ELEC4848 Senior Design Project 2023-2024

Tian Qijia 3035772729

Explainable Machine Learning Algorithm and Hardware for Surgical Robotics

Supervisor: Dr. C. Li
Second Examiner: Dr. Z. Wang

Abstract

Deep learning techniques have shown promise in accurate segmentation tasks in medical imaging, particularly in breast cancer diagnosis. This project endeavors to develop an AI-driven segmentation approach tailored for breast tumor delineation, focusing on efficiency and interpretability. The primary objective is to design an Attention UNet model capable of achieving high segmentation accuracy while addressing computational constraints commonly encountered in medical imaging tasks. The methodology involves extensive experimentation, including data preprocessing, model training, and evaluation, culminating in robust segmentation performance metrics. The results reveal the model's high and stable accuracy in segmenting images, though the Intersection over Union and Dice coefficients in the evaluation metrics indicate some underperformance. This highlights the imperative for continued research aimed at not only improving the model's interpretability but also expanding its utility across a variety of medical imaging modalities. Recommendations encompass acquiring more extensive annotated datasets and delving into sophisticated architectural enhancements to bolster the model's robustness and viability for deployment in real-world scenarios. Additionally, it is advised to swiftly design and implement high-performance, low-power hardware solutions to complement these advancements.

Acknowledgment

I extend my sincere appreciation to Dr. Can Li for their exceptional guidance, support, and mentorship throughout this project. His profound expertise, unwavering encouragement, and invaluable feedback have been pivotal in shaping the trajectory of my research and fostering my academic development.

Furthermore, I express my gratitude to Dr. Can Li 's esteemed doctoral student, Bo Wen, for his valuable assistance, insightful contributions, and collaborative spirit. His dedication and expertise have significantly enriched my understanding of the subject matter and enhanced the quality of my work.

Table of Contents

List of Figures

List of Abbreviations and Symbols

Abbreviation	Meaning
IoU	Intersection over Union
ReLU	Rectified Linear Unit
GradCAM	Gradient-weighted Class Activation Mapping
LIME	Local Interpretable Model-Agnostic Explanations
CNN	Convolutional Neural Network
DNN	Deep Neural Network

1. Introduction

Medical image segmentation, particularly in breast cancer diagnosis, has seen significant advancements with the emergence of deep learning techniques. Developing convolutional neural networks (CNNs) tailored for biomedical image segmentation, such as the UNet architecture introduced by Ronneberger et al. in 2015, has revolutionized the field [1]. UNet's elegant design, combining convolutional and downsampling layers with skip connections for upsampling, has proven to accurately delineate structures of interest within medical images, including breast tumors [1].

Furthermore, recent studies, such as the work by Singh et al., have underscored the efficacy of UNet-based models in various segmentation tasks, including breast tumor segmentation in ultrasound images [2]. These studies highlighted UNet's robust performance in terms of accuracy, Intersection over Union (IoU), and sensitivity, positioning it as a leading approach in medical image segmentation [2].

The discussion surrounding UNet emphasized its ability to achieve sound localization and contextual understanding, even with limited training data, thereby facilitating precise segmentations [1]. This unique characteristic of UNet addresses a critical need in medical imaging, where accurate delineation of pathological regions is essential for diagnosis and treatment planning.

While Transformer-based models have gained popularity in recent years, UNet and its variants remain the cornerstone of image segmentation in the medical field. Despite their widespread adoption, challenges persist in medical image segmentation, particularly regarding efficiency and interpretability.

This project aims to develop an AI-driven image segmentation approach to delineate breast tumors while providing interpretable segmentation maps efficiently. This approach aims to reduce the time and effort required for medical professionals to analyze medical images, thereby alleviating their workload and improving diagnostic accuracy.

The importance of this work lies in its potential to alleviate the burden on medical professionals by providing a fast and accurate method for breast tumor segmentation. By automating this process and providing interpretable results, the proposed approach can assist clinicians in making more informed decisions, ultimately improving patient care and outcomes.

This project focuses specifically on segmenting breast tumor regions from 2D ultrasound images, categorizing them into normal and pathological regions. The project's scope also encompasses working with a relatively low quantity of image data and limited computational resources, such as those available in environments like Google Colab.

The primary deliverable of this research is the Attention UNet model, which aims to achieve high segmentation accuracy while providing interpretable segmentation maps through mechanisms such as GradCAM and attention activation. In addition to developing the model architecture, this project also includes the implementation of evaluation metrics such as Intersection over Union (IoU) and Dice coefficient to measure the model's effectiveness and performance in accurately segmenting medical images. These deliverables collectively contribute to advancing the field of medical image segmentation and hold promise for improved diagnostic accuracy and patient care.

This report is organized into nine sections, each contributing a critical component to the comprehensive study of an AI-driven approach for breast tumor delineation using Attention UNet. Section 1 introduces the research topic, delineating the objectives and highlighting the significance of this study within the medical imaging domain. Section 2 delves into the UNet architecture and attention mechanisms in deep learning, laying down the theoretical groundwork essential for understanding the innovative approach undertaken. Section 3 outlines the methodology adopted for developing and evaluating the Attention UNet model, detailing the steps from data preprocessing and model training to the selection of evaluation metrics. Section 4 exhibits the results of the experiments, including segmentation performance metrics, qualitative evaluations, and visual representations of the segmentation outcomes. Section 5 interprets these results, contrasting the proposed model's efficacy and limitations against existing methodologies, thereby shedding light on its practical applicability and innovation. Section 6 focuses on the hardware design and specifications, emphasizing the synergy between algorithmic efficiency and hardware performance. Section 7 scrutinizes the limitations of the model, offering a transparent examination of areas requiring further research and improvement. Section 8 discusses future directions, proposing pathways for enhancing the model's accuracy, interpretability, and applicability across various medical imaging modalities. Finally, Section 9 concludes the report, summarizing the key findings, contributions, and the envisioned impact of this research on the field of medical imaging.

2. Theoretical Background

In medical image analysis, the choice of dataset is pivotal for developing robust models capable of accurate segmentation. Alongside dataset considerations, the model's design, evaluation metrics, and model interpretability are crucial aspects that contribute to the overall efficacy and trustworthiness of the segmentation framework.

2.1 Choice of Dataset

The model utilizes a well-known dataset, the Breast Ultrasound Images Dataset, sourced from Kaggle [3]. This dataset comprises images of breast ultrasounds. Noteworthy features of this dataset that led to its selection include its richness in data, sourced from females aged between 25 and 75 years, totaling 780 images. Additionally, the dataset provides explicit annotations, offering both original and annotated versions with ground truth labels. In the annotated versions, tumor regions are delineated (with white representing the tumor area and black representing the background), aiding in the model's training.

Furthermore, the dataset includes textual annotations categorizing images into three classes: normal, benign, and malignant. This comprehensive labeling scheme facilitates supervised learning and model evaluation. Notably, the dataset is easily accessible and available under the CC0 license, allowing unrestricted usage.

2.2 Model Design

The UNet architecture, proposed by Ronneberger et al. [1], is selected as the primary framework for this project. Introduced in 2015, UNet is designed explicitly for biomedical image segmentation tasks, demonstrating remarkable effectiveness even with limited training data.

2.2.1 UNet Structure

UNet is structured as a convolutional neural network consisting of a contracting path followed by an expansive path. The architecture is depicted in Figure 1 and described below.

Input to Output Process

(1) **Input Layer:** The process begins with an input image tile, typically a slice of a biomedical image, which undergoes convolutional operations.
(2) **Downsampling (Contraction Path):** Two consecutive 3x3 convolutions with ReLU activation are applied, followed by max-pooling to reduce spatial dimensions. This downsampling process progressively reduces the spatial dimensions while increasing the number of feature channels.
(3) **Bottleneck:** The lowest resolution is reached at the bottleneck, transitioning from the contraction to the expansion path.
(4) **Upsampling (Expansion Path):** Up-convolution operations are performed to increase spatial dimensions. Concatenation with corresponding feature maps from the contracting path enables precise localization. Convolutional layers with ReLU activation further process the feature maps.
(5) **Output Layer：** A 1x1 convolutional layer is applied to map the feature vector to the desired number of classes for segmentation.

The paragraphs below are a detailed description of the UNet Segmentation Model.

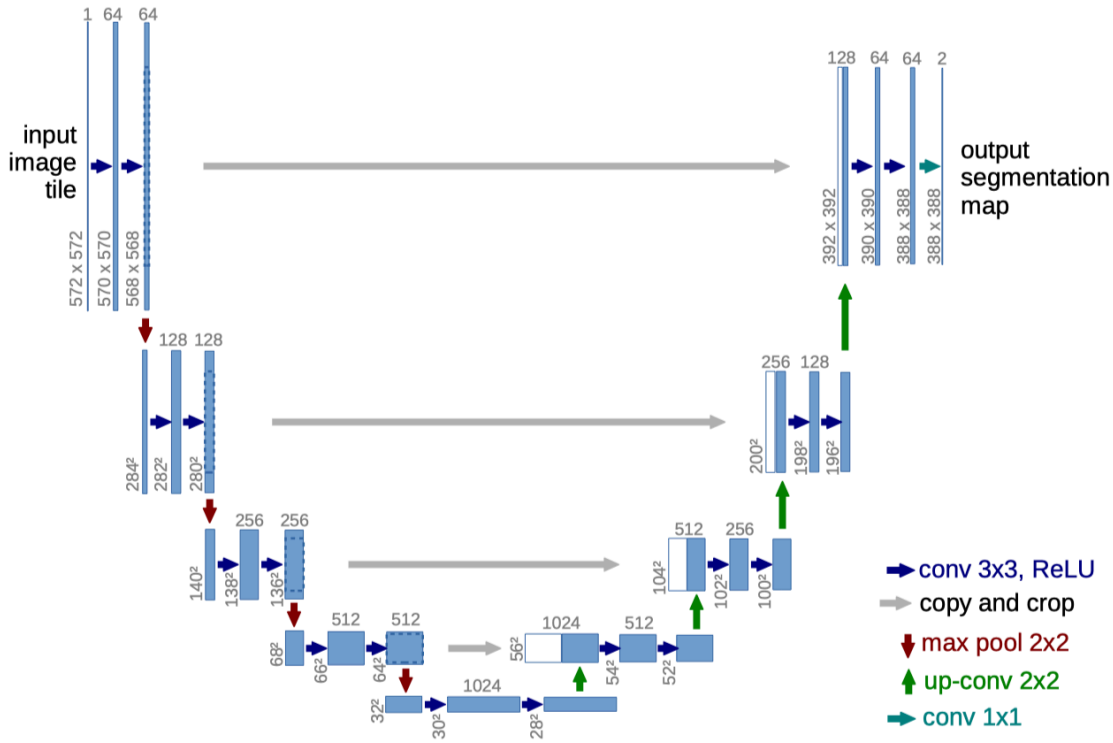

Figure 1. UNet Architecture [1]

The UNet architecture is a convolutional neural network designed specifically for biomedical image segmentation tasks, where the goal is to delineate different structures or regions within medical images accurately. It consists of two main components: a contracting path and an expansive path, which are responsible for capturing contextual information and performing precise localization, respectively.

The contracting path begins with an input layer, where the input image tile is processed through a series of convolutional layers. These convolutional layers, each followed by a ReLU activation function, help extract features from the input image. Subsequently, max-pooling operations are applied to reduce the spatial dimensions of the feature maps, effectively downsampling the input.

As the contracting path progresses, the spatial dimensions decrease while the number of feature channels increases. This process allows the network to capture increasingly abstract and high-level features from the input image.

At the network bottleneck, the lowest resolution is reached, marking the transition to the expansive path. In the expansive path, up-convolution operations are performed to increase the spatial dimensions of the feature maps. These up-convolutions are complemented by concatenation with corresponding feature maps from the contracting path, facilitating precise localization of features.

