

**ENHANCED CLASSIFICATION OF COLORECTAL  
TRACT ABNORMALITIES USING TRANSFER  
LEARNING ON ENDOSCOPIC IMAGES.**

**A PROJECT WORK II PHASE I REPORT**

Submitted by

**BHUVANESHWARI C  
21ADR009**

**SNEHAVARSHINI S  
21ADR048**

**SOUNDHARYA M  
21ADR050**

*in partial fulfillment of the requirements*

*for the award of the degree*

*of*

**BACHELOR OF TECHNOLOGY  
IN**

**ARTIFICIAL INTELLIGENCE  
AND DATA SCIENCE**

**DEPARTMENT OF ARTIFICIAL INTELLIGENCE**



**KONGU ENGINEERING COLLEGE  
(Autonomous)**

**PERUNDURAI, ERODE-638 060**

**OCTOBER 2024**

**DEPARTMENT OF ARTIFICIAL INTELLIGENCE**  
**KONGU ENGINEERING COLLEGE**  
(Autonomous)

**PERUNDURAI, ERODE – 638 060**

**OCTOBER 2024**

**BONAFIDE CERTIFICATE**

This is to certify that the Project Report titled **ENHANCED CLASSIFICATION OF COLORECTAL TRACT ABNORMALITIES USING TRANSFER LEARNING ON ENDOSCOPIC IMAGES** is the bonafide record of project work done by **BHUVANESHWARI C (21ADR009)**, **SNEHAVARSHINI S (21ADR048)**, **SOUNDHARYA M (21ADR050)** in partial fulfillment of the requirements for the award of Degree of Bachelor of Technology in Artificial Intelligence and Data Science of Anna University, Chennai during the year 2024-2025.

**SUPERVISOR**

**HEAD OF THE DEPARTMENT**

(Signature with seal)

Date:

Submitted for the end semester viva voce examination held on \_\_\_\_\_

**INTERNAL EXAMINER**

**EXTERNAL EXAMINER**

**DEPARTMENT OF ARTIFICIAL INTELLIGENCE**  
**KONGU ENGINEERING COLLEGE**  
**(Autonomous)**

**PERUNDURAI, ERODE – 638 060**

**OCTOBER 2024**

**DECLARATION**

We affirm that the Project Report titled **ENHANCED CLASSIFICATION OF COLORECTAL TRACT ABNORMALITIES USING TRANSFER LEARNING ON ENDOSCOPIC IMAGES** being submitted in partial fulfillment of the requirements for the award of Bachelor of Technology is the original work carried out by us. It has not formed part of any other project report or dissertation based on which a degree or award was conferred on an earlier occasion on this or any other candidate.

**Date:**

**BHUVANESHWARI C**

(21ADR009)

**SNEHAVARSHINI S**

(21ADR048)

**SOUNDHARYA M**

(21ADR050)

I certify that the declaration made by the above candidate is true to the best of my knowledge.

**Date:**

**Name & Signature of the Supervisor with seal**

## ABSTRACT

The goal of this research is to significantly enhance the classification of colorectal tract abnormalities from endoscopic images using transfer learning and attention modules. This study mainly focuses on the resnet model and enhances its classification throughput on endoscopic images. By incorporating attention mechanisms such as BAM and CBAM into the ResNet model, the study aims to improve the model's ability to focus on critical image regions, leading to more accurate and reliable classifications. This enhancement is expected to provide medical professionals with more precise diagnostic tools, helping them identify abnormalities early, optimize treatment plans, and improve patient outcomes. The study will also rigorously evaluate the model's performance using accuracy, precision, and F1-score metrics, comparing the results to baseline models without attention modules. Through this comprehensive approach, the research seeks to advance deep learning-based solutions for medical image analysis and contribute to better healthcare in colorectal cancer detection. In order to improve model generalization even further, the study will investigate the effects of various hyperparameter tuning methods and data augmentation approaches. By using cross-validation, the study hopes to guarantee the suggested model's resilience across various datasets, lowering overfitting and boosting its therapeutic usefulness. The ultimate goal of this research is to close the gap between state-of-the-art AI models and useful, real-world medical applications in the detection of colorectal cancer.

## ACKNOWLEDGEMENT

First and foremost, we acknowledge the abundant grace and presence of the almighty throughout different phases of the project and its successful completion.

We wish to express our sincere gratitude to our honorable Correspondent **Thiru.A.K.ILANGO B.Com., M.B.A., LLB.,** and other trust members for having providedus with all the necessary infrastructures to undertake this project.

We extend our hearty gratitude to our honorable Principal **Dr.V.BALUSAMY B.E.(Hons)., MTech., Ph.D.,** for his consistent encouragement throughout our college days.

We would like to express our profound interest and sincere gratitude to our respected Head of the department **Dr.C.S.KANIMOZHISELVI ME., Ph.D.,** for her valuable guidance.

A special debt is owed to the project coordinator **Ms. S. SANTHIYA B.E., M.E.,** Assistant Professor, Department of Artificial Intelligence for her encouragement and valuable advice that made us carry out project work successfully.

We extend our sincere gratitude to our beloved guide **Ms. O. ABHILA ANJU B.Tech., M.E.,** Department of Artificial Intelligence for her ideas and suggestions, which have been very helpful to complete the project.

We are grateful to all the faculty and staff members of the Department of Artificial Intelligence and persons who directly and indirectly supported this project.

## TABLE OF CONTENTS

CHAPTER No.	TITLE	PAGE No.
	<b>ABSTRACT</b>	<b>iv</b>
	<b>LIST OF TABLES</b>	<b>viii</b>
	<b>LIST OF FIGURES</b>	<b>ix</b>
	<b>LIST OF ABBREVIATIONS</b>	<b>xi</b>
<b>1</b>	<b>INTRODUCTION</b>	<b>1</b>
	1.1 INTRODUCTION	1
	1.2 OBJECTIVE	2
	1.3 SCOPE	2
<b>2</b>	<b>SYSTEM ANALYSIS</b>	<b>3</b>
	2.1 LITERATURE REVIEW	3
	2.2 SUMMARY	5
<b>3</b>	<b>SYSTEM REQUIREMENTS</b>	<b>6</b>
	3.1 HARDWARE REQUIREMENTS	6
	3.2 SOFTWARE REQUIREMENTS	6
	3.3 SOFTWARE DESCRIPTION	6
	3.3.1 Python	6
	3.3.2 Google Colab Notebook	7
<b>4</b>	<b>PROPOSED SYSTEM</b>	<b>8</b>
	4.1 SYSTEM ARCHITECTURE	8
	4.2 MODULE DESCRIPTION	9
	4.2.1 Dataset Collection	9
	4.2.2 Dataset pre-processing	11
	4.2.3 Implementation of Transfer Learning Algorithms	12

	4.2.3.1 ResNet	12
	4.2.3.2 ResNet with BAM	14
	4.2.3.3 ResNet with CBAM	16
	<b>4.3 GRAD CAM VISUALIZATION</b>	19
	<b>4.4 VISUALIZATION</b>	21
<b>5</b>	<b>PERFORMANCE ANALYSIS</b>	<b>24</b>
<b>6</b>	<b>RESULTS AND DISCUSSION</b>	<b>25</b>
<b>7</b>	<b>CONCLUSION</b>	<b>26</b>
<b>8</b>	<b>APPENDICES</b>	<b>27</b>
	8.1 APPENDIX – 1 CODING	27
	8.2 APPENDIX – 2 SCREENSHOTS	39
	<b>REFERENCES</b>	<b>40</b>

## LIST OF TABLES

<b>TABLE No.</b>	<b>TABLE NAME</b>	<b>PAGE No.</b>
4.1	Description of Kvasir dataset	<b>9</b>
5.1	Comparison of different models	<b>24</b>



## LIST OF FIGURES

<b>FIGURE No.</b>	<b>FIGURE NAME</b>	<b>PAGE No.</b>
4.1	Proposed model workflow	8
4.2	Dyed – lifted – Polyps	9
4.3	Dyed – resection - margins.	9
4.4	Esophagitis	10
4.5	Normal Cecum	10
4.6	Normal Pylorus	10
4.7	Normal Z line.	10
4.8	Polyps	10
4.9	Ucerative Colitis.	10
4.10	Load_image function	11
4.11	Encode_label function	11
4.12	Simple architecture of ResNet with identity and Residual Blocks.	13
4.13	Basic attention module	14
4.14	Work flow of an Attention module	14
4.15	Positioning of BAM module in ResNet	16
4.16	Positioning of CBAM module in ResNet	18
4.17.1	Original image used for visualizing GRAD-CAM.	19
4.17.2	Image processed by ResNet Model.	19
4.17.3	Image processed by BAM Model.	19
4.17.4	Image processed by CBAM Model.	19
4.18	Plotting of ResNet Model's accuracy	21

## LIST OF FIGURES

<b>FIGURE No.</b>	<b>FIGURE NAME</b>	<b>PAGE No.</b>
4.19	Plotting of ResNet Model's loss	21
4.20	Plotting of ResNet with BAM Model's accuracy	22
4.21	Plotting of ResNet with BAM Model's loss	22
4.22	Plotting of ResNet with CBAM Model's accuracy	23
4.23	Plotting of ResNet with CBAM Model's loss.	23
6.1	Comparison of various models	25
A2.1	BAM Model Architecture	39
A2.2	CBAM Model Architecture	39
A2.3	Test Accuracy of Resnet with BAM model	40
A2.4	Test Accuracy of Resnet with CBAM model	40

## LIST OF ABBREVIATIONS

<b>BAM</b>	:	Bottleneck Attention Modules
<b>CBAM</b>	:	Convolutional Block Attention Module
<b>GRAD – CAM</b>	:	Gradient-Weighted Class Activation Mapping

# CHAPTER 1

## INTRODUCTION

### 1.1 INTRODUCTION

Colorectal cancer is one of the leading causes of cancer-related deaths worldwide, making early detection and diagnosis critical for improving patient outcomes. Endoscopic images are commonly used by medical professionals to detect abnormalities in the colorectal tract, such as polyps, tumors, and other precancerous conditions. However, manual analysis of these images can be time-consuming and prone to errors, making automated detection systems highly valuable in clinical settings. With approximately 31% of colorectal cancer cases progressing without timely detection, developing advanced diagnostic tools can significantly improve survival rates.

