Faculty of
Mathematics
and Informatics

VILNIUS UNIVERSITY
FACULTY OF MATHEMATICS AND INFORMATICS
MASTER'S STUDY PROGRAMME
MODELLING AND DATA ANALYSIS

Interpretable Deep Learning Methods For Tissue Classification in Histopathology Images

Master's thesis

Author: Augustė Valatkaitė
VU email address: auguste.valatkaite@mif.stud.vu.lt
Supervisor: Dr. Mindaugas Morkūnas

Vilnius

2024

Contents

Abstract

Digital histopathology enables the investigation of whole slide images (WSI) in a digital environment and is currently employed in medical trials, research, telemedicine, and education. This technology, when combined with deep learning models, holds the potential to make significant strides in diagnostic processes, leading to more accurate treatment plans and enhanced diagnostic precision. Despite its promise, the adoption of this methodology in real medical practice is limited. Deep neural networks face challenges, with one of the most critical being the lack of model interpretation, creating a "black box" issue. Typically, deep learning models do not provide insights into how decisions are made, raising concerns about transparency, correctness, and potential biases. This research investigates the interpretation methods to increase the transparency of convolutional neural networks for tissue classification in histopathology images. Four interpretation methods - Integrated Gradients, Integrated Gradients with Noise Tunnel, LIME, and GradCAM - were examined. The obtained results were quantitively evaluated. The GradCAM method outperformed other methods by providing the most detailed and higher quality outputs than the other techniques. Through experimental analysis, it was determined that interpretation map quality is not influenced by classification outcomes.

Keywords: Interpretable methods, deep learning, digital histopathology, medical imaging field, GradCAM, Integrated Gradients, LIME, NoiseTunnel.

Santrauka

Skaitmeninė histopatologija tai nauja metodika, kuri leidžia tirti pilną pjūvio vaizdą (angl. whole slide image) skaitmeniniu būdu. Šiuo metu, ši technologija yra plačiai pritaikoma klinikinėje praktikoje, bei moksliniuose tyrimuose ir edukacijos tikslais. Sujungus skaitmeninę histopatalogiją ir giliuosius neuroninius tinklus, atsiveria galimybės pagerinti diagnostikos procesus, sudaryti tikslesnį gydymo planą, bei padidinti diagnozės tikslumą. Nors giliųjų neuroninių tinklų pritaikymas histopatologijos tyrimuose teikia daug vilčių, tačiau šiuo metu ji nėra plačiai naudojama realioje medicininėje praktikoje. Metodikoje naudojami giliojo mokymosi modeliai turi keletą silpnybių, kurios stabdo spartesnį technologijos pritaikymą realiame pasaulyje. Vienas iš trūkumų - tipiniai modeliai nepateikia paaiškinimo kaip buvo priimti sprendimai nusakant diagnozę. Tai vadinama "black box" problema. Trūkstant aiškumo modelyje, nėra tiksliai žinoma ar modelis veikia teisingai, ar sprendimai yra priimami naudojant svarius kintamuosius. Taip pat nėra užtikrinama, jog modelis nebūtų šališkas. Šiame tyrime yra apžvelgiami paaiškinamieji metodai, siekiant pagerinti giliųjų neuroninių tinklų aiškumą atliekant histopatologinių audinių klasifikavimą. Eksperimentas yra atliekamas su keturiais paaiškinamaisiais metodais (Integrated Gradients, Integrated Gradients su NoiseTunnel, LIME ir GradCAM). Atlikus kokybinę analizę buvo nustatya, kad GradCAM metodas sugeneravo geriausios kokybės ir labiausiai detalizuotus paaiškinimus, negu kiti metodai. Taip pat buvo pastebėta, kad interpretavimo metodams įtakos neturi klasifikavimo rezultatai.

1 Introduction

Histopathology analysis serves as the gold standard procedure for detecting cancer. In this process, pathologists examine microscopic portions of a patient's tissues obtained through biopsy, which are then stained with Hematoxylin and Eosin [7]. While the traditional method involves manually examining stained tissue slices on a glass slide under a microscope - critical for detecting abnormalities in a patient's body - it relies on the subjective evaluation of a pathologist. Moreover, the investigation process is manual, time-consuming, and labor-intensive [23]. Therefore, the digital histopathology approach is gaining increased attention as it aims to assist in transforming traditional histopathology into novel and advanced diagnostic methods.

Digital histopathology, also known as computational histopathology, is a technology that applies quantitative diagnosis to pathological samples. It aims to reduce the subjectivity of pathologist assessments, save time in examining histopathology images, minimize errors due to human bias, and decrease manual procedures for pathologists. This method utilizes RGB digitized histology images [28]. Typically, digitized images are obtained from a slide scanner, capturing a tissue slide digitally and saving it as a whole slide image (WSI). The acquired images contain extensive structural information that can be further utilized in computer-assisted diagnosis (CAD). Implementing computer-assisted analysis can benefit various areas of the medical imaging field.

By leveraging the morphology and genomic profile of a patient's tissue, pathologists can use computational tools for image classification or detecting regions of interest (RoI). Accurately distinguishing between healthy and unhealthy patients is crucial for determining treatment plans [15]. Using additional quantitative information, such as dimensions of different critical components in the tissue (for example: tumor morphology and architecture), and with software assistance, pathologists can address more challenging tasks. Therefore, by combining complex features, numerous parameters, and computational tools, pathologists advance diagnostic capabilities, providing a crucial clinical solution for determining a patient's diagnosis and creating more personalized treatment plans [15]. These solutions necessitate the implementation of machine learning methods into computer-assisted diagnosis, with machine learning methods rapidly expanding in various industries, including the medical imaging field.

There is a tendency towards the implementation of convolutional neural networks (CNNs) in the pathological diagnosis of various types of cancer [15]. The convolutional neural network is a type of machine learning method, which mimics the multilayered human cognitive system [42]. This machine learning method plays a crucial participant in enhancing the quality of the medical diagnostic field and offers several advantages, that can contribute to the popularity of computer-assisted diagnoses. As histopathological image analysis becomes more complex, convolutional neural networks are becoming increasingly important for detecting features, clustering, segmentation, and classification [3].

Several studies have demonstrated, that certain machine learning algorithms can be more sensitive than an experienced pathologist in detecting individual cancer cells in digital image [38]. Moreover, algorithm-assisted pathologists have demonstrated higher accuracy scores than either the algorithm or the pathologist alone.

However, deep learning models exhibit several weaknesses. Many machine learning models are created with limited versatility, suitable only for the specific task they were trained on [38]. Models

also demonstrate sensitivity to poor optical focus, leading to a decline in performance, especially when applied to external datasets during testing [41]. Furthermore, machine learning models are often illustrated as black boxes, lacking interpretation for the decision they make [30]. This lack of interpretability poses challenges for humans in comprehending the inner workings of deep learning methods and can result in erroneous decision-making due to systematic biases in the training dataset [41]. The absence of interpretation restricts the application of models in real-world medical practice. The incorporation of interpretability techniques is crucial in machine learning models for the analysis of medical images, enabling pathologists to trust the model, accept its results, and obtain regulatory approval [11].

In this thesis, the main goal was to propose an interpretable deep learning method that is transparent and provides better interpretability. The main contributions of this research are:

- apply four different post hoc interpretation methods to the trained convolutional neural network,
- conduct visual analysis and quality assessment for the generated interpretation maps,
- provide meaningful insights into the obtained results by elucidating the unique characteristics of each interpretation method observed during this research.

This research is arranged as follows. The second section presents a literature review, describes the current challenges of convolutional neural network implementation in computer-assisted diagnostic tools, and introduces numerous methods of promising interpretation methods. The third section explains the methodology. The fourth section presents the results of the performed experiment. The fifth discusses the obtained results. The research ends with conclusions in the sixth section.

2 Literature review

2.1 Digital histopathology

Cancer, being one of the most dangerous diseases, poses a significant threat, with certain types exhibiting a high mortality rate. To prevent the progression of cancer into incurable stages, scientists are actively working on developing sensitive screening and diagnostic methods capable of detecting cancer in its early stages. Various screening techniques play a crucial role in the endeavor, including magnetic resonance imaging (MRI), computed tomography (CT), histopathology slides, X-ray, and mammography [34]. These techniques enable the investigation of internal structures and metabolic processes within the human body without resorting to extensive invasive procedures [32].

The digitization trend is permeating clinical workflows, and the field of medical imaging is no exception. However, different screening techniques have demonstrated varying progress in transitioning into the digital and computational domains. MRI and CT have swiftly adapted to the digital radiology environment [31], while the histopathology process is gradually moving towards digitization, paving the for integration of digital histopathology into routine medical practice.

Currently, histology analysis stands out as one of the most reliable diagnostic tools for detecting abnormalities within a patient's body. Pathologists meticulously examine tissue biopsies on pathology slides, often stained with hematoxylin and eosin (H&E) for enhanced visualization [31] (Figure 1). These histopathology samples yield crucial phenotypic information at the cellular level, offering insights into cancer aggressiveness and the impact of cancer cell behavior on healthy tissues [3].

The histopathology analysis process, until now, involves two primary stages. Initially, the pathologist visually inspects tissue samples on the slide, identifying components such as epithelium, and necrosis. Subsequently, they highlight the region of interest (RoI) within the tissue and conduct a thorough evaluation based on criteria like tissue distribution, morphology, and topology [29]. Both stages of diagnosis are currently performed manually, leading to several challenges. The accuracy of identifying abnormal samples highly relies on the experience of the pathologist. Additionally, subtle changes in tissue, difficult for the human eye to detect, may result in misdiagnosis, posing a risk of false positives or false negatives. Furthermore, the manual analysis process is time-consuming, tedious, and expensive [42].

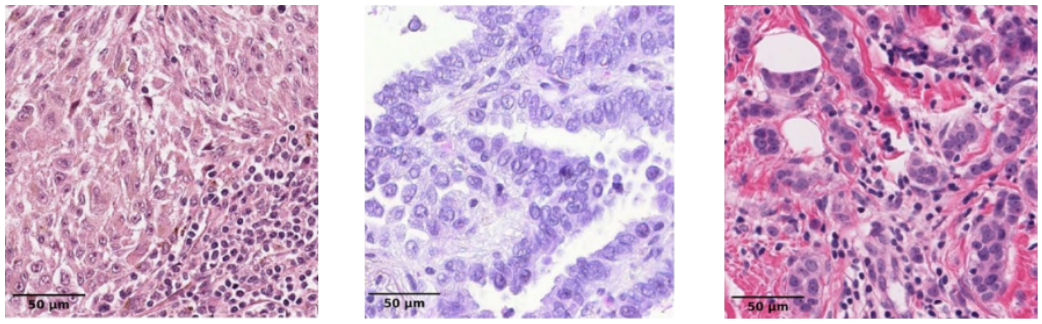

Figure 1: The examples of tissues stained with H&E [13].

The emergence of current challenges has increased the demand for automated solutions in cancer diagnosis. Over the recent decades, scientists and pathologists have adopted the practice of digitally

capturing the entire tissue slides using a slide scanner, resulting in a whole slide image (WSI). This development laid the foundation for digital pathology. Initially, digital pathology was primarily employed for archival purposes and remote consultations. However, early implementations did not significantly enhance the efficiency of a pathologist's primary workflow [38].

Nevertheless, digital pathology, also referred to as computational pathology, exhibits promising potential for application in decision-making tools and personalized medicine through the integration of machine learning methods. Digitized WSIs possess features that can be leveraged for the implementation of artificial intelligence (AI) applications. For instance, automatic feature extraction from images can augment information, providing more comprehensive diagnoses and prognoses for patients [15]. This approach facilitates quantitative analysis, extracting information about the tumor's architectural, shape, and texture-based characteristics - details often imperceptible to the human eye of an experienced pathologist [41]. Importantly, the extracted features can enhance the dataset for subsequent use in models handling classification, segmentation, or clustering of digital images. However, the extraction of these features from digital slide images requires the utilization of numerous parameters, surpassing the capabilities of simple models. Consequently, more sophisticated methods, such as machine learning applications, become necessary [3].

For the complicated task of digital image analysis, deep learning (DL) algorithms, a sub-type of machine learning architecture employing neural network models, are widely utilized. Deep learning methods excel in tackling complex problems where traditional machine learning approaches fall short, making them highly effective in the field of image recognition [34]. By combining deep learning algorithms with digital image analysis, pathologists can implement a computer-assisted diagnosis (CAD) tool. The tools can furnish information about disease indicators, patient survival time, and tumor size or propose personalized treatment plans [3]. Moreover, such solutions have the potential to enhance the existing pathologist's workflow by accelerating tissue analysis, minimizing misdiagnosis, and elevating the accuracy of cancer detection [38].

Despite the rapid evolution of machine learning methods and the increasing demand for their application in medical image analysis, only a limited number of AI-based applications have received approval for use in medical practice [34].

2.2 Machine learning methods

As mentioned earlier, machine learning models hold significant promise in the realm of medical image analysis. The primary function of a machine learning algorithm is to establish an optimal differentiation among multiple output classes, a feat achieved through the process of training. Currently, machine learning-based methods can be broadly categorized into two groups: traditional machine learning methods and deep learning methods. The first approach relies on the extraction of textural and morphological features to assess the provided tissue [43]. Traditional machine learning algorithms, such as linear regression, logistic regression, naive Bayes, decision trees, random forests (RF) analysis, K-nearest neighbors (KNN), and support vector machine (SVM) are commonly employed in this approach. The fundamental concept behind these algorithms is to identify patterns within existing data and subsequently train themselves to make predictions on unseen data [39]. These methods exhibit robust classification performance, particularly with small datasets. Furthermore, clinicians can comprehend

how the traditional machine learning methods arrive at decisions, as these models are more interpretable compared to other techniques.

However, these aforementioned models have certain limitations. For instance, they face challenges when tasked with analyzing large volumes of images, as the algorithms are designed to load all data into memory. Additionally, these models exhibit poorer performance when analyzing complex digital images, often learning only shallow representation of the input image [43]. In scenarios where a sufficiently large dataset is available and in-depth image details are essential, the implementation of deep learning methods becomes preferable.

2.2.1 Deep learning methods

Deep learning is a subset of machine learning that entails training artificial neural networks with multiple layers of artificial neurons to address problems deemed too complex for traditional machine learning models [7]. In this paradigm, algorithms learn from raw data to autonomously discover the details necessary for detection or classification. The core workflow of deep learning commences with simple nonlinear modules, each transforming the representation at one level into a higher, more abstract level, ultimately obtaining multiple depths of representation. When applied to digital images, deep learning models leverage pixel values as input, forming an array. In the initial layer, learned features represent the presence or absence of edges at specific orientations and locations in the image. The subsequent layer detects motifs by identifying specific arrangements of edges. As the model progresses through layers, motifs are assembled into larger combinations corresponding to parts of recognizable objects, with subsequent layers detecting objects as combinations of these parts [24].

Deep learning models operate without relying on extracted or selected features, mitigating errors stemming from faulty feature extraction [3]. There are several deep learning models, with the most common ones being convolutional neural networks (CNNs), generative adversarial networks (GANs), and recurrent neural networks (RNNs) [34].

Convolutional Neural Networks (CNNs) stand out as the most popular models of deep learning. The architecture of CNNs encompasses a substantial set of neurons, their interconnections, and associated parameters, featuring diverse convolutional and pooling layers. Convolutional layers serve as feature extractors, discerning key features from the input signal, while pooling layers focus on reducing network size while maintaining computational efficiency. In the implementation of these layers, the network is exposed to samples from various classes, such as positive and negative samples, and neurons learn to coordinate their operations to accomplish the designated task [39].

The initial step involves feeding the raw image into the input layer, designed to accommodate a three-dimensional spatial format of an image (width x height x depth), where depth signifies RGB (red, green, blue) - three color channels, or single - grey color channel. Typically, in medical image analysis, the raw image has RGB color channels. Subsequently, each convolution layer utilizes a filtering kernel of predetermined size (eg., 3x3 or 2x2) as a receptor to identify local features and basic patterns. Adjacent convolutional layers are connected along with pooling layers, which downsample data and selectively retain essential information. The final fully connected layers combine the features learned from the convolutional layers, facilitating domain-specific classification [3][7][43].

CNNs are predominantly employed for classification and segmentation tasks, with notable architec-

tures such as AlexNet, ResNet, and GoogLeNet being widely recognized in the field.

Generative Adversarial Networks (GANs) represent another deep learning paradigm, applicable in digital image analysis. Comprising two sub-models - a generator network and a discriminator network. GANs leverage the principles of game theory in their training. The generator is tasked with encoding and decoding images to produce new samples that closely resemble real ones [7]. Simultaneously, the discriminator is trained to distinguish between real and generated images, requiring exposure to extensive datasets of both types to effectively differentiate between them. The primary goal is to train the generator to produce samples that the discriminator can only classify as real images [39]. While GANs can enhance the training set of images before classification, their effectiveness heavily depends on the availability of large datasets.

Recurrent Neural Networks (RNNs) are commonly employed for analyzing sequential data. This algorithm not only processes standard inputs but also considers inputs from different time points in the past [39]. Unlike Convolutional Neural Networks (CNNs), which analyze data from a single time point, RNNs can learn from numerous discrete earlier steps. An RNN typically incorporates a Long short-term memory (LSTM) network, augmented by recurrent "forget" gates. This model learns tasks by reviewing propagated errors stored in recurrent feedback neurons on hidden layers and combines this historical information with present incoming inputs. The advantage lies in the ability to discern patterns from various data points, such as serial follow-up biopsy samples from cancer patients, and generate patient-level predictions by combining these patterns [5].

2.2.2 Current CNNs challenges

The implementation of deep learning in the field of medical image analysis holds promise and could significantly aid pathologists. However, it is important to note that despite the promise, only a limited number of deep learning-based methods have been put into real-world practice, and numerous challenges persist.

Whole slide images (WSI) in histopathology are extremely high-resolution images, often reaching gigapixel sizes, approximately 100,000 x 100,000 pixels [14]. Due to their size, they cannot be directly fed into CNNs without requiring substantial computational resources. To address this, the image is typically divided into smaller patches, preserving high resolution while reducing computational costs [3]. However, this approach introduces another challenge: pathologists must manually select the region of interest from the WSI, making the process labor-sensitive and increasing the workload. Furthermore, existing patch-based methods may not always align with the WSI-level label due to the large heterogeneity in WSI patterns [43].

Insufficient labeled images pose another challenge for training CNNs. CNNs demand a substantial quantity of well-annotated images for effective training. When patch-level images are used, some lack labels, as annotations are often applied only at the WSI level. Manual labeling by pathologists adds to the time-consuming nature of the annotation task, limiting the generalizability of CNNs trained on a limited dataset [3] [14].

These challenges contribute to a broader issue where current datasets do not sufficiently represent the diversity encountered in clinical practice. This discrepancy hinders the adaptability of trained networks to real-life scenarios, as the model's performance may falter when tested with data from

different sources than the training data [41]. There are two main solutions, that could help to create more versatile CNNs. The first solution, incorporating diverse images into the training set from various staining batches, scanners, and medical centers. Additionally, implement data augmentation techniques by randomly applying saturation, hue, brightness, rotation, adding noise [3]. The second solution, normalizing images to a common standard to remove variability. The latter, while potentially increasing computational costs, can enhance the algorithm's performance on a small set of training images [41].

Despite these potential solutions, the challenges in CNNs persist, affecting model quality, versatility, applicability in clinical practice, and integration into computer-assisted diagnosis tools. Furthermore, understanding the underlying functioning of CNNs remains elusive, emphasizing the need for explainable and interpretable methods in deep learning models.

2.3 Black box issue in CNNs

Decisions made by deep learning models are often referred to as black boxes. Since deep learning models are trained, not explicitly programmed, they do not provide an exact explanation or interpretation of how the function made decisions during the process [41]. The lack of explanation and interpretation in models, raises concerns about CNN transparency, limiting acceptability by the community. Additionally, there is a risk of creating a decision system that is not understandable for the community (e.g., pathologists using deep learning-based models for CAD), impacting safety and industrial liability [11].

The presence of black boxes in deep learning architecture can be dangerous, and create discrimination and trust issues. Several pitfalls in the models create room for the occurrence of black-boxes. For instance, a model can use variables (confounders) that should not be utilized as predictors in the model. Sometimes, it is difficult to detect if proper variables are used as predictors, especially when a poor choice for the evaluation metric is applied [1]. Deep learning-based models are data-driven and learn from collected information. Thus, obtained data may contain human biases, prejudices, or mistakes, which, when inserted into a model, can affect its ability to classify images correctly [11]. Additionally, it is difficult to have real-time error-checking. In conclusion, understanding the interactions among the vast number of neurons in the model without the assistance of dedicated computational tools is very challenging for a pathologist.

2.3.1 Types of explainable CNNs methods

Explainability is associated with the internal logic and mechanics within a model's architecture. Explainable models (XAI) provide an understanding of the internal procedures that occur during the training or testing phase [26]. There is a numerous amount of explainable machine learning and deep learning methods and they can be divided into global, local, ante-hoc, post-hoc and surrogate methods. The global method explains the model's behavior on the entire dataset, while the local method explains the prediction of a particular instance. Ante-hoc method explains the pre-training stage, while post-hoc method explains decisions on an already pre-trained model. The surrogate method suggests a simple model to mimic the prediction of the black-box model [16]. For CNNs, post-hoc local explanations are mostly adopted. The complex internal structure of CNN built as a sequence of convolutional and

pooling layers, poses a challenge to achieving explainability. The LIME framework is the most adopted approach for seeking explanations in CNNs. It combines visualization and feature relevance methods. In order to explain the process in CNN, LIME uses a simple linear model to produce an explanation of the deep learning model for the given input. However, it is greatly limited by huge computational costs [30].

