
70/70 [========================] - loss: ... - accuracy: ...
                        - val_loss: ... - val_accuracy: ...
Epoch 51/100
70/70 [========================] - loss: 0.0982 - accuracy: 0.9643
                        - val_loss: 0.4525 - val_accuracy: 0.8771
```

Figure 24: Results of Epochs

10.3 Model Accuracy Graph Analysis

Initial Model Evaluation

This section presents a detailed analysis of the results obtained during the training and evaluation of the image classification model. At the start of training, the model showed significantly lower accuracy on the training set compared to the validation set. This is mainly due to the implementation of data augmentation in the training set. Data augmentation introduces additional variations to the images, such as rotations, rescaling, and random cropping, which increases the initial difficulty for the model. However, these techniques are essential to improve the long-term generalization capacity of the model, since they make it more robust to variations in the data.

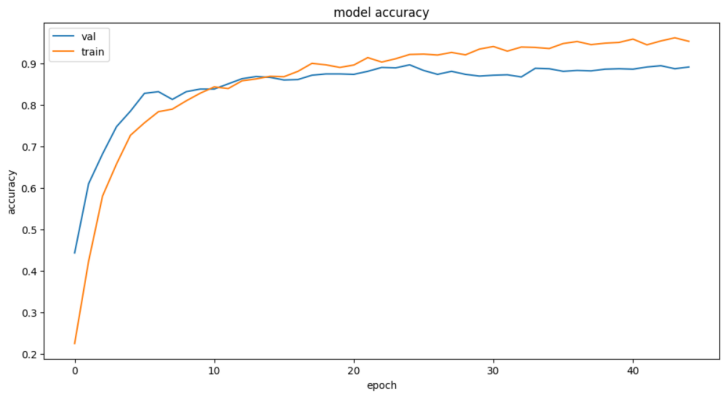


Figure 25: Model Accuracy: Test/Validation

Training Progress

Graph 1 shows the evolution of precision in both the training set (represented in orange) and the validation set (represented in blue) over the epochs.

Training Accuracy Curve: The accuracy in the training set, represented in orange, increases rapidly during the first 10-15 epochs, indicating effective learning of the data patterns. Subsequently, the improvement rate decreases and the accuracy stabilizes around values greater than 90%.

Validation Precision Curve: The precision in the validation set, represented in blue, also shows a rapid initial increase, reaching a significant value after the first 10-15 epochs. Unlike the training curve, the validation accuracy stabilizes earlier, around a value slightly lower than that of training, showing a tendency to stabilize and then slightly decrease towards the end of training.

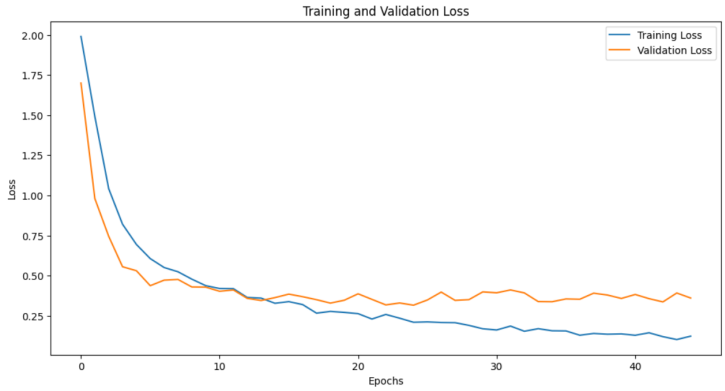


Figure 26: Model Loss: Test/Validation

Graph 2 shows the evolution of the loss in both the training set (represented in blue) and the validation set (represented in orange) over the epochs.

Training Loss Curve: The loss on the training set shows a rapid decrease during the first 10-15 epochs, indicating that the model is learning effectively and reducing the error in its predictions. After this initial period, the loss continues to decrease but at a slower rate, stabilizing at low values, reflecting that the model is well fitted to the training data.

Validation Loss Curve: The loss on the validation set also decreases rapidly at first, but stabilizes earlier than the loss on the training set. The stabilization and slight fluctuations in validation loss suggest that the model has reached a good balance between learning and generalization, although any further decrease in training loss does not translate into an improvement in validation loss.

Results analysis

The graphs show that the EfficientNetB7 model has effectively learned the features of the Kvasir dataset images. Training and validation accuracy suggest good overall model performance, with validation accuracy plateauing around 88-89%. The validation loss, although higher than the training loss in recent epochs, stabilizes and shows that the model has reached an optimal generalization point before starting to overfit.

Implementing techniques like EarlyStopping helped prevent excessive overfitting by stopping training at an appropriate point. These results serve as validation of the effectiveness of the approach used, achieving high precision in the test set and good overall performance, combining the robust architecture with regularization and data augmentation techniques.

10.4 Confusion Matrix

Below we present the results of the evaluation based on the confusion matrix, which is a fundamental tool to understand the detailed performance of the model. Two confusion matrices are presented: one unnormalized and the other normalized. Both matrices provide a complete view of the model performance in terms of correct and incorrect predictions for each class of the dataset.

Most classes have high precision, with values ranging between 82% and 97%. This indicates that the model is effective in classifying most of the images in the Kvasir dataset. But there are certain cases with Confusions in Specific Classes:

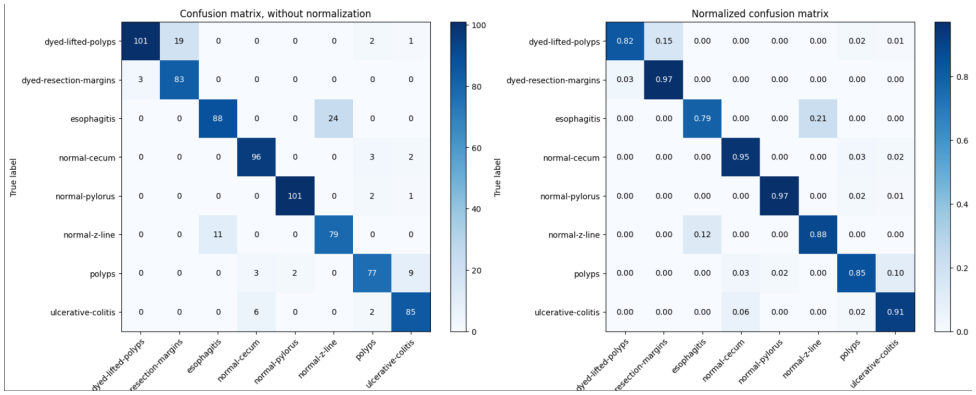


Figure 27: Confusion Matrix

Dyed-lifted-polyps and Dyed-resection-margins: There is some confusion between these two classes, if we represent them graphically, we can visually verify that this confusion is due to the visual similarity between the images of these categories. Since both of them, due to their original image characteristic, are the only ones that have pathological regions tinted blue.

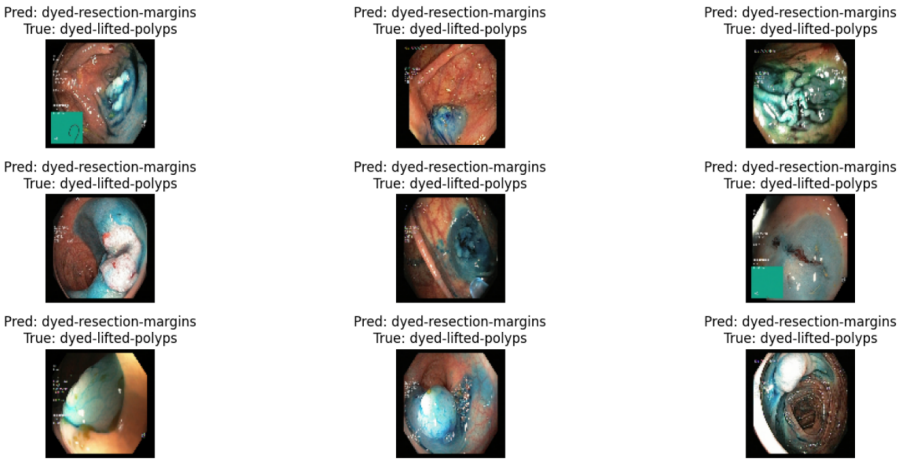


Figure 28: Dyed-lifted-polyps

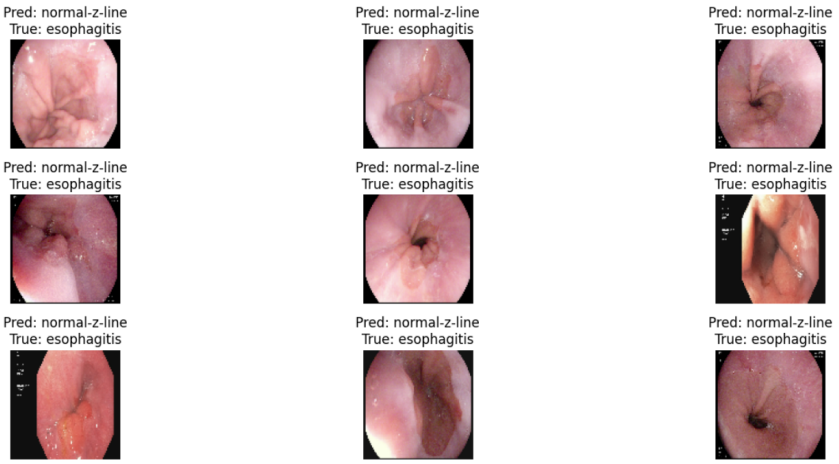


Figure 29: Esophagitis

Esophagitis and Normal z-line: There is notable confusion in the prediction of esophagitis as normal-z-line, taking into account that esophagitis is a malathia produced in the z-line area, so the intuition we carry out It is the ease of producing confusion when the patient is in the initial period of malaria, then as the length of the mucosal tears is defined by the degree of inflammation, it means that the IA is not able to clearly differentiate a mild inflammation.

Overall, although there are some confounds, most incorrect predictions are relatively low in proportion and high accuracy in most classes indicates good overall model performance, while specific confounds identify areas where work could be done to improve. discrimination between similar classes.

10.5 Heatmaps and Report

Now we come to the final part of this project, the generation of the heat map and the creation of the final report. As already mentioned, we will choose an image from each category and create its graphical representation using explainability techniques. Here, we are only going to illustrate some of the categories and perform such interpretation, regarding the representation of other categories you can find more information in the appendix.as we have trained the model using the efficientNet architecture, it is essential to use the same architecture preprocessing. This is because KernelSHAP needs to work with the same input data that the model has seen, in the same form and values. In more particular cases, other architectures can be used, but it is crucial to maintain consistency of the appropriate preprocessing for each specific architecture, making sure to use the corresponding preprocessing function from the library and adjusting the prediction function to be compatible with the new architecture. Since each architecture presents

different characters when generating graphs, you can introduce more cases in the appendix if you are interested.

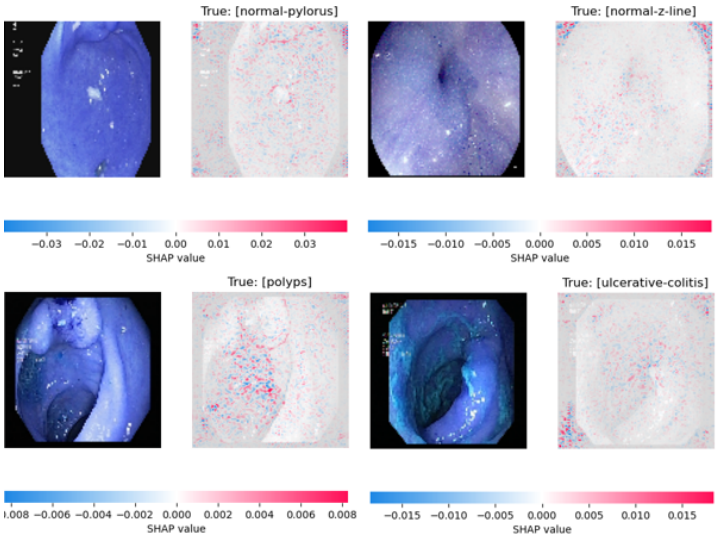


Figure 30: Kernel-Shap representation

This image shows examples of predictions from a computer vision model of Shap, together with their SHAP (SHapley Additive exPlanations) value maps. Each row presents an original image and its corresponding SHAP value map, highlighting the regions that most influenced the model prediction. Areas in red indicate regions that contribute positively to the prediction, while areas in blue indicate regions that contribute negatively.

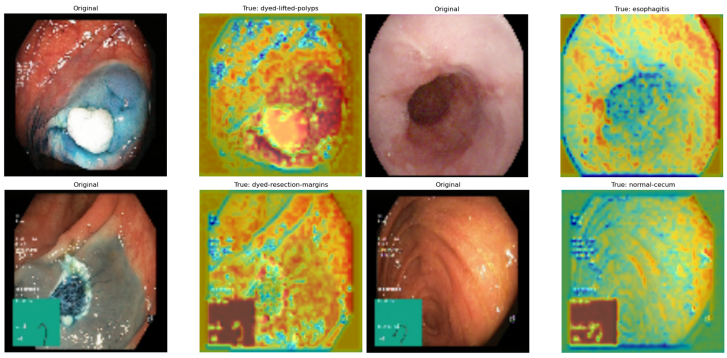


Figure 31: Grad-Cam Presentation

This image shows the Grad-Cam representation. Each image pair consists of the original image and a heat map indicating the true category (dyed-lifted-polyps...). The heat map highlights areas important for model prediction, using colours to indicate the contribution of different regions: areas in red and yellow indicate regions of high positive influence, while areas in blue and green indicate lower influence or negative influence.

Now, from the 8 samples of 8 categories, we are going to select the Dyed-lifted-polyps. From which we are going to make a detailed explanation about the graphical representation and the explanation of what we see. And that this sample will be used for the construction of the final report.

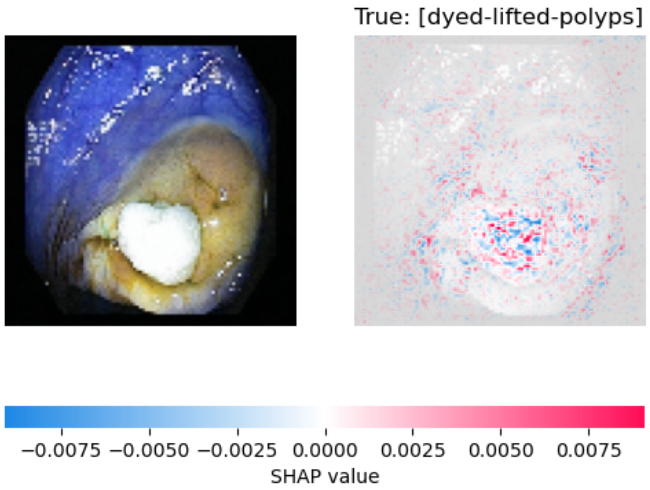


Figure 32: Kernel-Shap: Dyed-lifted-polyps

The image on the left is the original input used for prediction by the model. However, after applying the EfficientNet specific preprocessing (preprocess input), the image appears to have a blue colouration. This colour shift occurs because EfficientNet's preprocessing adjusts the pixel values to be suitable for the network, which can alter the image display by applying normalisations and colour scales that are not intended for direct visual interpretation.