The objective of this study is to develop a model for early detection and classification of colorectal tract abnormalities from endoscopic images. This is achieved by leveraging deep learning techniques with attention modules, specifically the ResNet model enhanced by BAM and CBAM. By integrating transfer learning with attention mechanisms, the proposed method aims to accurately classify endoscopic images based on abnormality types. This integration facilitates a more focused understanding of the regions contributing to classification, improving the model's precision in detecting abnormalities. Additionally, the study incorporates GRAD CAM to visualize the critical regions of the image that contribute most to the classification decision, making the process more transparent for healthcare providers.

To train and evaluate the model, a dataset consisting of 4,000 training images and 1,000 testing images from the Kvasir dataset is utilized. The performance of the attention-enhanced ResNet model is compared with baseline models lacking attention modules to determine the most effective architecture for classifying colorectal tract abnormalities. Metrics such as accuracy, precision, and F1-score are employed to assess the model's effectiveness in improving detection rates.

To further enhance the model's accuracy, techniques such as data augmentation and weighted averaging are applied. Additionally, the study investigates different abnormality types in endoscopic images to enable better understanding and classification of complex

cases. By combining deep learning models with attention mechanisms, this research aims to advance automated diagnostic tools for colorectal cancer detection, ultimately contributing to better patient outcomes and improved clinical decision-making. Evaluation of the model is conducted using standard metrics, ensuring a comprehensive assessment of its performance in real-world applications. Through these methods, the study offers the potential to revolutionize early detection strategies in colorectal cancer diagnostics.

## **1.2 OBJECTIVE**

- To develop a classification model that accurately detects and classifies colorectal tract abnormalities from endoscopic images for early diagnosis.
- To analyze the efficiency of attention modules, such as BAM and CBAM, integrated with the ResNet model using evaluation metrics like accuracy, precision, and F1-score.
- To compare the performance of the attention-enhanced ResNet model with baseline models to identify the better-performing architecture.
- To incorporate visualization techniques like GRAD CAM to identify critical regions contributing to the classification.

## **1.3 SCOPE**

- Conduct feature importance analysis using attention mechanisms and GRAD CAM to identify the key image regions driving the classification of abnormalities.
- The integration of transfer learning and attention modules enhances classification accuracy, aiding in real-world decision-making for colorectal cancer detection and treatment strategies.

## CHAPTER 2

### SYSTEM ANALYSIS

#### 2.1 LITERATURE REVIEW

In 2023, Noor et al. [1] undertook a study that presents a complete summary of current improvements in the categorization of gastrointestinal (GI) illnesses using deep learning models, concentrating on wireless capsule endoscopy (WCE) pictures. In order to increase the precision of GI illness identification, the study investigates a number of strategies, such as transfer learning and enhanced contrast-enhancement approaches. In order to address the poor contrast in WCE pictures, a prevalent problem in medical imaging, a unique brightness-controlled contrast enhancement employing a genetic algorithm was presented. The accuracy of diagnosis was previously hindered by intra-class and inter-class similarities, but this technique considerably decreased them. In order to guarantee robustness, further data augmentation techniques were used, along with machine learning classifiers for the ultimate categorization of the illness. The study's findings demonstrated a significant increase in classification accuracy, precision, recall, and F1 score, highlighting the value of integrating contrast-enhancement methods with deep learning models for the diagnosis of gastrointestinal diseases (GI using dcnn).

In 2023, Noor et al. [1] carried out a research to investigate the automation of endoscopic picture-based gastrointestinal illness detection. Convolutional Neural Networks (CNNs) were utilized, as they are well-known for their efficiency in image identification tasks and their capacity to collect local characteristics. The Kvasir dataset—which includes endoscopic pictures from eight distinct classes, including Z-line, Polyps, and Esophagitis—was employed by the authors. A residual model based on ResNet50 and a dense model based on DenseNet121 were both refined. The residual model earned 87.8% accuracy after 20 epochs of training, whereas the dense model scored 86.9%. These models act as decision-support tools, particularly in underdeveloped nations where there is a dearth of qualified medical professionals. Additionally, the study demonstrated how deep learning models may be used to automate medical identification jobs, relieving the workload of human specialists and offering trustworthy diagnostic assistance.

In 2023, Sivari et al. [3] undertook research on Disorders of the gastrointestinal (GI) tract which are common and that have a major impact on patient's health. For proper therapy and diagnosis, exact GI findings of categorization and identification from endoscopic images are essential. It can be laborious and subjective, but Manual analysis is a common component of the traditional approach. Direct visualization of the GI tract is made possible by endoscopic imaging, which offers comprehensive insights into diseases like polyps, tumors, and inflammation. In the diagnostic process, Automated classification methods are critical to assist doctors, given these findings. Combining multiple kinds of algorithms like (CNNs with conventional ML techniques) used to create hybrid models has been shown to increase performance in many categorization tasks and its performance, including those involving results from that gastrointestinal system. Metrics like precision, accuracy, recall, F1-score, and area under the receiver operating characteristic curve (AUC-ROC) will help to evaluate the performance

In 2023 Sujatha et al. [5] has discovered Globally over colorectal cancer (CRC) which continues to be the primary cause of cancer-related death. A early detection and precise diagnosis are essential for bettering the health of patient. The gold standard for identifying colorectal cancer (CRC) has histopathological examination for tissue samples. Recent developments in deep learning, particularly with regard to Generative Adversarial Networks (GANs), have effective means of improving the processing of histopathology images. Histopathology is the study of tissue samples under a microscope which helps in finding malignant cells. GAN is becoming more effective substitutes in producing high-quality synthetic images and enhancing certain segmentation tasks, but CNNs are frequently utilized. The suggested framework improves the segmentation of colorectal cancer in histological pictures by utilizing both DC-GAN and VAE-GAN. The design has a specific to increase cancer prediction accuracy and robustness.

In 2023 Mohapatra et al. [6] has undertaken the research of finding the abnormalities of the gastrointestinal system (GI) such as tumors, inflammation, and polyps which are common and if not it has been identified in a timely manner it can lead to major health consequences. Endoscopic imaging offers a clear and comprehensive picture of the gastrointestinal tract, but proper identification and categorization of the vast number of pictures necessitates the use of effective automated systems. Promising the pathways for enhancing diagnostic accuracy have been presented by recent developments in signal processing and with certain deep learning techniques, particularly through the combination of deep convolutional neural networks (CNNs) with empirical wavelet transform

(EWT).Endoscopy is a treatment which helps to identify the pictures of GI system that can be seen using a minimally invasive. It has the capacity of producing high-resolution pictures which may be examined for several kinds of anomalies. However, automated analysis is difficult due to the complexity of anatomical structures and the diversity in image quality.

In 2023 Tang et al.[7] has discovered about Gastrointestinal (GI) endoscopic screening which is commonly used to diagnose GI diseases but relies on experienced endoscopists, with missed diagnoses reported in about 20% of cases. AI, particularly convolutional neural networks (CNNs), shows promise in improving diagnosis by automating lesion detection. CNNs have been successfully applied in medical imaging, such as prostate cancer and skin lesions. However, CNNs struggle to capture long-range dependencies, essential for accurate GI tract lesion classification and segmentation. This limitation can hinder the differentiation and boundary identification of lesions, especially those with similar appearances in endoscopic images.

## **2.2 SUMMARY**

The suggested system design incorporates attention mechanisms like the Convolutional Block Attention Module (CBAM) and Bottleneck Attention Module (BAM) to improve the ResNet algorithm for medical image processing. By assisting the model in concentrating on the most pertinent aspects of the pictures, these modules increase the accuracy of categorization. Preprocessing operations are performed on the dataset (e.g., Kvasir) that include pictures of gastrointestinal anomalies, such as scaling and normalization. To enhance performance, BAM and CBAM are combined with the pre-trained ResNet-50 model by transfer learning. Grad-CAM is used to assess the efficacy of these improvements by seeing the regions of the pictures that are most important to the model's predictions. According to test findings, ResNet with CBAM obtains 84.375% accuracy, which is much higher than the baseline ResNet's 40.5% accuracy. ResNet with BAM achieves 77% accuracy.



## **CHAPTER 3**

### **SYSTEM REQUIREMENTS**

#### **3.1 HARDWARE REQUIREMENTS**

CPU type	:	Intel corei5 processors
Ram size	:	8 GB
Hard disk capacity	:	500 GB

#### **3.2 SOFTWARE REQUIREMENTS**

Operating System	:	Windows 10
Language	:	Python (version 3+)
Tool	:	Google Colab

#### **3.3 SOFTWARE DESCRIPTION**

##### **3.3.1 Python**

Python is simple to learn and a powerful programming language. Its extensive use of information structures and straightforward but effective approach to handling object-oriented programming. Python is a perfect language for prearranging and swift application advancement in many fields in the best stages due to its refined phrase structure and imperative composing, which are close to its made sense nature. The core ideas and capabilities of Python 3 are covered in this essay. A large number of the functions that come with Python when it is installed make up its standard library. There are numerous additional libraries that the Python language can use to experiment with more things available online. These libraries provide it strength and enable it to do a wide range of tasks. In the provided Python code snippets, several essential libraries and methods are employed to perform a range of tasks. The Pandas library is utilized for

data manipulation and analysis, particularly for reading and processing data from CSV files, enabling data preprocessing. Scikit- Learn (sklearn) comes into play as a machine learning library, assisting in model selection, data splitting for training and testing, and the calculation of accuracy scores.

### **3.3.2 Google Colab Notebook**

Users may easily execute and exchange code with Google Colaboratory, a cloud-based tool for working in Jupyter notebooks. It provides a deep learning, machine learning, and data analysis environment powered by GPUs and TPUs. Users may keep their work on Google Drive and collaborate in real time. The sharing of Colab notebooks is similar to that of Sheets or Google Docs. Either follow these instructions for sharing files from Google Drive, or click the Share icon in the top right corner of any Colab notebook. For Colab laptops, Google Drive search functions are provided. Clicking the Colab logo in the upper left corner of the notebook view will make all of the notebooks in Drive visible. Choosing File > Open Notebook will allow you to can furthermore search for notebooks that you have just opened. Code is executed on your account-only virtual computer. The Colab service enforces a limited lifetime for virtual computers, after which they are removed after a period of inactivity. You may access any Colab notebook you've made using the File menu in Colab, by following these instructions, or by downloading it from Google Drive.

## CHAPTER 4

### PROPOSED SYSTEM

#### 4.1 SYSTEM ARCHITECTURE

The proposed methodology shown in Fig 4.1 aims to enhance the ResNet algorithm for medical image analysis by integrating attention mechanisms, specifically the Bottleneck Attention Module (BAM) and the Convolutional Block Attention Module (CBAM). Given ResNet's effectiveness in handling medical images, incorporating these attention mechanisms will improve the classification process by allowing the model to focus on the most relevant features of the images.