2.3.2 Types of interpretable CNNs methods

Interpretable convolutional neural networks (IAI) aim to assist humans in understanding the link between features extracted by the model and its predictions, despite the algorithm's complexity [30]. The main idea in interpretation is not to explain mechanisms and internal logic inside the model, but to identify cause-and-effect relationships with the model's inputs and outputs [26]. There are several groups of interpretability methods, including the "white box" method, global method, and local methods.

The "white box" method adopts a transparency approach, providing internal information and the structure of a model. By offering internal information, such as gradient flow through a layer of the network, the model enhances the output and identifies regions that were important for prediction [30].

Global and local interpretation models function similarly to explainable global and local models. The global model offers information about patterns that were most crucial for the model's prediction, helping to demonstrate that learned patterns from the population align with existing domain knowledge. These methods can assist in detecting biases in training data. The local model focuses on explaining why the model makes a specific prediction for a given input [30].

Moving forward, this paper will concentrate on interpretable CNN methods and their application in the medical imaging field.

2.4 Interpretable CNNs methods

Interpretability methods can be integrated during the design process of the network or post-hoc interpretability methods can be added after the network is trained [26]. Current interpretability methods can be grouped into three main interpretation approaches: visualization, interpretation through influence functions, and semantic interpretations.

2.4.1 Interpretation through visual methods

Interpretation using visualization methods is one of the most powerful and extensively investigated fields. The primary purpose is to visualize the importance of input features to the model's output. Visual methods can elucidate how the model's prediction changes over the population (global) or individual (local) contributions of a feature. During the interpretation part, researchers expect, that major features will influence the model's predictions when their values are altered [30]. Thus, deep learning models can be explained by providing heat-map-based arguments and highlighting crucial regions of input image [32]. Visualization methods can be divided into three groups: gradient-based methods, perturbation-based methods, layerwise relevance propagation methods, and others.

Gradient-based methods, such as Class Activation Maps (CAMs), generate saliency maps by computing the gradient from the model's output to the input image space. Class Activation Maps indicate the distinguishable areas of an image used to identify the category during CNNs [26]. However, CAM implementation has limitations, requiring a specific structure in the final layers, leading to the need for architecture adaptation and re-training. Additionally, it provides interpretation only about the final convolutional layer of the model [26]. To address these issues, Gradient-Class Activation Maps (Grad-CAM) was introduced, allowing visual interpretation without re-training or changing the network architecture. Grad-CAM explains activation in any layer of the network, utilizing class-specific gradient information flow into the final convolutional layer of a network [32]. Nevertheless, Grad-CAM has limitations, such as the inability to localize multiple occurrences of an object and accurately describe class-regions coverage in an image [26]. Another gradient-based tool is the Integrated Gradient (IG) method, which mitigates the saturation problem of gradients. This method selects a baseline with a near-zero score (ex. complete black image) and, during calculation, shows how much of the input contributed to the final prediction [32].

Visual methods can applied using a perturbation-based approach, which investigates the effect on the model's prediction when systematically perturbing the image. This method provides an attribution map to observe the effect on the output [32]. By performing image alterations, it is expected to identify which part of the input is important. If a specific element is removed from the image and the image is assigned to another class, this element can be marked as a crucial object influencing classification. Local interpretable model-agnostic explanations (LIME) is a perturbation-based approach used for this purpose [18]. LIME creates a set of perturbed instances by dividing the input image into interpretable components, runs each perturbed instance through the model, receives probabilities, and provides interpretable components with the highest positive weights as an explanation [2].

For interpreting highly complex deep neural networks, the Layer-wise Relevance Propagation (LRP) technique can be implemented. The model uses the decomposition of the nonlinear classifiers approach and propagates network predictions backward [26]. LRP generates a heatmap by evaluating the relevance score of each neuron, enabling quantitative analysis of the individual value contribution of the input to the output [10]. Hägele et al (2020) proposed the implementation of the LRP method for a classification model, providing cell-level resolution and pixel-wise heatmaps. Heatmaps generated by this methodology were compared to labels provided by pathologists using the ROC (receiver operating characteristic) curve, demonstrating a high overlap with the provided labels. Additionally, the proposed methodology facilitated the discovery of class-correlated, dataset, and sample biases, addressing issues that often hinder the generalization abilities of the models [13].

2.4.2 Interpretation by using influence functions

Interpretation can be achieved not only by visualizing the input features but also by determining the most influential examples in the training set. Influence functions serve as a tool to assess which training images are most influential for a model. This function involves removing one image from the training set (leave-one-out approach), retraining the network, and then evaluating the change in model performance [30]. Fast Influence Functions (FastIF) is a tool belonging to the influence functions family. The model performs a k-nearest Neighbor (kNN) search to identify a small set of influence-worthy data points,

avoiding the need to examine the entire dataset for influential data points. This approach is typically chosen to expedite the search for influential data points. Additionally, the influence function can serve as a quality control mechanism for the training set, enabling the detection of potentially mislabeled images by utilizing the leave-one-out approach [12]. However, influence functions lack robustness and may not yield reliable results for architectures that are very deep and wide, such as ResNet-50 [4].

2.4.3 Interpretation through semantic methods

Instead of relying solely on visual context for interpretation, it is possible to enhance interpretability through a semantic approach. Methods utilizing semantic interpretation provide interpretations that describe model predictions [30]. Textual interpretation methods can be rule-based or template-based. Rule-based methods utilize structured diagnostic reports in conjunction with images for training, while template-based methods generate interpretation from auxiliary information. MDnet network is a rule-based method that provides visual and textual interpretation of medical images by generating diagnostic reports along with corresponding attention maps [27]. Other research has proposed a method in which the model predicts diagnosis labels and provides rich, interpretable predictions that are understandable to pathologists. The model generates a natural language description of microscopic findings and simultaneously visualizes the attention map. The model was trained on images with annotations of diagnostic reports provided by pathologists. The visual attention map was learned spontaneously by observing visual-semantic correspondences from image pixels to annotated description words of the data. The implemented method can serve as a valuable second opinion in the medical diagnostic field [44].

3 Material and Methods

In this section, the chosen dataset, along with the augmentation process, and CNN-based classification were introduced. Subsequently, a small group of interpretation methods was discussed, several evaluation metrics for network interpretability were proposed, and ground truth image extraction through segmentation model was presented.

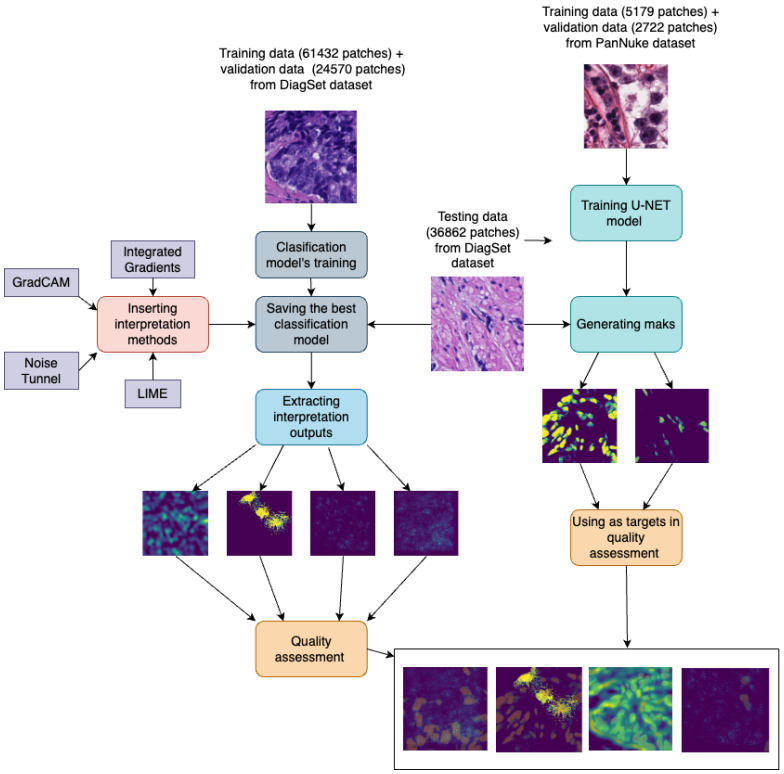

Figure 2: Workflow of experiment

The experiment begins with data collection and preparation from the DiagSet dataset. The dataset is then divided into three parts: the training set, validation set, and testing set. The training and validation data are utilized in the classification model, where the model learns to correctly classify given images. Subsequently, the model with the smallest loss is saved. Four interpretation methods and testing data are then applied to the saved model without altering its architecture, generating outputs. Quality assessments for those outputs are performed.

To conduct the quality evaluation for the generated interpretations, ground truth outputs are necessary. In this research, ground truth masks are obtained from segmentation model. A U-Net model is trained on images from the PanNuke dataset, and for the testing part, the same DiagSet testing images are used to generate two ground truth masks. The outputs received from the interpretation part are then compared with the generated masks. The entire process is visualized in Figure 2. The classification, segmentation, and visualization processes were implemented in Python 3.10, using the PyTorch library.

3.1 Classification dataset

For this research, the DiagSet dataset was utilized [22]. The proposed dataset consists of 2.6 million tissue patches extracted from 430 fully annotated scans, 4675 scans with assigned binary diagnoses, and 46 scans with diagnoses given independently by a group of histopathologists. The patches are derived from patients diagnosed with prostate cancer. Due to computational limits, patches were sampled only from DiagSet-A, a subset of the complete DiagSet dataset. Throughout this experiment, patches with a size of 256 x 256, 20x levels of magnification, and 3 colour channels were used. Each patch was assigned a single label of 7 possible classes: healthy tissue (N), acquisition artifact (A), or of 1-5 Gleason grades (R1-R5), representing tumor aggression/malignancy. Examples of the utilized patches is provided in Figure 3.

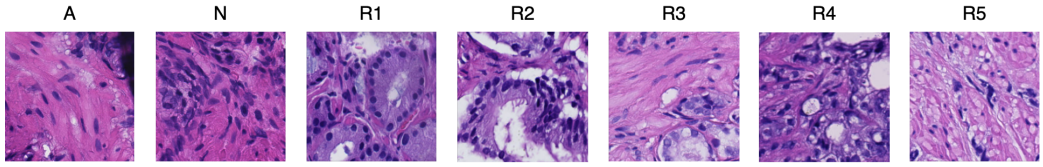

Figure 3: The examples of original tissues from DiagSet dataset.

In total, 395188 patches were collected and the distribution of patches for each class is presented in Table 1.

Class	Original number of patches	Number of patches after the augmentation	Number of patches for training	Number of patches for validation	Number of patches for testing
A	45706	45706	8776	3510	5266
N	200135	200135	8776	3510	5266
R1	4388	**17552**	**8776**	**3510**	**5266**
R2	4709	18836	8776	3510	5266
R3	36172	36172	8776	3510	5266
R4	87171	87171	8776	3510	5266
R5	16907	67628	8776	3510	5266

Table 1: Summary of DiagSet dataset. Classes that underwent data augmentation are highlighted in the table. Bolded numbers represent the smallest class set and, based on it, an equivalent number of images were selected from the other classes.

3.1.1 Data augmentation

Data augmentation is a valuable technique to influence the training of deep neural networks by constructing synthetic data from available datasets [35]. Various processes, such as adding noise, cropping, rotating, and flipping are employed for data augmentation. The primary objective of utilizing data augmentation is to increase the dataset's size and introduce more diverse objects during the model's training. In this research, the main purpose of data augmentation is to augment the number of patches for classes with the smallest amount of images.

As presented in Table 1, the dataset is not balanced, with some classes (A, N, R3, R4) having more patches than others. To ensure a sufficiently large number of patches from classes R1, R2, and R5,

augmentation techniques were applied by rotating patches 90, 180, and 270 degrees. The final number of patches in each class is detailed in Table 1.

To achieve a balanced dataset, an equal number of patches from each class were utilized for the training, validation, and testing phases. The specific amount was determined based on the smallest class, R1. Consequently, 8776 patches were selected from each class for training, 3510 for validation, and 5266 for testing. All details are highlighted and provided in Table 1.

3.2 Segmentation dataset

The segmentation process requires a different dataset that can provide images along with annotated masks. For this research, the segmentation model was trained using the PanNuke dataset, a semi-automated, quality-controlled dataset with class labels for five main types of nuclei across various cancerous tissue types [8]. Specifically for this experiment, the U-Net network was trained only on two types of nuclei, producing two masks for each image: tumor mask and non-tumor mask.

The tumor mask captures the epithelial class, encompassing all tumorous cells, and is marked as 'Neoplastic' in the PanNuke dataset. The non-tumor mask captures connective/soft tissue cells and is marked as 'non-neoplastic'. Examples of images and masks from the PanNuke dataset are presented in Figure 4.

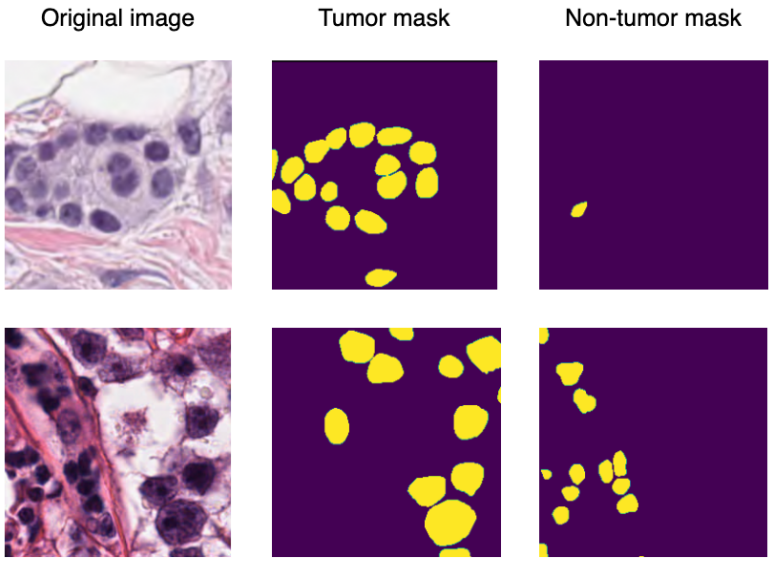

Figure 4: The examples of original tissues along with ground truth masks from the PanNuke dataset.

The dataset is divided into three folds, each containing different images and annotations. The first and second folds (5179 patches) were used for training the U-Net network, while the third fold (2722) was utilized in the network's validation part. All these details are summarized in Table 2.

Number of patches for training	Number of patches for testing
5179	2722

Table 2: Summary of Pannuke dataset

Subsequently, the trained U-Net network was saved and during the testing phase, patches from the

DiagSet dataset were fed into the saved model, generating two ground truth masks (patches) for each inserted patch.

3.3 Classification model

A CNN-based model with cross-entropy loss and Adam optimizer, utilizing a learning rate of 0.001, is implemented in this research. The overall framework is illustrated in Figure 5, and the cross-entropy loss function is defined in Equation 1,

$$\text{CrossEntropyLoss}(x, y) = -\frac{1}{N}\sum_{i=1}^{N}\log\left(\frac{e^{x_{i,y_i}}}{\sum_{j=1}^{C} e^{x_{i,j}}}\right) \tag{1}$$

where x_{ij} is the j-th element of the logits for the i-th example in the batch, y_i is the ground truth class index for the i-th example, N is the batch size, C is the number of classes and e denotes exponential function.

The model is composed of three convolutional layers, each equipped with ReLU activation and a max-pooling layer. The first convolutional layer employs a 3 x 3 filter and a 3 x 3 max-pooling operation with a 2 x 2 stride. The second and third layers use a 5 x 5 filter, maintaining the same max-pooling and stride parameters as the first layer. Subsequently, a flattening operation is applied, followed by seven fully connected linear layers. ReLU activation functions are incorporated after each linear layer. The final layer produces a vector of logits with seven values, representing confidence scores for the 7 classes in the classification task (Figure 5).

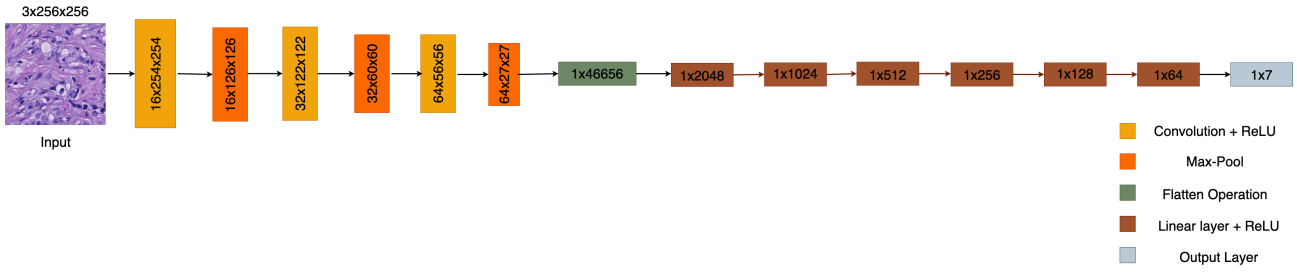

Figure 5: Representation of CNN-based model architecture

3.4 Interpretation methods

For this experiment four interpretation methods were investigated: Integrated Gradients, Integrated Gradients with NoiseTunnel, LIME, and GradCAM.

3.4.1 Integrated Gradients

Integrated Gradients (IG) is the gradient-based method, that employs a black image as baseline [40]. The method calculates attribution scores for each pixel of input by integrating gradients of the model's output based on input features along with a straight path from baseline x' to the actual input

x. Integrated gradient along the i^{th} pixel (feature) for input x and baseline x' is obtained by cumulating these gradients together (eq. 2). $\frac{\partial F(\mathbf{x})}{\partial x_i}$ represents the gradient of $F(\mathbf{x})$ along the i^{th} feature and α is the scaling coefficient.

$$\text{IntegratedGradients} = (x_i - x_i') \times \int_{\alpha=0}^{1} \frac{\partial F(\mathbf{x}' + \alpha \times (\mathbf{x} - \mathbf{x}'))}{\partial x_i} \, d\alpha \tag{2}$$

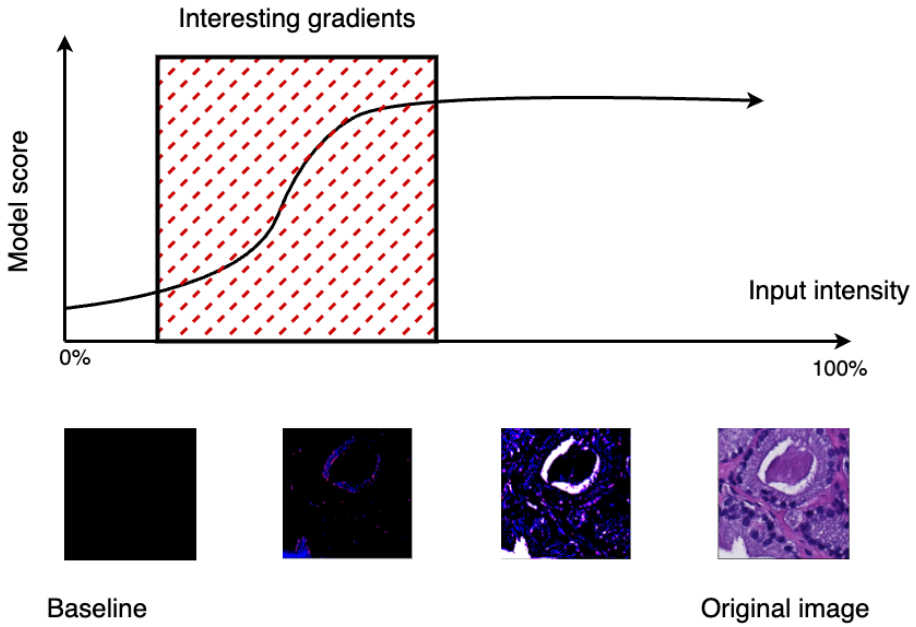

Figure 6: Schema of Integrated Gradients workflow. By changing input intensities, the integrated gradient method estimates the attribution of input towards predicting the output by averaging contributions.

The final integrated gradients represent the accumulated importance of each pixel in the input image along the integration path. The process of receiving visualization for gradients of scaled inputs is shown in Figure 6.

3.4.2 NoiseTunnel

NoiseTunnel (NT) is a smoothing technique, implemented along with Integrated Gradients. It reduces variability and sensitivity in the attributions, providing more stable and reliable attribution scores [19]. Computed attributions from the Integrated Gradients method are refined with a Gaussian kernel. Random samples in a neighborhood of an input x with Gaussian noise are taken, and an average of noisy sampled attributions is calculated [37]. The mathematical equation of this solution is presented in Equation. 3,

$$\text{NoiseTunnel}(\hat{M}_c) = \frac{1}{n} \sum_{j=1}^{n} M_c(\mathbf{x} + \mathcal{N}(0, \sigma^2)) \tag{3}$$

where n is the number of samples and $\mathcal{N}(0, \sigma^2)$ represents Gaussian noise with standard deviation σ, $\hat{M}_c$ is estimated smoothed gradient, and M_c is the unsmoothed gradient.

3.4.3 LIME

Local Interpretable Model-agnostic Explanations (LIME) delivers a local interpretation for the instance of interest by fitting a set of perturbed samples near the target sample using a linear model or decision trees as potentially interpretable models (Figure 7).