Finally, the output layer consists of a 1x1 convolutional layer, which maps the feature vector to the desired number of classes for segmentation. This layer produces the segmentation mask, where each pixel is classified into one of the target classes based on the features extracted by the network.

In summary, the UNet architecture combines the benefits of both contracting and expansive paths to achieve accurate and detailed segmentation of biomedical images, making it well-suited for various medical image analysis tasks.

2.2.2 Integration of Attention Mechanism

An attention mechanism is integrated into the architecture to enhance the interpretability of the model further. Specifically, an attention layer is introduced before concatenating feature maps in the shortcut connections. This mechanism aims to focus the model's attention on specific image regions, thereby aiding in understanding the decision-making process.

The UNet model is firmly chosen as the backbone architecture in this project due to its proven effectiveness in biomedical image segmentation tasks. However, an attention mechanism is introduced to improve the model's interpretability. This mechanism highlights certain regions of interest within the image by modulating the importance of different feature maps before they are concatenated during the shortcut connections.

The introduction of the attention mechanism aligns with the project's goal of achieving accurate segmentation and providing insights into the model's decision-making process. By emphasizing specific regions of interest, the attention mechanism enhances the interpretability of the model, allowing researchers and practitioners to gain deeper insights into which image features are most influential in the segmentation process.

Overall, integrating the attention mechanism into the UNet architecture represents a deliberate effort to enhance the model's interpretability, thereby making it more transparent and understandable for users. This approach contributes to the project's aim of developing a robust and interpretable model for biomedical image segmentation.

2.3 Evaluation Metric

Various methods exist for evaluating image segmentation's effectiveness, as Singh et al.

outlined [2]. This paper discussed at least five metrics for measuring segmentation performance: accuracy, Dice coefficient (Dice), Intersection over Union (IoU), Sensitivity, and Specificity. These metrics are defined based on the concepts of True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN), where A and B represent the ground-truth mask and the mask generated by the model, respectively. The formulas for these evaluation metrics are as follows:

$$TP = A \cap B \tag{1}$$

$$FP = \overline{A} \cap B \tag{2}$$

$$FN = A \cap \overline{B} \tag{3}$$

$$TN = \overline{A} \cap \overline{B} \tag{4}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5}$$

$$Dice = \frac{2TP}{2TP + FP + FN} \tag{6}$$

$$IoU = \frac{TP}{TP + FP + FN} \tag{7}$$

$$Sensitivity = \frac{TP}{TP + FN} \tag{8}$$

$$Specificity = \frac{TN}{TN + FP} \tag{9}$$

Among these standards, IoU and Dice are critical evaluations and will be key metrics for assessing this project model's performance. IoU measures segmentation accuracy by calculating the ratio of the intersection of the predicted segmentation result and the ground truth to their union. This provides an intuitive result ranging from 0 to 1, where 1 indicates perfect overlap, and 0 indicates no overlap. On the other hand, the Dice coefficient evaluates segmentation accuracy by computing the ratio of the overlap region to the average area of the two. It balances the relationship between true positives and false positives, making it suitable for imbalanced datasets or class segmentation. Additionally, compared to IoU, the Dice coefficient is more robust for segmentation results with fuzzy boundaries, as it is based on region overlap rather than exact pixel matching. These advantages make IoU and Dice coefficient essential metrics for evaluating the performance and robustness of image segmentation algorithms across different scenarios.

2.4 Model Interpretability

In the field of artificial intelligence, there exists a common trade-off between interpretability and model complexity. Complex models often achieve higher performance and accuracy but at the cost of interpretability. Complex models such as deep neural networks (DNNs) are effective in learning and extracting intricate features from images, leading to more accurate segmentation results, particularly in tasks like image segmentation. However, the complexity of these models makes their internal decision-making processes difficult to

interpret, even if their segmentation results are accurate, posing challenges for users to understand how these decisions are made.

Therefore, striking a balance between interpretability and complexity has become one of the significant challenges in the field of Explainable Image Segmentation. Designing image segmentation models that provide both high accuracy and good interpretability is a current hotspot and challenge in research.

Reliable and common interpretability methods have been reviewed [4] to enhance interpretability in Figure 2. The taxonomy of interpretability methods includes visualization-based, surrogate, and intrinsic approaches. Visualization-based methods visualize internal model workings, including back-propagation, CAM-based, and perturbation-based techniques. Surrogate methods approximate complex models with simpler ones, like LIME and knowledge distillation. Intrinsic methods embed interpretability directly into model architecture, such as attention mechanisms and decision trees. These methods offer insights into model decision-making, enhancing trust and facilitating model validation and improvement.

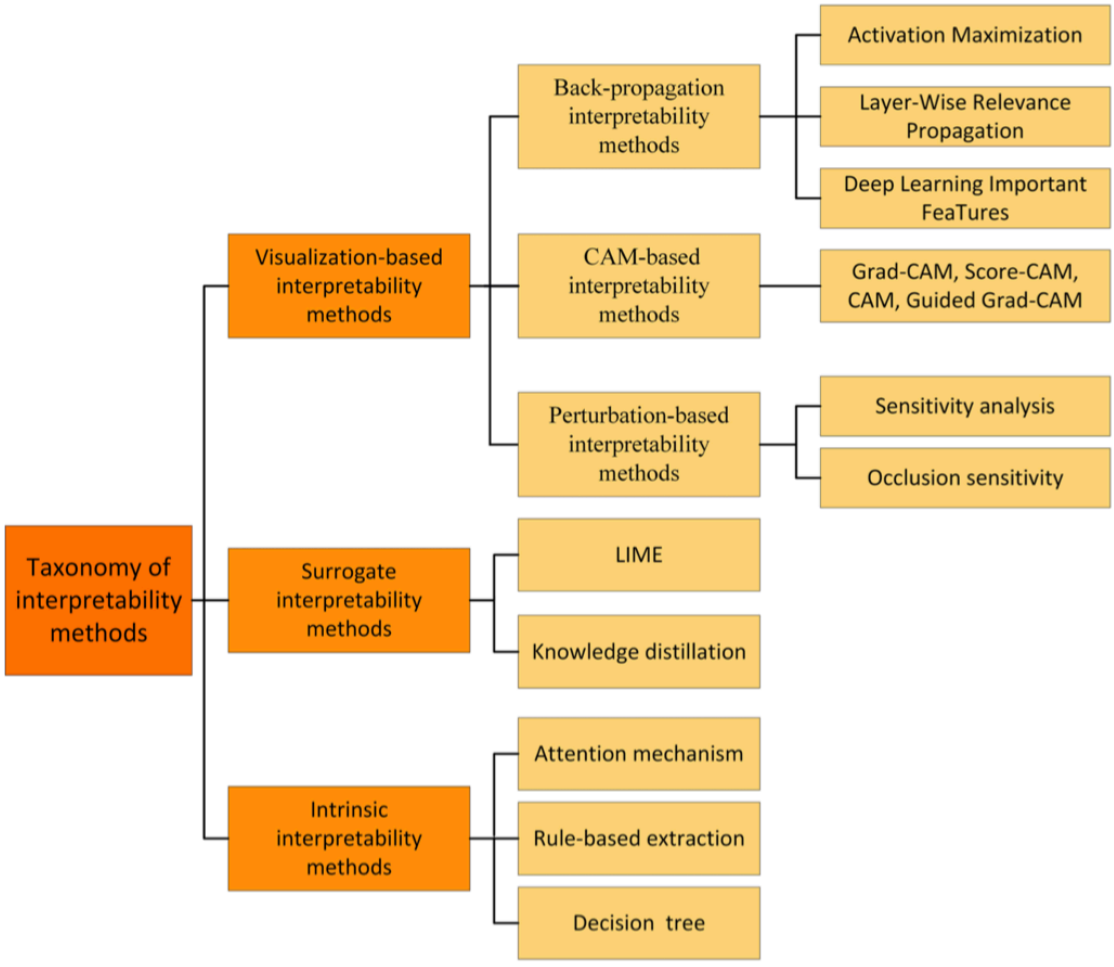

Figure 2. Taxonomy of interpretability methods [4]

To better explain the Attention UNet model, this project adopts various types of interpretability methods:

- Saliency Map and its variant guided backpropagation leverage gradient information from within the model to accurately identify regions of interest in image classification, aiding in understanding the model's decision-making process.
- Applying GradCAM to the Attention layer further enhances the model's interpretability, as the Attention layer typically indicates regions of focus, which can be intuitively visualized using GradCAM.
- The Occlusion Sensitivity method evaluates the impact of different parts of the image on the model's predictions by progressively occluding different regions, identifying critical areas on which the model relies when making decisions.
- The Attention mechanism inherent in the Attention UNet model itself enhances interpretability by enabling the model to perceive essential regions in the image better.
- While the LIME method is not used in this project, considering that it only involves two classes (normal and pathological regions) and the model covers all subclasses, LIME may not be necessary for further explaining the model's decisions.

The following provides a detailed explanation of the chosen interpretable techniques and their significance in the project context.

- **Saliency Map:** The technique allows for determining important pixels (features) in input images by generating saliency maps that depict the pixels' impact on the final score. Pixels with higher amplitudes can be interpreted as representing the target object's location in the input image. Moreover, mainstream AI frameworks such as PyTorch and Keras directly support gradients and backpropagation, facilitating their seamless integration into the project [5].
- **Guided Backpropagation:** While saliency maps rely on gradients, issues may arise due to the highly nonlinear nature of CNNs, where some critical features may contribute significantly to the global context but minimally to local scales. One approach to mitigate this issue is to modify the network such that only positive contributions are propagated during backpropagation, effectively discarding all negative values. Guided backpropagation achieves this by modifying the gradients of Rectified Linear Unit (ReLU) functions during backpropagation, ensuring that only positive contributions are retained. Although guided backpropagation results in clearer and more detailed visualizations compared to saliency maps, they are not entirely faithful to the model and do not truly capture class-specific features [5].
- **Occlusion Sensitivity Method:** It evaluates the impact of different image regions on the model's predictions by progressively occluding various regions, thereby identifying critical areas that the model relies on when making decisions. This explanation technique does not rely on any parameters within the model. Instead, it introduces local perturbations to test variations in model predictions, providing readers with an alternative perspective to observe the model's decision-making rationale.
- **GradCAM on Attention Layer:** GradCAM applied to attention layers offers direct visualization of the model's focal points across feature maps of varying sizes. This technique effectively demonstrates the attention mechanism's capabilities, emphasizing regions of interest within the input image. Understanding why GradCAM was chosen for this purpose requires delving into its utility in interpreting deep learning models' focus and the significance of attention mechanisms. In deep learning models, especially for semantic segmentation tasks, attention mechanisms play a pivotal role, allowing models to concentrate on task-relevant regions, thereby enhancing both performance and interpretability. Through GradCAM, the model's focus on critical areas can be visually represented in images. This is achieved by computing gradients to determine which areas are crucial for the model's predictions.

Applying GradCAM to attention layers enables the direct visualization of the model's focus across different-sized feature maps, which is critical for understanding how the model prioritizes regions in the input image across various levels of abstraction.

Moreover, showcasing the outcomes of attention layers underscores the attention mechanism's efficacy. This mechanism enables models to selectively focus on task-relevant areas within input images, thus improving both performance and interpretability. By displaying the outputs of attention layers, we gain insight into the model's focus across different regions, facilitating a deeper understanding of its decision-making processes and behaviors. Consequently, integrating GradCAM into attention layers not only provides a tangible representation of the model's focus points but also accentuates the role of attention mechanisms, thereby enhancing model interpretability and trustworthiness. It is crucial for elucidating the model's predictive process and aiding in comprehending the underlying principles of deep learning models.