Prior to model training, a preprocessing phase is implemented, which includes techniques such as resizing, flipping, and rescaling the images to improve image quality and analysis. These preprocessing steps ensure that the input images are uniform and enhance the model's ability to learn from the data effectively.

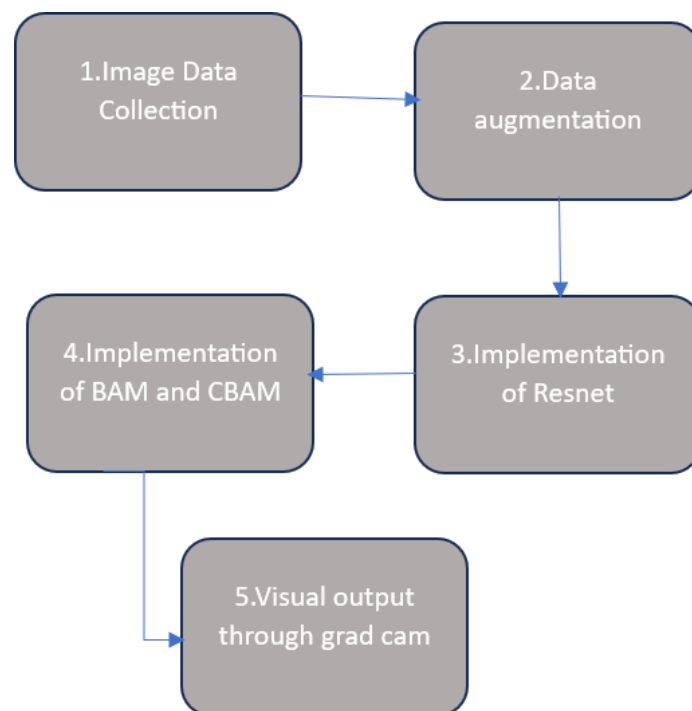


Fig. 4.1 Proposed model workflow

To analyze the learning outcomes of these enhanced models, Gradient-weighted Class Activation Mapping (Grad-CAM) is employed as a visualization tool.

Grad-CAM helps in interpreting the model's predictions by highlighting the regions of the input images that contribute most to the final classification decisions. This approach not only provides insights into the model's decision-making process but also aids in validating the effectiveness of the attention mechanisms in enhancing classification performance.


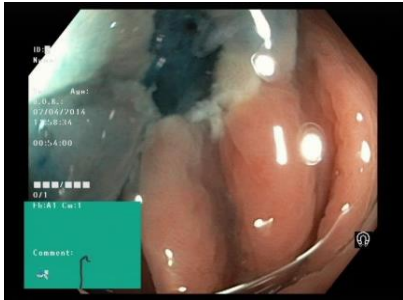
In summary, this methodology outlines the enhancement of the ResNet algorithm with BAM and CBAM, along with preprocessing techniques to improve image quality, followed by the application of Grad-CAM to visualize and understand the model's learning in the context of medical image classification.






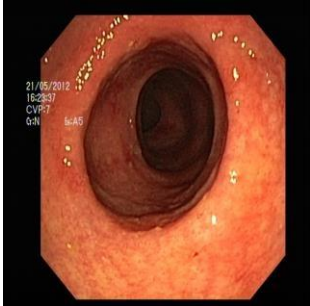
## 4.2 MODULE DESCRIPTION

### 4.2.1 Dataset Collection

The dataset is taken from kaggle named Kvasir dataset. It is widely used medical image dataset for research and analysis for detecting gastrointestinal abnormalities and anomalies. various examinations of the gastrointestinal tract, including the esophagus, stomach, and intestines. There are 8 different types of abnormalities were given in the dataset. The reference images for 8 types of abnormalities is included from Fig 4.2 – Fig 4.9.

Table 4.1 Description of Kvasir dataset.

Dyed – lifted - Polyps	Dyed – resection - margins
Polyps lifted and highlighted with dye during endoscopy.	Areas where tissue has been removed, visible with dye.
 <p data-bbox="395 1944 794 1977">Fig 4.2 Dyed – lifted – Polyps.</p>	 <p data-bbox="940 1944 1394 1977">Fig 4.3 Dyed – resection - margins.</p>

Esophagitis	Normal-cecum
<p>Inflammation or irritation of the esophagus.</p>	<p>Normal appearance of the cecum in the large intestine.</p>
 <p>Fig 4.4 Esophagitis</p>	 <p>Fig 4.5 Normal Cecum</p>
Normal - pylorus	Normal Z line
<p>Normal appearance of the stomach's exit to the small intestine.</p>	<p>Normal appearance of the junction between the esophagus and stomach.</p>
 <p>Fig 4.6 Normal Pylorus</p>	 <p>Fig 4.7 Normal Z line.</p>
Polyps	Ucerative – Colitis
<p>Abnormal growths in the colon or rectum.</p>	<p>Inflammation and ulcers in the colon's lining.</p>
 <p>Fig 4.8 Polyps</p>	 <p>Fig 4.9 Ucerative Colitis.</p>

### 4.2.2 Data pre-processing

```
def load_image(image_path, label, img_size=(224, 224)):
    image = tf.io.read_file(image_path)
    image = tf.image.decode_jpeg(image, channels=3)
    image = tf.image.resize(image, img_size)
    image = image / 255.0 # Normalize to [0, 1]
    return image, label
```

Fig 4.10 load\_image function

This function takes an image path and a label as inputs, reads the image file, and decodes it into a tensor format. It then resizes the image to the specified dimensions (defaulting to 224x224 pixels) and normalizes the pixel values to a range between 0 and 1 as sh. This normalization is important as it helps improve the model's performance during training by ensuring that the input data is on a consistent scale.

```
def encode_labels(label, class_names):
    label = tf.argmax(tf.constant(class_names) == label)
    return label

class_names = data['label'].unique().tolist()

train_data['label'] = train_data['label'].apply(lambda x: encode_labels(x, class_names))
test_data['label'] = test_data['label'].apply(lambda x: encode_labels(x, class_names))
```

Fig 4.11 encode\_label function

The 'encode\_labels', which converts categorical labels into numerical indices for each class in the dataset. By comparing the provided label against a list of unique class names, this function generates an integer index for each label, allowing the model to interpret the labels correctly during training. The transformation shown in Fig 4.11 is crucial for ensuring that the data is in the appropriate format for effective training and evaluation of the machine learning model.

### 4.2.3 Implementation of Transfer Learning Model.

Transfer learning is a category in deep learning that uses a pre-trained model, developed for one particular task, to perform another task. Instead of training a new model from scratch, transfer learning allows to choose a model that is trained on a similar type of task. By using the features learned from the previous task, it can apply them to the new task.

This approach starts with a model previously trained on a large dataset for a specific task. Identify models trained on datasets with general features. The pre-trained model is known as the base model. Additional task-specific transfer layers are employed on top of the model to capture relevant information for the new task. Finally, the model is fine-tuned to improve performance.

#### 4.2.3.1 ResNet

A ResNet-50 is part of the Residual Network family and stands out for its efficiency and depth. This ResNet-50 addresses the degradation problem and the vanishing gradient problem faced by deep neural networks. It contains 50 layers and uses pre-trained weights for these layers.

ResNet solves this issue by using residual blocks, which include skip connections that allow information to flow directly through the network. Therefore, the core building block is the Bottleneck Residual Block. The Fig 4.12 shows the sample architecture by keeping the blocks like residual blocks and identical blocks.

#### **Bottleneck Residual Block Architecture:**

- **ReLU Activation:** After each convolution and batch normalization layer, a ReLU (Rectified Linear Unit) activation function is applied. This introduces non-linearity into the model, which is essential for learning complex patterns by allowing only positive values to pass through.
- **Bottleneck Convolution Layers:**
  - The first layer uses 1x1 filters to reduce the number of channels in the input, helping compress the data for efficiency without losing much information.

- The second layer employs 3x3 filters to capture spatial features.
- The third layer again uses 1x1 filters to restore the original number of channels before adding the output to the shortcut connection.
- **Skip Connection:** The shortcut connection is a key component of the residual block, allowing the input to bypass the convolutional layers and be added directly to the block's output. This ensures that crucial information from earlier layers is retained and propagated through the network, even if the convolutional layers are unable to learn additional features in that particular block.

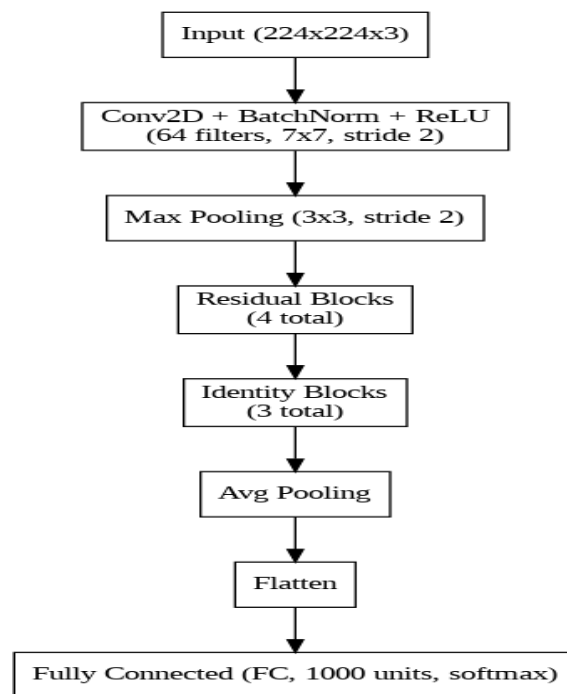


Fig 4.12 Simple architecture of ResNet with identity and Residual Blocks.

### Attention Modules

Attention modules as shown in Fig 4.13 help the network focus on the most relevant information, filtering out unnecessary and less useful details. In image processing tasks, only specific regions of an image contribute to classification, while the background is often irrelevant. Attention mechanisms are used to highlight and focus on these important regions, improving the model's ability to classify correctly. An attention module typically consists of a simple 2D or 3D convolutional layer, followed by a multi-layer perceptron (MLP) and a sigmoid function, which generates a mask over the input feature



map to emphasize the necessary regions.