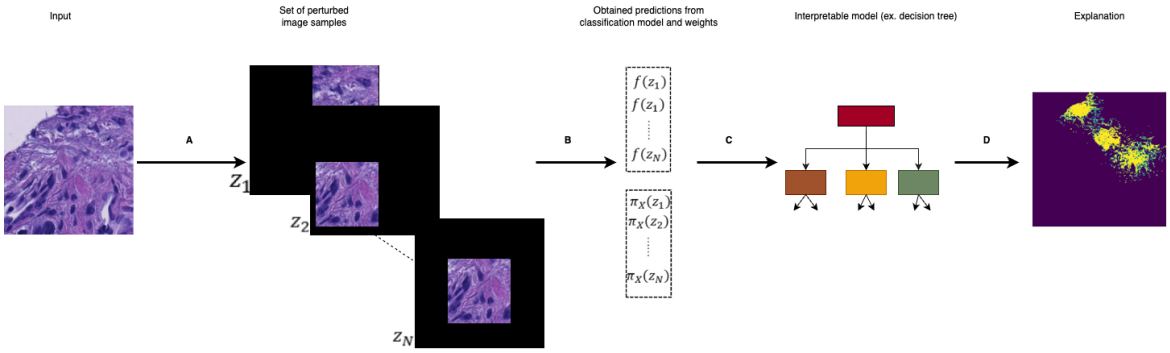

Figure 7: The flowchart of LIME method. A - perturbed images; B - insert perturbed images into deep learning model and obtain predictions for each perturbed instance; C - train the interpretable model with perturbed instances and their corresponding model predictions; D - visualize important features.

The LIME method for the target sample x provides interpretation from the computed explanation $\xi(z)$ in Equation 4 [25].

$$\xi(z') = \arg\min_{g \in G} L(f, g, \pi_x) + \Omega(g) \tag{4}$$

$$L(f, g, \pi_x) = \sum_{z, z' \in Z} \pi_x(z)(f(z) - g(z'))^2 \tag{5}$$

$$\pi_x(z) = e^{-\frac{D(x,z)^2}{\sigma^2}} \tag{6}$$

The model that requires interpretation is presented as f, g is an interpretable model, and $g \in G$, where G is a class of interpretable models. $L(f, g, \pi_x)$ is a locality-aware loss function that measures the discrepancy between f and g with weights given by π_x, and $\Omega(g)$, which is used for regularization by giving a penalty for the complexity of model g. The mathematical expression of $L(f, g, \pi_x)$ is shown in Equation 5, where the difference between the predictions of the models is measured. Perturbed instances are z, z' with dimensions corresponding to the input layer of f and g, respectively. The weight π_x, also known as similarity kernel, describes the locality of the interpreted sample x. The smaller the distance to the original sample x, the greater the weight. The similarity kernel is expressed

in Equation 6, where it is defined on distance measure function D (L2 distance for images) with width σ.

3.4.4 GradCAM

Gradient-weighted Class Activation Mapping (GradCAM) is a method that inspects the last convolutional layer for class-specific information in the image by using gradient information flowing into the last convolutional layer and assigning importance values to each neuron for a particular decision of interest (Figure 8).

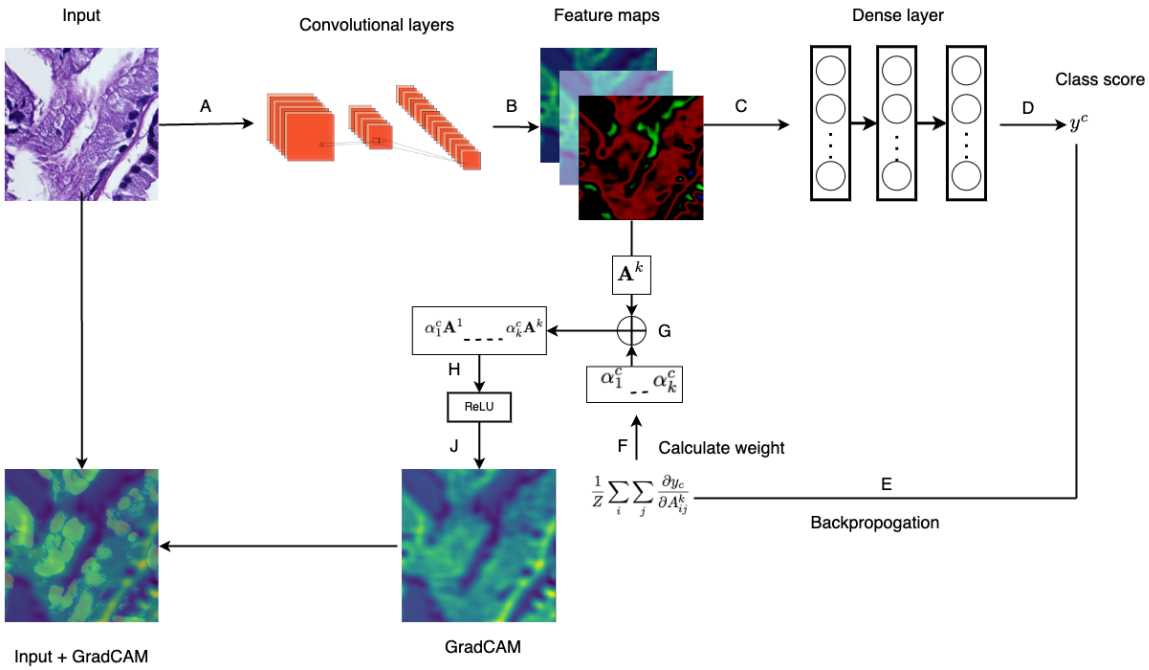

Figure 8: The flowchart of Grad-CAM method. A - load the input into pre-trained network; B - obtain feature maps of the last convolutional layer; C - perform forward pass; D - capture the output logits from the final layer; E - compute the gradients of the output logits; F - apply global average pooling; G - multiply the feature maps of the last convolutional layer by the computed weights; H - apply ReLU activation; J - visualize the heatmap that highlights regions contributing to the target class

To obtain the class-discriminating localization map L^c_{GradCAM} in GradCAM, the gradient score of y^c (before softmax) for class c with respect to feature map activations A^k of the unit k in a convolutional layer is computed. The obtained gradients are global-average-pooled over the width i and height j dimensions and the neuron importance weight α_k^c is computed (Equation 7).

$$\alpha_k = \frac{1}{Z}\sum_i\sum_j\frac{\partial y_c}{\partial A_{ij}^k} \tag{7}$$

$$L^c_{\text{Grad-CAM}} = \text{ReLU}\left(\sum_k \alpha_k^c \mathbf{A}^k\right) \tag{8}$$

Then, a weighted combination of forward activation maps and the application of ReLU to the linear combination of maps for extracting features that have a positive contribution to the class of interest, give localization map (GradCAM heatmap) in Equation 8 [33][25].

3.5 Intepretation quality assessment methods

The quality of interpretation methods is assessed using six metrics: Intersection-over-Union(IoU), precision, recall, F1-score, Spearman Rank correlation, and pixel accuracy.

3.5.1 Iou

Intersection-over-Union (IoU) - also known as the Jaccard Index is utilized for comparing similarity (overlap) between two arbitrary shapes.

$$\text{IoU} = \frac{\text{Area of Intersection}}{\text{Area of Union}} = \frac{|A \cap B|}{|A \cup B|} = \frac{|\{y = c\} \cap \{\tilde{y} = c\}|}{|\{y = c\} \cup \{\tilde{y} = c\}|} \tag{9}$$

In Equation 9, B represents a predicted class map and A represents a ground-truth class label map. For pixel-wise prediction, the evaluation is conducted on different variables. IoU for a label $c \in C$, where C is a finite set of classes to predict, gauges the overlap between the set of ground-truth pixels y and the set of predicted pixels $\tilde{y}$ [17]. A high IoU between predicted and ground truth output indicates accurate interpretation by the model.

3.5.2 Precision, recall and F1-score

Precisions, recalls, and F1-score metrics utilize True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN) notations for actual pixels in the image and predicted pixels in the generated image. TP represents the number of pixels correctly classified as belonging to the object in both the ground truth and the predicted output. TN is the count of pixels correctly classified as not belonging to the object in both the ground truth and the predicted output. Both TN and TP are accurate estimations.

FP is the number of pixels mistakenly classified as belonging to the predicted class, referred to as Type I error. FN is the count of pixels incorrectly classified as not belonging to the predicted class, known as Type II error. Each notation is summarized in Table 3. All three metrics are available in the sklearn module in the Python3.10 environment and are used for this research.

		Predicted pixel	
		Positive	Negative
Actual pixel	Positive	TP	FN
	Negative	FP	TN

Table 3: Confusion matrix of notations

Precision is the ratio of correctly predicted pixels belonging to the object to the total number of pixels predicted as belonging to the object. High precision indicates that the method accurately

identifies regions without introducing many false positives into the output (Eq. 10).

$$\text{Precision} = \frac{TP}{TP + FP} \tag{10}$$

Recall (Sensitivity) gives the ratio of correctly predicted pixels belonging to the object to the total number of pixels belonging to the object in the ground truth mask. High recall indicates that the model has good performance in capturing most pixels correctly (Eq. 11).

$$\text{Recall} = \frac{TP}{TP + FN} \tag{11}$$

$$\text{F1-score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{12}$$

F1-score is the weighted average of Precision and Recall metrics and evenly reflects the aforementioned evaluation results (Eq. 12) [6]. It is often employed for optimizing a model with a balance between precision and recall.

3.5.3 Spearman's rank correlation

Spearman's rank correlation estimates the correlation between the generated output and the ground truth mask, quantifying the similarity between them. It is a non-parametric measure for variables that may not have a linear relationship.

$$\rho = 1 - \frac{6 \sum_{i=1}^{n} d_i^2}{n(n^2 - 1)} \tag{13}$$

Firstly, each pixel in the ground truth mask (X) and in the generated output (Y) receives ranks based on their intensity values. Then, the difference (d_i) between the ranks of corresponding pixels in the two masks is computed: $d_i = rank_x - rank_y$ and the sum of the squared differences ($\sum_{i=1}^{n} d_i^2$) is obtained. Spearman's rank correlation (ρ) is then calculated using Equation 13, where n is the number of pixels. The metric shows results in the range from -1 to 1, where a perfect monotone relation between ground truth mask and generated output is described by 1 (positive) and -1 (negative), while 0 indicates no monotonic relationship [20]. Calculations are done using Python3.10 environment and utilizing SciPy module.

3.5.4 Pixel accuracy

Pixel accuracy computes the ratio between the number of matching pixels between the ground truth mask and generated output, providing a global measure of overall accuracy.

$$\text{Pixel Accuracy} = \frac{\text{Correct Pixels}}{\text{Total Pixels}} \tag{14}$$

Metric's logic is described in Equation 14, where correct pixels are determined by counting the number of matching pixels in ground truth mask X and received output Y: $\sum_{i=1}^{n} \mathbb{I}(X_i = Y_i)$, where n is the total number of pixels, $\mathbb{I}$ - indicator function [9].

3.6 Segmentation and ground truth mask extraction

As mentioned earlier, masks obtained from the U-Net segmentation model serve as ground truth masks for this research. The U-Net model comprises two main parts: encoder and decoder.

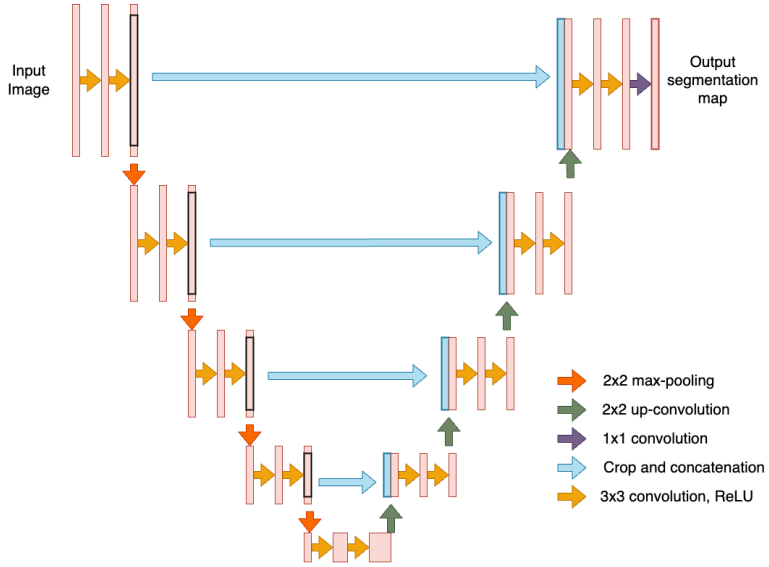

Figure 9: U-Net architecture

The encoder consists of typical CNN architecture, where two successive 3 x 3 convolutions followed by ReLU activation units and 2 x 2 max-pooling layer are repeated several times. This part of the network extracts classification information. Once the encoder path is completed, the decoder initiates the synthesis path. At each stage, the feature map is upsampled using 2 x 2 up-convolution, and then the feature map from the corresponding layer in the encoder part is cropped and concatenated onto the upsampled feature map. In the final stage, the decoder completes with two successive 3 x 3 convolutions with ReLU, followed by a final 1 x 1 convolution to reduce the feature map to the required number of channels, resulting in the production of the segmentation image [36]. A visual representation of the network is provided in Figure 9.

The research implements the U-Net model with BCE-Dice loss and Adam optimizer, utilizing a learning rate of 0.0001. The BCE-Dice loss function is the sum of Binary Cross-Entropy (BCE) and the Dice loss. The BCE loss measures the difference between true binary labels and the predicted probabilities. The Dice loss is responsible for measuring the overlap between the true mask and the predicted segmentation mask. The BCE-Dice loss function is defined below,

$$\text{BCE-Dice Loss} = \text{BCE Loss} + \text{Dice Loss} \tag{15}$$

The Binary Cross-Entropy (BCE) Loss is given by:

$$\text{BCE Loss} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(p_i) + (1 - y_i)\log(1 - p_i)] \tag{16}$$

And the Dice Loss is given by:

$$\text{Dice Loss} = 1 - \frac{2\sum_{i=1}^{N}(p_i \cdot y_i)}{\sum_{i=1}^{N} p_i^2 + \sum_{i=1}^{N} y_i^2} \tag{17}$$

Here, N is the number of pixels or elements in the binary masks, p_i is the predicted probability (the probability of the pixel being part of the object), and y_i is the ground truth label (either 0 or 1) [21].

4 Results

In this section, the performance of the proposed interpretation methods was evaluated. Firstly, the classification processes underwent evaluation. Secondly, the segmentation process underwent evaluation. Thirdly, the obtained interpretations underwent visual inspection. Finally, the outputs of four different interpretation methods were assessed using six different metrics.

4.1 Classification results

As aforementioned, a CNN-based model with cross-entropy loss and Adam optimizer was employed in this research. The model ran for 50 epochs with a batch size of 64, however after 22 epochs model stopped improving. Consequently, the training concluded at the 22nd epoch. The model's training loss score decreased from 0.89 to 0.05, consistently decreasing throughout the training period. However, after 19 epochs, the loss scores changed only by 0.001 points. While the validation loss score initially decreased in the first 4 epochs, it later increased by 0.04 points and remained unchanged thereafter (Figure 10).

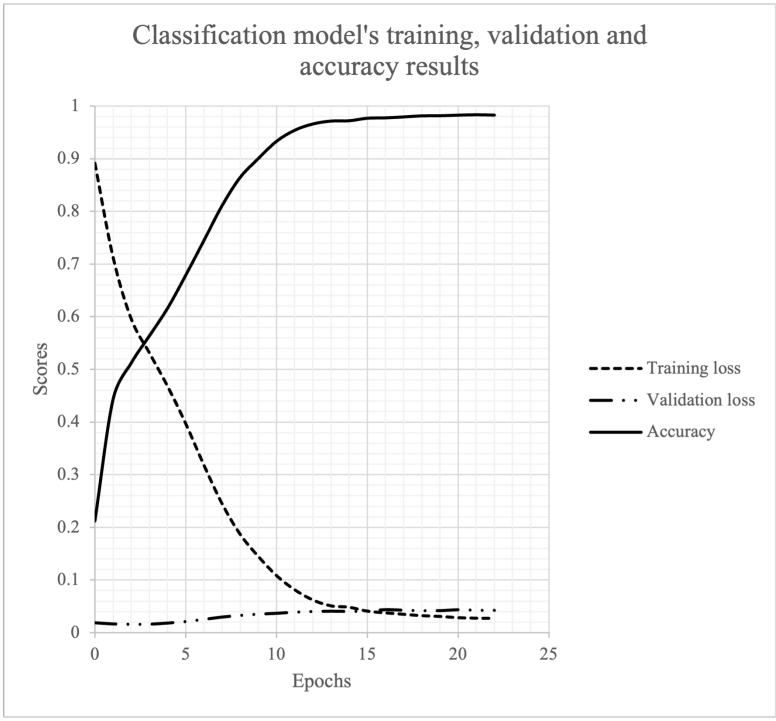

Figure 10: CNN-based model's training and validation results

During the training period, the model's classification accuracy increased for 18 epochs. Afterward, accuracy results remained consistent, reaching approximately 0.98.

N	A	R1	R2	R3	R4	R5	Average
66.27	82.4	60.98	47.17	52.45	54.99	72.46	62.39

Table 4: Classification accuracy per each class and average accuracy (%)

Once training was complete, the trained model was saved and utilized in the testing phase. During

the testing, new, unseen images from the DiagSet dataset were fed into the trained model, and the model's accuracy was evaluated (Table 4). The average model accuracy was 62.39%. The model exhibited the highest accuracy in classifying A class images (tissues with artifacts) at 82.4%, while the second-highest accuracy was for R5 class images (3rd stage of cancer) at 72.46% accuracy. However, the model did not perform well in classifying R2 class images (1st stage of cancer), achieving only 47.17% accuracy.

4.2 Segmentation results

U-Net model was implemented into the production of ground truth images. The BCE-Dice loss and Adam optimizer were employed as well. The model ran for 124 epochs, batch size of 1. All segmentation training and validation metrics are visualized in Figure 11.

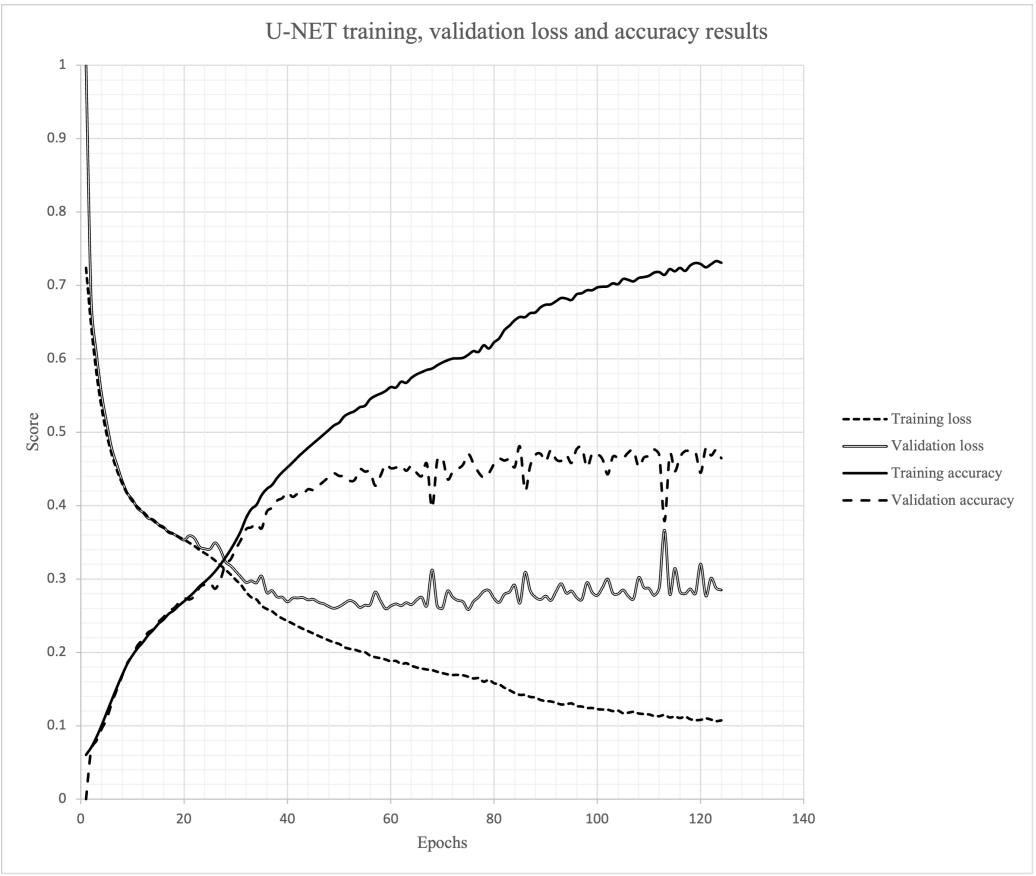

Figure 11: U-Net model's training and validation results

The model's training loss consistently decreased from 0.72 to 0.1 throughout the training process, while training accuracy steadily increased from 0.06 to 0.73. In contrast, the validation loss initially decreased in the first 50 epochs but later showed minor fluctuations without significant improvement. A similar manner was observed in validation accuracy, which gradually increased to approximately 0.46 in the first 50 epochs, and with minor fluctuations improved more slowly, reaching around 0.48.

4.3 Visual results of interpretations

The trained CNN-based model was checked with four different interpretation methods: Integrated Gradients, Integrated Gradients with Noise Tunnel, LIME, and GradCAM. Each interpretation method generated saliency maps. Firstly, each interpretation method's output characteristics will be described.

4.3.1 Integrated Gradients

Integrated Gradients generate a map of pixels, which are crucial for making decisions.