In summary, combining different interpretability techniques provides a comprehensive understanding of the model's decision-making process, enhancing its interpretability, credibility, and reliability.

3. Experimental Procedure

The experimental procedure is structured around a modular design approach to ensure clarity and conciseness throughout the project. It involves the sequential design and implementation of various components, including the Encoder Block, Decoder Block, Attention Gate, and Callback modules, followed by model training and result presentation. The following sections detail each of these processes.

3.1 Environment Setup

The following text describes the environment setup process for model interpretation and visualization using TensorFlow. It covers the installation of necessary libraries, importation of standard modules, and specific functionalities for Grad-CAM visualization.

The environment setup begins by installing the `tf_explain` library, a TensorFlow-based toolkit for explaining and visualizing deep learning models. This library enables the interpretation of model decisions and the visualization of internal workings, enhancing the understanding of model behavior.

Additionally, the environment setup involves importing various common libraries essential for deep learning tasks. These include:

- NumPy: A fundamental package for scientific computing in Python, providing support for large, multi-dimensional arrays and matrices.
- OpenCV: An open-source computer vision and machine learning software library offering a wide range of functions for image and video processing.
- TensorFlow: An end-to-end open-source platform for machine learning, encompassing tools, libraries, and community resources for building and deploying machine learning models.

Furthermore, the setup script imports modules for data preprocessing, visualization, model construction, callbacks, and metrics evaluation. These modules play crucial roles in different

stages of the model development pipeline, facilitating data manipulation, model training, and performance evaluation.

Finally, the script imports specific functionalities for Grad-CAM visualization from the `tf_explain` library. Grad-CAM is a technique for visualizing the regions of an input image most relevant to the model's prediction. By highlighting these regions, Grad-CAM provides insights into the model's decision-making process and helps identify the features it relies on for classification or segmentation tasks.

Overall, the environment setup script ensures that all necessary tools and functionalities are available for building, training, and interpreting deep learning models in TensorFlow, facilitating comprehensive analysis and understanding of model behavior.

3.2 Data Input

The following description outlines the process of loading and preprocessing images for a given dataset.

This code segment focuses on image handling within a machine-learning pipeline. Firstly, the `load_image` function reads an image from a specified file path using Keras, a high-level deep learning API. The image is then converted into a NumPy array, a fundamental data structure for numerical computing in Python.

Subsequently, the image's pixel values are normalized to fall within the range of 0 to 1, a common preprocessing step in deep learning tasks to ensure numerical stability and facilitate convergence during model training. Additionally, the image is resized to a specific dimension, enabling consistency in input size across the dataset.

Furthermore, the precision of pixel values is rounded to four decimal places, enhancing computational efficiency and reducing memory footprint without significantly sacrificing accuracy.

The `load_images` function iterates through a list of image paths, applying the aforementioned preprocessing steps to each image using the `load_image` function. If the images are masks, which typically represent ground truth annotations for segmentation tasks, only the first channel of the processed image is retained to represent binary segmentation masks.

Finally, the `load_images` function aggregates the processed images and masks into arrays, which are stored in separate variables for subsequent use. These variables, namely images and masks, contain the preprocessed image data and corresponding masks, respectively, enabling seamless integration into the training and evaluation pipeline of the machine learning model.

Overall, this data input segment efficiently prepares image data for training and evaluation, laying the groundwork for robust and accurate model development in machine learning applications.

3.3 Encoder Block

The Encoder Block serves as a fundamental building block for the encoder component of a

neural network architecture.

This class encapsulates the functionality required to process input data and extract meaningful features while reducing spatial dimensions through pooling operations. It comprises two convolutional layers, each augmented with dropout regularization to mitigate overfitting by randomly deactivating a fraction of neurons during training. Additionally, an optional max-pooling layer is included, enabling further reduction of spatial dimensions through downsampling.

The primary objective of the Encoder Block is to abstract essential features from input data, facilitating hierarchical feature extraction in subsequent layers of the neural network.

During the forward pass, the `call` method orchestrates the sequential application of defined layers to the input data. Depending on the configuration of the pooling parameter, this method returns either the output of the block or both the output and the input data before pooling, providing flexibility in network architecture design.

Furthermore, the Encoder Block class includes a `get_config` method, which enables retrieval of the configuration parameters associated with the block. This feature enhances reproducibility and facilitates model serialization, allowing for seamless model deployment and sharing across different environments.

In summary, the Encoder Block plays a crucial role in the neural network architecture, facilitating feature extraction and dimensionality reduction within the encoder component, thereby contributing to the overall efficacy and performance of the model.

3.4 Decoder Block

The Decoder Block, plays a crucial role in the neural network architecture, facilitating the reconstruction of spatial information lost during the encoding process.

This class is specifically designed to handle the upsampling and concatenation of feature maps from the encoder blocks, thereby enabling the recovery of spatial details essential for accurate segmentation or reconstruction tasks.

The Decoder Block consists of two primary components:

- **Upsampling Layer**: The `DecoderBlock` begins with an upsampling layer, which increases the spatial dimensions of the input feature map. This upsampling operation aims to restore the spatial resolution lost during the encoding process, thereby ensuring the fidelity of the reconstructed output.
- **`EncoderBlock` Instance**: The Decoder Block incorporates an `EncoderBlock` instance following the upsampling layer. This instance further processes the upsampled feature map, extracting and abstracting relevant features to generate the final output.

During the forward pass, the call method of the Decoder Block takes a tuple of input feature maps (X) and the corresponding skipped input from the encoder block (skip_X). It then performs the necessary upsampling operation, concatenates the upsampled feature map with the skip connection feature map, and passes the concatenated feature map through the `EncoderBlock` instance to generate the final output.

Additionally, the `DecoderBlock` class includes a `get_config` method, which returns the configuration parameters associated with the block. These parameters typically include details such as the number of filters and the dropout rate, providing insights into the architecture of the Decoder Block.

In summary, the Decoder Block plays a critical role in the neural network architecture, facilitating the reconstruction of spatial information and contributing to the overall effectiveness and performance of the model in tasks such as segmentation and image reconstruction.

3.5 Attention Gate

The `AttentionGate` class serves as a pivotal component within the architecture of a U-Net model, implementing an attention mechanism to focus on relevant spatial regions from skip connections selectively.

This class is designed to enhance the U-Net architecture by incorporating attention-based mechanisms, which dynamically adjust the contribution of skip connections based on their importance for the segmentation task.

The `AttentionGate` consists of several key components:

- **Convolutional Layers**: The `AttentionGate` begins with convolutional layers responsible for processing the input feature maps and downsampled skip connections. These layers apply learnable filters to extract relevant features and facilitate learning attention weights.
- **Element-wise Addition**: The call method of the `AttentionGate` class takes a tuple of input feature maps (X) and the corresponding skip connection feature maps (skip_X). It applies convolutional operations to both inputs and combines them through element-wise addition. This process enables the integration of information from both the original input feature maps and the skip connections.
- **Attention Weight Learning**: Following the element-wise addition, the `AttentionGate` learns attention weights to determine the relative importance of different spatial regions within the skip connections. These attention weights are applied to the skip connections, allowing the model to focus on regions crucial for accurate segmentation selectively.

Batch Normalization (Optional): Optionally, batch normalization can be applied to the output feature maps of the `AttentionGate`. *Batch normalization* is a technique used to stabilize and accelerate the training of deep neural networks by normalizing the activations of each layer.

Finally, the method `get_config` of the `AttentionGate` class returns the configuration parameters associated with the gate, including details such as the number of filters and whether batch normalization is used. This information provides insights into the architecture and configuration of the `AttentionGate` within the overall U-Net model.

In summary, the `AttentionGate` class plays a critical role in enhancing the U-Net architecture by introducing attention-based mechanisms, which enable the model to selectively focus on relevant spatial regions and improve the accuracy of segmentation tasks.

3.6 Custom Callback

The Custom Callback module is a unique tool that offers functionalities specifically designed to enhance model interpretation and training progress visualization. It does this without directly impacting the training process. In this section, we will delve into two key components of the Custom Callback module: the `compute_guided_backpropagation` function and the `ShowProgress` callback class.

The `compute_guided_backpropagation` function is a powerful tool for model interpretation during the evaluation phase. It calculates the gradients of the model's output with respect to the input image using guided backpropagation. This technique is particularly useful for identifying the input regions that significantly influence the model's predictions. By highlighting these influential regions through computed gradients, this function provides valuable insights into the model's decision-making process, which can be instrumental in model interpretation and debugging.

The `ShowProgress` Callback class is a comprehensive tool that visualizes the model's training progress and extends its capabilities to the evaluation phase. This class includes methods that are dedicated to visualizing various aspects of the model's behavior during evaluation, providing a clear and detailed view of the model's performance.

The `ShowProgress` Callback class incorporates methods for visualizing key aspects of the model's behavior during evaluation. This includes visualizing encoder blocks to understand feature extraction and transformation, attention gate GradCAM results highlighting crucial prediction regions, and saliency maps with guided backpropagation to identify important image regions for prediction, enhancing model interpretability and validation during evaluation.

Through these additional visualization techniques integrated into the ShowProgress Callback class, model interpretation, validation, and improvement are facilitated during training and evaluation. These visualizations enable a deeper understanding of the model's inner workings, aiding in model debugging, validation, and continuous improvement throughout the evaluation process.

3.7 Self-defined Metric

Custom metrics are utilized for quantitative evaluation, referenced as (6) and (7):

- Dice Coefficient Function: The `dice_coefficient` function (6) computes the similarity between two binary masks by measuring their overlap. It calculates the intersection and union of the masks and then applies the Dice formula to determine similarity.
- IoU Coefficient Function: The `iou_coefficient` function (7) computes the overlap between binary masks using the Intersection over Union (IoU) metric. It evaluates the ratio of intersection to union, providing a measure of segmentation accuracy.

3.8 Main Body

The main body of the code implements a custom U-Net architecture with attention gates for image segmentation, as shown in Figure 3. This architecture comprises several

interconnected components, each contributing to the segmentation process:

- **Encoder Blocks**: Four encoder blocks (EncoderBlock) are employed to downsample the input image and extract hierarchical features progressively. Each encoder block consists of two convolutional layers followed by optional max-pooling, generating pooled features (p1, p2, p3, p4) and corresponding feature maps (c1, c2, c3, c4) before pooling. These feature maps serve as skip connections for the decoder.
- **Encoding**: An additional encoding block is applied without pooling, resulting in the encoding tensor (encoding), which encapsulates high-level semantic information from the input image.
- **Attention Mechanism**: Attention gates (AttentionGate) are inserted between the encoding and decoder blocks to emphasize informative regions and suppress irrelevant features selectively. Four attention gates (a1, a2, a3, a4) are integrated into the network, each receiving input from the encoding tensor and the corresponding feature map from the encoder block.
- **Decoder Blocks**: Four decoder blocks (DecoderBlock) are utilized to upsample features and reconstruct the segmentation mask. Each decoder block incorporates an upsampling layer followed by convolutional layers, with skip connections from the corresponding encoder block and attention gate.
- **Output Layer**: The final output layer comprises a convolutional layer with a single filter and sigmoid activation function, yielding the segmentation mask.
- **Model Compilation and Training**: The model is compiled with binary cross-entropy loss and the Adam optimizer. Custom metrics such as accuracy, IoU, and Dice coefficient are employed to monitor model performance during training. The training process is configured with parameters such as batch size, epochs, and validation split, and the training progress is monitored using callbacks.