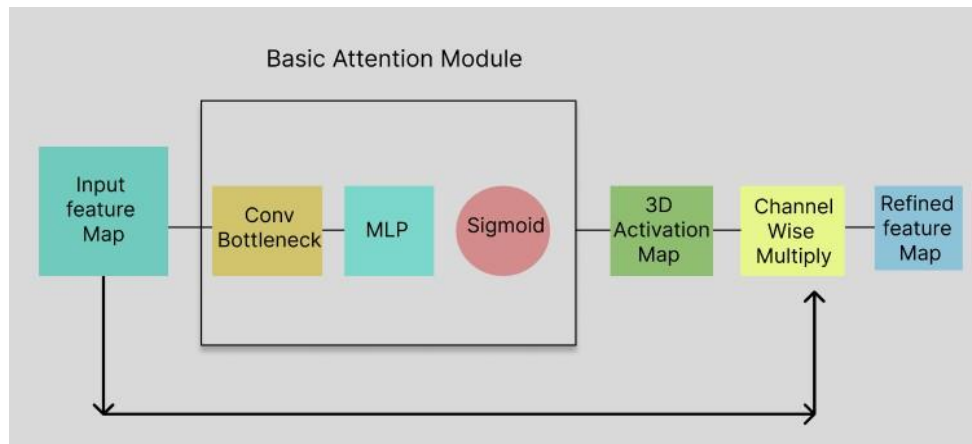


Fig 4.13 Basic attention module

The attention module takes a feature map of size  $C \times H \times W$  as input and produces an attention map, which can be either  $1 \times H \times W$  or  $C \times H \times W$  in the case of 3D attention. This attention map is then element-wise multiplied with the input feature map to obtain a more refined and emphasized output. Attention mechanisms typically focus on both spatial and channel dimensions. The Fig 4.14 shows the workflow of attention modules. The spatial and channel attention maps can be generated either sequentially or in parallel. Initially, these attention mechanisms were experimented with in residual network architectures.

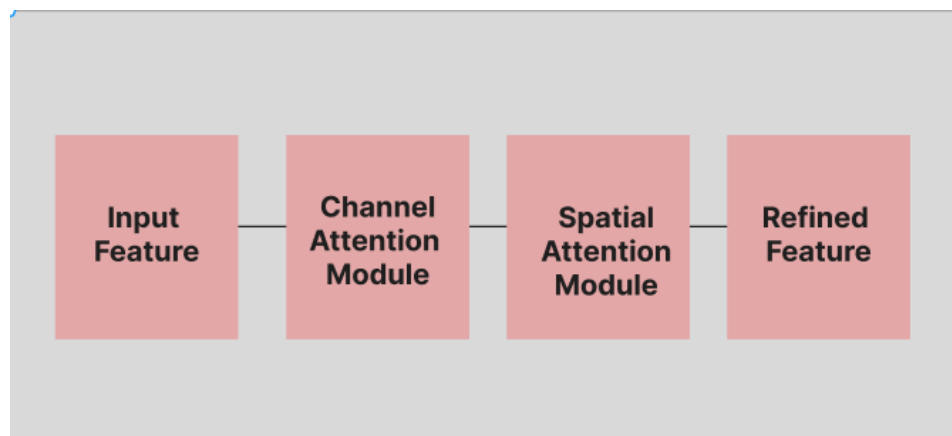


Fig 4.14 Work flow of an Attention module

#### 4.2.3.2 ResNet with BAM

The attention module consists of two main components: the Channel Attention Module and the Spatial Attention Module. For an input feature map  $F$  ( $C \times H \times W$ ), the BAM

(Bottleneck Attention Module) computes a 3D attention map  $M(F)$  ( $C \times H \times W$ ). The refined feature map  $F'$  is calculated as  $F' = M(F) \otimes F + F$ .

Here,  $\otimes$  denotes element-wise multiplication. The residual learning approach is integrated with the attention mechanism to support gradient flow during backpropagation. After multiplying the attention mask with the feature map, the original input tensor  $F$  is added back to the output to enhance the learning process. The attention module computes the channel attention map  $M_c(F)$  ( $C \times 1 \times 1$ ) and the spatial attention map  $M_s(F)$  ( $1 \times H \times W$ ) in separate branches. These outputs are combined to form the attention map  $M(F)$ , with a sigmoid function  $\sigma$  applied to both branches. The branch outputs are resized to  $C \times H \times W$  before being added together.

### Channel Attention Module

To generate the channel attention map, the process begins by performing Global Average Pooling on the input feature map  $F$ , which produces a channel vector  $F_c$  ( $C \times 1 \times 1$ ). This channel vector is passed through a small MLP (multi-layer perceptron) with a hidden layer dimension of  $C/r$ , where  $r$  is a reduction ratio. For example, if the input channel vector length is 1024 and the reduction ratio  $r$  is 16, the number of neurons in the hidden layer would be 64. A Batch Normalization Layer is applied after the MLP to normalize the output.

### Spatial Attention Module

The input feature map  $F$  is passed through a sequence of  $1 \times 1$  and  $3 \times 3$  convolutional layers. The  $3 \times 3$  convolutional layers utilize a dilation value of  $d = 4$ , which increases the network's receptive field. Larger dilation values result in a greater receptive field. The first  $1 \times 1$  convolutional layer reduces the channel dimensions to  $C/r$ . This reduced feature map is passed through two  $3 \times 3$  convolutional layers with a dilation value of  $d = 4$ . The final output channels are reduced to  $1 \times H \times W$  using another  $1 \times 1$  convolutional layer, followed by Batch Normalization to generate  $M_s(F)$ .

### Combination of Channel and Spatial Attention Maps

Once the channel attention  $M_c$  and spatial attention  $M_s$  maps are generated, they are combined through element-wise addition to produce the final attention map  $M(F)$ . Addition is chosen over multiplication for smoother gradient flow during backpropagation. As  $M_c$  and  $M_s$  have different dimensions ( $C \times 1 \times 1$  and  $1 \times H \times W$ , respectively), broadcasting is applied to ensure they match the final dimensions of  $C \times H \times W$ .

### Application of the Attention Map

The attention map  $M(F)$  is multiplied element-wise with the input feature map  $F$ , and then the original feature map is added back to obtain a more refined and highlighted feature map. This process allows the model to focus on the most relevant regions of the input, which is particularly beneficial for the task of colorectal tract classification. In this context, BAM helps the model emphasize important areas like polyps while ignoring irrelevant background textures.

### Positioning of BAM

In the ResNet architecture used for colorectal tract classification, BAM is placed at every bottleneck layer. At the early stages, BAM filters out low-level features such as background textures, while in the later stages, it focuses on high-level semantic features, such as the dyed-lifted polyps, which are critical for accurate classification.

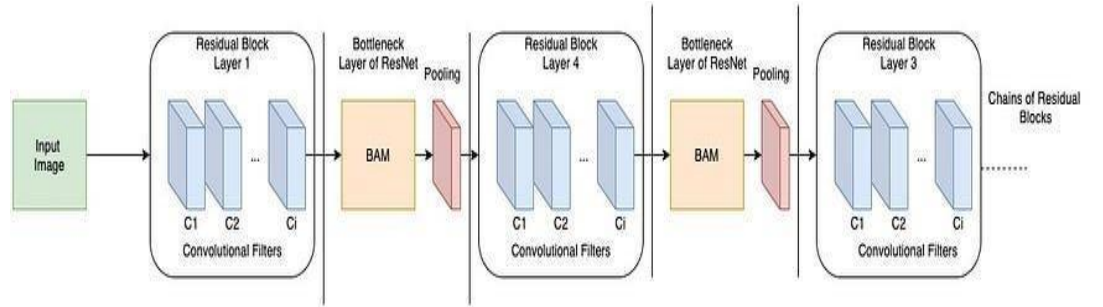


Fig 4.15 Positioning of BAM module in ResNet

#### 4.2.3.3 ResNet with CBAM

Given an intermediate feature map  $F$  ( $C \times H \times W$ ) as input, the Convolutional Block Attention Module (CBAM) generates a 1D channel attention map  $M_c$  ( $C \times 1 \times 1$ ) and a 2D spatial attention map  $M_s$  ( $1 \times H \times W$ ) sequentially. The attention mechanism can be summarized as follows:

In this context, the notation  $\otimes$  indicates element-wise multiplication. During this process, the attention values are appropriately broadcasted: the channel attention values are expanded along the spatial dimensions, while the spatial attention values are replicated across the channel dimensions. The final output after refinement is referred to as  $F'$ .

### Channel Attention Map

The generation of the Channel Attention Map in CBAM resembles that of the Bottleneck Attention Module (BAM), but it integrates both Average Pooling and Max Pooling to capture more distinct channel features. This procedure is illustrated in the accompanying figure.

In this setup, the symbol  $\sigma$  denotes the sigmoid function, and  $W_0$  ( $C/r \times C$ ) and  $W_1$  ( $C \times C/r$ ) represent shared weights for the Multi-Layer Perceptron (MLP) applied to both inputs. The ReLU activation function is used after the first layer of weights  $W_0$ .

### Spatial Attention Map

To create the Spatial Attention Map, the following steps are undertaken:

1. The input feature map  $F$  is processed to produce two intermediate feature maps:  $F_{\text{avg}}$  and  $F_{\text{max}}$  ( $1 \times H \times W$ ).
2. These outputs, derived from Global Average Pooling (GAP) and Max Pooling (MP), are concatenated and passed through a convolutional block with a kernel size of  $7 \times 7$ . Unlike BAM, which employs dilation for an extended receptive field, CBAM utilizes a larger kernel size to achieve a similar effect, with a dilation value of  $d = 1$ .

### Utilizing the Generated Attention Maps

The produced feature maps are arranged sequentially, as shown in the figure. The process begins with the application of the Channel Attention Map to the input feature map, followed by the Spatial Attention Map. The refined output, denoted as  $F''$ , is subsequently added back to the previous input from the convolutional layer in the ResNet architecture, functioning as the shortcut layer for that particular ResNet block.

### Placement of CBAM

In contrast to BAM, which is situated only at bottleneck layers, the CBAM modules are integrated within the residual blocks as well as at the bottlenecks in the network. This strategic positioning enhances the model's ability to focus on both low-level and high-level features, significantly improving its performance in classifying colorectal tract images.