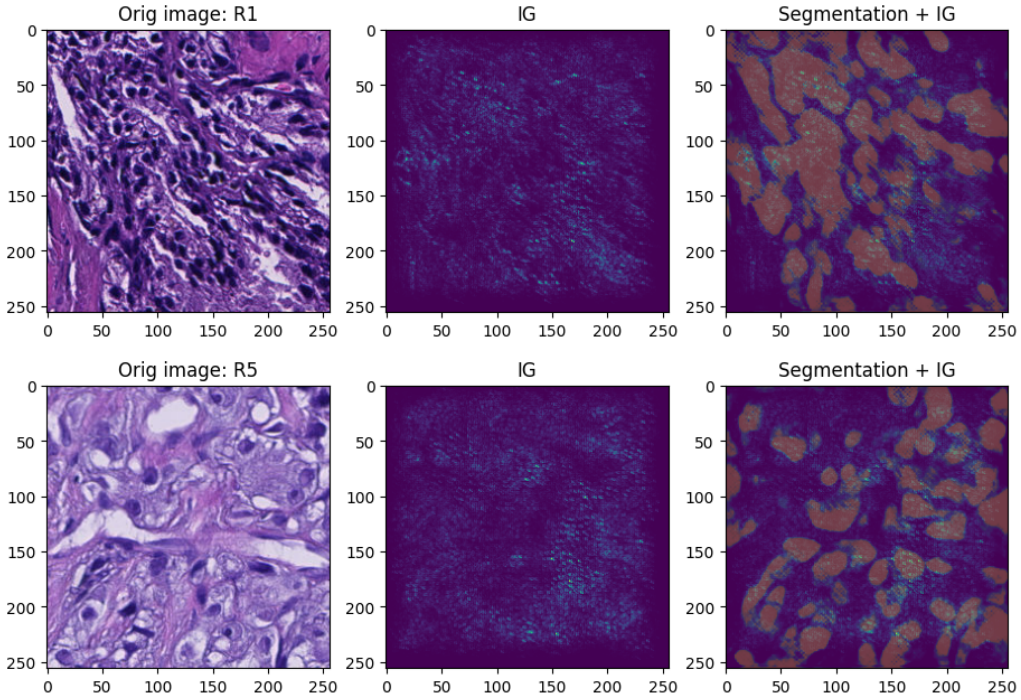

Figure 12: Example of two the original images from DiagSet dataset and outputs from Integrated Gradients (IG) methods, along with interpretation maps stacked on segmentation output for same images (selected best visual outputs)

Figure 12 illustrates an example of the Integrated Gradients (IG) output, along with the overlay of the segmentation mask. The resulting map displays bright, small dots, which represent important pixels.

4.3.2 NoiseTunnel

To reduce noise and sensitivity, an additional method, NoiseTunnel (NT) was implemented alongside Integrated Gradients. This method utilizes the output generated from Integrated Gradients and applies an additional NoiseTunnel function. The resulting outputs tend to exhibit less noise than those from Integrated Gradients.

Figure 13 displays the saliency map after applying the NoiseTunnel. As mentioned earlier, this method aims to reduce sensitivity and noise, emphasizing only the important pixels for decision-making. Therefore, generated maps have fewer bright pixels (dots).

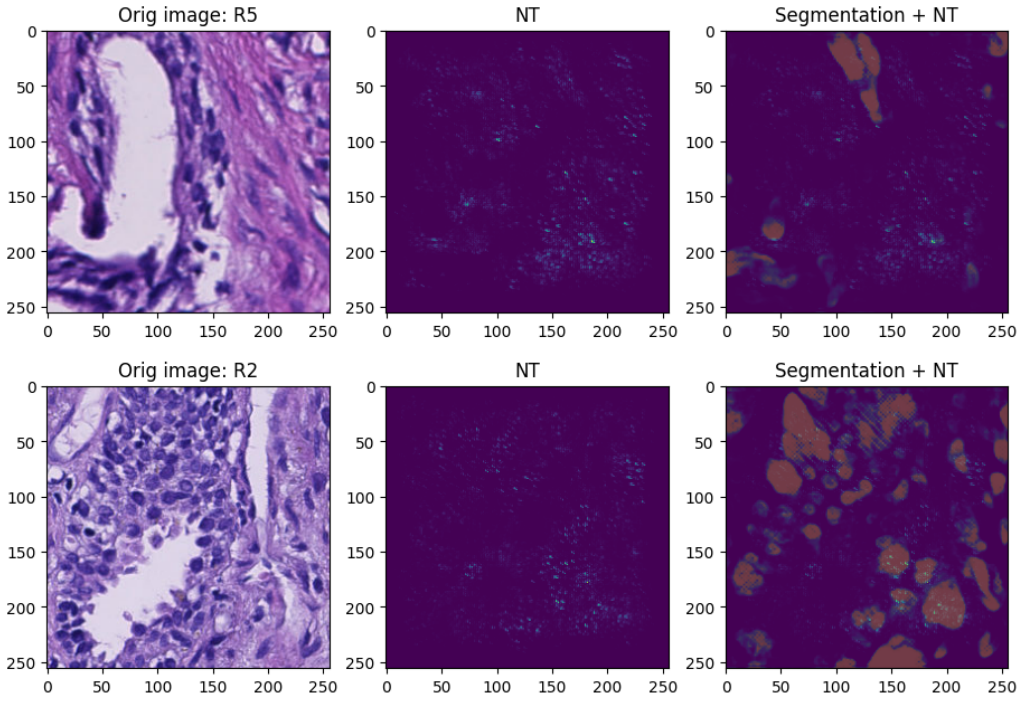

Figure 13: A pair of outputs received from NoiseTunnel (NT) method for different patches and same outputs stacked on segmentation masks (selected best visual outputs)

4.3.3 LIME

The third type of map was obtained from the LIME method. Instead of highlighting only important pixels for decision-making, this method generates areas (features) that are crucial for the model's decision. The LIME method allows the selection of the number of features to be presented in the interpretation. In this case, the method presented a maximum of three features in the generated map. This parameter was chosen due to the computational limitations and was deemed the most optimal.

Figure 14 showcases examples of generated saliency maps from the LIME method. As mentioned earlier, the method tends to highlight areas important for decision-making rather than focusing solely on important pixels.

4.3.4 GradCAM

The last type of interpretation map was generated using the GradCAM (GC) method. This method produces heat maps highlighting areas important for the network. GradCAM often generates the brightest and most detailed outputs. Figure 15 displays several examples of heat maps for patches from the testing dataset.

GradCAM focuses on features (areas) with positive attribution for decision-making. It is worth mentioning that this method does not have parameters specifying how many features should be provided during the interpretation. As a result, all relevant areas are displayed in the output.

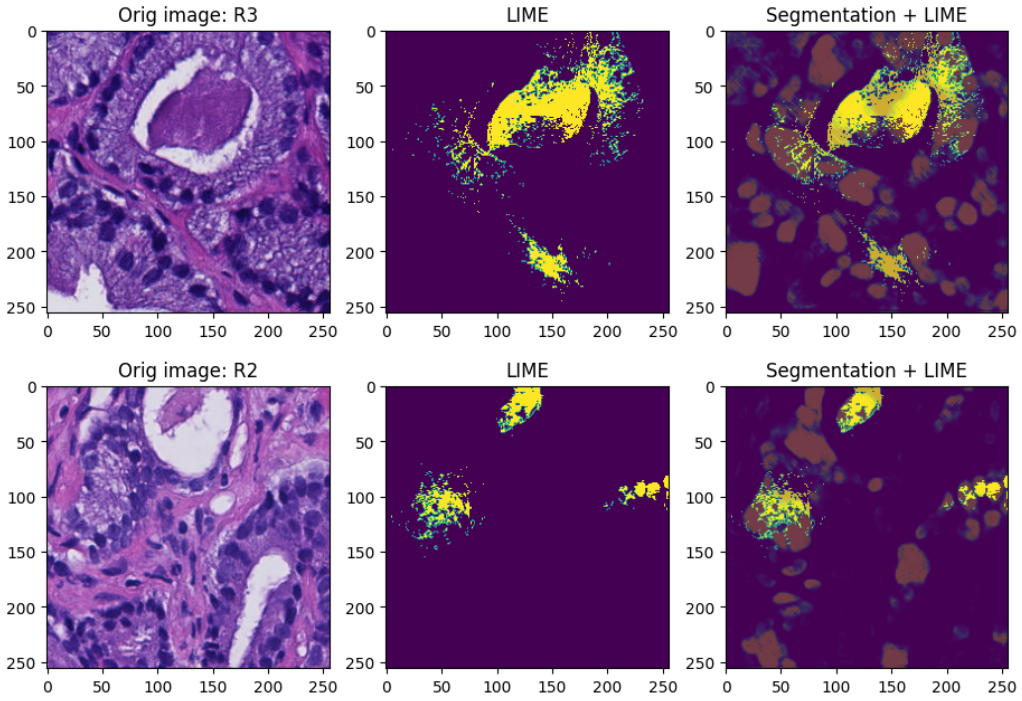

Figure 14: Several examples of saliency maps gathered from the LIME method along with the same outputs overlayed on segmentation masks (selected best visual outputs)

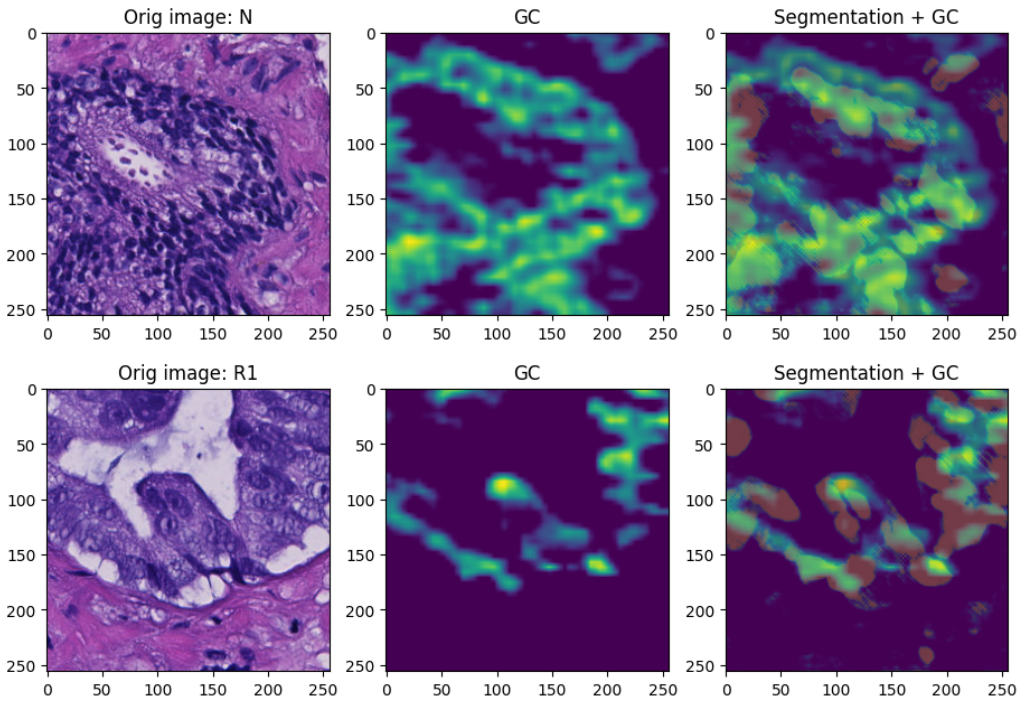

Figure 15: Two pairs of heat maps generated from GradCAM (GC) method and identical maps are stacked on segmentation outputs (selected best visual outputs)

4.3.5 Visual inspection of specific cases

Secondly, each interpretation will be visually compared with other interpretation methods. For visual inspection of these maps, they were overlaid on original patch images along with segmentation

masks. In the investigation, four tendencies of patches were chosen: 1) patches that were correctly classified with a high probability, 2) patches that were correctly classified but with a low probability, 3) patches that were incorrectly classified with a high probability, 4) patches that were incorrectly classified with a low probability. The summary is provided in Table 5.

	Classification results	Probability results
1st case	Correct	High ($\geq 60\%$)
2nd case	Correct	Low ($<60\%$)
3rd case	Incorrect	High ($\geq 60\%$)
4th case	Incorrect	Low ($<60\%$)

Table 5: Summary of cases (tendencies) that will be visually inspected

For each tendency, images were selected from N, R1-5 classes that best matched the previously mentioned descriptions (Table 5). In the images provided below, each layer is represented with a different color on the original image: green for the argumentation layer, red for the tumor segmentation mask, blue for the non-tumor segmentation mask, and yellow for the area where the tumor segmentation mask overlays with the argumentation layer.

To visualize only relevant pixels, an additional threshold was applied to the argumentation and segmentation outputs. In the argumentation output, pixels were set to 1 if the primary value was greater than 0.3 and 0 if the primary value was less than or equal to 0.3. For the segmentation mask, pixels with a value greater than 0.5 were set to 1, and pixels with a value less than or equal to 0.5 were set to 0.

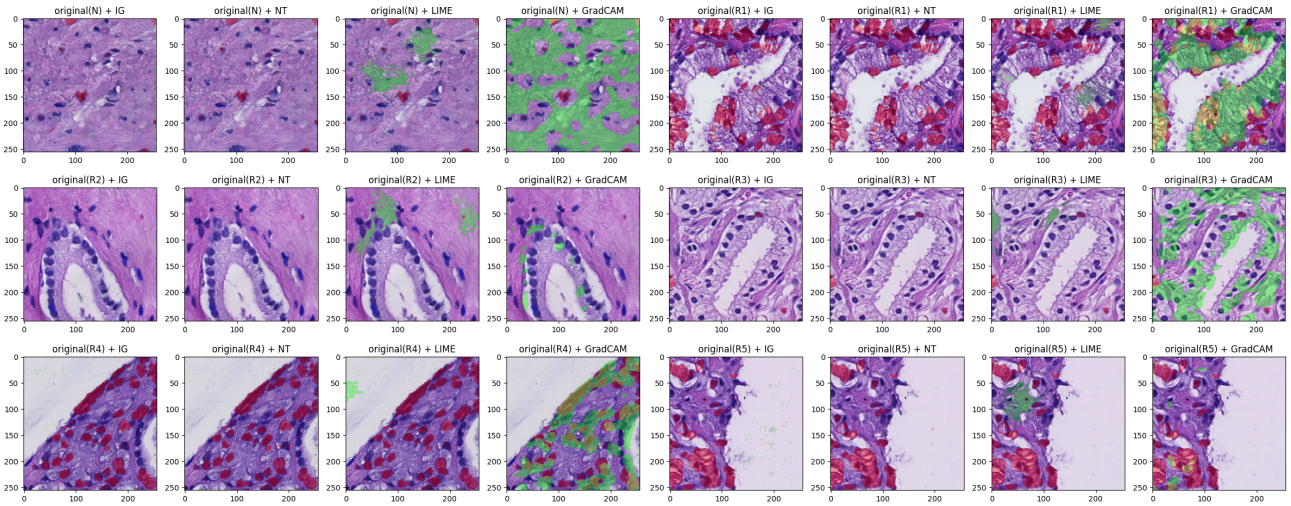

Figure 16: Chosen images from N, R1-5 classes that were correctly classified with the highest probability

The first case (Table 5) is illustrated in Figure 16. The GradCAM method exhibited the highest coverage on the original image, as evident in N, R1, R3, and R4 class images. Additionally, the largest number of yellow spots was observed on patches generated by GradCAM. However, there were some instances (ex. R2, R5), where the GradCAM-generated map covered only a small part of the patch. As mentioned earlier, the LIME method highlights up to three feature areas. Although, the LIME method marked important decision-making regions in patches, for this case, only a few small yellow

dots were observed on the R1 class image. Integrated Gradients and NoiseTunnel argumentation maps were barely noticeable in the provided example. Nevertheless, a few small green dots are identified on N, R2, R3, and R5.

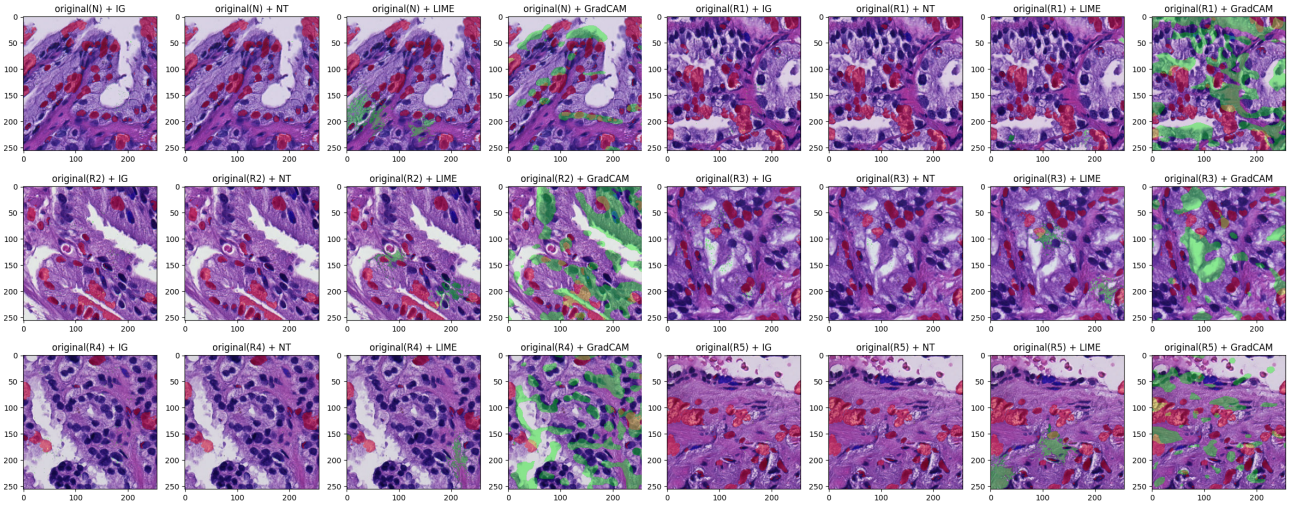

Figure 17: Selected images from N, R1-5 classes that were correctly classified with the low probability

The second tendency (Table 5) is depicted in Figure 17. The GradCAM method, as usual, generated the widest argumentation map, however a small number of yellow spots appeared from it. There were some instances in all provided patches where the GradCAM method focused on the areas around nuclei rather than targeting them. For this tendency, the LIME method highlighted quite small decision-making areas, but a few yellow spots appeared on the R2, R3, and R5 patches. The situation with Integrated Gradients and NoiseTunnel did not change in this tendency as well, argumentation maps were hardly visible. Small yellow dots appeared from Integrated Gradients and NoiseTunnel methods on R2 images.

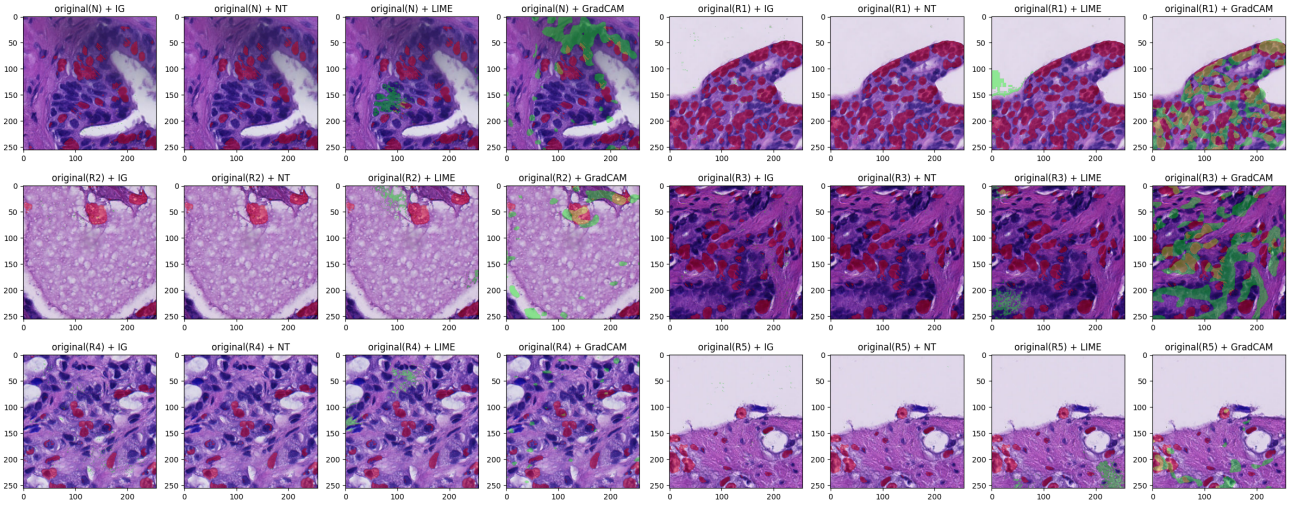

Figure 18: Chosen images from N, R1-5 classes that were incorrectly classified with the highest probability

The third case (Table 5) is illustrated in Figure 18. In several images (R1, R2, R3, R5), the Grad-

CAM method matched argumentation locations with tumor segmentation mask locations. However, there were some cases (R4) where this argumentation method provided small spots that should be responsible for decision-making. The LIME method provided several argumentation spots (R1, R2), which focused outside the tissue. It is worth mentioning that only a small yellow area was detected on the LIME argumentation for the R3 class patch. Nonetheless, other provided images did not have yellow spots from LIME argumentation. The same situation repeats for the Integrated Gradients and NoiseTunnel method, only a small amount of pixels were noticed on the stacked images (R4).