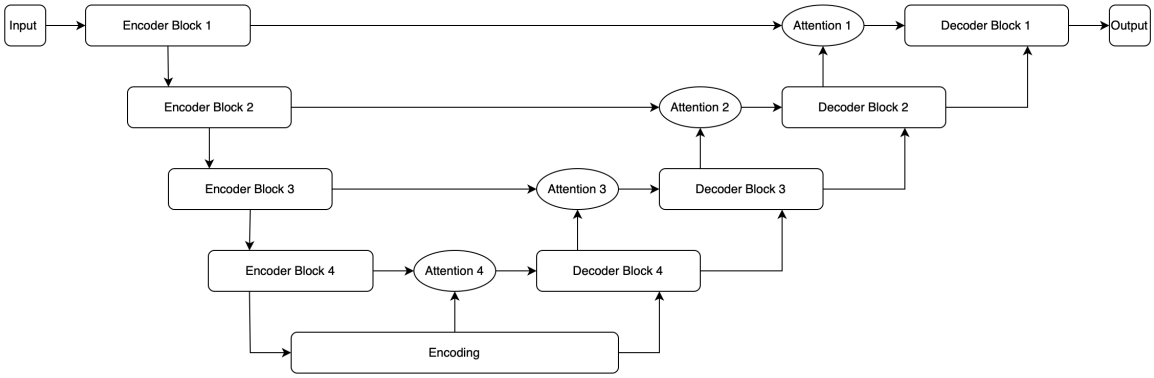

Figure 3. Attention UNet Structure

This comprehensive architecture enables the model to learn intricate spatial relationships and effectively segment input images into distinct regions of interest, rendering it suitable for various medical image analysis tasks.

3.9 Evaluation

The evaluation section encompasses visualizing the trained model's performance metrics and utilizing occlusion sensitivity mapping for model interpretation. Here is a breakdown of each component:

- Model Performance Visualization:
  Four subplots are generated to visualize the training and validation metrics across epochs: loss, accuracy, IoU, and Dice coefficient. These metrics are accessed from the results. history dictionary.
- Segmentation Results and Confidence Maps:
  Various visualizations are presented for a random subset of images, including the original image, ground truth mask, predicted mask, processed mask (thresholded predicted mask), and confidence map of the predicted mask.
  The predicted mask is thresholded using a confidence threshold 0.5 to generate the processed mask.
- Occlusion Sensitivity Mapping:
  An additional subplot is included to visualize the occlusion sensitivity map, elucidating the model's sensitivity to specific regions within the image.
  Multiple windows are iteratively slid across each image using a specified window size and stride. The model's predictions are compared at each window position between the original and occluded images to compute the sensitivity map.
  The sensitivity maps obtained from different windows are aggregated to produce the final occlusion sensitivity map, highlighting regions crucial for segmentation.
  - Visualizing Encoder Blocks: This method presents the activations of each encoder block within the neural network. By visualizing these activations, one can discern how the model extracts and transforms features at different stages of the encoding process, providing valuable insights into feature representation.
  - Visualizing Attention Gate GradCAM Results: GradCAM visualization is utilized to highlight regions of the input image crucial for making predictions, with a particular focus on attention gate layers. This visualization elucidates where the model directs its attention and how it integrates information from different image regions, enhancing interpretability.
  - Visualizing Saliency Maps and Guided Backpropagation: Saliency maps and guided backpropagation techniques highlight the most relevant regions of the input image for prediction. Saliency maps indicate regions of high importance, while guided backpropagation emphasizes these regions by illustrating gradients of the output with respect to the input image. These visualizations offer insights into the model's decision-making process and contribute to model understanding and validation.

These visualizations and analyses are crucial in evaluating the model's performance, understanding its segmentation decisions, and identifying significant image regions influencing the predictions. Furthermore, occlusion sensitivity mapping provides valuable insights into the model's attention and feature importance, enhancing its interpretability. The incorporation of additional visualization methods from the callback further enriches the evaluation process by providing deeper insights into the model's behavior and performance during training and evaluation phases.

4. Results

The result section presents a comprehensive overview of the U-Net model's performance and various visualization techniques employed for image segmentation.

4.1 Masks and Maps

Figure 4 illustrates various stages and outcomes associated with image segmentation. These images depict a single instance from the dataset following extensive training (over 100 epochs).

- **Original Mask:** Represents the ground truth of the region of interest.
- **Predicted Mask:** This illustrates the model's prediction for the segmentation. The predicted mask closely resembles the original mask, with minor discrepancies observed, particularly at the left edge.
- **Processed Mask:** This appears to be the predicted mask after post-processing (confidence > 0.5) to refine the segmentation results. It eliminates areas with low confidence (< 0.5) from the predicted mask.
- **Confidence Map:** Likely signifies the pixel-wise confidence of the model's predictions, with brighter areas indicating higher confidence. The confidence map primarily comprises bright areas against a dark background, with minimal gray regions along the boundaries. Although the Confidence Map is essentially another representation of the Predicted Mask, having a scale (color bar) makes it easier to observe the model's predicted confidence level.
- **Occlusion Sensitivity Map:** Visualizes the impact on model performance when different parts of the image are occluded sequentially, aiding in understanding which image regions are crucial for the model's predictions. The areas with the greatest impact are depicted in white, while those with the least impact are in black. The impact generally transitions between yellow and red, with areas of lighter color indicating greater influence. In this schematic, apart from some deep red areas in the top left corner indicating some misjudgment, the sensitivity map highlights critical regions around the boundary of the predicted mask, especially the brightest portion at the upper end of the tumor. However, occluding the central region of the tumor has little effect.

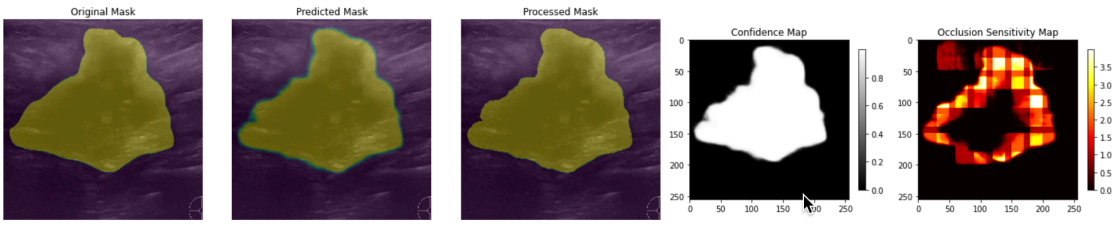

Figure 4. Masks and Maps

Figure 5 illustrates another explain technique for ultrasound image segmentation.

- **Original Mask:** Represents the true segmentation of the ultrasound images.
- **Predicted Mask:** Displays the segmentation output generated by the model. The predicted mask closely resembles the original mask regarding shape, boundary, and position.
- **Saliency Map:** Highlights regions of the image that strongly influence the model's output. Darker areas indicate less importance, while brighter red areas signify greater importance. It appears against a black background, suggesting that regions with smooth gradients are less critical, with the edges of the tumor exhibiting relatively higher gradient differences. Notably, the lower right boundary of the tumor is the brightest area in the entire image.
- **Guided Backpropagation:** Another visualization technique identifying important features contributing to the model's predictions. This method modifies the gradients of the ReLU functions, discarding negative values during the backpropagation calculation. The contour and distribution in the guided backpropagation results are similar to the saliency map, albeit with a sparser appearance.

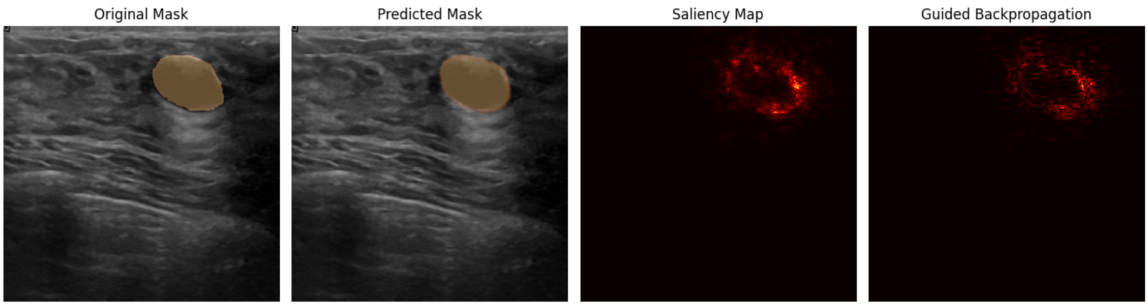

Figure 5. Saliency Map and Guided Backpropagation

4.2 Model Performance Metrics

Figure 6 showcases key performance metrics over epochs: loss, accuracy, IoU, and Dice coefficient.

- **Model Loss:** Both training and validation loss decrease steadily over epochs, which is typical of learning processes. The validation loss remains around 0.1, while the training loss decreases to approximately 0.01. Both lines exhibit minimal fluctuations.
- **Model Accuracy:** Fluctuations are observed, but there is an overall upward trend, particularly noticeable in training accuracy. Validation accuracy maintains around 0.98, with occasional larger fluctuations.
- **Model IoU:** Similarly, the IoU scores display fluctuations with an overall increasing trend. Training IoU peaks around 0.7, while validation IoU reaches approximately 0.61. Most of the time, the gap between training and validation metrics remains small.
- **The Dice Coefficient:** The graph closely resembles the IoU graph, reflecting similar trends and fluctuations.

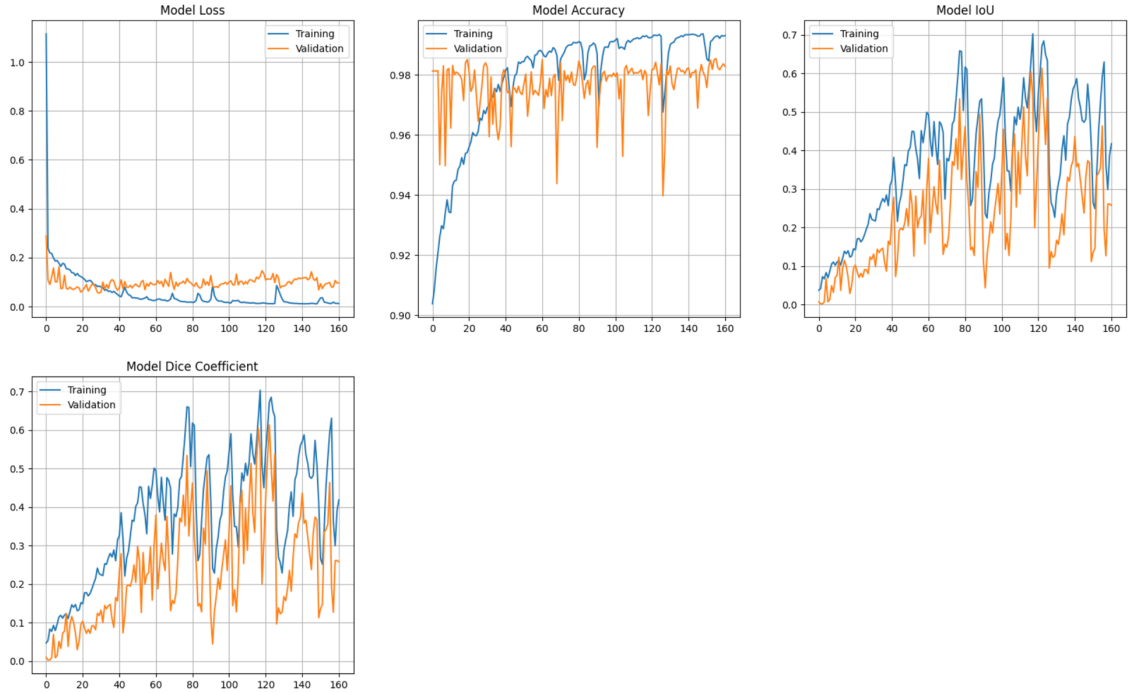

Figure 6. Model Performance Metrics

4.3 Grad-CAM Attention Maps

Figure 7 Grad-CAM attention maps provide valuable insights into the internal workings of the model, revealing the areas of the input image that are most influential in predicting the segmentation output. By visualizing the regions where the model directs its attention, it is easy to gain a deeper understanding of the underlying features and characteristics that contribute to the segmentation process.