The image on the right represents the SHAP values superimposed on the pre-processed image, indicating the contribution of each pixel to the model prediction. Pixels in red indicate a positive contribution, increasing the probability that the model predicts dyed-lifted-polyps, while pixels in blue indicate a negative contribution, decreasing the probability. The colour distribution shows that the central areas of the dyed polyp have a high positive influence on the prediction, helping to understand how the model has reached its conclusion.

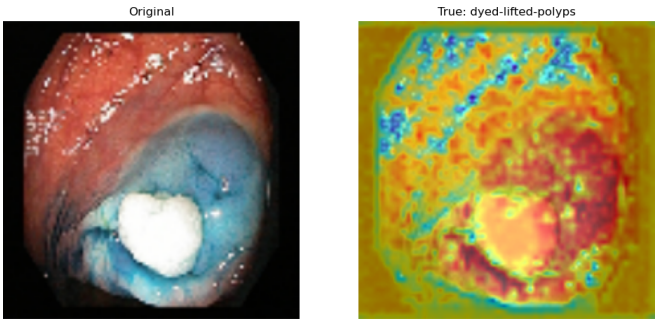


Figure 33: Grad-Cam: Dyed-lifted-polyps

The image shows the results of applying Grad-CAM from the second layer of the model.

The image on the left is the original input image, showing a polyp in the gastrointestinal tract, stained with a dye for easy visualisation. The image on the right shows the output of Grad-CAM, which visualises the regions of the image that contribute most to the model's prediction for the dyed-lifted-polyps class. The warm colours (red and yellow) indicate areas of high importance, where the model focused most on making its prediction. Cool colours (blue and green) indicate areas of lower importance. In this Grad-CAM output, it can be seen that the most prominent and relevant areas for prediction are around the stained polyp, especially in the central region. This suggests that the model is relying primarily on the visible features of the polyp to correctly identify the dyed-lifted-polyp class.

The heat distribution in both Shap and Grad-Cam confirms that the model is correctly focusing its analysis resources on the critical areas for polyp detection and classification, providing a double lock for the subsequent process of sending the language model to create a report from this data.

Endoscopic Report

Name: [Change for your Name]

Age: [Change for your Age]

Date: [Change for the Date]

Study

Upper gastrointestinal endoscopy to evaluate the gastric mucosa.

Findings

- A 5 mm antral polyp is observed, pedunculated, with a smooth surface and normal coloration.

Impression

Gastric antral polyp.

Recommendations

- Biopsy of the polyp to determine its histological nature.
- Endoscopic follow-up in 6 months.

Images

Heatmaps

Heatmaps have been provided for this case.

Doctor's Signature

Prepared by the Gemini model

Comments

In this case, only a 5 mm antral polyp is observed, pedunculated, with a smooth surface and normal coloration.

A biopsy of the polyp is necessary to determine its histological nature. If the polyp is benign, endoscopic follow-up can be performed in 6 months. If the polyp is precancerous or cancerous, endoscopic or surgical resection should be considered.

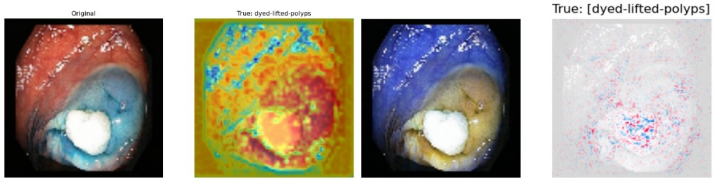


Figure 34: Pilot report

With the above images supplied, we now have here the report created by Google's Gemini model, a test report that shows the characteristics of the disease. As we can see, this report includes the sections that we have prefixed, and about the content of this report, apart from the most basic information, we can highlight the interesting point that the model is able to estimate with some accuracy the polyp size and location from the heatmaps. It measures the area where the most relevant colours are concentrated and makes an estimate, which is what we wanted to get to in the first place. It cannot be compared to a professional report, but it provides remarkable insights.

11 Conclusion

To conclude our work, we are going to evaluate the fulfilment of the objectives set out in the introduction, followed by a self-assessment of the results. As can be seen, the training model with the efficientNet-b7 architecture is quite successful, as it has achieved close to 90% accuracy. Of course, we are talking from a statistical perspective, for a health care use, the ideal accuracy value should be infinitely close to 100%, in order to avoid as many misdiagnosed cases as possible. However, it should also be noted that we are only training with 4000 images, which we consider to be an acceptable result.

Another point of consideration would be misclassified categories, especially in diseases that share similar features or cases that are in the early stages of pathology, so that currently training with small samples is not able to distinguish with perfection, and which theoretically will improve if the sample size is enlarged. Another case would be the methodology of staining the desired area during endoscopy, such as cases seen on polyps or resection. The staining technique is effective in improving detection, visualisation during human observation, but for artificial intelligence it is just the opposite. As the dye colour is clearly differentiated from the subtracted area, it stands out too much and creates a distracting region, making it difficult for the machine to focus on details and physical features.

Regarding the automatic creation of a medical report, what we have done during this project will be considered as a pilot test. The idea of completing the information based on heat mapping I consider to have significantly positive contribution to natural language processing. It helps to get to the point of pinpointing the region where the features of inflammation are located, by fixing the pixels that have light colours. And the fact of performing both Kernel-Shap and Grad-Cam, the purpose is that they complement each other's information. A pity about this project is that it has not been possible to make the code for this part more sophisticated. The best option would be to use an API key, then it would be a fully automated project, but due to my lack of knowledge and lack of time in researching this field of study, I have not been able to refine the code.

This project has allowed me to acquire new knowledge about neural networks, a field we had not explored during my degree. Throughout the learning process, I have had the opportunity to experiment with various interesting algorithms and understand the advantages of implementing machine learning in everyday life.

12 Bibliography

[1] Saranya A. and Subhashini R. "A systematic review of Explainable Artificial Intelligence models and applications: Recent developments and future trends". In: *Decision Analytics Journal* 7 (2023). Accessed: 2024-06-13, p. 100230. DOI: 10.1016/j.dajour.2023.100230. URL: https://doi.org/10.1016/j.dajour.2023.100230.

[2] Shaun Robert Commee. *Convolutional Neural Networks (CNN) for Image Classification*. Accessed: 2024-06-13. 2024. URL: https://medium.com/@shaunrobertcommee_37218/convolutional-neural-networks-cnn-for-image-classification-3d92f2ed7bef.

[3] DataCamp. *Introduction to Convolutional Neural Networks (CNNs)*. Accessed: 2024-06-13. 2024. URL: https://www.datacamp.com/es/tutorial/introduction-to-convolutional-neural-networks-cnns?dc_referrer=https%3A%2F%2Flens.google.com%2F.

[4] Hugging Face. *Tasks Explained*. Accessed: 2024-06-13. 2024. URL: https://github.com/huggingface/transformers/blob/main/docs/source/es/tasks_explained.md.

[5] GeeksforGeeks. *Explainable Artificial Intelligence (XAI)*. Accessed: 2024-06-13. 2024. URL: https://www.geeksforgeeks.org/explainable-artificial-intelligencexai/.

[6] Adam Geitgey. *Machine Learning is Fun! Part 3: Deep Learning and Convolutional Neural Networks*. Accessed: 2024-06-13. 2024. URL: https://medium.com/@ageitgey/machine-learning-is-fun-part-3-deep-learning-and-convolutional-neural-networks-f40359318721.

[7] Belajar Pembelajaran Mesin Indonesia. *Student Notes: Convolutional Neural Networks (CNN) Introduction*. Accessed: 2024-06-13. 2023. URL: https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/.

[8] Belajar Pembelajaran Mesin Indonesia. *Student Notes: Neural Networks and Deep Learning*. Accessed: 2024-06-13. 2023. URL: https://indoml.com/2018/02/23/student-notes-neural-networks-and-deep-learning/.

[9] Simula Research Laboratory. *Kvasir Dataset*. Accessed: 2024-06-13. 2024. URL: https://datasets.simula.no/kvasir/.

[10] Doniyorjon Mukhtorov et al. “Endoscopic Image Classification Based on Explainable Deep Learning”. In: *Sensors* 23 (2023). Accessed: 2024-06-13, p. 3176. DOI: 10.3390/s23063176. URL: https://doi.org/10.3390/s23063176.

[11] Packt Publishing. *What is Deep Learning?* Accessed: 2024-06-13. 2020. URL: https://subscription.packtpub.com/book/data/9781838642709/1/ch01lvl1sec9/what-is-deep-learning.

[12] Quora. *What is pooling in a convolutional neural network?* Accessed: 2024-06-13. 2024. URL: https://www.quora.com/What-is-pooling-in-a-convolutional-neural-network.

[13] SK124. *Bird Classification Using CNNs*. Accessed: 2024-06-13. 2024. URL: https://github.com/SK124/Bird-Classification-Using-CNNs.

[14] TensorFlow. *MNIST Dataset Catalog*. Accessed: 2024-06-13. 2024. URL: https://www.tensorflow.org/datasets/catalog/mnist?hl=es-419.

[15] Lingfeng Wang. “Forecast Model of TV Show Rating Based on Convolutional Neural Network”. In: *Complexity* 2021 (2021). Accessed: 2024-06-13, 10 pages. DOI: 10.1155/2021/6694538. URL: https://doi.org/10.1155/2021/6694538.

13 Appendix A: Source Code