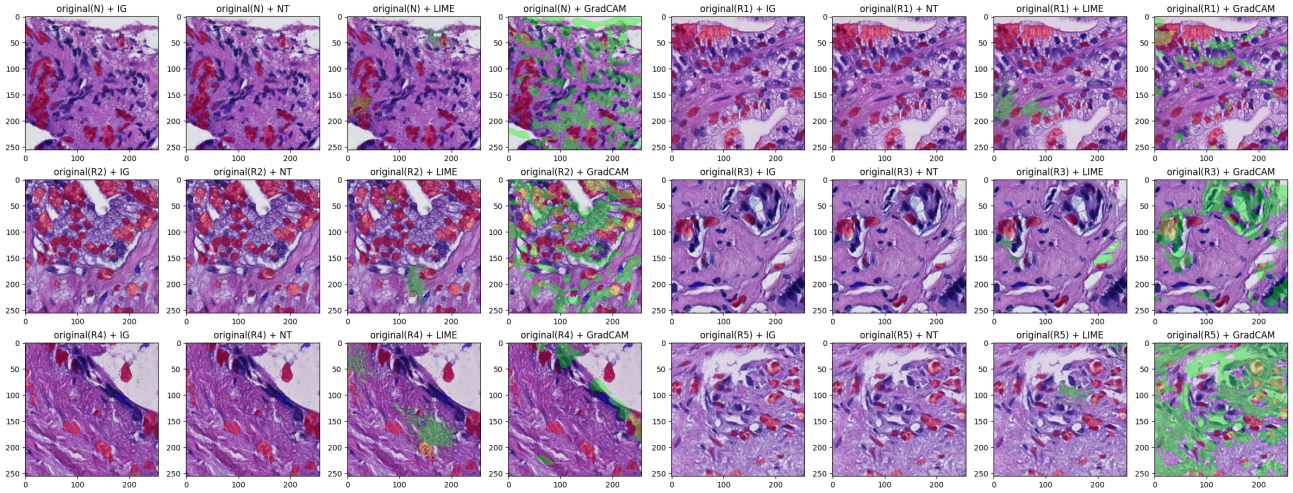

Figure 19: Chosen images from N, R1-5 classes that were incorrectly classified with the lowest probability

The last tendency (Table 5) is depicted in Figure 19. As usual, the GradCAM method's argumentation maps were the most noticeable. The method targeted areas outside nuclei or outside the tissue. Nevertheless, several yellow spots had appeared from GradCAM argumentation on all provided images. Likewise, the LIME method started to highlight areas outside nuclei. A few yellow spots were received from LIME argumentation (N, R2, R4). The Integrated Gradients method provided a few green argumentation dots. The NoiseTunnel did not provide any green spots.

4.4 Quality metrics results

In this section, results from the quality assessment will be presented. The quality of interpretation methods was evaluated by six methods: Intersection-over-Union (IoU), precision, recall, F1-score, pixel accuracy, and Spearman's rank correlation coefficient. All results will be displayed in violin plots. Additionally, average points from 4 different tendencies (Table 5) will presented on all plots to check the quality of argumentation for specific cases.

4.4.1 Iou results

The Intersection-over-Union (IoU) method measures the overlap between the generated interpretation output and the segmentation mask. In this case, measurements were conducted to evaluate the same interpretation output on both the tumor segmentation mask and the non-tumor segmentation mask. Results are presented in Figure 20.

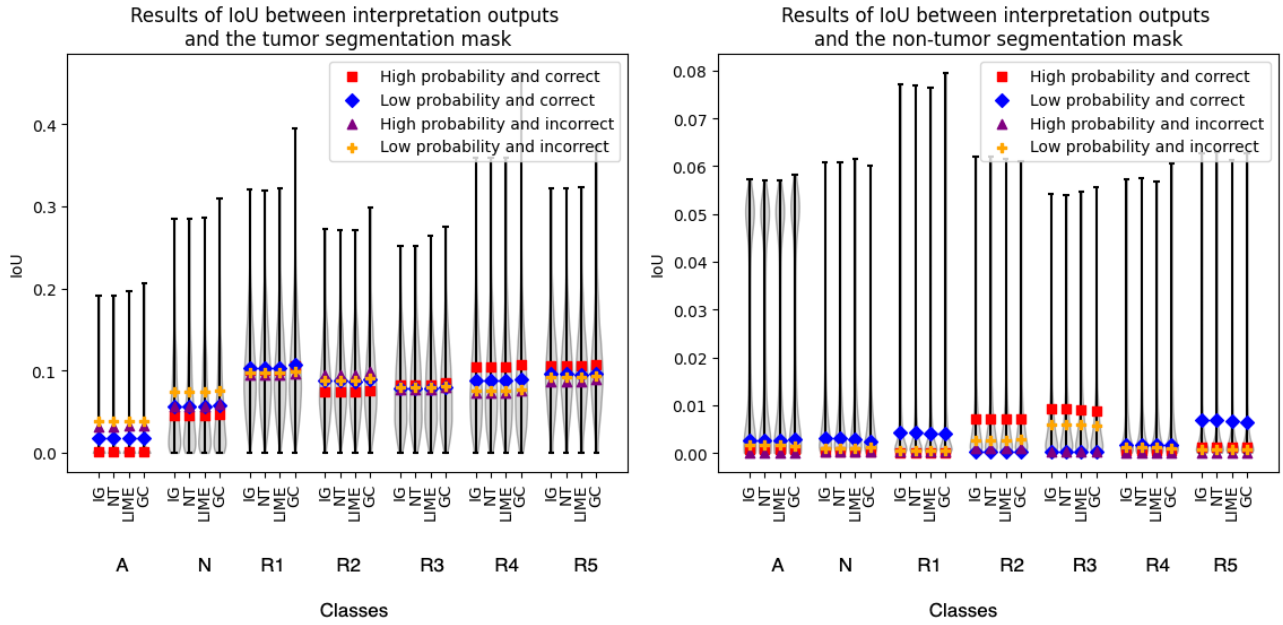

Figure 20: IoU results for each interpretation method for tumor and non-tumor segmentation masks. Points for the four analyzed tendencies are visualized on each violin plot

Higher IoU results were obtained when measuring the argumentation map with the tumor segmentation mask. IoU results on non-tumor segmentation masks varied from 0.00 to 0.08, while on tumor segmentation masks, it reached up to 0.45. Some patches with R4 labels had the highest scores, and all interpretation methods generated similar quality argumentation maps, except GradCAM maps, which had slightly higher results. The R1 class showed the second-best IoU results. It is worth mentioning that the argumentation map for images from the R4 class with correct classification and a high probability score had a higher IoU score than maps from other cases. All argumentation maps for the R1 class had similar scores, but argumentation maps for images that were correctly classified with a low probability score had a slightly higher score than the other tendencies. The lowest results were obtained for the A and R3 classes. Nevertheless, scores of A and N classes were distributed higher than scores for the other classes on the non-tumor segmentation mask. Additionally, all four different interpretation maps had similar IoU scores.

4.4.2 Precision results

Precision is responsible for identifying regions that are correctly classified as belonging to the class without introducing many false positive regions. Figure 21 visualizes precision results between the argumentation map and tumor/non-tumor masks.

Overall results between argumentation outputs and the tumor segmentation masks were higher than between argumentation outputs and non-tumor segmentation masks. The GradCAM method demonstrated higher results than other methods in the evaluation with the tumor segmentation mask. The second-best maps were received from the LIME method. The smallest precision was demonstrated by maps from the NoiseTunnel method. Argumentation maps for patches from R1 and R2 classes had the highest precision scores. Although precision results between generated interpretation maps and

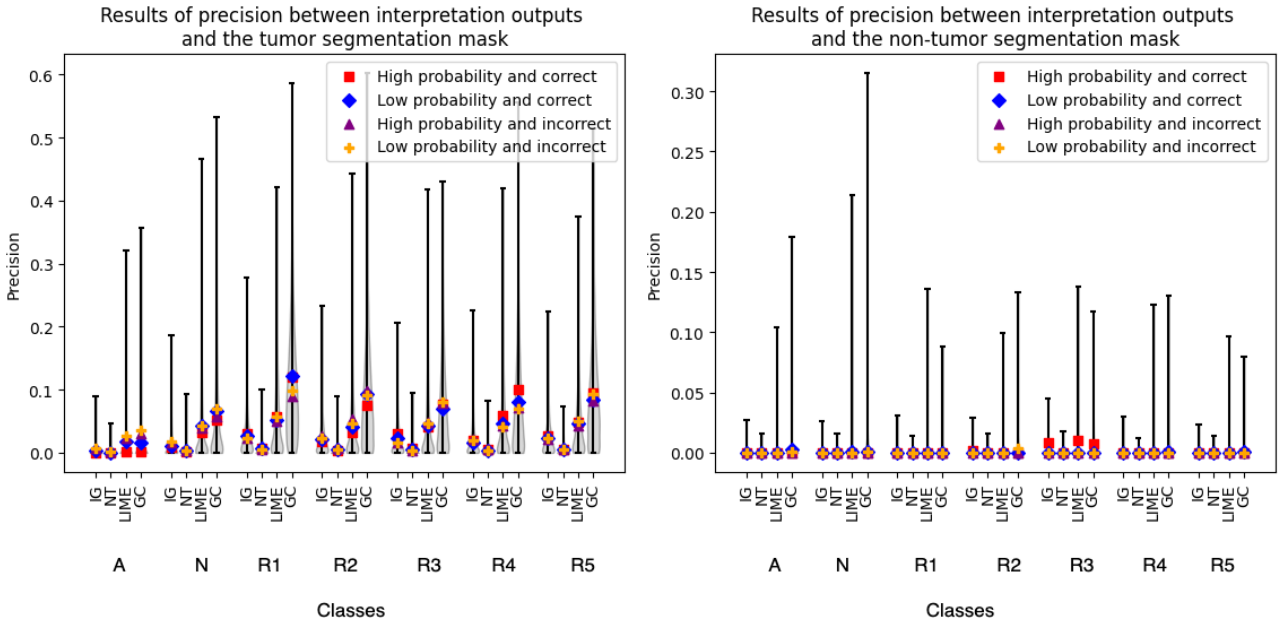

Figure 21: Precision results for each interpretation method for tumor and non-tumor segmentation masks. Points for the four analyzed tendencies are visualized on each violin plot

non-tumor segmentation masks were smaller than between the same maps and tumor segmentation masks, the GradCAM method generated slightly more precise outputs for A, N, R2, R4 classes, while the LIME method provided better interpretation for outputs from R1, R3, and R5 classes.

4.4.3 Recall results

Recall, also known as sensitivity, checks if the method captured as many regions as possible that belong to the class. Recall scores are demonstrated in Figure 22.

Most of the results between argumentation maps and tumor segmentation masks were distributed across a range from 0.1 and 0.4. However, some maps demonstrated high recall scores and showed results higher than 0.8. All high-recall results were received from the GradCAM method, and they occurred in all classes. Other interpretation methods did not show results higher than 0.2. Many recall results between argumentation maps and non-tumor segmentation masks distributed from 0.0 to 0.1. Nevertheless, there were interpretation maps, that reached scores up to 0.8 and these scores were from the GradCAM method.

4.4.4 F1-score results

F1-score gives a harmonic mean of precision and recall results and considers both false-negative and false-positive outputs. The evaluation of interpretation outputs between two segmentation masks is displayed in Figure 23.

Most of F1-scores between interpretation maps and tumor segmentation masks spread out from 0.0 to 0.2. There were some exceptions where the F1-score reached more than 0.3, and all these cases were for outputs from the GradCAM method. Other methods did not reach scores above 0.3. The LIME

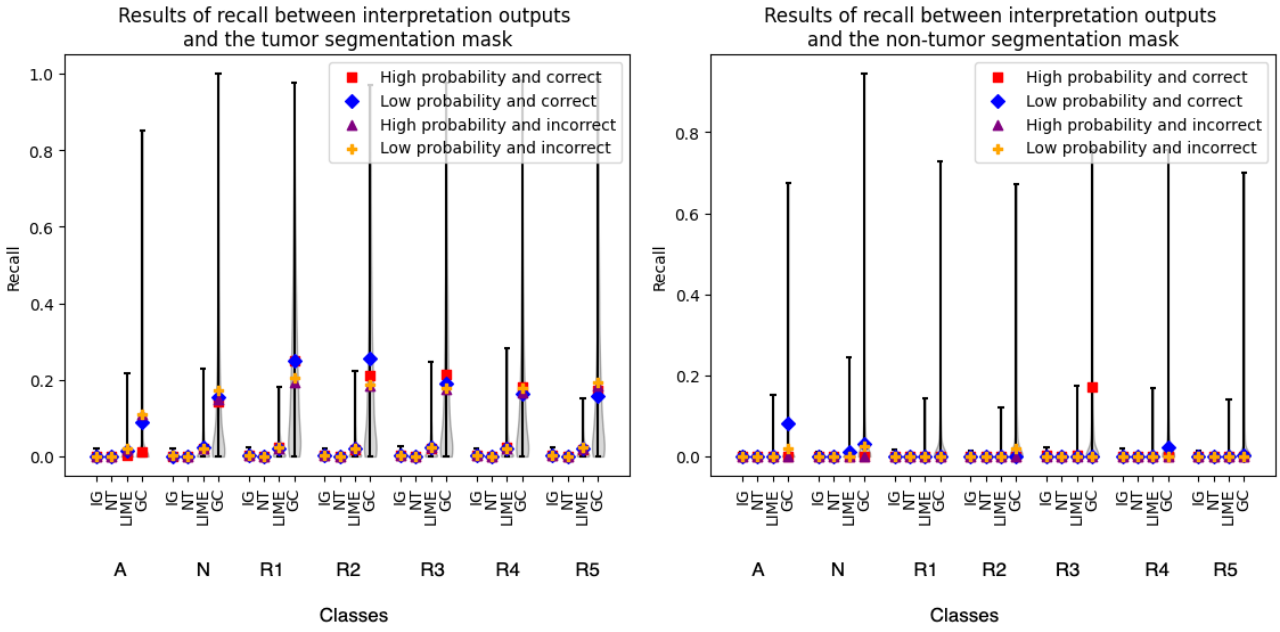

Figure 22: Recall results for each interpretation method for tumor and non-tumor segmentation masks. Points for the four analyzed tendencies are visualized on each violin plot

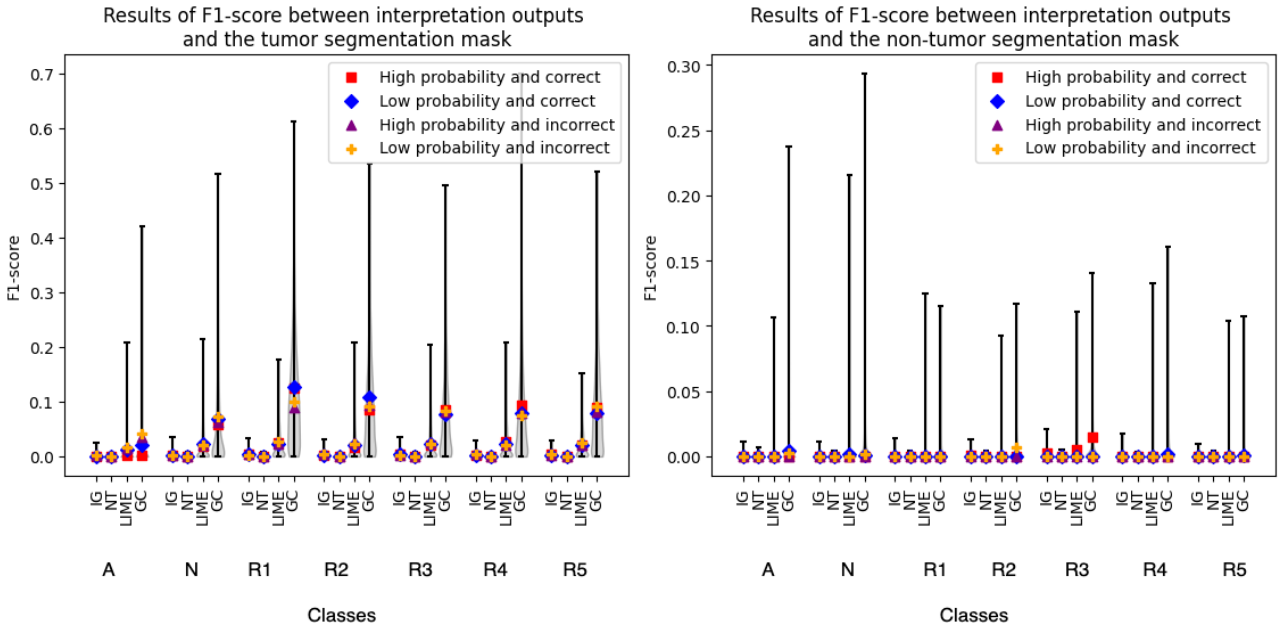

Figure 23: F1-score results for each interpretation method for tumor and non-tumor segmentation masks. Points for the four analyzed tendencies are visualized on each violin plot

method showed slightly better performance than the Integrated Gradients and NoiseTunnel approach. Results between interpretation outputs and non-tumor segmentation masks varied from 0.00 to 0.03. There were several cases where the F1-score varied from 0.05 to 0.3 and these scores were received from LIME and GradCAM methods.

4.4.5 Spearman's rank correlation results

Spearman's rank correlation shows a correlation between the generated interpretation output and segmentation masks. Results may vary from -1 to 1, and if the result is closer to -1 or 1, then a more perfect relation between the ground truth mask and obtained output exists. Computed Spearman's rank correlation is provided in Figure 24.

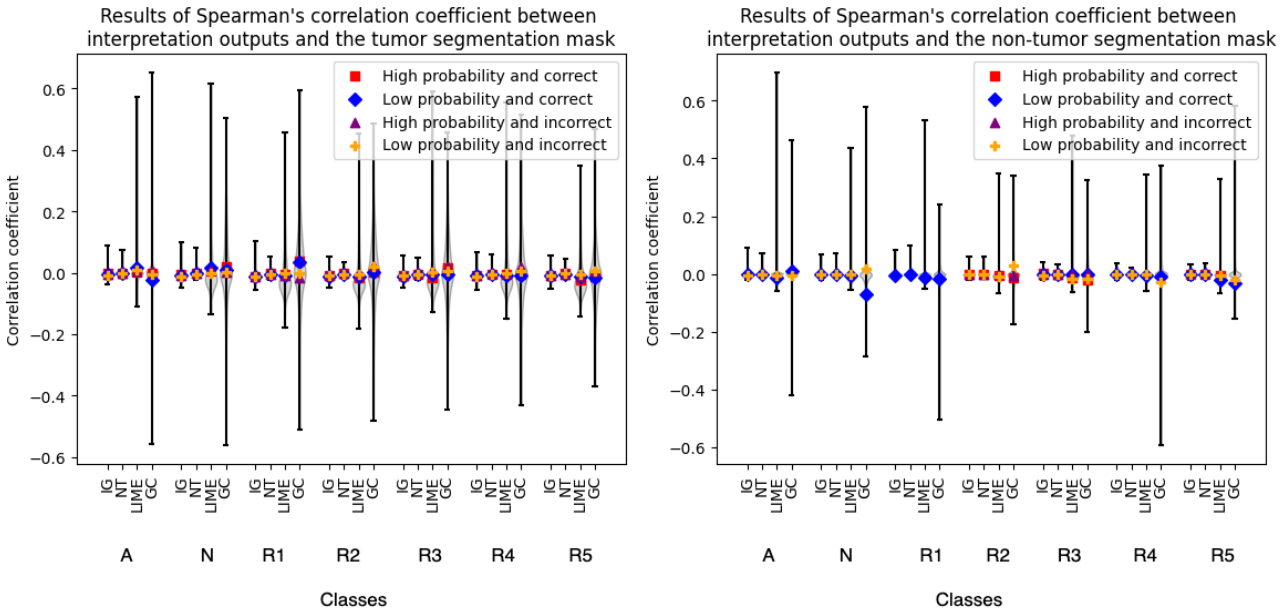

Figure 24: Spearman's rank correlation results for each interpretation method for tumor and non-tumor segmentation masks. Points for the four analyzed tendencies are visualized on each violin plot

Results between output maps and tumor segmentation masks were more scattered than compared to the same outputs with non-tumor segmentation masks. In the first pair, there were some results where generated maps reached results close to 0.6 or -0.6. Most of these scores were from the GradCAM method. Several interpretation maps from the LIME method reached coefficients higher than 0.3. Almost all coefficients were close to 0 from the Integrated Gradients and NoiseTunnel methods. In the second pair, there were some results above 0.3 or below -0.3, and these coefficients were obtained from the GradCAM and LIME methods. However, most of the results were centered near the 0 coefficient. Additionally, all classes from Table 5 showed the same manner in comparison with both ground truth objects. The average results of each tendency were gathered around 0.

4.4.6 Pixel accuracy results

The last metric for evaluating the quality of the generated interpretation output is pixel accuracy. It calculates the global accuracy between the obtained output map and segmentation masks. The measurement of pixel accuracy is provided in Figure 25.