In the four layers' attention maps, distinct focus patterns emerge.

- **Attention Gate 1:** This attention map primarily highlights the region below the tumor, indicating that the model assigns significance to features in this area when making segmentation decisions. The attention here may be directed toward subtle structural cues or contextual information surrounding the tumor region.
- **Attention Gate 2:** In contrast, the attention map for this layer appears to emphasize the image texture and the lower portion of the tumor region. This suggests that the model is sensitive to the image's overall structure and the specific details within the tumor area, utilizing a combination of global and local features for segmentation.
- **Attention Gates 3 and 4:** These attention maps predominantly focus on the tumor's edges. Edge detection is crucial in segmentation tasks as it helps delineate the boundary between different regions of interest. The model's attention to these edges indicates its awareness of the spatial boundaries within the image, enabling precise delineation of the tumor region.

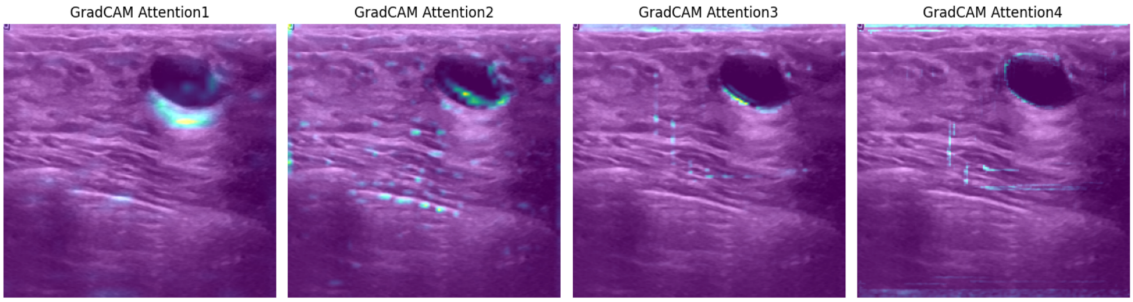

Figure 7. GradCAM Attention Map

4.4 Encoder Block Activations

The activation maps depicted in Figure 8 provide valuable insights into the feature extraction process within the encoder part of the U-Net architecture. These maps showcase the regions of the input image that elicit strong responses from the network at different stages of the encoding process.

As we observe the activation maps from left to right, a noticeable trend emerges: information density decreases progressively. This observation aligns with the encoding process's downsampling characteristic, where the feature maps' spatial dimensions are reduced to capture increasingly abstract representations of the input image.

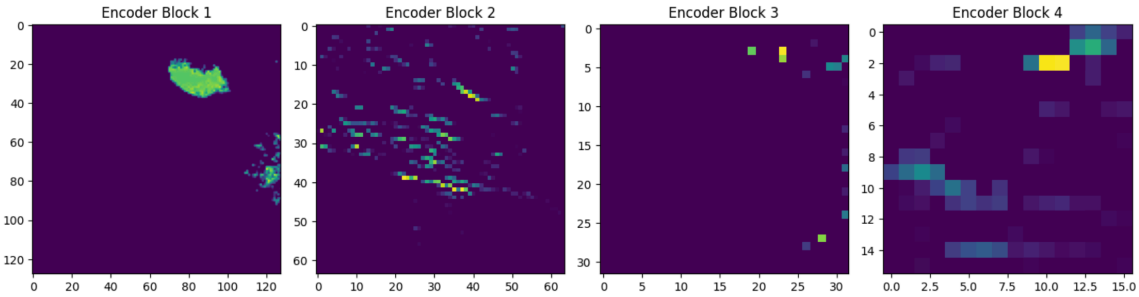

Figure 8. Encoder Block Activation

4.5 Training Console Output

The displayed snippet from the training console provides valuable insights into the training progress of the U-Net model over different epochs. Key performance metrics such as loss, accuracy, IoU coefficient, dice coefficient, and their validation counterparts are monitored and updated throughout the training process. As shown in Figure 9, at the time of display, the training had progressed to nearly 40%. The loss and accuracy could maintain relatively low and high levels, respectively. Although the IoU coefficient is not high overall, the fluctuations in IoU on both the training set and the validation set are roughly consistent. The same pattern applies to the Dice coefficient. Meanwhile, it is evident that the performance in Epochs 75 and 76 is regressing compared to Epoch 74.

```
Epoch 73/200
97/97 [==============================] - 23s 242ms/step - loss: 0.0228 - accuracy: 0.9892 - i
ou_coefficient: 0.3645 - dice_coefficient: 0.3667 - val_loss: 0.0912 - val_accuracy: 0.9786 -
val_iou_coefficient: 0.1987 - val_dice_coefficient: 0.1995
Epoch 74/200
97/97 [==============================] - 23s 242ms/step - loss: 0.0190 - accuracy: 0.9906 - i
ou_coefficient: 0.4756 - dice_coefficient: 0.4775 - val_loss: 0.0986 - val_accuracy: 0.9792 -
val_iou_coefficient: 0.2730 - val_dice_coefficient: 0.2737
Epoch 75/200
97/97 [==============================] - 24s 242ms/step - loss: 0.0229 - accuracy: 0.9894 - i
ou_coefficient: 0.4401 - dice_coefficient: 0.4422 - val_loss: 0.1090 - val_accuracy: 0.9769 -
val_iou_coefficient: 0.2630 - val_dice_coefficient: 0.2635
Epoch 76/200
97/97 [==============================] - 23s 242ms/step - loss: 0.0364 - accuracy: 0.9847 - i
ou_coefficient: 0.2866 - dice_coefficient: 0.2897 - val_loss: 0.0958 - val_accuracy: 0.9606 -
val_iou_coefficient: 0.0987 - val_dice_coefficient: 0.0998
```

Figure 9. Training Console Output

5. Analyses

The analysis section delves into the interpretation of the results, highlighting key findings and insights derived from the model's performance metrics and visualization outputs.

5.1 Masks and Maps

In Figure 4, the similarity between the original and predicted masks suggests effective segmentation performance by the model. Despite potential fluctuations in model metrics in the plotted graphs, the actual segmentation results demonstrate significant accuracy and determination in delineating segmentation boundaries.

The processed mask exhibits fewer false positives and negatives, indicating successful post-processing, enhancing segmentation accuracy and uncovering subtle differences from the original image.

The high brightness in the confidence map within the region of interest indicates the model's confidence in its predictions, reflecting rapid convergence during training and successful model learning.

The occlusion sensitivity map reveals that certain regions within the mask are critical for the model's decision-making process, suggesting the model has learned informative features for segmentation, particularly around the tumor's edges.

From Figure 5, it shows that the close resemblance between the original and predicted masks suggests accurate segmentation performance by the model.

Both the saliency map and guided backpropagation results highlight areas of the image that significantly impact the model's prediction. This insight into the model's focus indicates that the model correctly attends to and utilizes regions with higher gradients, potentially identifying diagnostically significant features in the images.

5.2 Model Performance Metrics

As shown in Figure 6, the convergence of training and validation loss suggests minimal overfitting of the model. The choice of cross-entropy loss contributes to the well-behaved loss graph.

Improvements in accuracy and IoU metrics over epochs indicate that the model's generalization ability is improving. However, fluctuations suggest potential instability, possibly due to learning rate adjustments by the Adam optimizer or data variability.

The similarity between validation and training metrics can stem from factors such as a small validation set or the inclusion of regularization techniques.

The close resemblance between the Dice Coefficient and IoU graphs is due to their data similarity during training, with differences typically within 0.01.

In the referenced study by V. K. Singh et al. [2], the UNet model achieved state-of-the-art results in breast tumor segmentation, with IoU scores ranging from 0.66 to 0.77, depending on dataset annotation and difficulty. The performance metrics observed here are competitive with these benchmarks, indicating the effectiveness of the implemented model.

5.3 Grad-CAM Attention Maps

Figure 7 tells that the varied patterns of attention across different layers of the model

highlight its ability to capture and leverage a diverse range of features for segmentation. The multi-scale understanding demonstrated by the model suggests that it can effectively integrate information from various spatial scales, enhancing its ability to segment ultrasound images accurately.

Furthermore, the specificity of attention to different regions within the image indicates that the model's segmentation decisions are based on a nuanced analysis of image features. The model can produce segmentation outputs that align closely with the ground truth by focusing on relevant regions and features, demonstrating its robustness and efficacy in medical image analysis tasks.

5.4 Encoder Block Activations

The activation patterns within the encoder blocks shown in Figure 8 offer clues about the nature of the features being extracted at various levels of abstraction.

Early Encoder Blocks: The activation maps from the initial encoder blocks likely correspond to low-level features such as edges, textures, and basic shapes. These blocks serve as the foundation for higher-level feature learning, capturing fundamental elements of the input image.

Deeper Encoder Blocks: In contrast, the activation maps from deeper encoder blocks reveal more abstract and complex features. These blocks are adept at capturing high-level semantic information and contextual relationships within the input image. By hierarchically combining features from earlier blocks, deeper blocks can discern more intricate patterns and structures in the data.

The variation in activation patterns across different encoder blocks underscores the hierarchical nature of feature learning in CNNs. Each block specializes in extracting specific features relevant to its position in the network, collectively contributing to the model's ability to understand and interpret complex visual information.

5.5 Training Console Output

The training console output in Figure 9 reveals several important observations about the model's performance and training dynamics:

- **Progressive Improvement:** There is a discernible trend of improvement in all tracked metrics as the training progresses over epochs. This consistent enhancement in metrics, including accuracy, IoU coefficient, and dice coefficient, signifies that the model is learning from the training data and refining its segmentation capabilities over time.
- **Convergence of Validation Metrics:** The validation counterparts of the metrics also exhibit a similar trend of improvement over epochs, converging towards higher values. This convergence indicates that the model's performance generalizes well to unseen data, as evidenced by the consistent validation performance across different metrics.
- **Progress Bar Benefits:** The use of a Progress Bar during training offers several benefits, including real-time monitoring of training progress, visualization of epoch-wise metrics, and early detection of any potential issues, such as overfitting or convergence problems. Additionally, it provides valuable feedback to researchers

and practitioners, enabling them to make informed decisions about model optimization and hyperparameter tuning. The Progress Bar serves as a powerful tool for tracking training dynamics and ensuring the successful convergence of the model.

Overall, the training console output provides comprehensive feedback on the model's performance and training trajectory, guiding the optimization process toward achieving optimal segmentation results.

6. Hardware Design

Incorporating the Attention-UNet image segmentation technology into the medical sector necessitates hardware that is energy-efficient yet powerful to ensure seamless integration with medical devices. While the hardware model remains untested in simulation, an effort is made to articulate its workflow.