```
import os
import cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
import glob
import numpy as np
import pandas as pd
import random
import tensorflow as tf
import zipfile

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.utils import to_categorical
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import SGD, Adam
from keras.models import Sequential, load_model
from keras.layers import Flatten, Dense, Dropout
```

```
# Seed setting for reproducibility
def set_seed(seed):
    os.environ['PYTHONHASHSEED'] = str(seed)
    random.seed(seed)
    np.random.seed(seed)
    tf.random.set_seed(seed)

# Establishing the seed
set_seed(42)
```

```
from google.colab import drive
drive.mount('/content/drive')

# Suppose the ZIP file is in 'My Drive/kvasir-dataset.zip'.
zip_path = '/content/drive/My Drive/kvasir-dataset.zip'
extract_path = '/content/kvasir-dataset'

with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)

os.listdir(extract_path)
```

```
['normal-cecum',
 'normal-pylorus',
 'dyed-lifted-polyps',
 'esophagitis',
 'polyps',
 'normal-z-line',
 'ulcerative-colitis',
 'dyed-resection-margins']
```

```
def get_dataCategories(dataset_dir):
    if not os.path.exists(dataset_dir):
        raise FileNotFoundError(f"The directory {dataset_dir} does not exist.")

    categories = []
    for folder_name in os.listdir(dataset_dir):
        folder_path = os.path.join(dataset_dir, folder_name)
        if os.path.isdir(folder_path):
            nbr_files = len(glob.glob(os.path.join(folder_path, "*.jpg")))
            categories.append(np.array([folder_name, nbr_files]))

    categories.sort(key=lambda a: a[0])
    cat = np.array(categories)

    return list(cat[:, 0]), list(cat[:, 1])

# The path to the unzipped directory
dataset_dir = '/content/kvasir-dataset'
try:
    categories, nbr_files = get_dataCategories(dataset_dir)
```

```
    # Create DataFrame
    df = pd.DataFrame({"categorie": categories, "number of files": nbr_files})
    print("number of categories: ", len(categories))
    print(df)
except FileNotFoundError as e:
    print(e)
```

```
number of categories: 8
              categorie  number of files
0    dyed-lifted-polyps              500
1  dyed-resection-margins            500
2           esophagitis              500
3         normal-cecum               500
4        normal-pylorus              500
5         normal-z-line              500
6               polyps               500
7  ulcerative-colitis                 500
```

```
def create_dataset(datadir, categories, img_wid, img_high):