GradCAM method, for both comparison cases between output and tumor segmentation mask and between output and non-tumor segmentation mask, showed the most scattered results than other methods. There were some cases where no pixel accuracy was found. In the measurement between the interpretation map and tumor segmentation mask, Integrated Gradients, NoiseTunnel, and LIME

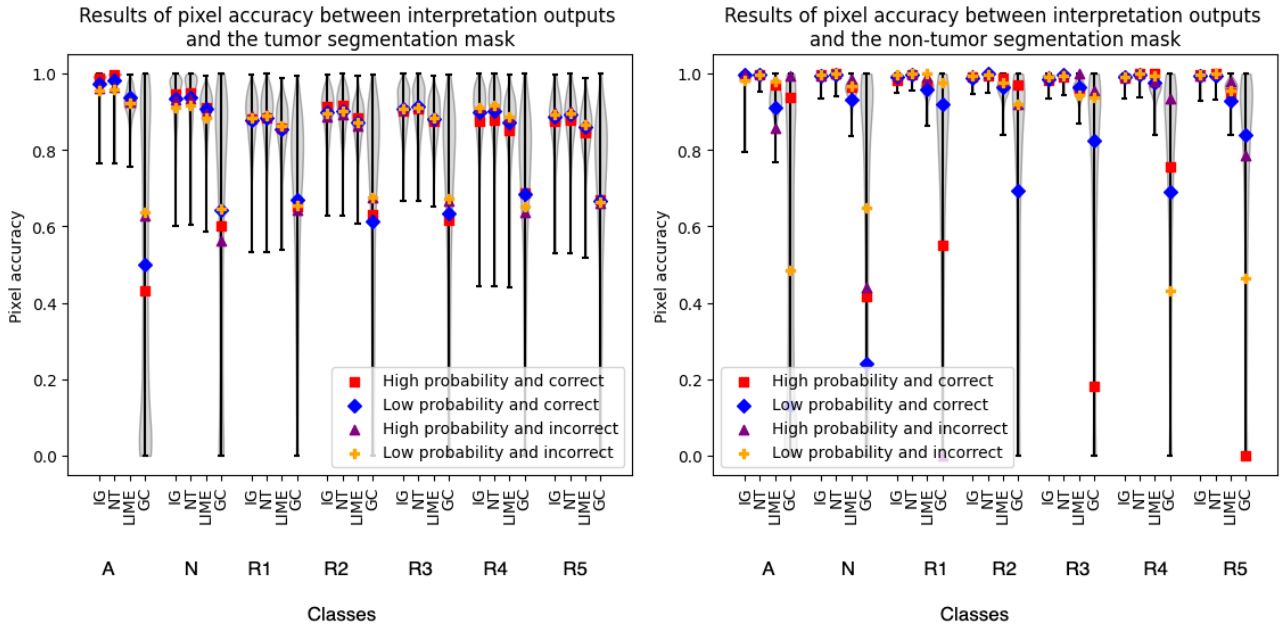

Figure 25: Pixel accuracy results for each interpretation method for tumor and non-tumor segmentation masks. Points for the four analyzed tendencies are visualized on each violin plot

methods had similar results for the same classes. Pixel accuracy for these methods did not drop below 0.4. In the measurement between the interpretation map and the non-tumor segmentation mask, Integrated Gradients, and NoiseTunnel methods showed similar results and they gathered around 1.0, except in the A class, where Integrated Gradients had several results lower than 0.9. The LIME method slightly differed from these two methods and had several cases, where accuracy dropped below 0.9.

4.4.7 Summary of results

The average results obtained from all evaluations for all four interpretation methods were gathered and presented in Table 6.

IoU evaluation showed that the GradCAM method had a better overlay with the ground truth outputs than other methods. Interpretation outputs for patches from the R5 class (highlighted in Table 6) had higher scores than other classes in comparison with the tumor segmentation mask. N class images had the highest scores between interpretation outputs and the non-tumor segmentation mask, than patches with R1-5 labels.

Precision calculation indicated that the GradCAM method demonstrated better results than other methods in comparison with both ground truth outputs. The second best method is the LIME technique (highlighted in Table 6).

Recall results did not yield any surprising outcomes. The GradCAM method distinguished itself from other techniques by obtaining the best results for both ground truth masks and all classes.

F1-score results followed a similar pattern to previous metrics. The GradCAM method exhibited the best performance. Very low results were obtained from the evaluation of NoiseTunnel outputs.

Type of masks														
	Tumor mask							Non-tumor mask						
Class / Metrics	A	N	R1	R2	R3	R4	R5	A	N	R1	R2	R3	R4	R5
IoU														
IG	0.0083	0.0558	0.0991	0.0856	0.0797	0.0864	0.1012	0.0218	**0.0062**	0.004	0.0056	0.006	0.005	0.0036
NT	0.0083	0.0558	0.0992	0.0857	0.0798	0.0864	0.1013	0.0216	**0.0062**	0.00397	0.0056	0.006	0.005	0.0036
LIME	0.0084	0.0561	0.0993	0.0856	0.0799	0.0865	0.1011	0.0217	**0.0062**	0.004	0.0056	0.005973	0.005	0.0035
GC	**0.0085**	**0.0576**	**0.1028**	**0.088**	**0.0816**	**0.0882**	**0.1028**	0.022	**0.0062**	0.004	0.0056	0.006	0.005	0.0036
Precision														
IG	0.0017	0.0123	0.0273	0.022	0.0218	0.0193	0.0255	0.0003	0.0008	0.0004	0.0006	0.0011	0.0007	0.0003
NT	0.0004	0.0029	0.0066	0.0052	0.0048	0.0044	0.0059	0.00005	0.0001	0.00007	0.0001	0.0002	0.00009	0.00004
LIME	0.0061	0.0375	0.055	0.0433	0.0441	0.0475	0.0479	0.0009	0.0025	0.0014	0.0019	0.0025	0.0019	0.001
GC	**0.008**	**0.0602**	**0.1106**	**0.0891**	**0.0767**	**0.0813**	**0.0921**	**0.0012**	**0.0044**	**0.0028**	**0.004**	**0.0048**	**0.004**	**0.0023**
Recall														
IG	0.0003	0.0013	0.0021	0.0019	0.0022	0.0018	0.002	0.0001	0.0003	0.0002	0.0003	0.0005	0.0003	0.0001
NT	0.00004	0.0002	0.0003	0.0002	0.0002	0.0002	0.0002	0.00001	0.00003	0.00002	0.00002	0.00004	0.00002	0.00001
LIME	0.0051	0.0215	0.0225	0.0202	0.022	0.0219	0.0197	0.0014	0.0045	0.0024	0.0031	0.0041	0.003	0.0018
GC	**0.0308**	**0.153**	**0.2298**	**0.2047**	**0.1906**	**0.1736**	**0.1754**	**0.0131**	**0.0474**	**0.0363**	**0.0449**	**0.0493**	**0.0401**	**0.0271**
F1-score														
IG	0.0004	0.002	0.0035	0.0031	0.0035	0.0028	0.0032	0.00013	0.0003	0.0002	0.0003	0.0005	0.0003	0.0001
NT	0.0001	0.0003	0.0005	0.0004	0.0004	0.0004	0.0004	0.00002	0.00004	0.00003	0.00003	0.00006	0.00003	0.00001
LIME	0.0041	0.0203	0.0244	0.0208	0.0225	0.0228	0.0213	0.0008	0.0024	0.0013	0.0017	0.0023	0.0018	0.001
GC	**0.0096**	**0.0649**	**0.1134**	**0.0927**	**0.0818**	**0.0819**	**0.0875**	**0.0018**	**0.0065**	**0.0044**	**0.0059**	**0.0072**	**0.0061**	**0.0036**
Correlation coefficient														
IG	-0.0016	-0.0079	-0.0111	-0.0101	**-0.0101**	**-0.0101**	-0.0093	-0.0007	-0.003	-0.0024	-0.0032	-0.0032	-0.0026	-0.0021
NT	-0.0005	-0.0028	-0.0036	-0.0033	-0.0037	-0.0037	-0.0036	-0.0003	-0.0011	-0.0009	-0.0012	-0.0012	-0.001	-0.0008
LIME	**0.003**	0.0062	-0.0061	-0.0105	-0.0049	-0.0051	**-0.0185**	-0.0003	-0.0033	**-0.0033**	**-0.0053**	**-0.0039**	**-0.0034**	**-0.0039**
GC	-0.0012	**0.0105**	**0.0186**	**0.0124**	0.0081	0.0007	-0.0089	**-0.0023**	**-0.0042**	-0.0027	-0.0016	0.0004	0.0009	-0.0015
Pixel accuracy														
IG	0.9827	0.9336	0.8824	0.8998	0.9057	0.898	0.8808	0.9899	0.9894	0.9911	0.9896	0.9884	0.9905	0.9925
NT	**0.9899**	**0.9379**	**0.8866**	**0.9039**	**0.9104**	**0.9019**	**0.8841**	**0.9972**	**0.994**	**0.9958**	**0.9941**	**0.9937**	**0.9948**	**0.9964**
LIME	0.9255	0.9024	0.8588	0.8732	0.8799	0.8727	0.8527	0.9319	0.954	0.9616	0.9585	0.9578	0.9599	0.9594
GC	0.4687	0.6171	0.6523	0.6512	0.6467	0.6677	0.6669	0.4716	0.6397	0.6967	0.6899	0.6841	0.7168	0.731

Table 6: A summary of the average results for each interpretation method across different quality assessment metrics. Results and method names are highlighted to facilitate the distinction of obtained results.

The Spearman's correlation coefficient metrics did not identify one particularly superior interpretation method. Different interpretation techniques showed better results for different classes. However, average results were concentrated near 0.

Pixel accuracy evaluation indicates that the GradCAM method had lower results than other methods. In most cases, the best results were obtained from the NoiseTunnel method.

5 Discussion

After conducting several quality assessment measurements, various tendencies related to different interpretation methods were observed. The discussion will integrate comments about interpretation maps from a visual perspective, along with the results of quality metrics.

Different interpretation methods generated distinct types of interpretation outputs. The Integrated Gradients method tended to produce a collection of dots (crucial pixels) responsible for decision-making. These dots may be scattered across all image areas or concentrated in a particular region where a potential feature exists. While dots remained visible when they stacked on the segmentation mask, observing them in the original image can be challenging. Consequently, evaluating whether the output from the Integrated Gradients method overlays with crucial features on the original image can be difficult. The obtained IoU results showed that Integrated Gradients had one of the lowest scores (Table 6). A similar situation arose when implementing the Noise Tunnel method. The output consisted of separate dots representing pixels crucial for decision-making. The number of dots was much smaller than in Integrated Gradients because the Noise Tunnel method reduced sensitivity. Thus, on segmentation masks, dots remained quite visible, allowing for quality assessment, but on the regular image, these dots are hardly noticeable and challenging to discern by the human eye. Both these methods generated the lowest-quality outputs (Table 6). However, pixel accuracy evaluation showed relatively high scores for Integrated Gradients and NoiseTunnel methods. Pixel accuracy compares whether the pixel locations from the ground truth mask match those in the generated map. Therefore, in these two methods, some pixels matched locations along with the segmentation mask.

Contrasting interpretation maps were generated from the LIME method. As mentioned earlier, the method highlights up to three of the most important decision-making regions in the image. Highlighted areas can be quite large, covering a noticeable part of the image, but also there were cases, where only small spots were marked. Based on the examples provided in Figure 16-19, the interpretation maps obtained from the LIME method can be seen on the original image, making it easier to evaluate if the LIME method marks crucial decision-making regions. According to the results in Table 6, the LIME technique is the second-best interpretation method. Although the LIME method did not show higher results in any metrics, quite remarkable outcomes were obtained from precision evaluation (Figure 21). Precision assesses whether outputs of the interpretation method highlight regions that belong to the given class with fewer false positives. The LIME method demonstrated characteristics of generating interpretation maps with features important for decision-making without introducing many false-positive regions. Additionally, outputs from the LIME method tended to have positive correlation coefficients (Figure 24). This implies that the method generated interpretation maps highlighting features in the nuclei, rather than focusing on the area around them.

The last method, GradCAM tended to generate the most detailed interpretation maps. Based on the examples in Figure 16-19, GradCAM maps are visible on the original image, making it easy to assess which areas are presented as crucial for decision-making by the CNN model. Quality metrics have shown that the GradCAM method had the best interpretation maps, distinguishing itself in IoU, precision, recall, and F1-sore metrics (Table 6). In the evaluation of Spearman's rank correlation (Figure 24), several outputs from the GradCAM method showed a negative relation between obtained

maps and ground truth masks. Scores closer to -1 suggested that generated outputs were focusing on areas outside the important regions. The same tendency could be noticed in the provided examples in Figures 16-19, where interpretation maps from GradCAM covered areas around nuclei or connective tissue. For the same reason, some outputs from GradCAM had small pixel accuracy scores, as the ground truth pixels' location did not match with the generated output pixels' location. Therefore, some interpretation maps showed low pixel accuracy results.

Outputs from all interpretation methods assigned to R1-5 classes showed higher results for all metrics on the tumor segmentation mask than the non-tumor segmentation mask. Therefore, it is safe to assume that interpretation methods tended to focus on nuclei regions in the image, than connective tissue. Among all R1-5 classes, the R1 class showed the highest results in precision and F1-score. All interpretation methods, except Integrated Gradients had the highest recall results in the R1 class. The interpretation with GradCAM and Integrated Gradients of the R1 class had the best evaluation in Spearman's correlation coefficient metric. Interpretation of images with R5 class labels had the best performance in IoU metrics. However, the same outputs had one of the lowest scores in pixel accuracy evaluation. Nevertheless, the R1 and R5 classes had the highest scores in classification accuracy. Both of these classes were additionally augmented before training. The worst results were received from the R3 and R2 classes, and these classes had the lowest classification accuracy (Table 4). The R2 class was augmented before classification, but the R3 class was not augmented. Therefore, image augmentation did not have an influence of classification accuracy.

It is worth mentioning that during the investigation of four different tendencies in Figure 16-19, there were no exceptional cases that would be related only to specific tendencies (Table 5). Additionally, each tendency was marked in the quality assessment violin plots (Figure 20-25) to check if one of these cases had higher results than others. Nevertheless, no remarkable pattern was observed. In different quality assessment methodologies, different tendencies showed higher results. Therefore, without a comprehensive inspection, it is safe to assume that interpretation methods can generate good-quality interpretation maps not only for images, that were correctly classified with high probability but also for correctly classified images with low probability scores and mistakenly classified images with low/high probability scores. The ability to provide detailed interpretations for mistakenly classified images would help assess which regions were utilized for incorrect decisions by the CNN-based model.

6 Conclusions

In this research, a convolutional neural network for histopathological image classification was implemented along with four different interpretation methods. The primary objective was to determine which of interpretation method could offer the best and the most transparent insights into the decision-making process of the CNN-based model. Integrated Gradients, Integrated Gradients with NoiseTunnel, LIME, GradCAM were used as interpretation methods and their outputs were thoroughly examined. The Investigation comprised two main parts: visual inspection and quality assessment. The generated interpretation outputs were compared to two segmentation masks: tumor and non-tumor masks.

The findings of this study are summarized in the following enumerated points, explaining the key outcomes and insights derived from the conducted analyses:

1. The interpretation of images with cancer labels yielded higher metrics results when compared to the tumor mask than the non-tumor mask. This suggests that the network is emphasizing nuclei regions in the image, and interpretation methods validate this decision.

2. The GradCAM method generated the most detailed interpretation maps, and based on almost all quality assessment methods, it had the best interpretation maps.

3. There is some evidence, that suggests that the GradCAM method tended to highlight regions outside nuclei or tissue areas.

4. The LIME method emerged as the second-best interpretation method, with characteristics to highlight features in the nuclei rather than focusing on the surrounding areas.

5. Integrated Gradients and Integrated Gradients with NoiseTunnel provided the lowest quality outputs.

6. For images with tumor labels, the best interpretation maps were obtained for classes that had one of the highest classification accuracies: images with the lowest aggression cancer (R1 class) and images with the highest aggression cancer (R5 class).

7. It was identified that interpretation methods can generate similar quality interpretation maps for correctly and incorrectly classified images with low and high probability scores. This suggests that these maps can highlight crucial decision-making regions in all cases.

8. Further improvements should be implemented in interpretation methods and classification methods. Firstly, the CNN network should be trained on a larger dataset to obtain a more versatile model. Secondly, interpretation method parameters should be investigated more deeply, for instance implementing the LIME method to produce more than three important feature maps or increasing the sensitivity of the NoiseTunnel method.

In summary, interpretation methods can enhance the transparency of the CNN-based model's decision and provide more insights into the implemented classification method. Further enhancements, including training on larger datasets and refining method parameters, are essential for future improvements.

References

[1] Michael Anis Mihdi Afnan et al. *Interpretable, not black-box, artificial intelligence should be used for embryo selection*. 2021.

[2] Alejandro Barredo Arrieta et al. "Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI". In: *Information fusion* 58 (2020), pp. 82–115.

[3] Sugata Banerji and Sushmita Mitra. "Deep learning in histopathology: A review". In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 12.1 (2022), e1439.

[4] Samyadeep Basu, Philip Pope, and Soheil Feizi. "Influence functions in deep learning are fragile". In: *arXiv preprint arXiv:2006.14651* (2020).

[5] Kaustav Bera et al. "Artificial intelligence in digital pathology—new tools for diagnosis and precision oncology". In: *Nature reviews Clinical oncology* 16.11 (2019), pp. 703–715.

[6] Yeong-Jun Cho. "Weighted Intersection over Union (wIoU): A New Evaluation Metric for Image Segmentation". In: *arXiv preprint arXiv:2107.09858* (2021).

[7] Athena Davri et al. "Deep Learning on Histopathological Images for Colorectal Cancer Diagnosis: A Systematic Review". In: *Diagnostics* 12.4 (2022), p. 837.

[8] Jevgenij Gamper et al. "Pannuke dataset extension, insights and baselines". In: *arXiv preprint arXiv:2003.10778* (2020).

[9] Alberto Garcia-Garcia et al. "A review on deep learning techniques applied to semantic segmentation". In: *arXiv preprint arXiv:1704.06857* (2017).

[10] John Grezmak et al. "Interpretable convolutional neural network through layer-wise relevance propagation for machine fault diagnosis". In: *IEEE Sensors Journal* 20.6 (2019), pp. 3172–3181.

[11] Riccardo Guidotti et al. "A survey of methods for explaining black box models". In: *ACM computing surveys (CSUR)* 51.5 (2018), pp. 1–42.

[12] Han Guo et al. "Fastif: Scalable influence functions for efficient model interpretation and debugging". In: *arXiv preprint arXiv:2012.15781* (2020).

[13] Miriam Hägele et al. "Resolving challenges in deep learning-based analyses of histopathological images using explanation methods". In: *Scientific reports* 10.1 (2020), p. 6423.

[14] A Ben Hamida et al. "Deep learning for colon cancer histopathological images analysis". In: *Computers in Biology and Medicine* 136 (2021), p. 104730.

[15] Ralf Huss and Sarah E Coupland. "Software-assisted decision support in digital histopathology". In: *The Journal of Pathology* 250.5 (2020), pp. 685–692.

[16] Sheikh Rabiul Islam et al. "Explainable artificial intelligence approaches: A survey". In: *arXiv preprint arXiv:2101.09429* (2021).

[17] Jiaqian Yu et al. "Learning generalized intersection over union for dense pixelwise prediction". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 12198–12207.

[18] Maksims Ivanovs, Roberts Kadikis, and Kaspars Ozols. "Perturbation-based methods for explaining deep neural networks: A survey". In: *Pattern Recognition Letters* 150 (2021), pp. 228–234.

[19] Sumit Kumar Jha et al. "Neural Stochastic Differential Equations for Robust and Explainable Analysis of Electromagnetic Unintended Radiated Emissions". In: *arXiv preprint arXiv:2309.15386* (2023).

[20] Alain Jungo, Fabian Balsiger, and Mauricio Reyes. "Analyzing the quality and challenges of uncertainty estimations for brain tumor segmentation". In: *Frontiers in neuroscience* 14 (2020), p. 282.

[21] Donya Khaledyan et al. "Enhancing breast ultrasound segmentation through fine-tuning and optimization techniques: Sharp attention UNet". In: *Plos one* 18.12 (2023), e0289195.

[22] Michał Koziarski et al. "DiagSet: a dataset for prostate cancer histopathological image classification". In: *arXiv preprint arXiv:2105.04014* (2021).

[23] Abhinav Kumar et al. "Deep feature learning for histopathological image classification of canine mammary tumors and human breast cancer". In: *Information Sciences* 508 (2020), pp. 405–421.

[24] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), pp. 436–444.

[25] Xuhong Li et al. "Interpretable deep learning: Interpretation, interpretability, trustworthiness, and beyond". In: *Knowledge and Information Systems* 64.12 (2022), pp. 3197–3234.

[26] Pantelis Linardatos, Vasilis Papastefanopoulos, and Sotiris Kotsiantis. "Explainable ai: A review of machine learning interpretability methods". In: *Entropy* 23.1 (2020), p. 18.

[27] Adriano Lucieri et al. "On interpretability of deep learning based skin lesion classifiers using concept activation vectors". In: *2020 international joint conference on neural networks (IJCNN)*. IEEE. 2020, pp. 1–10.

[28] Samuel Ortega et al. "Hyperspectral and multispectral imaging in digital and computational pathology: a systematic review". In: *Biomedical Optics Express* 11.6 (2020), pp. 3195–3233.

[29] Pushpak Pati et al. *HACT-Net: A Hierarchical Cell-to-Tissue Graph Neural Network for Histopathological Image Classification*. 2020. arXiv: 2007.00584 [cs.CV].

[30] Mauricio Reyes et al. "On the interpretability of artificial intelligence in radiology: challenges and opportunities". In: *Radiology: artificial intelligence* 2.3 (2020), e190043.

[31] Yair Rivenson et al. "Emerging advances to transform histopathology using virtual staining". In: *BME Frontiers* 2020 (2020).