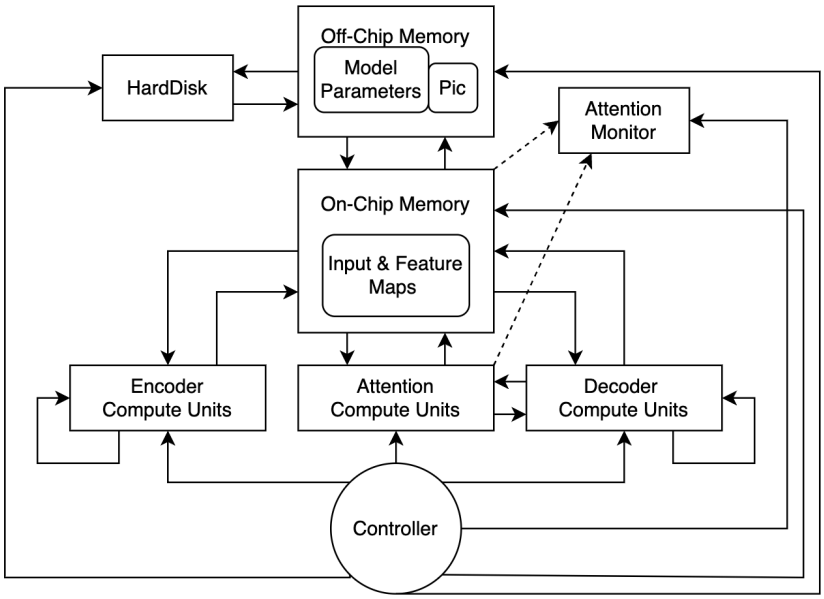

Figure 10. Hardware Structure

6.1 Architecture Overview

Figure 10 showcases the dedicated hardware's configuration, bifurcated into two primary memory types: off-chip and on-chip memory. Off-chip memory is utilized for interfacing with the hard disk to store the segmented images and accommodate camera-acquired image or video signals. It also houses extensive model parameters. Conversely, the on-chip memory, with its reduced size and expedited throughput, directly interfaces with the off-chip memory, compute units and the Attention Monitor—which displays content from the Attention Gate (model's focus areas). Owing to its rapid processing and minimal latency, the Attention Monitor can nearly instantaneously track the model's status. Alterations to monitoring needs or controller directives can be implemented swiftly due to this architecture. Additionally, the on-chip memory temporarily retains the inputs and Feature Maps, facilitating pipelining to diminish recurrent memory access operations.

6.2 Computational Workflow

Computationally, the model is underpinned by three distinct compute units structured in alignment with the algorithm's architecture—Encoder, Attention, and Decoder Compute Units.

- **Encoder Compute Unit**: Sources input from the on-chip memory, performs iterative calculations, and consecutively outputs the results of five cycles back to the on-chip memory.
- **Attention Compute Unit**: Sequentially assimilates results from the Encoder Compute Unit in reverse chronological order, alongside outputs from the previous clock cycle of the Decoder Compute Unit, dispatching the attention distribution map to the Decoder Compute Unit.
- **Decoder Compute Unit**: Directly ingests inputs from the Attention Compute Unit along with its outputs from the prior clock cycle.

The data flow from the Attention Compute Unit to the on-chip memory predominantly aids in monitoring, while the Decoder Compute Unit, barring the initial and terminal stages, operates independently of the on-chip memory, enhancing operational efficiency. This streamlining of interactions bypassing memory further augments the training and inference performance.

6.3 Centralized Control

All hardware elements are orchestrated by a central controller that maintains the operational harmony of the system.

6.4 Performance Benchmarks

Evaluating the performance of the hardware design tailored for the Attention-UNet image segmentation model extends beyond traditional metrics such as accuracy, loss, Intersection over Union (IoU), and Dice coefficients. The hardware-specific benchmarks cater to a more nuanced analysis, crucial for assessing the model's deployment in real-world medical applications.

- **Execution Time**: A critical measure of performance is the execution time, which is the duration the hardware takes to complete a segmentation task. This metric is pivotal as it reflects the raw computational speed of the system. A lower execution time indicates a robust capability to process image data swiftly, which is vital for real-time applications such as intraoperative imaging during surgeries. By closely observing the time taken for each segment of the pipeline to process data—particularly during the encoder, attention gate, and decoder stages—we gain insight into potential bottlenecks and the efficiency of the hardware design.
- **Memory Usage (On-Chip and Off-Chip)**: Memory usage metrics provide insight into the hardware's optimization for data handling. On-chip memory usage is particularly important due to its limited size and faster access speeds. Efficient use of on-chip memory can significantly reduce the need for costly data transfers to and from the slower off-chip memory, thus enhancing overall performance. Off-chip memory usage is also monitored to ensure that the model parameters and image

storage do not exceed the system's capacity, which could lead to performance degradation.

- **Instructions Per Cycle (IPC)**: IPC is a direct measure of the hardware's operational efficiency. A higher IPC value indicates that more instructions are being executed per cycle, which implies better utilization of the compute resources. For the Attention-UNet hardware, IPC can reveal how effectively the compute units are being utilized, showing the parallelism achieved in processing various layers and aspects of the neural network.
- **Cycles Per Instruction (CPI)**: Conversely, CPI measures the number of cycles required to execute a single instruction. An elevated CPI can signal inefficiencies, potentially from complex instructions that take multiple cycles to complete or from stalling due to dependencies and memory latency. CPI helps in identifying the cost of the instructions executed and aids in tuning the hardware for better performance.

6.5 Debugging and Optimization

For model debugging and validation, datasets from within the algorithm are used as inputs fed into the off-chip memory. Post-simulation validation, real-world trials with camera image data streams can be compared against the simulation results to ensure reliability and efficiency.

7. Limitations

While the Attention-UNet model has achieved satisfactory results in breast cancer image segmentation, the project still has significant limitations, and there is a considerable gap in actual application in the medical field.

Firstly, the limitation lies in the restriction of the dataset. The 780 images are far from the size of a real database. Although the UNet model excels in training with small samples, the more data there is, the more memorable cases there will be, which can better test or improve the model's adaptability. This is especially true when the model faces irregularly shaped tumors, where the segmentation can sometimes be significantly off. Having more unusual samples could partially solve the model's generalization ability.

Secondly, there is instability during training. The model experiences periodic severe fluctuations in key indicators such as IoU and Dice, sometimes severe enough to approach or reach pre-training levels. This fluctuation does not align with the stability and efficiency required when introducing new technologies in the medical field, as no one knows when the best state to pause training is. Even if the gap between the training set and the validation set gradually narrows, and the actual segmentation effect improves with the increase in training rounds, it cannot guarantee steady improvement. AI image segmentation assists doctors, and even a slight correct hint or guidance can potentially lead to early detection of lesions and timely intervention in treatment. Therefore, only when both the training indicators and the actual effects are good can we say that this model has value for application in the medical field.

The third limitation is that the model's environment is overly ideal. This model only considers identifying the area of breast cancer in ultrasound images, with binary image segmentation simply classifying the entire image into normal and tumor areas. In reality, images can be divided into many different areas, so multi-class image segmentation is needed, which can make use of more explanatory techniques such as LIME. If the model

learns the different features of specific areas through training, theoretically, it has a greater possibility of more accurate segmentation. It can also better segment difficult images. For example, by annotating normal wrinkle areas more in the training set, the model can learn the features and potential manifestations of wrinkles through extensive training, thereby largely avoiding the misidentification of normal wrinkles on difficult images. Of course, this more detailed classification or annotation is bound to increase the difficulty in the selection and use of the dataset, to some extent increasing the cost of training. It's also worth noting that in medical imaging, not only 2D black and white images need to be analyzed. Other images, such as 3D and color images, still have a certain market. If the model's versatility is not strong, doctors will inevitably consider whether it is necessary to introduce artificial intelligence. After all, artificial intelligence in the medical field can hardly completely replace the doctor's final diagnosis. Accurate segmentation results may not alleviate the pressure and time needed for the doctor's final diagnosis. It's not that the doctor can directly listen to the model's segmentation results just because the model says it's okay. Therefore, the model's approach to the real environment should be a necessary path for its practicality.

The fourth limitation is that this model is a supervised learning model, and the dataset still requires ground truth annotation. This can be done under normal circumstances. However, doctors can easily identify simple image situations without the help of AI. For images where tumor identification is very difficult, it is hard for doctors to annotate, so the model will hardly have a reference ground truth, and thus, the model will hardly have the opportunity and ability to train with difficult images. When doctors most need reference and advice, this model may not be able to provide suggestions. Therefore, if the model makes breakthroughs in the field of unsupervised learning, it may solve the problem of the training set.

The fifth limitation is that the persuasiveness of interpretability technology is still not enough. Although many image-based interpretability technologies, including this project, have, to some extent, explained the model's focus and segmentation basis, they still lack the combination with medical theoretical knowledge, lack references, and derivations, and rely solely on pattern recognition-like methods, which lack the rigor of scientific practice. Therefore, combining large language models to further explain images may have surprising effects.

The final notable limitation is the conceptual stage of the hardware design. Although a framework has been outlined, intricate aspects such as the translation of the Attention U-Net model's parameters into corresponding hardware attributes—resistance, conductance, and voltage values—remain unresolved. This gap underscores the need for a detailed engineering approach to ensure hardware components can effectively mirror the model's computational functions.

8. Future Work

Considering the limitations mentioned above, several improvements can be made to the project:

Firstly, acquiring ultrasound images of breast cancer without restrictions can lead to a more diverse dataset. Introducing noise into the dataset can enhance the model's generalization ability and prevent overfitting. Moreover, if the model supports unlabeled or semi-labeled samples, it would significantly reduce the burden before training. Adapting the model to support multi-class segmentation and analyzing 3D or color images under certain conditions would also be beneficial.

Secondly, updating the evaluation metrics by removing inferior and excellent outliers before averaging them can stabilize metrics like Intersection over Union (IoU) and Dice, ensuring they remain representative. Additionally, to terminate training more effectively, implementing a monitoring thread to observe training progress and interrupt training if there is no improvement for several epochs can be beneficial.

Thirdly, enhancing the hardware setup to exceed the efficiency of general-purpose computers by two to three times is crucial. This involves not only refining the hardware architecture but also conducting thorough testing to ensure that the system can handle the computational demands of advanced deep learning models effectively.

Finally, integrating the training, inference, and interpretation results with a large language model can provide more logical and readable explanations. Furthermore, recent advancements in academia have strengthened the Attention UNet model. Introducing a Transformer architecture, as demonstrated in U-Net Transformer by Petit et al. [6], can significantly improve the model's predictive ability. U-Net Transformer accurately segments irregularly shaped organs in medical image segmentation tasks.

9. Conclusion

This project was dedicated to developing a unique medical image segmentation model - Attention UNet. This model, specifically designed for detecting breast cancer in two-dimensional ultrasound chest images, integrates artificial intelligence in medical image analysis. This integration alleviates the burden on healthcare professionals and can potentially reduce misdiagnosis and missed diagnosis cases. The model's unique feature is the addition of interpretable techniques, which enhances medical personnel's understanding of the model's segmentation basis, thereby increasing their acceptance of new technologies. Moreover, the Attention UNet model was selected considering GPU computing costs and sample capacity, and it operates efficiently with limited data and computational resources. The advantages are achieved through the modular design of Encoder Blocks, Decoder Blocks, and Attention Gates, effectively capturing image details and contours while achieving relatively accurate segmentation at a lower cost.

The overall performance of the Attention UNet model is commendable. The loss metric consistently decreases during training, with both training and validation accuracies reaching high levels. Although IoU and Dice's coefficients displayed fluctuations, they showed an overall increasing trend, indicating improved segmentation performance. Notably, with rapid predictions, the model demonstrated precise segmentation results for most samples. While challenges such as irregular tumor shapes persisted, the model's performance remained robust, with minimal overfitting tendencies.

The developed model has significant practical implications. It contributes to automating medical image preprocessing, thereby reducing physicians' workload and mitigating healthcare resource constraints. The model's findings also highlight the need for further research in enhancing model interpretability and extending the model's applicability to diverse medical image modalities. Despite these challenges, the model represents a significant advancement in medical image analysis, laying the groundwork for future research endeavors.

Despite the model's achievements, several limitations and challenges were encountered during its development. The fluctuating IoU and Dice coefficients posed difficulties in achieving consistent segmentation performance. The limited dataset size and computational

constraints also hindered extensive training, leading to potential model performance limitations. Furthermore, attempts to integrate local interpretability techniques like LIME were impeded by mismatches in mask sizes. The study's reliance on the UNet architecture also overlooks emerging approaches like Transformer UNet, which warrants further exploration.