    X, y = [], []
    for category in categories:
        path = os.path.join(datadir, category)
        class_num = categories.index(category)
        for img in os.listdir(path):
            try:
                img_array = cv2.imread(os.path.join(path, img))
                ima_resize_rgb = cv2.resize(img_array, (img_wid, img_high))

                X.append(ima_resize_rgb)
                y.append(class_num)

            except Exception as e:
                pass

    y = np.array(y)
    X = np.array(X).reshape(y.shape[0], img_wid, img_wid, 3)
    return X, y


img_wid, img_high = 100, 100
X, y = create_dataset(dataset_dir, categories, img_wid, img_high)
```

```
print(f"X: {X.shape}")
print(f"y: {y.shape}")

plt.figure(figsize=(12, 5))
st, end = 0, 500
for i in range(8):
    plt.subplot(2, 4, i + 1)
    idx = np.random.randint(st, end)
    st = end + 1
    end = (i + 2) * 500
    plt.imshow(X[idx][:, :, ::-1])
    plt.title(f"{i}. {categories[y[idx]]}")
    plt.axis("off")
plt.show()
```

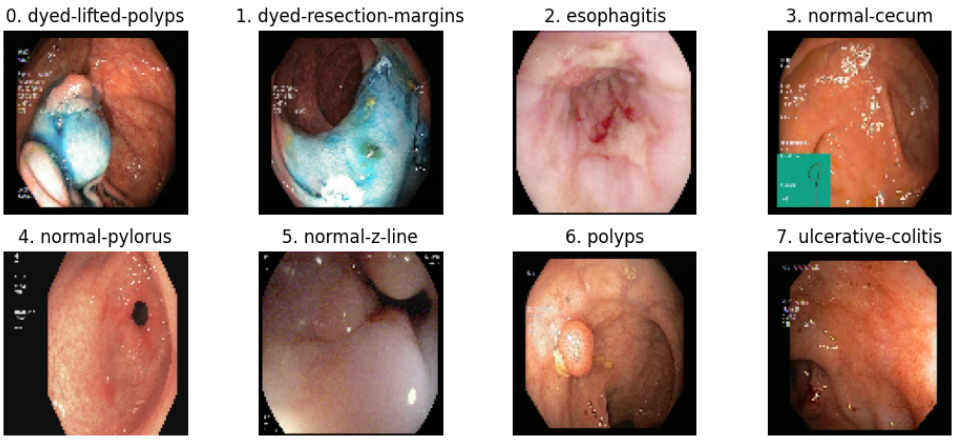


Figure 35: Representation of a sample of each category

```
# Converting y to scaler format
Y = np.reshape(y, (len(y), 1))

# split dataset to train and test set
X_train, X_test, y_train, y_test = train_test_split(
    X, Y, train_size=0.8, random_state=42
)
print(f"X_train: {X_train.shape}")
print(f"t_train: {y_train.shape}")
print(f"X_test: {X_test.shape}")
print(f"y_test: {y_test.shape}")


# defining training and test sets
x_train, x_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.3)
x_test = X_test
```

```python
# Dimension of the dataset
print(f"x_train:{x_train.shape},  y_train:{y_train.shape}")
print(f"x_train:{x_val.shape},  y_train:{y_val.shape}")
print(f"x_train:{x_test.shape},  y_train:{y_test.shape}")

# One Hot Encoding
y_train = to_categorical(y_train)
y_val = to_categorical(y_val)
y_test = to_categorical(y_test)

# Verifying the dimension after one hot encoding
print(f"x_train:{x_train.shape},  y_train:{y_train.shape}")
print(f"x_train:{x_val.shape},  y_train:{y_val.shape}")
print(f"x_train:{x_test.shape},  y_train:{y_test.shape}")

# Data augmentation
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
```

```python
from tensorflow.keras.applications import VGG19, ResNet50, EfficientNetB0, EfficientNetB7
from tensorflow.keras.models import load_model
import os

def get_model(architecture, input_shape, num_classes):
    # select the architecture
    if architecture == "vgg19":
        base_model = VGG19(include_top=False, weights="imagenet", input_shape=input_shape)
    elif architecture == "resnet50":
        base_model = ResNet50(include_top=False, weights="imagenet", input_shape=input_shape)
    elif architecture == "efficientnetb0":
        base_model = EfficientNetB0(include_top=False, weights="imagenet", input_shape=input_shape)
    elif architecture == "efficientnetb7":
        base_model = EfficientNetB7(include_top=False, weights="imagenet", input_shape=input_shape)
    else:
        raise ValueError("Unknown architecture")

    model = Sequential()
    model.add(base_model)
```

```
    model.add(Flatten())
    model.add(Dense(1024, activation='relu'))
    model.add(Dense(512, activation='relu'))
    model.add(Dense(256, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(128, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))

    return model

# Usage
architecture = "efficientnetb7"  # Change to "vgg19", "resnet50", "efficientnetb0", or "efficientnet
input_shape = (100, 100, 3)
num_classes = y_train.shape[1]

model = get_model(architecture, input_shape, num_classes)
model.summary()
```

```
Layer (type)                 Output Shape              Param #
=================================================================
efficientnetb7 (Functional) (None, 4, 4, 2560)        64097687
_________________________________________________________________
flatten (Flatten)            (None, 40960)             0
_________________________________________________________________
dense (Dense)                (None, 1024)              41944064
_________________________________________________________________
dense_1 (Dense)              (None, 512)               524800
_________________________________________________________________
dense_2 (Dense)              (None, 256)               131328
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense_3 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_4 (Dense)              (None, 8)                 1032
=================================================================
Total params: 106731807 (407.15 MB)
Trainable params: 106421080 (405.96 MB)
Non-trainable params: 310727 (1.19 MB)
```

```
# Initialise no. of training samples for each batch
batch_size = 32
```

```
# No. of iterations
epochs = 100

# Learning rate
learn_rate = 0.001

# Using Gradient Descent
sgd = SGD(learning_rate=learn_rate, momentum=0.9, nesterov=False)

# Using adam optimizer
# adam = Adam( learning_rate=learn_rate, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=

# Compiling the model
model.compile(optimizer=sgd, loss="categorical_crossentropy", metrics=["accuracy"])

from tensorflow.keras.callbacks import EarlyStopping

lrr = ReduceLROnPlateau(monitor="val_acc", factor=0.01, patience=3, min_lr=1e-5)

# Early Stopping
early_stop = EarlyStopping(monitor='val_loss', patience=20, restore_best_weights=True)

history = model.fit(
    datagen.flow(x_train, y_train, batch_size=batch_size),
    epochs=epochs,
    steps_per_epoch=x_train.shape[0] // batch_size,
    validation_data=(x_val, y_val),
    callbacks=[early_stop],
    verbose=1,
)
history = history.history
```

```
score = model.evaluate(x_test, y_test, verbose=0)
print("Test loss:", round(score[0], 3))
print("Test accuracy:", round(score[1], 3))

import matplotlib.pyplot as plt2
plt2.figure(figsize=(12, 6))
plt2.plot(history['val_accuracy'])
plt2.plot(history['accuracy'])
plt2.title('model accuracy')
plt2.ylabel('accuracy')
plt2.xlabel('epoch')
plt2.legend(['val', 'train'], loc='upper left')
plt2.show()
```

```
# Loss graph
plt.figure(figsize=(12, 6))
plt.plot(history['loss'], label='Training Loss')
plt.plot(history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
#Confusion Matrix
def cm_plt(ax, cm, classes, cmap, title, normalize):
    im = ax.imshow(cm, interpolation="nearest", cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
    # We want to show all ticks...
    ax.set(
        xticks=np.arange(cm.shape[1]),
        yticks=np.arange(cm.shape[0]),
        # ... and label them with the respective list entries
        xticklabels=classes,
        yticklabels=classes,
        title=title,
        ylabel="True label",
        xlabel="Predicted label",
    )

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right", rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
    fmt = ".2f" if normalize else "d"
    thresh = cm.max() / 2.0
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(
                j,
                i,
                format(cm[i, j], fmt),
                ha="center",
                va="center",
                color="white" if cm[i, j] > thresh else "black",
            )

    return ax

# Defining function for confusion matrix plot
```

```
def plt_confusion_mat(cm, classes, fig_size, cmap=plt.cm.Blues):
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=fig_size)
    ax1 = cm_plt(
        ax1,
        cm,
        classes,
        cmap,
        title="Confusion matrix, without normalization",
        normalize=False,
    )

    cmn = cm.astype("float") / cm.sum(axis=1)[:, np.newaxis]
    ax2 = cm_plt(
        ax2,
        cmn,
        classes,
        cmap,
        title="Normalized confusion matrix",
        normalize=True,
    )

# Making the predictions
y_pred = np.argmax(model.predict(x_test), axis=1)
y_true = np.argmax(y_test, axis=1)

# get confusion matrix
confuision_mat = confusion_matrix(y_true, y_pred)
# plot confusion_mat
plt_confusion_mat(confuision_mat, classes=categories, fig_size=(20, 7))
```