[32] Zohaib Salahuddin et al. "Transparency of deep neural networks for medical image analysis: A review of interpretability methods". In: *Computers in biology and medicine* 140 (2022), p. 105111.

[33] Ramprasaath R. Selvaraju et al. "Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization". In: *International Journal of Computer Vision* 128.2 (Oct. 2019), pp. 336–359. ISSN: 1573-1405. DOI: 10.1007/s11263-019-01228-7. URL: http://dx.doi.org/10.1007/s11263-019-01228-7.

[34] Dan Shao et al. “Artificial intelligence in clinical research of cancers”. In: *Briefings in Bioinformatics* 23.1 (2022), bbab523.

[35] Connor Shorten, Taghi M Khoshgoftaar, and Borko Furht. “Text data augmentation for deep learning”. In: *Journal of big Data* 8 (2021), pp. 1–34.

[36] Nahian Siddique et al. “U-net and its variants for medical image segmentation: A review of theory and applications”. In: *Ieee Access* 9 (2021), pp. 82031–82057.

[37] Daniel Smilkov et al. *SmoothGrad: removing noise by adding noise*. 2017. arXiv: `1706.03825 [cs.LG]`.

[38] David F Steiner et al. “Impact of deep learning assistance on the histopathologic review of lymph nodes for metastatic breast cancer”. In: *The American journal of surgical pathology* 42.12 (2018), p. 1636.

[39] Ahmed S Sultan et al. “The use of artificial intelligence, machine learning and deep learning in oncologic histopathology”. In: *Journal of Oral Pathology & Medicine* 49.9 (2020), pp. 849–856.

[40] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. *Axiomatic Attribution for Deep Networks*. 2017. arXiv: `1703.01365 [cs.LG]`.

[41] Jeroen Van der Laak, Geert Litjens, and Francesco Ciompi. “Deep learning in histopathology: the path to the clinic”. In: *Nature medicine* 27.5 (2021), pp. 775–784.

[42] Yunjun Wang et al. “Using deep convolutional neural networks for multi-classification of thyroid tumor by histopathology: a large-scale pilot study”. In: *Annals of translational medicine* 7.18 (2019).

[43] Yawen Wu et al. “Recent advances of deep learning for computational histopathology: Principles and applications”. In: *Cancers* 14.5 (2022), p. 1199.

[44] Zizhao Zhang et al. “Pathologist-level interpretable whole-slide cancer diagnosis with deep learning”. In: *Nature Machine Intelligence* 1.5 (2019), pp. 236–245.

Appendix A

Python code of classification model's training, validation:

```
import datetime
import sys

import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.tensorboard import SummaryWriter
from torch.utils.data import DataLoader
from dataset import CustomImageDataset
import config as cfg
import time

batch_size = 64
n_epochs = 50
lr = 0.001
datadate = datetime.datetime.now().strftime("%Y%m%d")

#Pytorch NN
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(in_channels = 3, out_channels = 16, kernel_size = 3)
        #print(self.conv1.weight.shape)
        self.conv1_bn = nn.BatchNorm2d(16)
        self.conv2 = nn.Conv2d(in_channels = 16, out_channels = 32,  kernel_size = 5)
        self.conv3 = nn.Conv2d(in_channels = 32, out_channels = 64,  kernel_size = 5)

        self.fc1 = nn.Linear(in_features=64 * 27 * 27, out_features=2048)
        self.fc1_bn = nn.BatchNorm1d(2048)
        self.fc2 = nn.Linear(in_features=2048, out_features=1024)
        self.fc3 = nn.Linear(in_features=1024, out_features=512)
        self.fc4 = nn.Linear(in_features=512, out_features=256)
        self.fc5 = nn.Linear(in_features=256, out_features=128)
        self.fc6 = nn.Linear(in_features=128, out_features=64)
        self.out = nn.Linear(in_features=64, out_features=2)

    def forward(self, t):
```

```
        t = self.conv1(t)
        t = F.relu(self.conv1_bn(t))
        t = F.max_pool2d (t, kernel_size = 3, stride = 2)
        t = F.relu(self.conv2(t))
        t = F.max_pool2d(t, kernel_size = 3, stride = 2)
        t = F.relu(self.conv3(t))
        t = F.max_pool2d(t, kernel_size = 3, stride = 2)

        t=t.view(t.size(0), -1)

        t = F.relu(self.fc1(t))
        t = F.relu(self.fc1_bn(t))
        t = F.relu(self.fc2(t))
        t = F.relu(self.fc3(t))
        t = F.relu(self.fc4(t))
        t = F.relu(self.fc5(t))
        t = F.relu(self.fc6(t))
        t = self.out(t)
        return t

class cnnModel():

    def get_num_correct(self, preds, labels):
        return preds.argmax(dim=1).eq(labels).sum().item()

    def model_param(self, num_clss):
        model = CNN()
        model.out = nn.Linear(in_features=64, out_features=num_clss)
        criterion = nn.CrossEntropyLoss()
        optimizer = torch.optim.Adam(model.parameters(), lr=lr)
        return criterion, optimizer, model

    def training_validation_part(self, train_dataset, valid_dataset, n_epochs, batch_size):
        min_valid_loss = np.inf
        valid_loss = 0.0
        patience = 5
        triggertimes = 0
        criterion, optimizer, model = self.model_param(train_dataset.num_clss)
        model.to(DEVICE)
        train_dataloader = DataLoader(train_dataset,batch_size=batch_size, shuffle=True)
        valid_dataloader = DataLoader(valid_dataset,batch_size=batch_size, shuffle=True)
```

```
for epoch in range(n_epochs+1):
    total_loss = 0
    total_correct = 0
    current_valid_loss = 0.0
    train_length = 0
    valid_length = 0
    sublist_index = 0
    for train_images, train_labels in train_dataloader:
        train_labels = train_labels.to(DEVICE)
        train_images = train_images.to(DEVICE)
        train_length += len(train_images)
        prediction = model(train_images)
        loss = criterion(prediction, train_labels.long())
        optimizer.zero_grad() #clear gradients
        loss.backward() #backpropogation
        optimizer.step() #update weights

        #for analytics
        total_loss += loss.item() * train_images.size(0)
        total_correct += self.get_num_correct(prediction, train_labels)

    for valid_images, valid_labels in valid_dataloader:
        valid_length += len(valid_images)
        valid_images = valid_images.to(DEVICE)
        valid_labels = valid_labels.to(DEVICE)
        target = model(valid_images)
        loss = criterion(target, valid_labels.long())
        current_valid_loss += loss.item() * valid_images.size(0)

    if min_valid_loss > current_valid_loss:
        min_valid_loss = current_valid_loss
        triggertimes = 0
        # Saving State Dict
        torch.save(model.state_dict(), saved_model_name)

    #early stopping
    elif min_valid_loss >= valid_loss:
        triggertimes += 1
        if triggertimes >= patience:
            break
```

```
            else:
                triggertimes = 0

            valid_loss = current_valid_loss
            accuracy = total_correct/train_length
            print(f'accuracy: {accuracy}')
            sys.stdout.flush()


if __name__ == '__main__':
    train_dataset = CustomImageDataset(cfg.root_pth, cfg.clss_dict,
    cfg.augmented_root_pth, json_pth=train_index_file)
    valid_dataset = CustomImageDataset(cfg.root_pth, cfg.clss_dict,
    cfg.augmented_root_pth, json_pth=valid_index_file)
    train_and_validate = cnnModel().training_validation_part(train_dataset,
    valid_dataset, n_epochs, batch_size)
```

Python code for classification model's testing:

```
import numpy as np
import torch
import torch.nn as nn
from torch.utils.data import DataLoader
from dataset import CustomImageDataset
import config as cfg
import h5py
from cnn_train_test import CNN

DEVICE = "cuda" if torch.cuda.is_available() else "cpu"

class_codes = {'A': 0, 'N': 1, 'R1': 2, 'R2': 3, 'R3': 4, 'R4': 5, 'R5': 6}

batch_size = 1
test_index_file = '../index_list/test_indexes.txt'
out_file = f"../cnn_20231026_balanced_rez.h5"

true_labels = []
predicted_labels = []
prediction_scores = []

test_dataset = CustomImageDataset(cfg.root_pth, cfg.clss_dict, cfg.augmented_root_pth,
```

```
json_pth=test_index_file)
test_dataloader = DataLoader(test_dataset,batch_size=batch_size, shuffle=False)

model = CNN()
model.out = nn.Linear(in_features=64, out_features=7)
criterion = nn.CrossEntropyLoss()

model.load_state_dict(torch.load('../saved_models/cnn_20231026_balanced.pth'))
model.to(DEVICE)
model.eval()

passed = 0
test_loss = 0
correct = 0
test_length = 0
rez = []

with torch.no_grad():
    for img_num, (test_images, test_labels) in enumerate(test_dataloader):
        test_length += len(test_images)
        test_images = test_images.to(DEVICE)
        test_labels = test_labels.to(DEVICE)
        passed +=1
        output = model(test_images)
        test_loss += criterion(output, test_labels.long()).item() * test_images.size(0)
        rez.append(output)
        pred = output.data.max(1, keepdim=True)[1]
        correct += pred.eq(test_labels.data.view_as(pred)).sum()

preds = np.concatenate(rez, axis=0)
with h5py.File(out_file, 'w') as f:
    f.create_dataset('preds', data = preds)
```

Python code for data augmentation:

```
import os
import numpy as np
import sys

path = '../DiagSet-A/blobs/S/20x/'
path_to_save_images = '../DiagSet-A/blobs/S/Augmented_Patches'

class_codes = {'A': 0, 'N': 1, 'R1': 2, 'R2': 3, 'R3': 4, 'R4': 5, 'R5': 6}
```

```
def save_image(array, name, group):
    name = name.split('/')[-1]
    if not os.path.isdir(f'{path_to_save_images}/{group}'):
        os.makedirs(f'{path_to_save_images}/{group}')
    np.save(f'{path_to_save_images}/{group}/{name}', array)
    print(f'Image was saved: {group}/{name}, shape: {array.shape}')

def upsampling(path_and_number, smallest_class):
    all_arrays = []
    for path, number in path_and_number.items():
        if smallest_class in path.split('/')[-2]:
            upsampled_array = []
            orig_image = np.load(path)
            for i in range(orig_image.shape[0]):
                #upsampled_array.append(orig_image[i])
                upsampled_array.append(np.rot90(orig_image[i]))
                upsampled_array.append(np.rot90(orig_image[i], 2))
                upsampled_array.append(np.rot90(orig_image[i], 3))
            save_image(np.array(upsampled_array), path, smallest_class)
            all_arrays.extend(upsampled_array)
    return np.array(all_arrays)

def prepare_for_upsampling_and_downsampling(class_and_number, path_and_number):
    smallest_class = (min(class_and_number, key=class_and_number.get))
    smallest_class_shape = upsampling(path_and_number, smallest_class)
    class_and_number[smallest_class] += smallest_class_shape.shape[0]
    other_small_classes = [other_small_classes for other_small_classes,
    number in class_and_number.items() if number<=class_and_number[smallest_class]
    and other_small_classes != smallest_class]
    for classes in other_small_classes:
        upsampled_class = upsampling(path_and_number, classes)
        class_and_number[classes] += upsampled_class.shape[0]
    smallest_class = (min(class_and_number, key=class_and_number.get))
    set_train_length = class_and_number[smallest_class]//2
    set_valid_length = int(class_and_number[smallest_class] * 0.2)
    return set_train_length, set_valid_length

def list_directories(path):
    list_of_directories = os.listdir(path)
    if '.DS_Store' in list_of_directories:
```

```
        list_of_directories.remove('.DS_Store')
    return list_of_directories

def extract_from_numpy(filename):
    set_of_patches = np.load(filename).shape[0]
    return set_of_patches

def extract_files(path):
    image_arrays = 0
    path_and_number = {}
    class_and_number = {}
    for wsi_folder in list_directories(path):
        sys.stdout.flush()
        for classes in list_directories(f'{path}/{wsi_folder}'):
            if classes in class_codes:
                for files_in_class in list_directories(f'{path}/{wsi_folder}/{classes}'):
                    number_of_patches = extract_from_numpy(f'{path}/{wsi_folder}/{classes}/
                    {files_in_class}')
                    image_arrays += number_of_patches
                    path_and_number[f'{path}/{wsi_folder}/{classes}/
                    {files_in_class}'] = number_of_patches
                    class_and_number[classes] = class_and_number.get(classes, 0)
                    + number_of_patches

    return image_arrays, path_and_number, class_and_number

image_arrays, path_and_number, class_and_number = extract_files(path)
train_length, valid_length =
prepare_for_upsampling_and_downsampling(class_and_number, path_and_number)
```

Python code for segmentation model's training and validation:

```
import sys
import os
import glob
import torch
from torch.utils.data import DataLoader, Dataset
import torch.nn.functional as F
from torchvision import transforms
import time
import numpy as np
import h5py
```

```
import gc

pref = 'torch_UNET_small'
suff = time.strftime('_%Y%m%d%H%M',time.localtime())

best_model_path = './seg_models/' + pref + suff + '.pt'
train_metrics_path = './seg_graph/' + pref + suff + '.h5'

out_chans = 5

def divisorGenerator(n):
    large_divisors = []
    for i in range(1, int((n**.5) + 1)):
        if n % i == 0:
            yield i
            if i * i != n:
                large_divisors.append(n / i)
    for divisor in reversed(large_divisors):
        yield divisor

class H5ImageDataset(Dataset):
    def __init__(self, path_to_msk_file, path_to_img_file,
    transform = None, target_transform = None):
        self.msk_dir = path_to_msk_file
        self.img_dir = path_to_img_file
        self.transform = transform
        self.target_transform = target_transform

    def __len__(self):
        with h5py.File(self.img_dir, 'r') as f:
            len_ = f['images'].shape[0]
        return len_

    def __getitem__(self, idx):
        with h5py.File(self.img_dir, 'r') as f:
            image = f['images'][idx]
        with h5py.File(self.msk_dir, 'r') as f:
            label = f['images'][idx]

        if self.transform:
            image = self.transform(image)
```

```
        if self.target_transform:
            label = self.target_transform(label)
        return image, label

class UNET(torch.nn.Module):
    def __init__(self, in_channels, out_channels):
        super().__init__()
        ch = 24 # 32
        self.conv1 = self.contract_block(in_channels, ch, 7, 3)
        self.conv2 = self.contract_block(ch, ch*2, 3, 1)
        self.conv3 = self.contract_block(ch*2, ch*4, 3, 1)

        self.upconv3 = self.expand_block(ch*4, ch*2, 3, 1)
        self.upconv2 = self.expand_block(ch*2 * 2, ch, 3, 1)
        self.upconv1 = self.expand_block(ch * 2, out_channels, 3, 1)

    def __call__(self, x):
        # downsampling part
        conv1 = self.conv1(x)
        conv2 = self.conv2(conv1)
        conv3 = self.conv3(conv2)

        upconv3 = self.upconv3(conv3)

        upconv2 = self.upconv2(torch.cat([upconv3, conv2], 1))
        upconv1 = self.upconv1(torch.cat([upconv2, conv1], 1))
        return upconv1

    def contract_block(self, in_channels, out_channels, kernel_size, padding):
        contract = torch.nn.Sequential(
            torch.nn.Conv2d(in_channels, out_channels,
            kernel_size = kernel_size, stride = 1, padding = padding),
            torch.nn.BatchNorm2d(out_channels),
            torch.nn.ReLU(),
            torch.nn.Conv2d(out_channels, out_channels,
            kernel_size = kernel_size, stride = 1, padding = padding),
            torch.nn.BatchNorm2d(out_channels),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size = 3, stride = 2, padding = 1))
        return contract
```

```
    def expand_block(self, in_channels, out_channels, kernel_size, padding):
        expand = torch.nn.Sequential(
            torch.nn.Conv2d(in_channels, out_channels, kernel_size,
            stride = 1, padding = padding),
            torch.nn.BatchNorm2d(out_channels),
            torch.nn.ReLU(),
            torch.nn.Conv2d(out_channels, out_channels, kernel_size,
            stride = 1, padding = padding),
            torch.nn.BatchNorm2d(out_channels),
            torch.nn.ReLU(),
            torch.nn.ConvTranspose2d(out_channels, out_channels,
            kernel_size = 3, stride = 2, padding = 1, output_padding = 1))
        return expand


def train(model, train_dl, valid_dl, loss_fn, optimizer, acc_fn, epochs = 1, patience = 1):
    patience_,patience = patience,patience
    model.cuda()
    train_loss, valid_loss = [], []
    train_acc, valid_acc = [], []
    best_ep = 0
    best_acc = 0.0
    best_valid_loss = np.Inf
    valid_loss.append(best_valid_loss)
    valid_acc.append(best_acc)

    for epoch in range(1,epochs+1,1):
        for phase in ['train', 'valid']:
            if phase == 'train':
                model.train(True)  # Set trainind mode = true
                dataloader = train_dl
                optimizer.zero_grad()
            else:
                model.train(False)  # Set model to evaluate mode
                dataloader = valid_dl

            running_loss = 0.0
            running_acc = 0.0
            step = 0
            for x, y in dataloader:
                step += 1
```

```
    x = x.to(device = device,dtype = torch.float32)
    y = y.to(device = device,dtype = torch.float32)
    # forward pass
    if phase == 'train':
        outputs = model(x)
        loss = loss_fn(outputs, y)
        loss.backward()
        if step % 500 == 0:
            sys.stdout.flush()
            optimizer.step()
            optimizer.zero_grad()
            gc.collect()
            torch.cuda.empty_cache()
    else:
        with torch.no_grad():
            outputs = model(x)
            loss = loss_fn(outputs, y)

    acc = acc_fn(outputs, y)

    running_acc  += acc * dataloader.batch_size
    running_loss += loss * dataloader.batch_size

epoch_loss = running_loss / len(dataloader.dataset)
epoch_acc = running_acc / len(dataloader.dataset)
sys.stdout.flush()
train_loss.append(epoch_loss.detach().cpu().resolve_conj().
resolve_neg().numpy()) \
if phase == 'train' else
valid_loss.append(epoch_loss.detach().cpu().resolve_conj()
.resolve_neg().numpy())
train_acc.append(epoch_acc.detach().cpu().resolve_conj().
resolve_neg().numpy()) \
if phase == 'train' else
valid_acc.append(epoch_acc.detach().cpu().resolve_conj().
resolve_neg().numpy())
if (phase == 'valid') & (best_valid_loss > epoch_loss):
    best_valid_loss = epoch_loss
    best_acc = valid_acc[-1]
    best_ep = epoch
    patience = patience_
```

```
                checkpoint = {'epoch' : epoch,
                    'valid_loss_min' : best_valid_loss,
                    'state_dict' : model.state_dict(),
                    'optimizer' : optimizer.state_dict()}
                #save checkpoint
                torch.save(checkpoint, best_model_path)
                sys.stdout.flush()
            elif (phase == 'valid') & (best_valid_loss <= epoch_loss):
                patience -= 1
        sys.stdout.flush()
        if patience < 1:
            break
    return train_loss, valid_loss, train_acc, valid_acc

def acc_metric(predb, yb):
    return (predb.argmax(dim = 1) == yb.cuda()).float().mean()

class DiceBCELoss(torch.nn.Module):
    def __init__(self, weight = None, size_average = True):
        super(DiceBCELoss, self).__init__()
    def forward(self, inputs, targets, smooth = 1):
        inputs = torch.sigmoid(inputs)
        inputs = inputs.view(-1)
        targets = targets.view(-1)
        intersection = (inputs * targets).sum()
        dice_loss = 1 - (2. * intersection + smooth)/(inputs.sum() + targets.sum() + smooth)
        BCE = F.binary_cross_entropy(inputs, targets, reduction='mean')
        Dice_BCE = BCE + dice_loss
        return Dice_BCE

class Iou(torch.nn.Module):
    def __init__(self, weight = None, size_average = True):
        super(Iou, self).__init__()
    def forward(self, inputs, targets, smooth = 1):
        inputs = torch.sigmoid(inputs)
        inputs = inputs.view(-1)
        targets = targets.view(-1)
        intersection = (inputs * targets).sum()
        total = (inputs + targets).sum()
        union = total - intersection
        IoU = (intersection + smooth)/(union + smooth)
```

```
        return IoU

model = UNET(3,out_chans)
trans = transforms.Compose([transforms.ToTensor()])

training_data = H5ImageDataset('../PanNuke/fold1_2_masks.h5',
                '../PanNuke/fold1_2_images.h5', trans, trans,)
validation_data = H5ImageDataset('../PanNuke/Fold1/masks/fold3/masks.h5',
                  '../PanNuke/Fold1/images/fold3/images.h5', trans, trans,)

train_bs = 1
train_dl = DataLoader(training_data, batch_size = train_bs,
shuffle = True, num_workers = 4)
valid_dl = DataLoader(validation_data, batch_size = valid_bs,
shuffle = True, num_workers = 4)

loss_fn = DiceBCELoss()
acc_fn = Iou()
opt = torch.optim.Adam(model.parameters(),lr = 0.0001)

train_loss, valid_loss, train_acc, valid_acc = train(model, train_dl, valid_dl,
                           loss_fn, opt, acc_fn, epochs = 5000, patience = 50)

with h5py.File(train_metrics_path, 'w') as f:
    f.create_dataset('train_loss', data = np.array(train_loss))
    f.create_dataset('train_acc', data = np.array(train_acc))
    f.create_dataset('valid_loss', data = np.array(valid_loss))
    f.create_dataset('valid_acc', data = np.array(valid_acc))
```