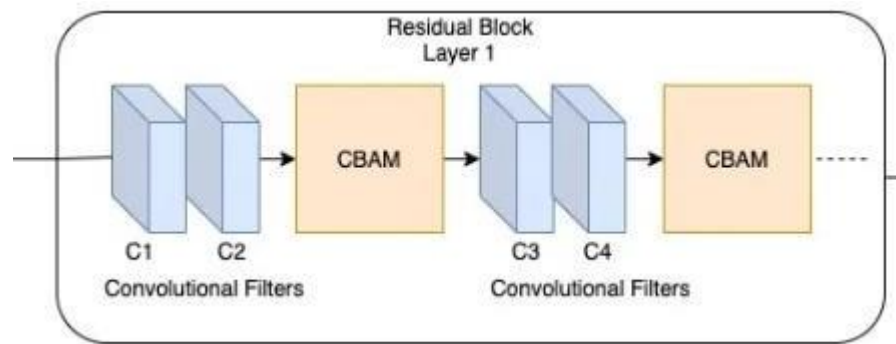


Fig 4.16 Positioning of CBAM module in ResNet

### 4.3 GRAD CAM VISUALISATION



Fig 4.17.1 Original image used for visualizing GRAD-CAM.

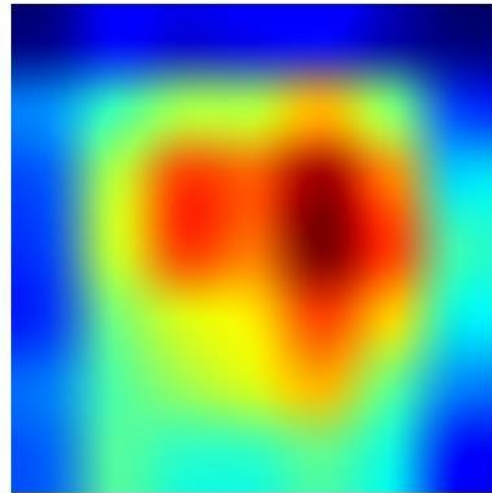


Fig 4.17.2 Image processed by ResNet Model

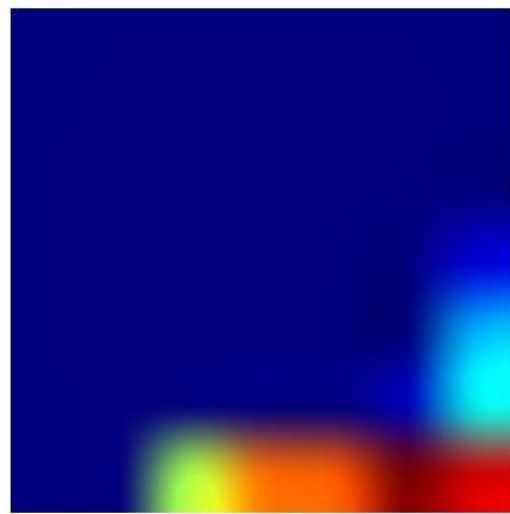


Fig 4.17.3 Image processed by BAM Model.

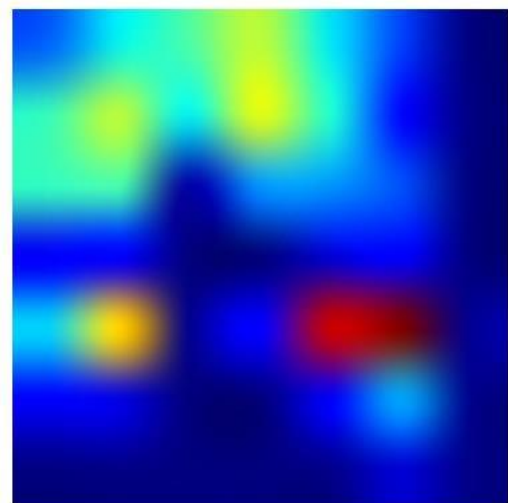


Fig 4.17.4 Image processed by CBAM Model

- **Grad-CAM Overview**

Grad-CAM (Gradient-weighted Class Activation Mapping) is a visualization method that enhances the interpretability of convolutional neural networks (CNNs) in image classification tasks. It offers insights into the specific regions of an image that contribute significantly to the model's predictions, allowing users to better understand how deep learning models make decisions.

The technique involves computing the gradients of the score for a specific class with respect to the feature maps obtained from the last convolutional layer. These gradients provide information about the importance of each feature map for the predicted class. By aggregating these gradients, weights for each feature map are generated, leading to the creation of a weighted sum of the feature maps. This results in a localization map that indicates which parts of the image the model focuses on when making a classification.

- **Uses in Image Classification**

In image classification scenarios, Grad-CAM is invaluable for visualizing the areas that influence the model's predictions. By superimposing the Grad-CAM heatmap onto the original image, one can observe the critical regions that impact the classification outcome. This not only aids in understanding the model's behavior but also helps identify potential weaknesses or biases in its predictions.

- **Color Gradients and Interpretation**

In Grad-CAM visualizations, color gradients typically range from warm shades (like red and yellow) to cooler hues (like blue and green). The warm colors signify areas deemed important by the model for the predicted class, while cooler colors indicate less relevance. By analyzing these color gradients, users can discern which features the model prioritizes in its decision-making process. For example, in the context of medical imaging, Grad-CAM can highlight regions associated with abnormalities, which is crucial for accurate diagnosis.