```
# Counting wrong images
def predict_categorie_img(img, model, categories):
    try:
        img = img[None, :, :, :]
    except:
        raise TypeError("test image dimension != 3")
    predict = model.predict(img)
    idx_cat = np.argmax(predict, axis=1)[0]
    return idx_cat, categories[idx_cat]

# Setting random seed
np.random.seed(42)

# Ready to save the selected images
selected_images = []
incorrect_images = []
```

```
for i in range(len(y)):
    img = X[i]
    pred_class_idx, pred_class_name = predict_categorie_img(img, model, categories)
    true_class_idx = y[i]
    true_class_name = categories[true_class_idx]

    if pred_class_idx != true_class_idx:
        incorrect_images.append((img, pred_class_name, true_class_name))

# Function for displaying incorrect images
def mostrar_imagenes_incorrectas(num_images):
    rows = 4  # Número de filas
    cols = (num_images // rows) + (1 if num_images % rows != 0 else 0)  # Calculate columns
    fig, axes = plt.subplots(rows, cols, figsize=(20, 8))
    axes = axes.flatten()

    for i in range(min(len(incorrect_images), num_images)):  # Mostrar hasta num_images imágenes inc
        img, pred_class_name, true_class_name = incorrect_images[i]

        axes[i].imshow(img[:, :, ::-1])
        axes[i].set_title(f"Pred: {pred_class_name}\nTrue: {true_class_name}")
        axes[i].axis("off")

    # Hide empty axes
    for i in range(num_images, len(axes)):
        axes[i].axis("off")

    plt.tight_layout()
    plt.show()

    len(incorrect_images)
```

```
# Function to display incorrect images of a specific category
def mostrar_imagenes_incorrectas_por_categoria(num_images, categoria_especifica):
    filtered_images = [img for img in incorrect_images if img[2] == categoria_especifica]
    print(f"Cantidad de imágenes incorrectas en la categoría '{categoria_especifica}': {len(filtered_

    rows = 4  # Número de filas
    cols = (num_images // rows) + (1 if num_images % rows != 0 else 0)  # Calculate columns
    fig, axes = plt.subplots(rows, cols, figsize=(20, 8))
    axes = axes.flatten()

    for i in range(min(len(filtered_images), num_images)):  # Mostrar hasta num_images imágenes inco
        img, pred_class_name, true_class_name = filtered_images[i]
```

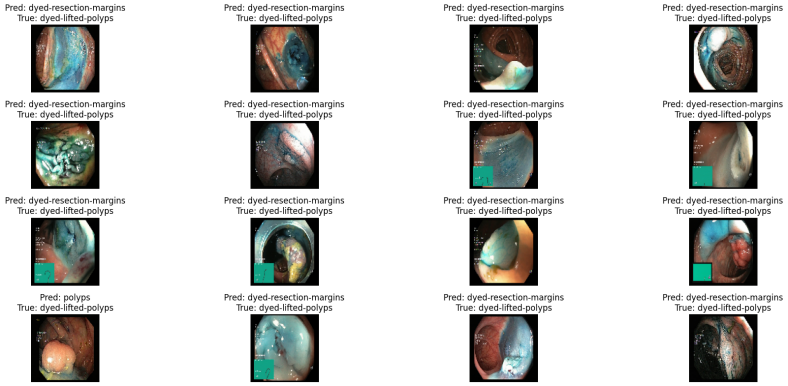


Figure 36: dyed-lifted-polyps

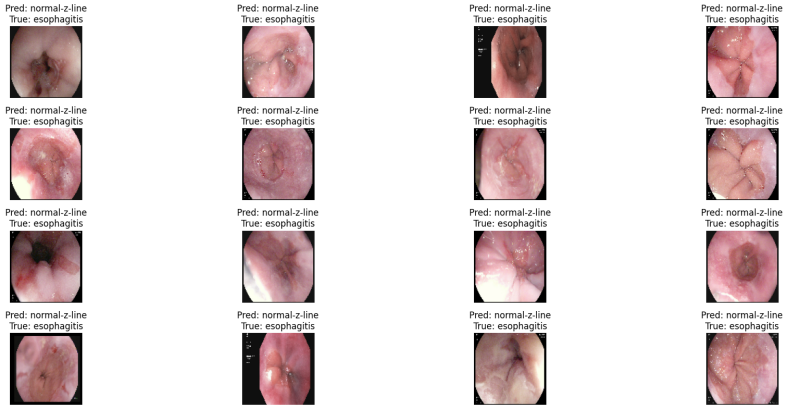


Figure 37: esophagitis

```
        axes[i].imshow(img[:, :, ::-1])
        axes[i].set_title(f"Pred: {pred_class_name}\nTrue: {true_class_name}")
        axes[i].axis("off")

    #
    for i in range(num_images, len(axes)):
        axes[i].axis("off")

    plt.tight_layout()
    plt.show()

# Call the function with the number of incorrect images you want to display and the specific categor
categoria_especifica = 'dyed-lifted-polyps'  # Cambia esto a la categoría que desees filtrar
mostrar_imagenes_incorrectas_por_categoria(16, categoria_especifica)

categoria_especifica2 ='esophagitis'
mostrar_imagenes_incorrectas_por_categoria(16,categoria_especifica2)
```

```
!pip install shap
```

```
import shap
from tensorflow.keras.applications.efficientnet import preprocess_input as preprocess_input_efficien

# Define the output directory in Google Drive
output_directory = '/content/drive/My Drive/model_outputs'
if not os.path.exists(output_directory):
    os.makedirs(output_directory)
```

```
# ES POSIBLE QUE NO SALEN IMAGENES IGUALES QUE EN EL DOCUMENTO DE TFG, DEBIDO AL PROBLEMA
# DE DESCOMPRIR Y EXTRACCION, DE MANERA QUE ORDENA DE MANERA DIFERENTE LAS IMAGENES,
# A PESAR DE QUE ESTOY SACANDO EL PRIMERO DE CADA. CON EL SORT A VECES NO FUNCIONA,
# DEPENDE MUCHO DE SOFTWARE, LA CONFIGURACION DEL SISTEMA Y MANERA DE DESCOMPRIMIR.

# Sort images and tags by category
sorted_indices = np.argsort(y)
X_sorted = X[sorted_indices]
y_sorted = y[sorted_indices]

# Function to obtain the first image of each category
def get_first_images_per_category(X, y, categories):
    selected_images = []
    selected_indices = []
    for category in range(len(categories)):
        for i in range(len(y)):
            if y[i] == category:
                selected_images.append(X[i])
                selected_indices.append(i)
                break
    return selected_images, selected_indices

# Get the first image from each category after sorting
selected_images, selected_indices = get_first_images_per_category(X_sorted, y_sorted, categories)

# Ready to save the selected images
true_class_names = [categories[y_sorted[idx]] for idx in selected_indices]

# Display selected images
plt.figure(figsize=(20, 8))
for i, img in enumerate(selected_images):
    true_class = y_sorted[selected_indices[i]], categories[y_sorted[selected_indices[i]]]

    plt.subplot(2, 5, i + 1)
    plt.imshow(img[:, :, ::-1])
    plt.title(f"True:[{true_class}]")
    plt.axis("off")
```

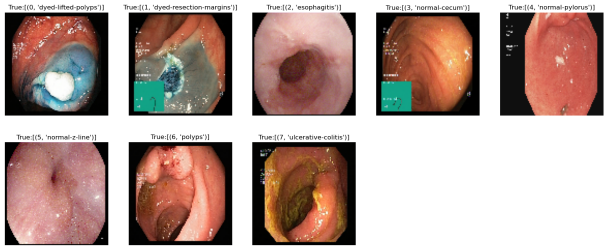


Figure 38: Selected images

```
plt.show()
```

```
# GOOGLE COLAB PRESENTA INCOMPABILIDAD CON LA FUNCION DE SHAP.IMAGE.PLOT, DE MANERA
# QUE, LAS GRAFICAS SALEN DESCUADRADAS, POR LO TANTO, PARA MEJORAR LA PRESENTACION,
# LOS RESULTADOS QUE HE PRESENTADO AL DOCOMENTO LO HE EFECTUADO DESDE OTRA PLATILLA,
# CON ESTE MISMO TROZO DE CODIGO.