Python code for segmentation model's testing:

```
import sys
import os
import glob
import torch
from torch.utils.data import DataLoader, Dataset
import torch.nn.functional as F
from torchvision import transforms
import numpy as np
from dataset import CustomImageDataset

out_chans = 2
```

```
class UNET(torch.nn.Module):
    def __init__(self, in_channels, out_channels):
        super().__init__()
        self.conv1 = self.contract_block(in_channels, 32, 7, 3)
        self.conv2 = self.contract_block(32, 64, 3, 1)
        self.conv3 = self.contract_block(64, 128, 3, 1)
        self.upconv3 = self.expand_block(128, 64, 3, 1)
        self.upconv2 = self.expand_block(64 * 2, 32, 3, 1)
        self.upconv1 = self.expand_block(32 * 2, out_channels, 3, 1)

    def __call__(self, x):
        # downsampling part
        conv1 = self.conv1(x)
        conv2 = self.conv2(conv1)
        conv3 = self.conv3(conv2)
        upconv3 = self.upconv3(conv3)
        upconv2 = self.upconv2(torch.cat([upconv3, conv2], 1))
        upconv1 = self.upconv1(torch.cat([upconv2, conv1], 1))
        return upconv1

    def contract_block(self, in_channels, out_channels, kernel_size, padding):
        contract = torch.nn.Sequential(
            torch.nn.Conv2d(in_channels, out_channels,
            kernel_size = kernel_size, stride = 1, padding = padding),
            torch.nn.BatchNorm2d(out_channels),
            torch.nn.ReLU(),
            torch.nn.Conv2d(out_channels, out_channels,
            kernel_size = kernel_size, stride = 1, padding = padding),
            torch.nn.BatchNorm2d(out_channels),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size = 3, stride = 2, padding = 1))
        return contract

    def expand_block(self, in_channels, out_channels, kernel_size, padding):
        expand = torch.nn.Sequential(
            torch.nn.Conv2d(in_channels, out_channels,
            kernel_size, stride = 1, padding = padding),
            torch.nn.BatchNorm2d(out_channels),
            torch.nn.ReLU(),
            torch.nn.Conv2d(out_channels, out_channels,
```

```
            kernel_size, stride = 1, padding = padding),
            torch.nn.BatchNorm2d(out_channels),
            torch.nn.ReLU(),
            torch.nn.ConvTranspose2d(out_channels, out_channels,
            kernel_size = 3, stride = 2, padding = 1, output_padding = 1))
        return expand

model = UNET(3,out_chans)
model.load_state_dict(torch.load(model_file)['state_dict'])
model.to(device)
model.eval()

test_data = CustomImageDataset(cfg.root_pth, cfg.clss_dict,
cfg.augmented_root_pth, json_pth=test_index_file)
test_bs = 1
test_dl = DataLoader(test_data, batch_size = test_bs, shuffle = False)

model_file = '../seg_models/torch_UNET_small_202311081855.pt'
out_file = f"../seg_rez/{model_file.split('/')[-1].split('.')[0]}.h5"

preds = []
for x, y in test_dl:
    x = x.to(device = device,dtype = torch.float)
    y = y.to(device = device,dtype = torch.float)
    with torch.no_grad():
        outputs = model(x)
        outputs = torch.sigmoid(outputs)
        preds.append(outputs.detach().cpu().resolve_conj().resolve_neg().numpy())

preds = np.concatenate(preds, axis=0)

with h5py.File(out_file, 'w') as f:
    f.create_dataset('preds', data = preds)
```

Python code for obtaining outputs from Integrated Gradients and NoiseTunnel methods:

```
import torch
import numpy as np
import torch.nn as nn
import torch.nn.functional as F
from cnn_train_test import CNN
from torch.utils.data import DataLoader
```

```
from captum.attr import IntegratedGradients
from captum.attr import NoiseTunnel

from dataset import CustomImageDataset
import config as cfg

class_codes = {'A': 0, 'N': 1, 'R1': 2, 'R2': 3, 'R3': 4, 'R4': 5, 'R5': 6}

batch_size = 1
test_index_file = '../index_list/test_indexes.txt'

test_dataset = CustomImageDataset(cfg.root_pth, cfg.clss_dict,
cfg.augmented_root_pth, json_pth=test_index_file)
test_dataloader = DataLoader(test_dataset,batch_size=batch_size, shuffle=False)

model = CNN()
model.out = nn.Linear(in_features=64, out_features=7)
criterion = nn.CrossEntropyLoss()

model.load_state_dict(torch.load('../saved_models/cnn_20231026_balanced.pth'))
model.eval()

with torch.no_grad():
    for img_num, (test_images, test_labels) in enumerate(test_dataloader):
        outfname = f'../saved_imgs/{img_num:05d}_ig_CNN.npy'
        output = model(test_images)
        test_loss += criterion(output, test_labels.long()).item() *test_images.size(0)
        #IG heatmap extraction
        output = F.softmax(output, dim=1)

        true_lbl = ''.join([k for k, v in class_codes.items()
        if v == (test_labels[0]).numpy()])
        pred_score, pred_label_idx = torch.topk(output, 1)
        pred_label_idx.squeeze_()
        predicted_label = next((k for k, v in class_codes.items()
        if v == pred_label_idx), None)

        #Create IntegratedGradients object and get attributes
        integrated_gradients = IntegratedGradients(model)
```

```
        #Request the algorithm to assign our output target to
        attributions_ig = integrated_gradients.attribute(test_images,
        target=pred_label_idx, n_steps=100)
        ig_imgs = attributions_ig.squeeze().cpu().detach().numpy()
        np.save(outfname, ig_imgs)

        #Create NoiseTunnel output
        outfname = f'../saved_imgs/{img_num:05d}_nt_CNN.npy'
        noise_tunnel = NoiseTunnel(integrated_gradients)
        attributions_ig_nt = noise_tunnel.attribute(test_images, nt_samples=5,
        nt_type='smoothgrad_sq', target=pred_label_idx)
        nt_imgs = attributions_ig_nt.squeeze().cpu().detach().numpy()
        np.save(outfname, nt_imgs)
```

Python code for obtaining outputs from LIME method:

```
import torch
import numpy as np
import torch.nn as nn
import torch.nn.functional as F
from cnn_train_test import CNN
from torch.utils.data import DataLoader
from dataset import CustomImageDataset
import config as cfg
from lime import lime_image

DEVICE = "cuda" if torch.cuda.is_available() else "cpu"
class_codes = {'A': 0, 'N': 1, 'R1': 2, 'R2': 3, 'R3': 4, 'R4': 5, 'R5': 6}

batch_size = 1
test_index_file = '../index_list/test_indexes.txt'
output_path = '../index_list'
LIME_output = 'LIME_output_CNN.txt'

test_dataset = CustomImageDataset(cfg.root_pth, cfg.clss_dict,
cfg.augmented_root_pth, json_pth=test_index_file)

model = CNN()
model.out = nn.Linear(in_features=64, out_features=7)
criterion = nn.CrossEntropyLoss()
model.load_state_dict(torch.load('../saved_models/cnn_20231026_balanced.pth'))
model.eval()
```

```
test_dataloader = DataLoader(test_dataset,batch_size=batch_size, shuffle=False)
explainer = lime_image.LimeImageExplainer()

def batch_predict(test_images):
    test_images = torch.permute((torch.from_numpy(test_images)),(0, 3, 1, 2))
    with torch.no_grad():
        device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        model.to(device)
        test_images = test_images.to(device)
        output = model(test_images)
        probs = F.softmax(output, dim=1)

    return probs.detach().cpu().numpy()

for img_num, (test_images, test_labels) in enumerate(test_data
    outfname = f'../saved_imgs/{img_num:05d}_LIME_CNN.npy'
    explanation = explainer.explain_instance(test_images.squeeze(0).permute(1, 2, 0).numpy(),
    batch_predict, labels=test_labels, num_samples=500, top_labels=1,)
    temp, mask = explanation.get_image_and_mask(explanation.top_labels[0],
    positive_only=False, num_features=3, hide_rest=False)
    mask = abs(mask)
    mask = (mask-np.min(mask))/(np.max(mask)-np.min(mask))

    if not os.path.isdir(output_path):
        os.makedirs(output_path)
    np.save(outfname, mask)
```

Python code for obtaining outputs from GradCAM method:

```
import torch
import numpy as np
import torch.nn as nn
import torch.nn.functional as F
from cnn_train_test import CNN
from torch.utils.data import DataLoader
from dataset import CustomImageDataset
import config as cfg

from pytorch_grad_cam import GradCAM

DEVICE = "cuda" if torch.cuda.is_available() else "cpu"
```

```
class_codes = {'A': 0, 'N': 1, 'R1': 2, 'R2': 3, 'R3': 4, 'R4': 5, 'R5': 6}

batch_size = 1
test_index_file = '../index_list/test_indexes.txt'
output_path = '../index_list'

test_dataset = CustomImageDataset(cfg.root_pth, cfg.clss_dict, cfg.augmented_root_pth,
json_pth=test_index_file)
test_dataloader = DataLoader(test_dataset,batch_size=batch_size, shuffle=False)

model = CNN()
model.out = nn.Linear(in_features=64, out_features=7)
criterion = nn.CrossEntropyLoss()

model.load_state_dict(torch.load('../saved_models/cnn_20231026_balanced.pth'))
model.to(DEVICE)
model.eval()

target_layer = [model.conv3]
cam = GradCAM(model=model, target_layers=target_layer)

for img_num, (test_images, test_labels) in enumerate(test_dataloader):
    outfname = f'{output_path}/{img_num:05d}_GradCAM_CNN.npy'
    sys.stdout.flush()
    test_images = test_images.to(DEVICE)
    output = cam(input_tensor=test_images)
    output = output[0, :]
    heatmap = abs(output)

    if not os.path.isdir(output_path):
        os.makedirs(output_path)
        print(f'Path: {output_path} was created')
    np.save(outfname, heatmap)
```

Python code for IoU metric's calculation:

```
import glob
import h5py
import numpy as np
from matplotlib import pyplot as plt
from torch import nn
```

```
import torch
import json

test_index_file = '../DiagSet/index_list/short_test_indexes.txt'
segmentation_file = h5py.File(f'../seg_rez_torch_UNET_small_202311081855.h5')
GradCAM_outputs = glob.glob(f'../gradcam_rez/GRADCAM_*.npy')
IG_outputs = glob.glob(f'..}/igrad_rez/IG_*.npy')
NT_outputs =  glob.glob(f'../igrad_rez/NT_*.npy')
LIME_outputs = glob.glob(f'../lime_rez/LIME_*.npy')
seg_treshold = 0.5
intrp_treshold = 0.3

class Iou(nn.Module): #intersection over union
    def __init__(self, weight=None, size_average=True):
        super(Iou, self).__init__()

    def forward(self, inputs, targets, smooth=1):
        inputs = torch.sigmoid(inputs)
        inputs = inputs.reshape(-1)
        targets = targets.reshape(-1)
        intersection = (inputs * targets).sum()
        total = (inputs + targets).sum()
        union = total - intersection
        IoU = (intersection + smooth)/(union + smooth)
        return IoU

def process_img(img):
    IMG = []
    img = abs(img)
    for i in range(3):
        IMG.append((img[i,:,:]-np.min(img[i,:,:]))/(np.max(img[i,:,:])-np.min(img[i,:,:])))
    IMG = np.sum(np.dstack(IMG),axis=2)
    return((IMG-np.min(IMG))/(np.max(IMG)-np.min(IMG)))

def return_tensor(npy_image, treshold):
    npy_image = (npy_image-np.min(npy_image))/(np.max(npy_image)-np.min(npy_image))
    image = torch.tensor(np.expand_dims(npy_image, -1),
                             dtype=torch.float)
    image = torch.permute(image, (2, 0, 1))
    binary_image = torch.where(image >= treshold, torch.tensor(1.0), torch.tensor(0.0))
    return binary_image
```

```
def calculate_acc(map, seg_img_0, seg_img_1):
    class_0 = accuracy_metric(map, seg_img_0).item()
    class_1 = accuracy_metric(map, seg_img_1).item()
    return (class_0, class_1)

accuracy_metric = Iou()

with open(test_index_file,'r') as f:
    test_d = json.load(f)

seg_id = 0
passed = 0

IoU_IG = []
IoU_NT = []
IoU_LIME = []
IoU_GC = []

for n in range(len(IG_outputs)):
    load_array_IG = np.load(IG_outputs[n])
    load_array_NT = np.load(NT_outputs[n])
    load_array_LIME = np.load(LIME_outputs[n])
    load_array_GC = np.load(GradCAM_outputs[n])
    for id in range(load_array_IG.shape[0]):
        IG_img = process_img(load_array_IG[id])
        NT_img = process_img(load_array_NT[id])
        ig_img = return_tensor(np.rot90(np.flip(IG_img, 1), 1), intrp_treshold)
        nt_img = return_tensor(np.rot90(np.flip(NT_img, 1), 1), intrp_treshold)
        seg_img_0 = return_tensor(np.rot90(np.flip(segmentation_file['preds']
        [seg_id][0], 1),1), seg_treshold)
        seg_img_1 = return_tensor(np.rot90(np.flip(segmentation_file['preds']
        [seg_id][1], 1), 1), seg_treshold)
        lime_img = return_tensor(np.rot90(np.flip(abs(load_array_LIME[id]), 1), 1),
        intrp_treshold)
        gc_img = return_tensor(np.rot90(np.flip(abs(load_array_GC[id]), 1), 1),
        intrp_treshold)
        seg_id += 1
        ig_scores = calculate_acc(ig_img, seg_img_0, seg_img_1)
        nt_scores = calculate_acc(nt_img, seg_img_0, seg_img_1)
        lime_scores = calculate_acc(lime_img, seg_img_0, seg_img_1)
```

```
        gc_scores = calculate_acc(gc_img, seg_img_0, seg_img_1)
        IoU_IG.append(ig_scores)
        IoU_NT.append(nt_scores)
        IoU_LIME.append(lime_scores)
        IoU_GC.append(gc_scores)
```

Python code for precision, recall and F1-score calculation:

```
from sklearn.metrics import precision_score, recall_score, f1_score
def prep_numpy(npy_image, treshold):
    image = np.rot90(np.flip(npy_image, 1), 1)
    binary_image = (image > treshold).astype(int)
    return binary_image

def metrics(binary_image, masks, treshold):
    precisions = ()
    recalls = ()
    f1s = ()
    for idx in range(len(masks)):
        mask = prep_numpy(masks[idx], treshold)
        precisions += (precision_score(mask, binary_image, average='macro'),)
        recalls += (recall_score(mask, binary_image, average='macro'),)
        f1s += (f1_score(mask, binary_image, average='macro'),) #DICE
    return precisions, recalls, f1s
seg_id = 0
passed = 0

precision_IG = []
precision_NT = []
precision_LIME = []
precision_GC = []

recall_IG= []
recall_NT = []
recall_LIME = []
recall_GC = []

f1_score_IG = []
f1_score_NT = []
f1_score_LIME = []
f1_score_GC = []
```

```
for n in range(len(IG_outputs)):
    load_array_IG = np.load(IG_outputs[n])
    load_array_NT = np.load(NT_outputs[n])
    load_array_LIME = np.load(LIME_outputs[n])
    load_array_GC = np.load(GradCAM_outputs[n])

    for id in range(load_array_IG.shape[0]):

        IG_img = process_img(load_array_IG[id])
        NT_img = process_img(load_array_NT[id])

        ig_img = prep_numpy(IG_img, intrp_treshold)
        nt_img = prep_numpy(NT_img, intrp_treshold)

        lime_img = prep_numpy(abs(load_array_LIME[id]), intrp_treshold)
        gc_img = prep_numpy(abs(load_array_GC[id]), intrp_treshold)

        lime_img = np.nan_to_num((lime_img-np.min(lime_img))
        /(np.max(lime_img)-np.min(lime_img)))
        gc_img = np.nan_to_num((gc_img-np.min(gc_img))/(np.max(gc_img)-np.min(gc_img)))

        ig_precision, ig_recall, ig_f1 = metrics(ig_img, segmentation_file['preds'][seg_id],
        seg_treshold)
        nt_precision, nt_recall, nt_f1 = metrics(nt_img, segmentation_file['preds'][seg_id],
        seg_treshold)
        lime_precision, lime_recall, lime_f1 = metrics(lime_img,
        segmentation_file['preds'][seg_id], seg_treshold)
        gc_precision, gc_recall, gc_f1 = metrics(gc_img,
        segmentation_file['preds'][seg_id], seg_treshold)

        seg_id += 1

        precision_IG.append(ig_precision)
        precision_NT.append(nt_precision)
        precision_LIME.append(lime_precision)
        precision_GC.append(gc_precision)

        recall_IG.append(ig_recall)
        recall_NT.append(nt_recall)
        recall_LIME.append(lime_recall)
```

```
        recall_GC.append(gc_recall)

        f1_score_IG.append(ig_f1)
        f1_score_NT.append(nt_f1)
        f1_score_LIME.append(lime_f1)
        f1_score_GC.append(gc_f1)
```

Python code for Spearman's correlation coefficient metric's:

```
from scipy.stats import spearmanr

def prep_flatten_numpy(npy_image, treshold):
    image = np.rot90(np.flip(npy_image, 1), 1)
    binary_image = (image > treshold).astype(int)
    flat_image = binary_image.flatten()
    return flat_image

def correlation_coef(flat_image, masks, treshold):
    correlations = ()
    for idx in range(len(masks)):
        mask = prep_flatten_numpy(masks[idx], treshold)
        correlation_coefficient, _ = spearmanr(flat_image, mask)
        correlations += (correlation_coefficient, )
    return correlations

seg_id = 0
passed = 0

correlation_IG = []
correlation_NT = []
correlation_LIME = []
correlation_GC = []

for n in range(len(IG_outputs)):
    load_array_IG = np.load(IG_outputs[n])
    load_array_NT = np.load(NT_outputs[n])
    load_array_LIME = np.load(LIME_outputs[n])
    load_array_GC = np.load(GradCAM_outputs[n])

    for id in range(load_array_IG.shape[0]):

        IG_img = process_img(load_array_IG[id])
```

```
        NT_img = process_img(load_array_NT[id])

        ig_img = prep_flatten_numpy(IG_img, intrp_treshold)
        nt_img = prep_flatten_numpy(NT_img, intrp_treshold)

        lime_img = prep_flatten_numpy(abs(load_array_LIME[id]), intrp_treshold)
        gc_img = prep_flatten_numpy(abs(load_array_GC[id]), intrp_treshold)

        lime_img = (lime_img-np.min(lime_img))/(np.max(lime_img)-np.min(lime_img))
        gc_img = (gc_img-np.min(gc_img))/(np.max(gc_img)-np.min(gc_img))

        ig_correlation = correlation_coef(ig_img, segmentation_file['preds'][seg_id],
        seg_treshold)
        nt_correlation = correlation_coef(nt_img, segmentation_file['preds'][seg_id],
        seg_treshold)
        lime_correlation = correlation_coef(lime_img, segmentation_file['preds'][seg_id],
        seg_treshold)
        gc_correlation = correlation_coef(gc_img, segmentation_file['preds'][seg_id],
        seg_treshold)

        correlation_IG.append(ig_correlation)
        correlation_NT.append(nt_correlation)
        correlation_LIME.append(lime_correlation)
        correlation_GC.append(gc_correlation)

        seg_id += 1
```

Python code for pixel accuracy metrics:

```
def calculate_pixel_accuracy(binary_image, masks, treshold):
    pixel_accuracy = ()
    for idx in range(len(masks)):
        mask = prep_numpy(masks[idx], treshold)
        correct_pixels = np.sum(mask == binary_image)
        total_pixels = mask.size
        pixel_accuracy += ((correct_pixels/total_pixels),)
    return pixel_accuracy
seg_id = 0
passed = 0

pixel_accuracy_IG = []
pixel_accuracy_NT = []
pixel_accuracy_LIME = []
```

```
pixel_accuracy_GC = []


for n in range(len(IG_outputs)):
    load_array_IG = np.load(IG_outputs[n])
    load_array_NT = np.load(NT_outputs[n])
    load_array_LIME = np.load(LIME_outputs[n])
    load_array_GC = np.load(GradCAM_outputs[n])

    for id in range(load_array_IG.shape[0]):

        IG_img = process_img(load_array_IG[id])
        NT_img = process_img(load_array_NT[id])

        ig_img = prep_numpy(IG_img, intrp_treshold)
        nt_img = prep_numpy(NT_img, intrp_treshold)

        lime_img = prep_numpy(abs(load_array_LIME[id]), intrp_treshold)
        gc_img = prep_numpy(abs(load_array_GC[id]), intrp_treshold)

        lime_img = (lime_img-np.min(lime_img))/(np.max(lime_img)-np.min(lime_img))
        gc_img = (gc_img-np.min(gc_img))/(np.max(gc_img)-np.min(gc_img))

        ig_pixel_accuracy = calculate_pixel_accuracy(ig_img,
        segmentation_file['preds'][seg_id], seg_treshold)
        nt_pixel_accuracy = calculate_pixel_accuracy(nt_img,
        segmentation_file['preds'][seg_id], seg_treshold)
        lime_pixel_accuracy = calculate_pixel_accuracy(lime_img,
        segmentation_file['preds'][seg_id], seg_treshold)
        gc_pixel_accuracy = calculate_pixel_accuracy(gc_img,
        segmentation_file['preds'][seg_id], seg_treshold)

        seg_id += 1

        pixel_accuracy_IG.append(ig_pixel_accuracy)
        pixel_accuracy_NT.append(nt_pixel_accuracy)
        pixel_accuracy_LIME.append(lime_pixel_accuracy)
        pixel_accuracy_GC.append(gc_pixel_accuracy)
```