## 4.4 VISUALIZATION

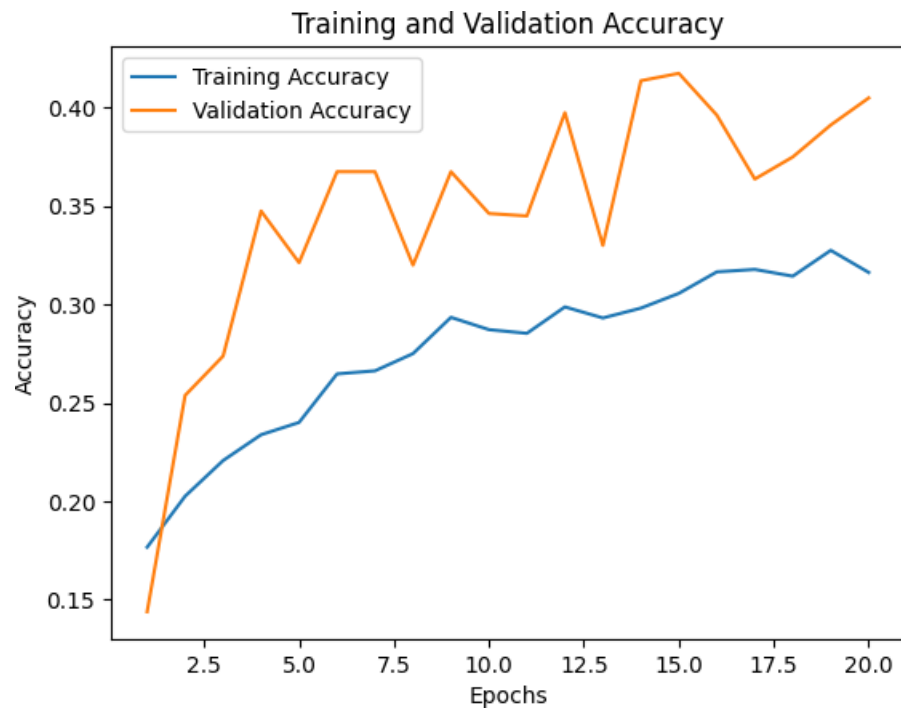


Fig 4.18 Plotting of ResNet Model's accuracy

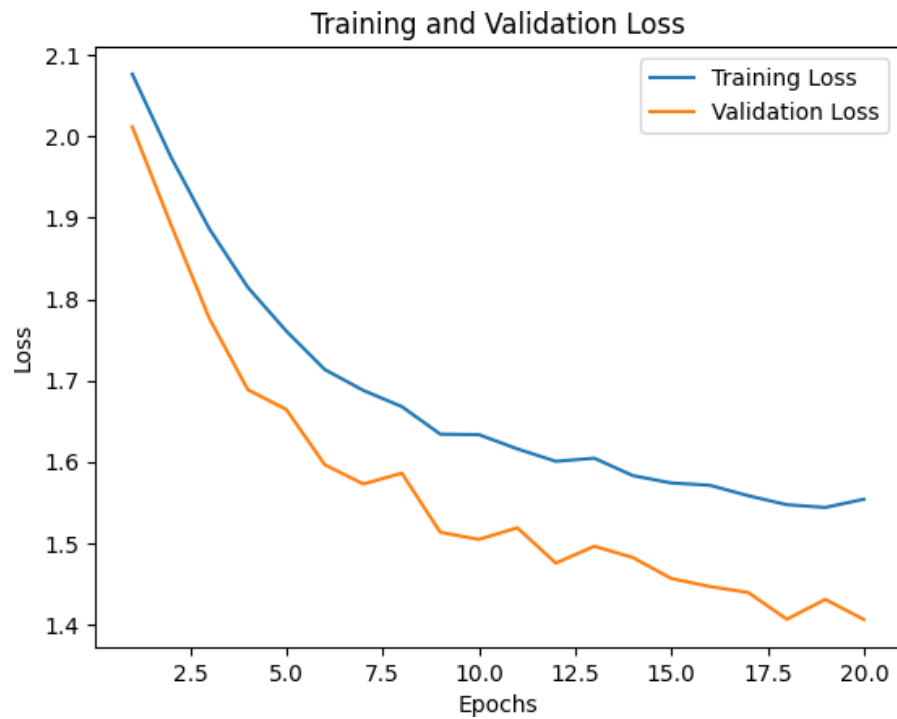


Fig 4.19 Plotting of ResNet Model's loss



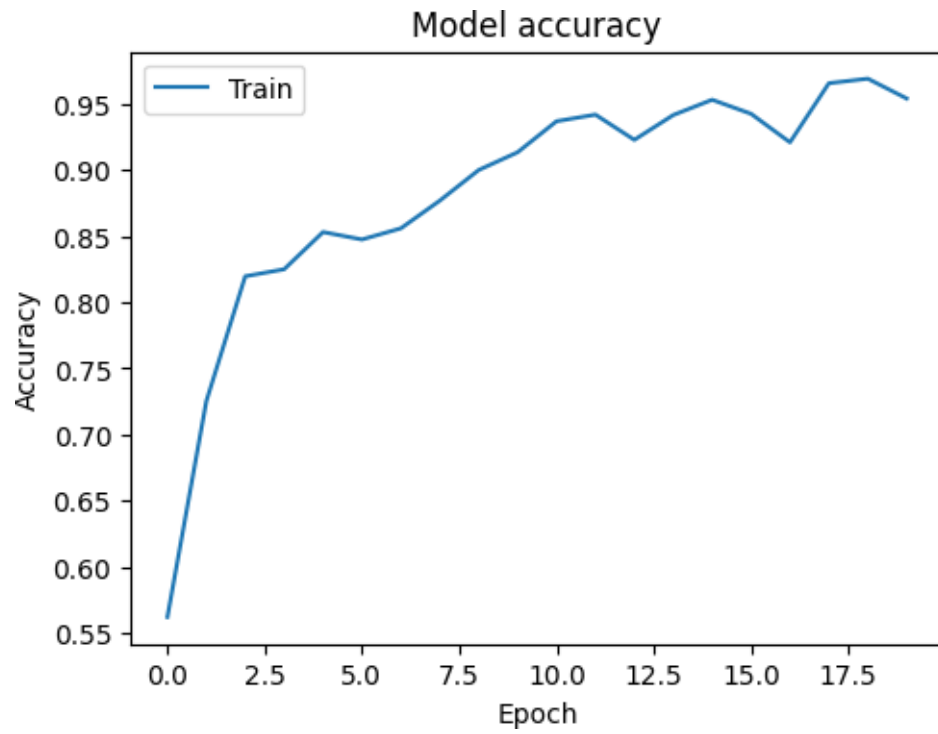


Fig 4.20 Plotting of ResNet with BAM Model's accuracy

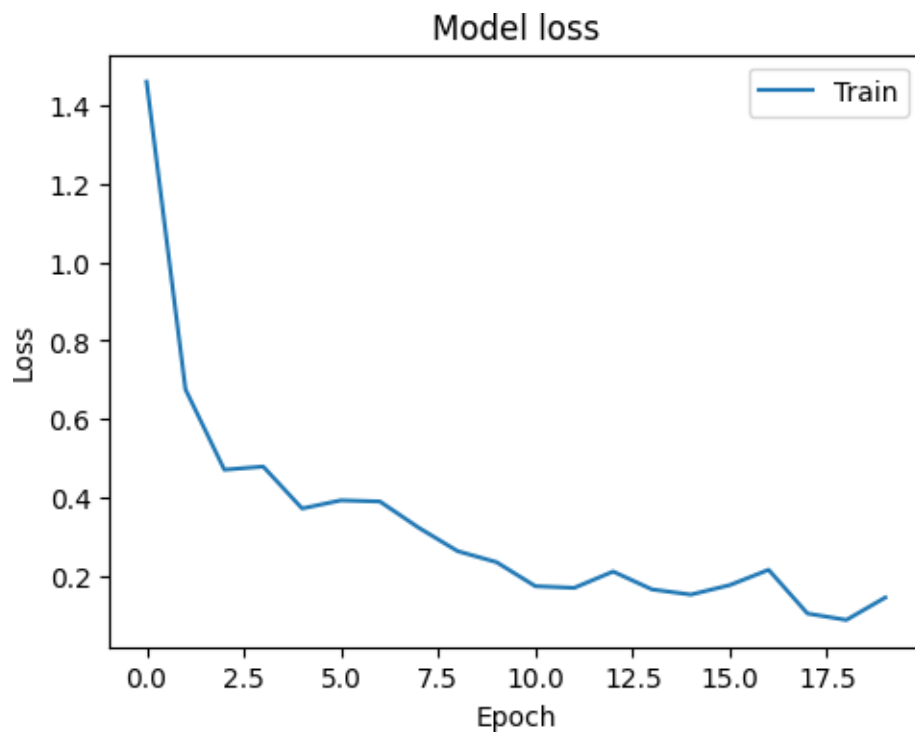


Fig 4.21 Plotting of ResNet with BAM Model's loss

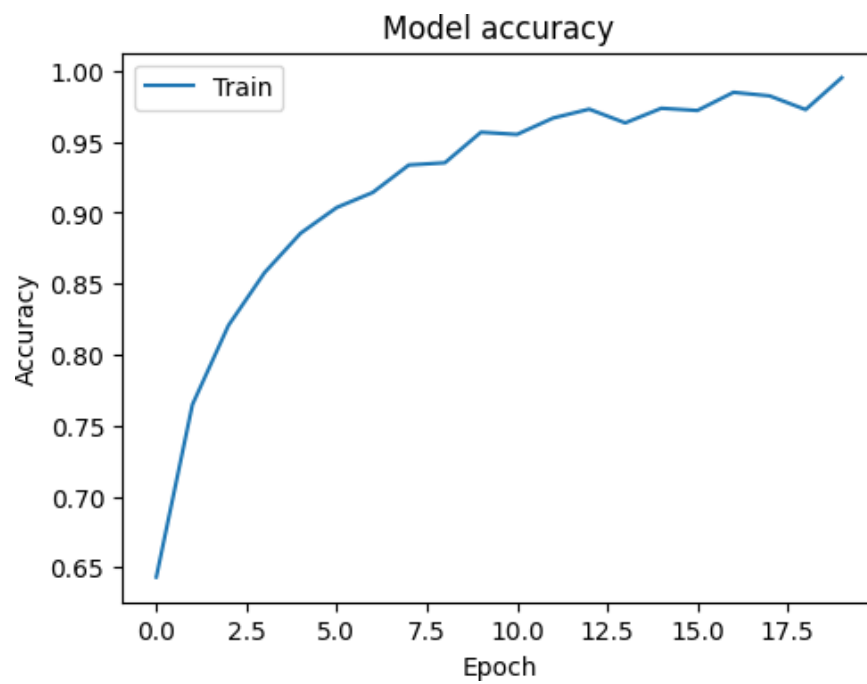


Fig 4.22 Plotting of ResNet with CBAM Model's accuracy

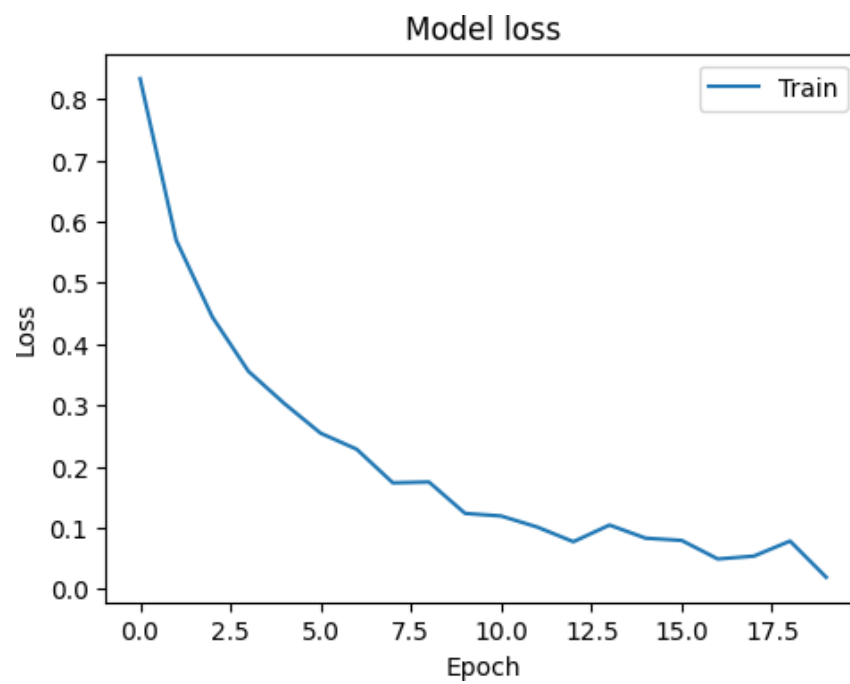


Fig 4.23 Plotting of ResNet with CBAM Model's loss.

## CHAPTER 5

### PERFORMANCE ANALYSIS

The effectiveness of a classification model can be assessed using metrics like Accuracy and loss.

Accuracy represents the proportion of correct predictions made by the model compared to the total number of predictions.

$$\text{Accuracy} = \frac{\text{Number of correct Predictions from Output}}{\text{Total Number of Output}} \quad (5.1)$$

Categorical Cross-Entropy quantifies the difference between the actual class labels and the predicted probabilities for each class in a classification problem. It calculates how closely the predicted probability distribution matches the true distribution of classes. The formula sums the negative logarithm of the predicted probabilities corresponding to the true labels, averaging this across all samples. This loss function effectively assesses the model's accuracy in predicting the likelihood of each category.

$$\text{Categorical cross entropy} = -1/n \sum \sum Y_{ij} (\log (\hat{Y}_{ij})) \quad (5.2)$$

Table 5.1 Comparison of different models

Methodologies	Test Accuracy	Test Loss
ResNet	40.5 %	1.4061
ResNet with BAM	77 %	0.8712
ResNet with CBAM	84. 375 %	0.5640

## CHAPTER 6

### RESULTS AND DISCUSSION

The study's findings demonstrate how adding attention mechanisms to the ResNet model significantly improved its performance in classifying medical pictures. The benchmark ResNet model's test accuracy of 40.5% and high test loss of 1.4061 reveal its shortcomings in terms of efficiently locating pertinent features. However, the model's accuracy increased to 77% with a lower test loss of 0.8712 when the Bottleneck Attention Module (BAM) was added. This illustrates how BAM aids in the model's ability to concentrate on important areas of the picture.

The greatest performance was obtained with further refinement using the Convolutional Block Attention Module (CBAM), with a test accuracy of 84.375% and a smaller test loss of 0.5640. The model can perform better thanks to the coupled channel and spatial attention processes of the CBAM. comprehend the characteristics of the image, producing better outcomes than BAM. Preprocessing methods including flipping, resizing, and normalizing were crucial in enhancing data homogeneity and facilitating the model's learning process. Grad-CAM representations, which emphasize the areas of the picture that contributed most to the classification, also provide insightful information about the model's decision-making process. ResNet performs much better in medical image analysis overall when attention mechanisms are included, with CBAM yielding the best results.