Future research will follow a clear roadmap to address the identified limitations and advance the field of medical image segmentation. It will include acquiring more extensive and more precisely annotated image datasets, which is crucial for model training and evaluation. Improvements in evaluation metrics tailored to real-world segmentation outcomes will also be imperative. Additionally, exploring advanced architectures like Transformer UNet and transitioning to multi-class segmentation can enhance model robustness and applicability. Furthermore, iterative improvements in interpretability techniques akin to large language models could revolutionize medical image interpretation and analysis. In culmination, the development and rigorous testing of hardware specifically designed to optimize the performance of the Attention-UNet algorithm will be pivotal, ensuring that these technological advancements are fully realized and effectively implemented in clinical settings.

References

[1] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," arXiv.org, https://arxiv.org/abs/1505.04597 (accessed Mar. 7, 2024).

[2] V. K. Singh et al., "Breast tumor segmentation in ultrasound images using contextual-information-aware deep adversarial learning framework," Expert Systems with Applications, vol. 162, p. 113870, Dec. 2020. doi:10.1016/j.eswa.2020.113870

[3] W. Al-Dhabyani, M. Gomaa, H. Khaled, and A. Fahmy, "Dataset of breast ultrasound images," Data in Brief, vol. 28, p. 104863, Feb. 2020. doi:10.1016/j.dib.2019.104863

[4] Q. Teng, Z. Liu, Y. Song, K. Han, and Y. Lu, "A survey on the interpretability of deep learning in medical diagnosis," Multimedia Systems, vol. 28, no. 6, pp. 2335–2355, Jun. 2022. doi:10.1007/s00530-022-00960-4

[5] C. Schorr, P. Goodarzi, F. Chen, and T. Dahmen, "Neuroscope: An explainable AI toolbox for semantic segmentation and image classification of convolutional neural nets," Applied Sciences, vol. 11, no. 5, p. 2199, Mar. 2021. doi:10.3390/app11052199

[6] O. Petit, N. Thome, C. Rambour, and L. Soler, "U-Net Transformer: Self and cross attention for medical image segmentation," arXiv.org, https://arxiv.org/abs/2103.06104 (accessed Mar. 26, 2024).