# Make sure that 'selected_images' is defined and is a numpy array.
selected_images = np.array(selected_images)

# Pre-process selected images
selected_images_preprocessed = preprocess_input(selected_images)

# Create the explainer with a subset of x_train
explainer = shap.GradientExplainer(model, x_train[:100], local_smoothing=0)

# Get SHAP values for selected images
shap_values = explainer.shap_values(selected_images_preprocessed)

# Visualise the explanations
for i in range(len(selected_images)):
    plt.figure(figsize=(10, 10))
    shap.image_plot(shap_values[np.argmax(y_test[i])][i], selected_images_preprocessed[i], show=Fals
    plt.title(f"True: [{true_class_names[i]}]")
    # Save figure
    shap_output_path = os.path.join(output_directory, f"shap_sample_{i+1}.png")
    plt.savefig(shap_output_path)
    plt.close()
```

```
output_directory_gradcam = '/content/drive/My Drive/model_outputs_gradcam'
# DENSE SELECTION
```

```
base_model = EfficientNetB7(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
base_model.summary()
```

```
from tensorflow.keras.models import Model
from tensorflow.keras.applications.efficientnet import EfficientNetB7, preprocess_input
from tensorflow.keras.preprocessing import image
import cv2
import numpy as np
import os
import matplotlib.pyplot as plt
import tensorflow as tf

output_directory_gradcam = '/content/drive/My Drive/model_outputs_gradcam'

if not os.path.exists(output_directory_gradcam):
    os.makedirs(output_directory_gradcam)

# Grad-CAM Function
def grad_cam(model, img_array, category_index, layer_name):
    grad_model = tf.keras.models.Model([model.inputs], [model.get_layer(layer_name).output, model.ou
    with tf.GradientTape() as tape:
        conv_outputs, predictions = grad_model(img_array)
        loss = predictions[:, category_index]

    output = conv_outputs[0]
    grads = tape.gradient(loss, conv_outputs)[0]

    gate_f = tf.cast(output > 0, 'float32')
    gate_r = tf.cast(grads > 0, 'float32')
    guided_grads = gate_f * gate_r * grads

    weights = tf.reduce_mean(guided_grads, axis=(0, 1))
    cam = tf.reduce_sum(tf.multiply(weights, output), axis=-1)

    cam = np.maximum(cam, 0)
    cam = (cam - cam.min()) / (cam.max() - cam.min())
    heatmap = cv2.resize(cam, (224, 224))
    return heatmap

# Image upload and pre-processing function
def load_and_preprocess_image(img_array):
    img_array = cv2.resize(img_array, (224, 224))
    img_array = np.expand_dims(img_array, axis=0)
    return preprocess_input(img_array)

# Function to plot heatmap together with the original image
```

```
def plot_heatmap(heatmap, img_array, true_class_name, output_path, alpha=0.6):
    original_img = cv2.cvtColor(img_array, cv2.COLOR_BGR2RGB)
    original_img = cv2.resize(original_img, (224, 224))

    heatmap = np.uint8(255 * heatmap)
    heatmap = cv2.applyColorMap(heatmap, cv2.COLORMAP_JET)

    superimposed_img = heatmap * alpha + original_img * (1 - alpha)
    superimposed_img = np.uint8(superimposed_img)

    plt.figure(figsize=(12, 6))

    # Show original image
    plt.subplot(1, 2, 1)
    plt.imshow(original_img)
    plt.title('Original')
    plt.axis('off')

    # Show image with Grad-CAM
    plt.subplot(1, 2, 2)
    plt.imshow(superimposed_img)
    plt.title(f'True: {true_class_name}')
    plt.axis('off')

    plt.savefig(output_path)
    plt.close()

# Create the model with EfficientNetB7
base_model = EfficientNetB7(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
x = base_model.output
x = Flatten()(x)
x = Dense(1024, activation='relu')(x)
output_layer = Dense(len(categories), activation='softmax')(x)  # Asegúrate de que la capa de salida
model = Model(inputs=base_model.input, outputs=output_layer)

# Pre-process selected images
selected_images_preprocessed = np.array([load_and_preprocess_image(img) for img in selected_images])

# Check that the lengths match
assert len(selected_images) == len(true_class_names), "Las longitudes de selected_images y true_class

# Aplication Grad-CAM
for i in range(len(selected_images)):
    preprocessed_image = selected_images_preprocessed[i]
    predictions = model.predict(preprocessed_image)
    predicted_class = np.argmax(predictions[0])

    true_class_name = true_class_names[i]
```

```
    # Print for debugging
    print(f'Predicted class index: {predicted_class}')
    print(f'Number of categories: {len(categories)}')
    print(f'True class name: {true_class_name}')

    if predicted_class < len(categories):
        heatmap = grad_cam(model, preprocessed_image, predicted_class, 'block2a_project_conv')

        # Save the figure in Grad-CAM folder
        gradcam_output_path = os.path.join(output_directory_gradcam, f"gradcam_sample_{i+1}.png")
        plot_heatmap(heatmap, selected_images[i], true_class_name, gradcam_output_path)
    else:
        print(f'Error: predicted class index {predicted_class} is out of range for categories.')
```

```
#Save to google Drive files

#!pip install pillow

import shutil
from PIL import Image
# Define the paths to the source folders and the new destination folder.
carpeta_a = '/content/drive/My Drive/model_outputs_gradcam'
carpeta_b = '/content/drive/My Drive/model_outputs'
carpeta_destino = '/content/drive/My Drive/carpeta_destino'

# Create the destination folder if it does not exist
if not os.path.exists(carpeta_destino):
    os.makedirs(carpeta_destino)

# Get lists of files in each folder
imagenes_a = sorted(os.listdir(carpeta_a))
imagenes_b = sorted(os.listdir(carpeta_b))

# Make sure that both folders have the same number of images.
assert len(imagenes_a) == len(imagenes_b), "Las carpetas no tienen el mismo número de imágenes"

# Merge the images in the destination folder
for i in range(len(imagenes_a)):
    # Crear una subcarpeta para cada par de imágenes
    subcarpeta = os.path.join(carpeta_destino, f"subcarpeta_{i+1}")
    if not os.path.exists(subcarpeta):
        os.makedirs(subcarpeta)

    # Get the full path to each image
    imagen_a_path = os.path.join(carpeta_a, imagenes_a[i])
```

```
    imagen_b_path = os.path.join(carpeta_b, imagenes_b[i])

    imagen_a = Image.open(imagen_a_path)
    imagen_b = Image.open(imagen_b_path)

    width_a, height_a = imagen_a.size
    width_b, height_b = imagen_b.size

    # Create a new image with appropriate dimensions to contain both images.
    combined_width = width_a + width_b
    combined_height = max(height_a, height_b)
    combined_image = Image.new('RGB', (combined_width, combined_height))

    # Paste the two images into the new image
    combined_image.paste(imagen_a, (0, 0))
    combined_image.paste(imagen_b, (width_a, 0))

    # Define the name of the merged image in the destination subfolder
    imagen_combinada_destino = os.path.join(subcarpeta, f"imagen_combinada_{i+1}.jpg")

    # Save the merged image in the target subfolder
    combined_image.save(imagen_combinada_destino)

print("Imágenes combinadas y copiadas a las subcarpetas de destino con éxito.")

#!pip install pdfkit
#!apt-get install -y wkhtmltopdf
```

```
#Function to transform html to PDF, and save in the path

import pdfkit

# Define the URLs of the images
image_url_1 = "https://drive.google.com/uc?export=view&id=YOUR_IMAGE_ID_1"
image_url_2 = "https://drive.google.com/uc?export=view&id=YOUR_IMAGE_ID_2"

# Defines the HTML content with the images included
html_content = fr"""
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Endoscopic Report</title>
  <style>
        body {{
```

```
            font-family: Arial, sans-serif;
            margin: 20px;
        }}

        h1 {{
            font-size: 24px;
            font-weight: bold;
        }}

        h2 {{
            font-size: 18px;
            font-weight: bold;
        }}

        p {{
            font-size: 16px;
        }}

        .image-container {{
            margin-bottom: 10px;
        }}

        .image {{
            width: 200px;
            height: 150px;
            border: 1px solid #ccc;
            margin-bottom: 5px;
        }}
  </style>
</head>
<body>

Output

"""

# Define the path where you want to save the PDF file in Google Drive
drive_folder = '/content/drive/My Drive/'
pdf_filename = "medical_report.pdf"
pdf_path = drive_folder + pdf_filename