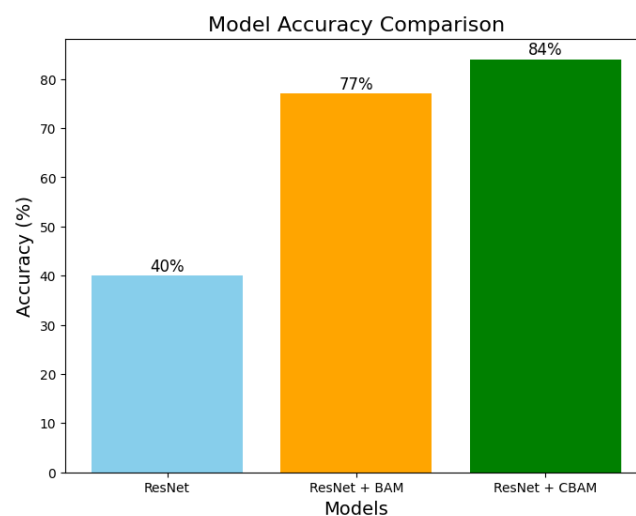


Fig 6.1 Comparison of various models

## **CHAPTER 7**

### **CONCLUSION**

The presence of attention mechanisms, namely BAM and CBAM, in the ResNet architecture leads to a notable improvement in the classification performance of medical images. The attention modules enable the model to focus on the most relevant portions of the pictures, resulting to better accuracy and lower loss relative to the baseline ResNet model. The CBAM module fared better than BAM, indicating that it is useful for fine-tuning feature maps since it combines both channel and spatial attention. Preprocessing methods including flipping, resizing, and normalization improved the model's capacity to learn from the input. Grad-CAM also offered comprehensible graphics that supported the attention mechanisms by emphasizing the regions that affected the model's predictions. All things considered, this method shows how useful attention mechanisms are for enhancing deep learning models for medical image analysis, utilizing CBAM.

## CHAPTER 8

### APPENDICES

#### 8.1 APPENDIX – 1 CODING

```

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Flatten, Dropout , Input

from tensorflow.keras.optimizers import RMSprop, Adam

!kaggle datasets download -d meetnagadia/kvasir-dataset/

!unzip "/content/kvasir-dataset.zip"

import os

import glob

import numpy as np

import pandas as pd

from sklearn.model_selection import train_test_split

from PIL import Image

from tqdm import tqdm

# Path to the dataset

# Path to the dataset

dataset_path = '/content/kvasir-dataset'

# Lists to store file paths and labels

image_paths = []

labels = []

# Loop through each folder in the dataset

for folder_name in os.listdir(dataset_path):

    folder_path = os.path.join(dataset_path, folder_name)

    # Check if the path is a directory

```

```

if os.path.isdir(folder_path):

    # Find all image files in the folder

    for img_file in glob.glob(os.path.join(folder_path, '*.jpg')):

        image_paths.append(img_file)

        labels.append(folder_name)

# Convert lists to pandas DataFrame

data = pd.DataFrame({

    'image_path': image_paths,

    'label': labels

})

# Perform train-test split

train_data, test_data = train_test_split(data, test_size=0.2, stratify=data['label'], random_state=42)

print(f'Training samples: {len(train_data)}')

print(f'Test samples: {len(test_data)}')

def load_image(image_path, label, img_size=(224, 224)):

    image = tf.io.read_file(image_path)

    image = tf.image.decode_jpeg(image, channels=3)

    image = tf.image.resize(image, img_size)

    image = image / 255.0 # Normalize to [0, 1]

    return image, label

def encode_labels(label, class_names):

    label = tf.argmax(tf.constant(class_names) == label)

    return label

class_names = data['label'].unique().tolist()

train_data['label'] = train_data['label'].apply(lambda x: encode_labels(x, class_names))

test_data['label'] = test_data['label'].apply(lambda x: encode_labels(x, class_names))

train_dataset = tf.data.Dataset.from_tensor_slices((train_data['image_path'].values.tolist(),
train_data['label'].values.tolist()))

test_dataset = tf.data.Dataset.from_tensor_slices((test_data['image_path'].values.tolist(),
test_data['label'].values.tolist()))

```

```

# Map loading function to datasets

train_dataset = train_dataset.map(load_image, num_parallel_calls=tf.data.AUTOTUNE)

test_dataset = test_dataset.map(load_image, num_parallel_calls=tf.data.AUTOTUNE)

# Shuffle and batch the datasets

BATCH_SIZE = 32

train_dataset = train_dataset.shuffle(len(train_data)).batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)

test_dataset = test_dataset.batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)

import tensorflow as tf

from tensorflow.keras.applications import ResNet50

from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout

base_model = ResNet50(weights='imagenet', include_top=False)

base_model.trainable = False

x = base_model.output

x = GlobalAveragePooling2D()(x)

x = Dense(128, activation='relu')(x)

x = Dropout(0.5)(x)

predictions = Dense(8, activation='softmax')(x)

model = tf.keras.Model(inputs=base_model.input, outputs=predictions)

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

history = model.fit(train_dataset, epochs=20, validation_data=test_dataset)

loss_res, accuracy_res = model.evaluate(test_dataset)

print("Test loss:", loss_res)

print("Test accuracy:", accuracy_res*100)

import matplotlib.pyplot as plt

train_acc = history.history['accuracy']

val_acc = history.history['val_accuracy']

plt.plot(range(1, len(train_acc) + 1), train_acc, label='Training Accuracy')

plt.plot(range(1, len(val_acc) + 1), val_acc, label='Validation Accuracy')

plt.title("Training and Validation Accuracy")

plt.xlabel('Epochs')

```



```

plt.ylabel('Accuracy')

plt.legend()

plt.show()

import matplotlib.pyplot as plt

train_loss = history.history['loss']

val_loss = history.history['val_loss']

plt.plot(range(1, len(train_loss) + 1), train_loss, label='Training Loss')

plt.plot(range(1, len(val_loss) + 1), val_loss, label='Validation Loss')

plt.title('Training and Validation Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()

class Flatten(Layer):
    def call(self, x):
        return tf.keras.layers.Flatten()(x)

class ChannelGate(Layer):
    def __init__(self, gate_channel, reduction_ratio=16, num_layers=1, **kwargs):
        super(ChannelGate, self).__init__(**kwargs)
        self.flatten = Flatten()
        self.gate_channels = [gate_channel]
        self.gate_channels += [gate_channel // reduction_ratio] * num_layers
        self.gate_channels += [gate_channel]
        self.dense_layers = []
        for i in range(len(self.gate_channels) - 2):
            self.dense_layers.append(Dense(self.gate_channels[i+1], use_bias=False))
            self.dense_layers.append(BatchNormalization())
            self.dense_layers.append(ReLU())
        self.dense_layers.append(Dense(self.gate_channels[-1], use_bias=False))
    def call(self, x):
        avg_pool = tf.reduce_mean(x, axis=[1, 2], keepdims=True)
        x = self.flatten(avg_pool)
        for layer in self.dense_layers:
            x = layer(x)
        x = tf.reshape(x, (-1, 1, 1, x.shape[-1]))

```

```

        return x

class SpatialGate(Layer):
    def __init__(self, gate_channel, reduction_ratio=16, dilation_conv_num=2, dilation_val=4, **kwargs):
        super(SpatialGate, self).__init__(**kwargs)
        self.conv_reduce = Conv2D(gate_channel // reduction_ratio, kernel_size=1, padding='same',
        use_bias=False)
        self.bn_reduce = BatchNormalization()
        self.relu = ReLU()
        self.dilation_conv_num = dilation_conv_num
        self.dilation_val = dilation_val
        self.dilation_convs = []
        for _ in range(dilation_conv_num):
            self.dilation_convs.append(Conv2D(gate_channel // reduction_ratio, kernel_size=3, padding='same',
        dilation_rate=dilation_val, use_bias=False))
            self.dilation_convs.append(BatchNormalization())
            self.dilation_convs.append(ReLU())
        self.conv_final = Conv2D(1, kernel_size=1, padding='same', use_bias=False)
    def call(self, x):
        x = self.conv_reduce(x)
        x = self.bn_reduce(x)
        x = self.relu(x)
        for conv in self.dilation_convs:
            x = conv(x)
        x = self.conv_final(x)
        return x

class BAM(Layer):
    def __init__(self, gate_channel, **kwargs):
        super(BAM, self).__init__(**kwargs)
        self.channel_att = ChannelGate(gate_channel)
        self.spatial_att = SpatialGate(gate_channel)
    def call(self, x):
        channel_att = self.channel_att(x)
        spatial_att = self.spatial_att(x)
        att = 1 + tf.sigmoid(channel_att * spatial_att)
        return Multiply()([att, x])

from tensorflow.keras.applications import ResNet50
def create_model(input_shape):
    inputs = Input(shape=input_shape)

```

```

# Add ResNet as a feature extractor
resnet = ResNet50(weights='imagenet', include_top=False, input_shape=input_shape)
x = resnet(inputs)
x = Conv2D(64, kernel_size=3, padding='same')(x)
x = BAM(64)(x)
x = MaxPooling2D(pool_size=2)(x)
x = Flatten()(x)
x = Dense(128, activation='relu')(x)
x = Dense(64, activation='relu')(x)
x = Dense(32, activation='relu')(x)
outputs = Dense(8, activation='softmax')(x) # Example for 8 classes
model = Model(inputs, outputs)
return model

bam_model = create_model((224, 224, 3)) # Example input shape
bam_model.summary()
from tensorflow.keras.utils import plot_model
plot_model(bam_model, to_file='BAM.png', show_shapes=True, show_layer_names=True)
# Compile the model
bam_model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
# Train the model
history = bam_model.fit(train_dataset, epochs=20)
# Evaluate the model
test_loss, test_acc = bam_model.evaluate(test_dataset)
print(f'\nTest accuracy: {test_acc:.2f}')
plt.figure(figsize=(12, 4))
# Plot training & validation accuracy values
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.title('Model accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'])
# Plot training & validation loss values
plt.subplot(1, 2, 2)

```

```

plt.plot(history.history['loss'])
plt.title('Model loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'])
plt.show()

import tensorflow as tf
from tensorflow.keras.layers import Layer, Conv2D, Dense, BatchNormalization, ReLU,
GlobalAveragePooling2D, GlobalMaxPooling2D, Add, Multiply, Concatenate, Input
from tensorflow.keras.applications import InceptionV3
import tensorflow.keras.backend as K
import matplotlib.pyplot as plt

# Define CBAM Module
class Flatten(Layer):
    def call(self, x):
        return tf.keras.layers.Flatten()(x)

class BasicConv(Layer):
    def __init__(self, in_planes, out_planes, kernel_size, stride=1, padding='same', dilation=1, groups=1,
relu=True, bn=True, bias=False, **kwargs):
        super(BasicConv, self).__init__(**kwargs)
        self.conv = Conv2D(out_planes, kernel_size=kernel_size, strides=stride, padding=padding,
dilation_rate=dilation, groups=groups, use_bias=bias)
        self.bn = BatchNormalization(epsilon=1e-5, momentum=0.01) if bn else None
        self.relu = ReLU() if relu else None
    def call(self, x):
        x = self.conv(x)
        if self.bn is not None:
            x = self.bn(x)
        if self.relu is not None:
            x = self.relu(x)
        return x

class ChannelGate(Layer):
    def __init__(self, gate_channels, reduction_ratio=16, pool_types=['avg', 'max'], **kwargs):
        super(ChannelGate, self).__init__(**kwargs)
        self.gate_channels = gate_channels
        self.reduction_ratio = reduction_ratio
        self.pool_types = pool_types
        self.flatten = Flatten()