Appendix

The source code for this project is as follows:

```
# **Imports**


# tf-explain is tensorflow library for model interpretation and visualization
from IPython.display import clear_output
!pip install tf_explain
clear_output()

# common
import os
import cv2
import keras
import numpy as np
import pandas as pd
from glob import glob
import tensorflow as tf
import tensorflow.image as tfi

# Data
# loading images and converting them to arrays
# to_categorical is used for one-hot encoding
from keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.utils import to_categorical

# Data Viz
import matplotlib.pyplot as plt

# Model
from keras.models import Model
from keras.layers import Layer
from keras.layers import Conv2D
from keras.layers import Dropout
from keras.layers import UpSampling2D
from keras.layers import concatenate
from keras.layers import Add
from keras.layers import Multiply
from keras.layers import Input
from keras.layers import MaxPool2D
from keras.layers import BatchNormalization

# Callbacks
from keras.callbacks import Callback
from keras.callbacks import EarlyStopping
from keras.callbacks import ModelCheckpoint
from tf_explain.core.grad_cam import GradCAM
```

```
# Metrics
from keras import backend as K
from keras.metrics import MeanIoU

from tensorflow.keras.models import load_model

"""# **Data**

Adding error handling,image prepocessing techinique
"""

# load image from keras and convert it to a NumPy Array
# normalize and resized the image to 0-1 value and the specific size
# round for 4 decimal places
def load_image(image, SIZE):
  return np.round(tfi.resize(img_to_array(load_img(image))/255.,(SIZE, SIZE)),4)

# If trim is specified, it trims the image paths to the specified number.
# It initializes an array (images) to store the processed images based on the specified size and
    whether it's a mask or not.
# It iterates through the image paths, loading and processing each image using the load_image
    function.
# If it's a mask, it takes only the first channel of the processed image.
def load_images(image_paths, SIZE, mask=False, trim=None):
  if trim is not None:
    image_paths = image_paths[:trim]

  if mask:
    images = np.zeros(shape=(len(image_paths), SIZE, SIZE, 1))
  else:
    images = np.zeros(shape=(len(image_paths), SIZE, SIZE, 3))

  for i,image in enumerate(image_paths):
    img = load_image(image,SIZE)
    if mask:
      images[i] = img[:,:,:1]
    else:
      images[i] = img

  return images

def show_image(image, title=None, cmap=None, alpha=1):
  plt.imshow(image, cmap=cmap, alpha=alpha)
  if title is not None:
    plt.title(title)
  plt.axis('off')

def show_mask(image, mask, cmap=None, alpha=0.4):
  plt.imshow(image)
  plt.imshow(tf.squeeze(mask), cmap=cmap, alpha=alpha)
```

```
    plt.axis('off')

def show_img(image, title=None, cmap='viridis', alpha=0.5):
    if len(image.shape) == 3 and image.shape[0] == 1:
        image = image.numpy().squeeze()  # Convert to NumPy array and remove the batch
                    dimension if present

    if len(image.shape) == 2:
        plt.imshow(image, cmap=cmap, alpha=alpha)
    else:
        raise ValueError("Invalid shape for image data")

    if title is not None:
        plt.title(title)
    plt.axis('off')

SIZE = 256

root_path = '../input/breast-ultrasound-images-dataset/Dataset_BUSI_with_GT/'
classes = sorted(os.listdir(root_path))
classes

single_mask_paths = sorted([sorted(glob(root_path + name + "/*mask.png")) for name in
    classes])
double_mask_paths = sorted([sorted(glob(root_path + name + "/*mask_1.png")) for name in
    classes])

image_paths = []
mask_paths = []
for class_path in single_mask_paths:
    for path in class_path:
        img_path = path.replace('_mask','')
        image_paths.append(img_path)
        mask_paths.append(path)

show_image(load_image(image_paths[0], SIZE))

show_mask(load_image(image_paths[0], SIZE), load_image(mask_paths[0], SIZE)[:,:,0],
    alpha=0.6)

"""## **Data Work**"""

images = load_images(image_paths, SIZE)
masks = load_images(mask_paths, SIZE, mask=True)

plt.figure(figsize=(13,8))
for i in range(15):
    plt.subplot(3,5,i+1)
    id = np.random.randint(len(images))
    show_mask(images[id], masks[id], cmap='binary')
```

```
plt.tight_layout()
plt.show()

plt.figure(figsize=(13,8))
for i in range(15):
  plt.subplot(3,5,i+1)
  id = np.random.randint(len(images))
  show_mask(images[id], masks[id], cmap='copper')
plt.tight_layout()
plt.show()

"""# **Encoder**"""

class EncoderBlock(Layer):

  def __init__(self, filters, rate, pooling=True, **kwargs):
    super(EncoderBlock, self).__init__(**kwargs)

    self.filters = filters
    self.rate = rate
    self.pooling = pooling

    #It consists of two convolutional layers with dropout and optional max pooling.
    self.c1 = Conv2D(filters, kernel_size=3, strides=1, padding='same', activation='relu',
                     kernel_initializer='he_normal')
    self.drop = Dropout(rate)
    self.c2 = Conv2D(filters, kernel_size=3, strides=1, padding='same', activation='relu',
                     kernel_initializer='he_normal')
    self.pool = MaxPool2D()

  # The call method performs forward pass and returns either the output
  # or both the output and the skipped input depending on the pooling parameter.
  def call(self, X):
    x = self.c1(X)
    x = self.drop(x)
    x = self.c2(x)
    if self.pooling:
      y = self.pool(x)
      return y, x
    else:
      return x

  def get_config(self):
    base_config = super().get_config()
    return {
      **base_config,
      "filters":self.filters,
      'rate':self.rate,
      'pooling':self.pooling
    }
```

```
"""# **Decoder**"""

class DecoderBlock(Layer):

    def __init__(self, filters, rate, **kwargs):
        super(DecoderBlock, self).__init__(**kwargs)

        self.filters = filters
        self.rate = rate

        self.up = UpSampling2D()
        self.net = EncoderBlock(filters, rate, pooling=False)

    def call(self, X):
        X, skip_X = X
        x = self.up(X)
        c_ = concatenate([x, skip_X])
        x = self.net(c_)
        return x

    def get_config(self):
        base_config = super().get_config()
        return {
            **base_config,
            "filters":self.filters,
            'rate':self.rate,
        }

"""# **Attention Gate**"""

class AttentionGate(Layer):

    def __init__(self, filters, bn, **kwargs):
        super(AttentionGate, self).__init__(**kwargs)

        self.filters = filters
        self.bn = bn

        self.normal = Conv2D(filters, kernel_size=3, padding='same', activation='relu',
                             kernel_initializer='he_normal')
        self.down = Conv2D(filters, kernel_size=3, strides=2, padding='same', activation='relu',
                           kernel_initializer='he_normal')
        self.learn = Conv2D(1, kernel_size=1, padding='same', activation='sigmoid')
        self.resample = UpSampling2D()
        self.BN = BatchNormalization()

    def call(self, X):
        X, skip_X = X
```

```
        x = self.normal(X)
        skip = self.down(skip_X)
        x = Add()([x, skip])
        x = self.learn(x)
        x = self.resample(x)
        f = Multiply()([x, skip_X])
        if self.bn:
            return self.BN(f)
        else:
            return f
        # return f

    def get_config(self):
        base_config = super().get_config()
        return {
            **base_config,
            "filters":self.filters,
            "bn":self.bn
        }

"""# **Custom Callback**"""

def compute_guided_backpropagation(model, image):
    # Define GradientTape to record gradient operations
    with tf.GradientTape() as tape:
        # Watch the input image tensor
        tape.watch(image)
        # Perform a forward pass
        predictions = model(image)
    # Compute gradients of the output with respect to the input image
    gradients = tape.gradient(predictions, image)
    # Discard all negative gradients (set them to zero)
    gradients = tf.where(gradients > 0, gradients, tf.zeros_like(gradients))
    return gradients


class ShowProgress(Callback):
    def __init__(self, images, masks, model):
        self.images = images
        self.masks = masks
        self.model = model
        self.counter = 0

    def visualize_encoder_blocks(self, image):
        # Get the layers for the four Encoder Blocks
        encoder1_layer = self.model.get_layer("Encoder1")
        encoder2_layer = self.model.get_layer("Encoder2")
        encoder3_layer = self.model.get_layer("Encoder3")
        encoder4_layer = self.model.get_layer("Encoder4")
```

```
        image = np.expand_dims(image, axis=0)

        # Create subplots for visualization
        plt.figure(figsize=(14, 5))

        # Visualize the activations for each Encoder Block
        plt.subplot(1, 4, 1)
        plt.title("Encoder Block 1")
        activations = tf.keras.Model(inputs=self.model.inputs,
                                     outputs=encoder1_layer.output)(image)
        plt.imshow(activations[0][0, ..., 0], cmap='viridis')

        plt.subplot(1, 4, 2)
        plt.title("Encoder Block 2")
        activations = tf.keras.Model(inputs=self.model.inputs,
                                     outputs=encoder2_layer.output)(image)
        plt.imshow(activations[0][0, ..., 0], cmap='viridis')

        plt.subplot(1, 4, 3)
        plt.title("Encoder Block 3")
        activations = tf.keras.Model(inputs=self.model.inputs,
                                     outputs=encoder3_layer.output)(image)
        plt.imshow(activations[0][0, ..., 0], cmap='viridis')

        plt.subplot(1, 4, 4)
        plt.title("Encoder Block 4")
        activations = tf.keras.Model(inputs=self.model.inputs,
                                     outputs=encoder4_layer.output)(image)
        plt.imshow(activations[0][0, ..., 0], cmap='viridis')

        plt.tight_layout()
        plt.show()

    def visualize_gradcam(self, image, mask, pred_mask):
        # Initialize GradCAM explainer
        exp = GradCAM()

        # Use GradCAM to explain the image and obtain the gradient-weighted class activation
                                     map for each Attention layer
        cam1 = exp.explain(
            validation_data=(image[np.newaxis, ...], mask),
            class_index=1,
            layer_name='Attention1',
            model=self.model
        )
        cam2 = exp.explain(
            validation_data=(image[np.newaxis, ...], mask),
            class_index=1,
            layer_name='Attention2',
            model=self.model
```

```
    )
    cam3 = exp.explain(
        validation_data=(image[np.newaxis, ...], mask),
        class_index=1,
        layer_name='Attention3',
        model=self.model
    )
    cam4 = exp.explain(
        validation_data=(image[np.newaxis, ...], mask),
        class_index=1,
        layer_name='Attention4',
        model=self.model
    )

     # Visualize the GradCAM results

    plt.figure(figsize=(14,5))

    plt.subplot(1,4,1)
    show_image(cam1,title="GradCAM Attention1")

    plt.subplot(1,4,2)
    show_image(cam2,title="GradCAM Attention2")

    plt.subplot(1,4,3)
    show_image(cam3,title="GradCAM Attention3")

    plt.subplot(1,4,4)
    show_image(cam4,title="GradCAM Attention4")


    plt.tight_layout()
    plt.show()


def visualize_saliency_maps_and_guided_backpropagation(self, image, mask, pred_mask):
    # Compute the gradients of the output with respect to the input image
    input_tensor = tf.convert_to_tensor(image[np.newaxis, ...], dtype=tf.float32)
    with tf.GradientTape() as tape:
        tape.watch(input_tensor)
        predictions = self.model(input_tensor)

    gradients = tape.gradient(predictions, input_tensor)
    gradients_abs = tf.abs(gradients)
    saliency_map = tf.reduce_max(gradients_abs, axis=-1)

    # Compute guided backpropagation gradients
    GBgradients = compute_guided_backpropagation(self.model, input_tensor)
    # Compute saliency map from guided backpropagation gradients
    GBsaliency_map = np.abs(GBgradients[0]).max(axis=-1)
```

```
    # Visualize the saliency map
    plt.figure(figsize=(14, 5))
    plt.subplot(1, 4, 1)
    plt.title("Original Mask")
    show_mask(image, mask, cmap='copper')

    plt.subplot(1, 4, 2)
    plt.title("Predicted Mask")
    show_mask(image, pred_mask, cmap='copper')

    plt.subplot(1, 4, 3)
    plt.title("Saliency Map")
    show_img(saliency_map, title="Saliency Map", cmap='hot', alpha=1)

    plt.subplot(1, 4, 4)
    plt.title("Guided Backpropagation")
    plt.imshow(GBsaliency_map, cmap='hot')
    plt.axis('off')

    plt.tight_layout()
    plt.show()

def on_epoch_end(self, epoch, logs=None):
    self.counter += 1
    if self.counter % 10 == 0:
        # Reset counter
        self.counter = 0

        # Randomly select an index for an image
        id = np.random.randint(200)

        # Choose an image
        image = self.images[id]
        mask = self.masks[id]

        # Predict the mask using the model
        pred_mask = self.model.predict(image[np.newaxis, ...])

        plt.figure(figsize=(14, 5))
        plt.subplot(1, 2, 1)
        plt.title("Original Mask")
        show_mask(image, mask, cmap='copper')

        plt.subplot(1, 2, 2)
        plt.title("Predicted Mask")
        show_mask(image, pred_mask, cmap='copper')

        plt.tight_layout()
```

```
        plt.show()

"""# **Dice Metric**"""

def dice_coefficient(y_true, y_pred):
    smooth = 0.000000001
    intersection = K.sum(K.abs(y_true * y_pred), axis=-1)
    union = K.sum(y_true, axis=-1) + K.sum(y_pred, axis=-1) - intersection
    return (2.0 * intersection + smooth) / (intersection + union + smooth)

"""# **IoU Metric**"""

def iou_coefficient(y_true, y_pred):
    smooth = 0.000000001
    intersection = K.sum(K.abs(y_true * y_pred), axis=-1)
    union = K.sum(y_true, axis=-1) + K.sum(y_pred, axis=-1) - intersection
    return (intersection + smooth) / (union + smooth)

"""# **Attention UNet**"""

# Custom contrast enhancement function using OpenCV
def enhance_contrast(image):
    # Convert the image to uint8
    image = (image * 255).astype(np.uint8)

    # Convert the image to LAB color space
    lab = cv2.cvtColor(image, cv2.COLOR_RGB2LAB)

    # Split the LAB image into L, A, and B channels
    l, a, b = cv2.split(lab)

    # Apply CLAHE (Contrast Limited Adaptive Histogram Equalization) to the L channel
    clahe = cv2.createCLAHE(clipLimit=3.0, tileGridSize=(8, 8))
    cl = clahe.apply(l)

    # Merge the enhanced L channel with the original A and B channels
    enhanced_lab = cv2.merge([cl, a, b])

    # Convert the LAB image back to RGB
    enhanced_image = cv2.cvtColor(enhanced_lab, cv2.COLOR_LAB2RGB)

    return enhanced_image

try:
    # load model
    model = load_model('/content/drive/MyDrive/AttentionCustomUNet.h5',
                custom_objects={"iou_coefficient": iou_coefficient, "dice_coefficient":
                dice_coefficient})
    print("load previous model successfully")
except (OSError, IOError):
```

```
# failed to load the model
# Apply contrast enhancement to each image in the dataset
enhanced_images = [enhance_contrast(image) for image in images]

# Convert the list of enhanced images to a numpy array
enhanced_images = np.array(enhanced_images)

# Inputs from last 3 element
# (batch_size, height, width, channels)
input_layer = Input(shape=images.shape[-3:])

# Encoder
# p1, p2, p3, and p4 represent the pooled features at each stage.
# c1, c2, c3, and c4 represent the features before pooling,
# which are used for skip connections in the decoder.
p1, c1 = EncoderBlock(32,0.1, name="Encoder1")(input_layer)
p2, c2 = EncoderBlock(64,0.1, name="Encoder2")(p1)
p3, c3 = EncoderBlock(128,0.2, name="Encoder3")(p2)
p4, c4 = EncoderBlock(256,0.2, name="Encoder4")(p3)

# Encoding
# This line defines an additional encoding block without pooling, resulting in the encoding
        tensor.
encoding = EncoderBlock(512,0.3, pooling=False, name="Encoding")(p4)

# Attention + Decoder

a1 = AttentionGate(256, bn=True, name="Attention1")([encoding, c4])
d1 = DecoderBlock(256,0.2, name="Decoder1")([encoding, a1])

a2 = AttentionGate(128, bn=True, name="Attention2")([d1, c3])
d2 = DecoderBlock(128,0.2, name="Decoder2")([d1, a2])

a3 = AttentionGate(64, bn=True, name="Attention3")([d2, c2])
d3 = DecoderBlock(64,0.1, name="Decoder3")([d2, a3])


a4 = AttentionGate(32, bn=True, name="Attention4")([d3, c1])
d4 = DecoderBlock(32,0.1, name="Decoder4")([d3, a4])

# Output
# The output layer with a convolutional layer having one filter, using the sigmoid activation
        function.
# It produces the final segmentation mask.
output_layer = Conv2D(1, kernel_size=1, activation='sigmoid', padding='same')(d4)

# Model
model = Model(
   inputs=[input_layer],
   outputs=[output_layer]
```

```
    )

    # Compile
    model.compile(
        loss='binary_crossentropy',
        optimizer='adam',
        metrics=['accuracy', iou_coefficient, dice_coefficient]
    )

    print("create new model succesfully")

# Callbacks
# saves the model's weights to a file during training
cb = [
    # EarlyStopping(patience=3, restore_best_weight=True), # With Segmentation I trust on eyes
                rather than on metrics
    ModelCheckpoint("AttentionCustomUNet.h5", save_best_only=True),
    ShowProgress(images, masks, model)
]

"""# **Training**"""

# Config Training
BATCH_SIZE = 8
SPE = len(images)//BATCH_SIZE

# Training
results = model.fit(
    images, masks,
    validation_split=0.2,
    epochs=250, # 15 will be enough for a good Model for better model go with 20+
    steps_per_epoch=SPE,
    batch_size=BATCH_SIZE,
    callbacks=cb
)

# Evaluation

loss, accuracy, iou, dice = results.history['loss'], results.history['accuracy'],
      results.history['iou_coefficient'], results.history['dice_coefficient']
val_loss, val_accuracy, val_iou, val_dice = results.history['val_loss'],
      results.history['val_accuracy'], results.history['val_iou_coefficient'],
      results.history['val_dice_coefficient']

plt.figure(figsize=(30, 12))

plt.subplot(1, 4, 1)
plt.title("Model Loss")
plt.plot(loss, label="Training")
plt.plot(val_loss, label="Validation")
```

```
plt.legend()
plt.grid()

plt.subplot(1, 4, 2)
plt.title("Model Accuracy")
plt.plot(accuracy, label="Training")
plt.plot(val_accuracy, label="Validation")
plt.legend()
plt.grid()

plt.subplot(1, 4, 3)
plt.title("Model IoU")
plt.plot(iou, label="Training")
plt.plot(val_iou, label="Validation")
plt.legend()
plt.grid()

plt.subplot(1, 4, 4)
plt.title("Model Dice")
plt.plot(dice, label="Training")
plt.plot(val_dice, label="Validation")
plt.legend()
plt.grid()

plt.tight_layout()
plt.show()

numPic = 20
num_samples_per_image = 100
window_size = (50, 50)
stride = (30, 30)
progress = ShowProgress(images, masks, model)

for i in range(numPic):
    plt.figure(figsize=(20, 25))
    id = np.random.randint(len(images))
    image = images[id]
    mask = masks[id]
    pred_mask = model.predict(image[np.newaxis, ...])

    plt.subplot(1, 5, 1)
    plt.title("Original Mask")
    show_mask(image, mask)

    plt.subplot(1, 5, 2)
    plt.title("Predicted Mask")
    show_mask(image, pred_mask)

    plt.subplot(1, 5, 3)
    processed_mask = (pred_mask > 0.5).astype('float')
```

```
    plt.title("Processed Mask")
    show_mask(image, processed_mask)

    plt.subplot(1, 5, 4)
    plt.title("Confidence Map")
    plt.imshow(pred_mask[0], cmap='gray')
    plt.colorbar(shrink=0.1)

    plt.subplot(1, 5, 5)
    plt.title("Occlusion Sensitivity Map")
    occlusion_map = np.zeros((image.shape[0], image.shape[1]), dtype=np.float32)

    # traverse all the windows
    for y in range(0, image.shape[0] - window_size[0] + 1, stride[0]):
        for x in range(0, image.shape[1] - window_size[1] + 1, stride[1]):
            occluded_image = np.copy(image)
            # particularly window
            occluded_image[y:y+window_size[0], x:x+window_size[1]] = 0

            pred_original = model.predict(occluded_image[np.newaxis, ...])
            pred_occluded = model.predict(image[np.newaxis, ...])

            sensitivity_map = np.abs(pred_original[0, y:y+window_size[0], x:x+window_size[1], 0] -
                                     pred_occluded[0, y:y+window_size[0], x:x+window_size[1], 0])

            occlusion_map[y:y+window_size[0], x:x+window_size[1]] += sensitivity_map

    plt.imshow(occlusion_map, cmap='hot')
    plt.colorbar(shrink=0.1)

    plt.tight_layout()
    plt.show()

    progress.visualize_encoder_blocks(image)
    progress.visualize_gradcam(image, mask, pred_mask)

    progress.visualize_saliency_maps_and_guided_backpropagation(image, mask,
pred_mask)
```