# Generate PDF
pdfkit.from_string(html_content, pdf_path)

print(f"PDF successfully created and saved to Google Drive: {pdf_path}")
```

14 Appendix B: Supplementary information

14.1 Practical example of Shap-Value

Let's combine an example to visualize how the Shapley value distributes earnings fairly.

Today overtime, a program $C = 500$ lines of code need to be written, today the product manager found three programmers to complete, according to the completion of the amount of bonus:

$$v(\{1\}) = 100, \quad v(\{2\}) = 125, \quad v(\{3\}) = 50$$

Explanation: programmer 1 can write 100 lines independently, programmer 2 can write 125 lines independently, programmer 3 can write 50 lines.

$$v(\{1,2\}) = 270, \quad v(\{2,3\}) = 350, \quad v(\{1,3\}) = 375$$

Explanation: 1,2 can write 270 lines cooperatively, 2,3 can write 350 lines cooperatively, and 1,3 can write 375 lines cooperatively.

$$v(\{1,2,3\}) = 500$$

Explanation: 3 people together can complete 500 rows.

Then the co-operation process will have 6 possible scenarios to consider:

A. No. 1 invites 2 to join to form an S-coalition, and 1,2 invites 3 to join in co-writing.

B. No. 1 invites No. 3 to join as Panel S and No. 2 to join Panel S.

C. No. 2 invites No. 1 to join as Panel S and No. 3 to join Panel S.

D. No. 2 invites No. 3 to join as Panel S. No. 1 joins Panel S.

E. No. 3 invites No. 1 to join as Panel S. No. 2 joins Panel S.

F. No. 3 invites No. 2 to join as Panel S. No. 1 joins Panel S.

Then according to the Shapley value method, the rational allocation, whether it is fair or not mainly examines the marginal contribution. The Shapley method defines the marginal contribution of i joining organisation S:

$$\text{Marginal contribution} = v(S \cup \{i\}) - v(S)$$

The marginal contribution of the above examples can then be expressed as follows:

Probability	**Order of arrival**	**1's marginal contribution**	**2's marginal contribution**	**3's marginal contribution**
$\frac{1}{6}$	$1 \to 2 \to 3$: 123	$v(\{1\}) = 100$	$v(\{1,2\}) - v(\{1\}) = 270 - 100 = 170$	$v(\{1,2,3\}) - v(\{1,2\}) = 500 - 270 = 230$
$\frac{1}{6}$	$1 \to 3 \to 2$: 132	$v(\{1\}) = 100$	$v(\{1,2,3\}) - v(\{1,3\}) = 500 - 375 = 125$	$v(\{1,3\}) - v(\{1\}) = 375 - 100 = 275$
$\frac{1}{6}$	$2 \to 1 \to 3$: 213	$v(\{1,2\}) - v(\{2\}) = 270 - 125 = 145$	$v(\{2\}) = 125$	$v(\{1,2,3\}) - v(\{1,2\}) = 500 - 270 = 230$
$\frac{1}{6}$	$2 \to 3 \to 1$: 231	$v(\{1,2,3\}) - v(\{2,3\}) = 500 - 350 = 150$	$v(\{2\}) = 125$	$v(\{2,3\}) - v(\{2\}) = 350 - 125 = 225$
$\frac{1}{6}$	$3 \to 1 \to 2$: 312	$v(\{1,3\}) - v(\{3\}) = 375 - 50 = 325$	$v(\{1,2,3\}) - v(\{1,3\}) = 500 - 375 = 125$	$v(\{3\}) = 50$
$\frac{1}{6}$	$3 \to 2 \to 1$: 321	$v(\{1,2,3\}) - v(\{2,3\}) = 500 - 350 = 150$	$v(\{2,3\}) - v(\{3\}) = 350 - 50 = 300$	$v(\{3\}) = 50$

- Programmer 1 value for Shapley is:

$$\frac{1}{6}(100 + 100 + 145 + 150 + 325 + 150) = \frac{970}{6}$$

- Programmer 2 value for Shapley is:

$$\frac{1}{6}170 + \frac{1}{6}125 + \frac{1}{6}125 + \frac{1}{6}125 + \frac{1}{6}125 + \frac{1}{6}300 = \frac{970}{6}$$

- Programmer 3 value for Shapley is:

$$\frac{1}{6}230 + \frac{1}{6}275 + \frac{1}{6}230 + \frac{1}{6}225 + \frac{1}{6}50 + \frac{1}{6}50 = \frac{1060}{6}$$

It can be seen that, as stated in the premise, the sum is $v(\{1,2,3\}) = 500$.

To sum up, Number 1 should get 32.3% of the total bonus.
Number 2 should receive 32.3% of the total prize money.
Number 3 should get 35.3% of the total bonus.

14.2 LIME

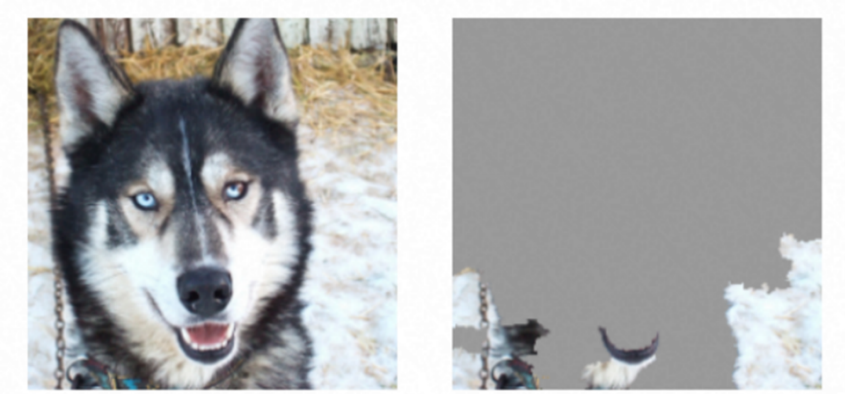
LIME, Local Interpretable Model-Agnostic Explanations, is a model-agnostic method for explaining individual sample predictions, proposed in 2017. The core idea is to interpret the sample to be explained by partitioning its features into several thousand groups, then sampling locally around these groups to obtain a new artificial sample set S. Subsequently, a new linear model is trained to learn the output of the original model near sample S.

The formal expression is:

$$\xi(x) = \arg\min_{g \in G} L(f, g, \pi_x) + \Omega(g)$$

- $g(\cdot)$ is the interpretable function.
- π_x is the weight of the artificial sample.
- $\Omega(\cdot)$ constrains the complexity of g.

If we concretise the above explanation of the joint formula, then the result obtained will be as follows: The figure uses LIME to explain that the reason the model incorrectly predicted a Husky as a Wolf is that it initially focused only on the snow in the image. The figure is an example from the LIME paper. The original features of the image are pixels, and it selects image regions to create super-pixel representations. Through interpretability analysis, it verifies that the model learned the correlation between the wolf and the snow, rather than understanding the causal relationship between the wolf and the snow.



(a) Husky classified as wolf (b) Explanation

Figure 39: How LIME works

14.3 Others Visualization Methods for Kernel-Shap

1. Force Plot

A force plot shows how individual feature contributions (Shapley values) combine to form the model's prediction. It typically illustrates:

- The baseline model prediction.
- Positive contributions (features that push the prediction higher).
- Negative contributions (features that push the prediction lower).
- The final prediction for the sample.

Force plots help visualize the push-and-pull effect of each feature on the model's prediction, making it clear how each feature influences the outcome.

2. Bar Plot

A bar plot displays the Shapley values of features for a single prediction or aggregated over multiple predictions. It helps in:

- Comparing the relative importance of different features.

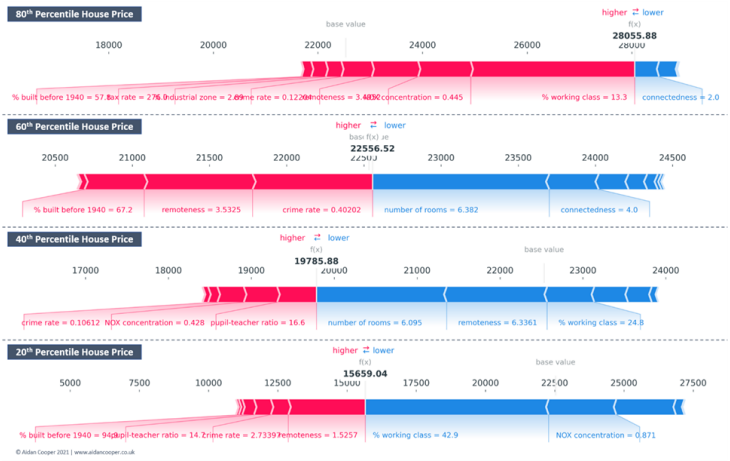


Figure 40: Force Plot

- Identifying which features have the most significant impact on the model's predictions.

Bar plots provide a straightforward way to rank features by their importance, highlighting which features are driving the model's decisions.

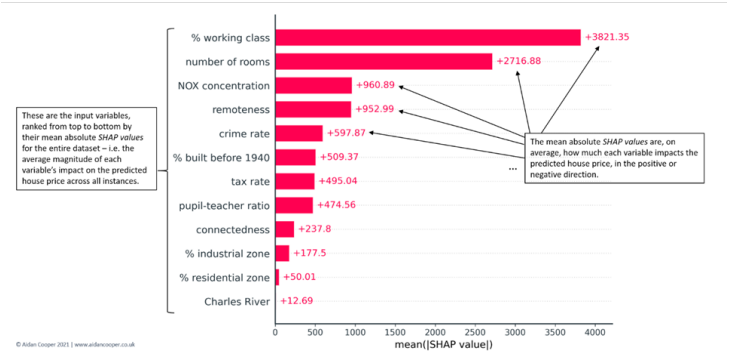


Figure 41: Bar Plot

3. Beesworm Plot

SHAP feature importance bar plots are a superior approach to traditional alternatives but in isolation, they provide little additional value beyond their more rigorous theoretical underpinnings. Beeswarm plots are a more complex and information-rich display of SHAP values that reveal not just the relative importance of features, but their actual relationships with the predicted outcome.

4. Dependence Plot

A dependence plot shows how the Shapley value of a particular feature changes as the value

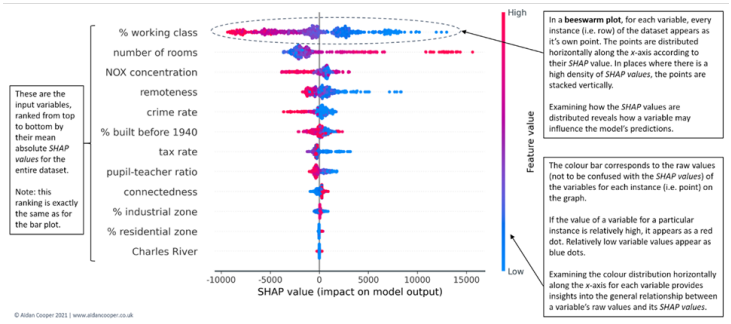


Figure 42: Beesworm Plot

of that feature changes. This plot typically:

- Plots the value of the feature on the x-axis.
- Plots the corresponding Shapley value on the y-axis.
- May include colour coding to show the interaction with another feature.

Dependence plots are useful for identifying the nature of the relationship between a feature and the model's prediction, including any nonlinear effects or interactions with other features.

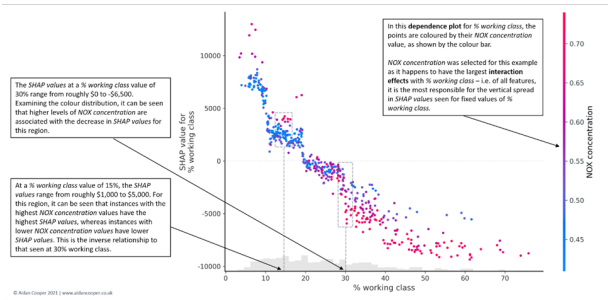


Figure 43: Dependence Plot

Using various visualization techniques such as force plots, bar plots, and dependence plots, users can gain a deeper understanding of how each feature influences the model's decisions, ensuring transparency and trust in the model's outputs.

In summary, Kernel Shap offers the following benefits:

- Interpretability: Provides intuitive interpretation by attributing model predictions to individual features.

- Model-independent: Can be applied to any machine learning model, making it a versatile tool for interpretability.
- Theoretical Foundation: Combines the intuition of LIME with the theoretical foundation of Shapley values.

Kernel-Shap combines the strengths of LIME and Shapley values to provide a powerful and unified approach to interpreting machine learning models, ensuring that interpretations are both intuitive and theoretically sound.

15 Appendix C: Supplementary images

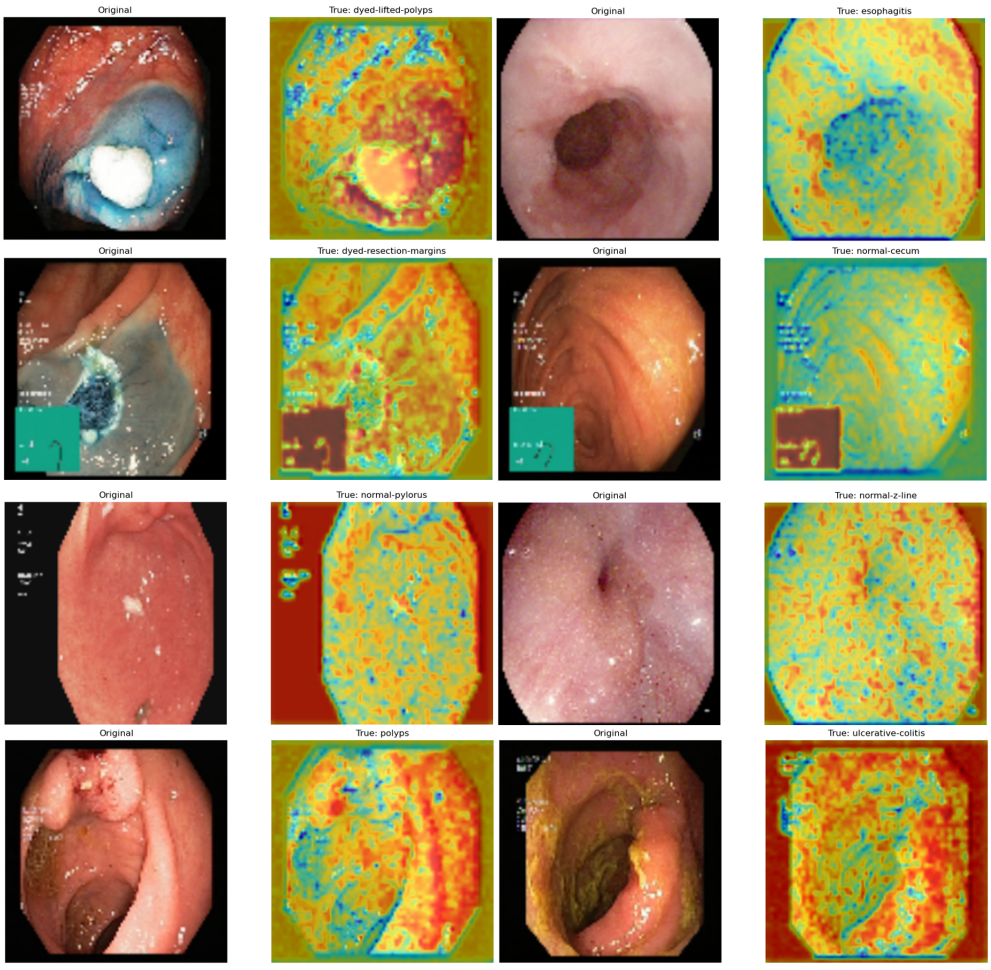


Figure 44: Representation of Grad-CAM

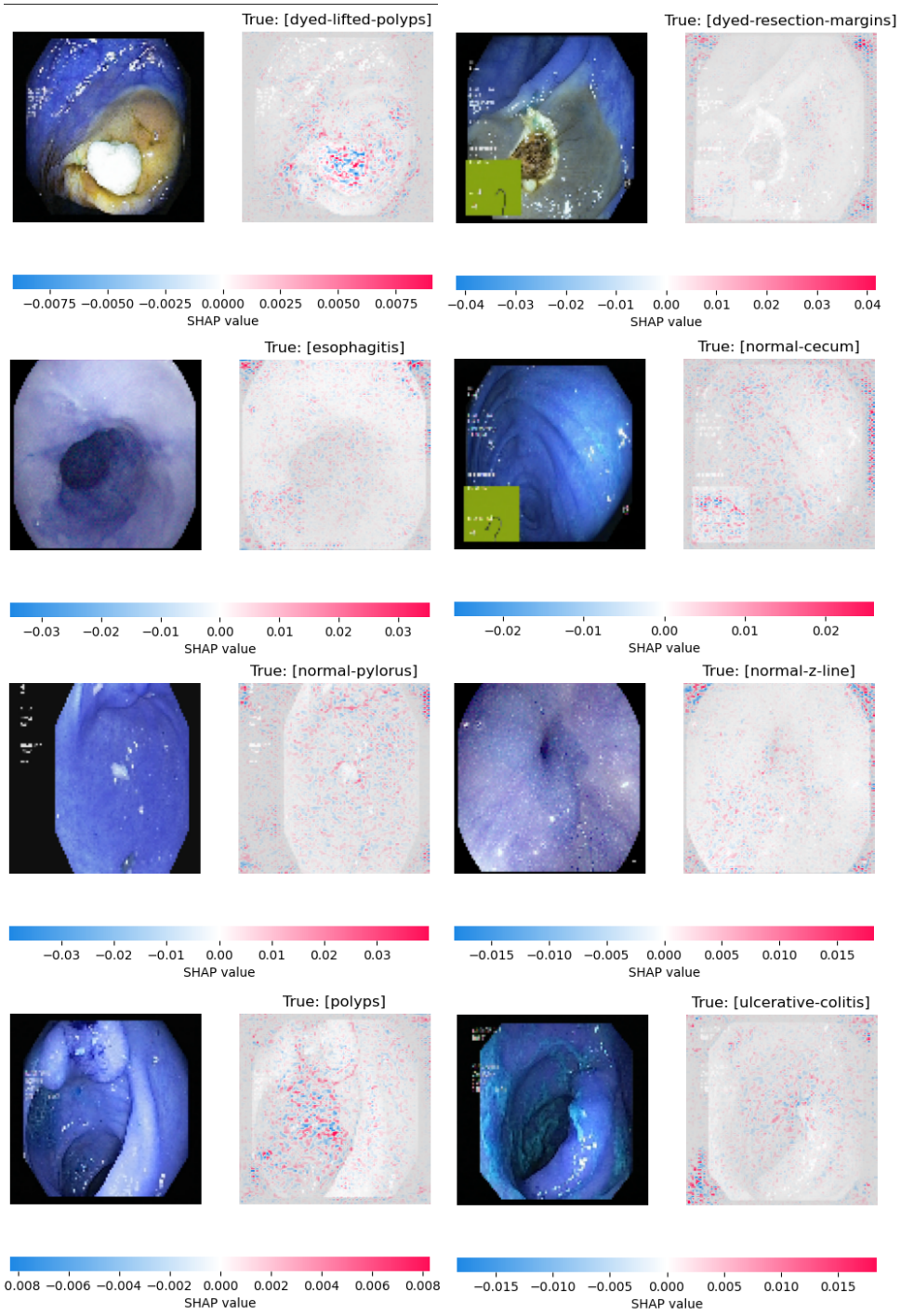


Figure 45: Heatmaps of Kernel-Shap