```

```

        self.mlp = tf.keras.Sequential([
            Dense(gate_channels // reduction_ratio, activation='relu'),
            Dense(gate_channels)
        ])
    def call(self, x):
        channel_att_sum = None
        for pool_type in self.pool_types:
            if pool_type == 'avg':
                avg_pool = GlobalAveragePooling2D()(x)
                channel_att_raw = self.mlp(avg_pool)
            elif pool_type == 'max':
                max_pool = GlobalMaxPooling2D()(x)
                channel_att_raw = self.mlp(max_pool)
            elif pool_type == 'lp':
                lp_pool = self.lp_pool2d(x)
                channel_att_raw = self.mlp(lp_pool)
            elif pool_type == 'lse':
                lse_pool = self.logsumexp_2d(x)
                channel_att_raw = self.mlp(lse_pool)
        if channel_att_sum is None:
            channel_att_sum = channel_att_raw
        else:
            channel_att_sum = Add()([channel_att_sum, channel_att_raw])
        scale = tf.keras.activations.sigmoid(channel_att_sum)
        scale = tf.reshape(scale, (-1, 1, 1, self.gate_channels))
    def lp_pool2d(self, x, p=2):
        return tf.reduce_sum(tf.pow(x, p), axis=[1, 2], keepdims=True)
    def logsumexp_2d(self, x):
        x_flatten = tf.reshape(x, (tf.shape(x)[0], tf.shape(x)[3], -1))
        s = tf.reduce_max(x_flatten, axis=2, keepdims=True)
        outputs = s + tf.math.log(tf.reduce_sum(tf.exp(x_flatten - s), axis=2, keepdims=True))
        return outputs
class ChannelPool(Layer):
    def call(self, x):
        return Concatenate(axis=-1)([tf.reduce_max(x, axis=3, keepdims=True), tf.reduce_mean(x, axis=3,
keepdims=True)])
class SpatialGate(Layer):
    def __init__(self, **kwargs):
        super(SpatialGate, self).__init__(**kwargs)

```

```

self.compress = ChannelPool()

self.spatial = BasicConv(2, 1, kernel_size=7, stride=1, padding='same', relu=False)

def call(self, x):
    x_compress = self.compress(x)
    x_out = self.spatial(x_compress)
    scale = tf.keras.activations.sigmoid(x_out)
    return Multiply()([x, scale])

class CBAM(Layer):
    def __init__(self, gate_channels, reduction_ratio=16, pool_types=['avg', 'max'], no_spatial=False,
    **kwargs):
        super(CBAM, self).__init__(**kwargs)
        self.channel_gate = ChannelGate(gate_channels, reduction_ratio, pool_types)
        self.no_spatial = no_spatial
        if not no_spatial:
            self.spatial_gate = SpatialGate()
    def call(self, x):
        x_out = self.channel_gate(x)
        if not self.no_spatial:
            x_out = self.spatial_gate(x_out)
        return x_out

def build_model(input_shape, num_classes):
    inputs = Input(shape=input_shape)
    # Load InceptionV3 model without top layers
    base_model = ResNet50(include_top=False, weights='imagenet', input_tensor=inputs)
    # Output from InceptionV3 base model
    x = base_model.output
    # Apply CBAM to the output of InceptionV3
    x = CBAM(gate_channels=x.shape[-1])(x) # Adjust gate_channels to match the output of InceptionV3
    # Add additional layers
    x = BasicConv(x.shape[-1], 256, kernel_size=3)(x)
    x = BasicConv(256, 512, kernel_size=3)(x)
    x = GlobalAveragePooling2D()(x)
    x = Dense(num_classes, activation='softmax')(x)
    model = tf.keras.Model(inputs=inputs, outputs=x)
    return model

# Model compilation, training, and evaluation
input_shape = (224, 224, 3) # Example input shape
num_classes = 8 # number of classes

```

```

cbam_model = build_model(input_shape, num_classes)
cbam_model.summary()
cbam_model.compile(optimizer='adam',
                   loss='sparse_categorical_crossentropy',
                   metrics=['accuracy'])
# Train the model
history = cbam_model.fit(train_dataset, epochs=20, batch_size=32)
# Evaluate the model
test_loss, test_acc = cbam_model.evaluate(test_dataset, verbose=2)
print(f'Test accuracy: {test_acc}')
# Plotting training history
plt.figure(figsize=(12, 4))
# Plot training & validation accuracy values
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.title('Model accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'])
# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.title('Model loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'])
plt.show()
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
def get_grad_cam(model, img, class_index, last_conv_layer_name):
    # Create a model that maps the input image to the activations of the last conv layer
    grad_model = tf.keras.models.Model(
        [model.inputs], [model.get_layer(last_conv_layer_name).output, model.output]
    )
    # Record operations for automatic differentiation
    with tf.GradientTape() as tape:
        # Forward pass

```

```

conv_outputs, predictions = grad_model(np.array([img]))
loss = predictions[:, class_index]
# Compute the gradient of the class score with respect to the feature map
grads = tape.gradient(loss, conv_outputs)
# Compute the mean intensity of the gradient over each channel
pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))
# Multiply each channel by its corresponding gradient
conv_outputs = conv_outputs[0]
conv_outputs = conv_outputs * pooled_grads
# Generate heatmap
heatmap = np.mean(conv_outputs, axis=-1)
# Normalize the heatmap
heatmap = np.maximum(heatmap, 0)
heatmap /= np.max(heatmap)
return heatmap

def display_grad_cam(img, heatmap,
alpha=0.4): # Rescale heatmap to a range of 0-
255 heatmap = np.uint8(255 * heatmap)
# Use jet colormap to colorize heatmap
jet = plt.cm.get_cmap("jet")
# Use jet colormap to apply color to heatmap
jet_colors = jet(np.arange(256))[:, :3]
jet_heatmap = jet_colors[heatmap]
# Create an image with RGB colored heatmap
jet_heatmap = tf.keras.preprocessing.image.array_to_img(jet_heatmap)
jet_heatmap = jet_heatmap.resize((img.shape[1], img.shape[0]))
jet_heatmap = tf.keras.preprocessing.image.img_to_array(jet_heatmap)
# Superimpose the heatmap on the original image
superimposed_img = jet_heatmap * alpha + img
superimposed_img = tf.keras.preprocessing.image.array_to_img(superimposed_img)
return superimposed_img

# Sample image from the test dataset
sample_image, label = next(iter(test_dataset.take(1)))
plt.imshow(sample_image[0])
plt.axis('off')
plt.title(f'Original Image - Label: {class_names[label[0]]}') # Display the label
plt.show()
sample_image = sample_image[0].numpy()

```



```

# Get model's prediction for the sample image
predictions = model.predict(np.array([sample_image]))
predicted_class = np.argmax(predictions)
# Generate Grad-CAM heatmap for the predicted class
heatmap = get_grad_cam(model, sample_image, predicted_class,
last_conv_layer_name='conv5_block3_3_conv')
# Display the image with the Grad-CAM heatmap
superimposed_img = display_grad_cam(sample_image, heatmap)
plt.imshow(superimposed_img)
plt.axis('off') # Hide axis
plt.show()

# Get model's prediction for the sample image
predictions = bam_model.predict(np.array([sample_image]))
predicted_class = np.argmax(predictions)
# Generate Grad-CAM heatmap for the predicted class
heatmap = get_grad_cam(bam_model, sample_image, predicted_class, last_conv_layer_name='bam')
# Display the image with the Grad-CAM heatmap
superimposed_img = display_grad_cam(sample_image, heatmap)
plt.imshow(superimposed_img)
plt.axis('off') # Hide axis
plt.show()

# Get model's prediction for the sample image
predictions = cbam_model.predict(np.array([sample_image]))
predicted_class = np.argmax(predictions)
# Generate Grad-CAM heatmap for the predicted class
heatmap = get_grad_cam(cbam_model, sample_image, predicted_class, last_conv_layer_name='cbam_1')
# Display the image with the Grad-CAM heatmap
superimposed_img = display_grad_cam(sample_image, heatmap)
plt.imshow(superimposed_img)
plt.axis('off') # Hide axis
plt.show()

```

## 8.2 APPENDIX – 2 SCREENSHOTS

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 224, 224, 3)	0
resnet50 (Functional)	(None, 7, 7, 2048)	23,587,712
conv2d (Conv2D)	(None, 7, 7, 64)	1,179,712
bam (BAM)	(None, 7, 7, 64)	1,124
max_pooling2d (MaxPooling2D)	(None, 3, 3, 64)	0
flatten_4 (Flatten)	(None, 576)	0
dense_2 (Dense)	(None, 128)	73,856
dense_3 (Dense)	(None, 64)	8,256
dense_4 (Dense)	(None, 32)	2,080
dense_5 (Dense)	(None, 8)	264

Total params: 24,853,004 (94.81 MB)

Trainable params: 24,799,852 (94.60 MB)

Non-trainable params: 53,152 (207.62 KB)

### A2.1 BAM Model Architecture


conv5_block3_add (Add)	(None, 7, 7, 2048)	0	conv5_block2_out[0][0... conv5_block3_3_bn[0][...
conv5_block3_out (Activation)	(None, 7, 7, 2048)	0	conv5_block3_add[0][0]
cbam_1 (CBAM)	(None, 7, 7, 2048)	526,566	conv5_block3_out[0][0]
basic_conv_2 (BasicConv)	(None, 7, 7, 256)	4,719,616	cbam_1[0][0]
basic_conv_3 (BasicConv)	(None, 7, 7, 512)	1,181,696	basic_conv_2[0][0]
global_average_pooling2d... (GlobalAveragePooling2D)	(None, 512)	0	basic_conv_3[0][0]
dense_10 (Dense)	(None, 8)	4,104	global_average_poolin...

Total params: 30,019,694 (114.52 MB)

Trainable params: 29,965,036 (114.31 MB)

Non-trainable params: 54,658 (213.51 KB)

### A2.2 CBAM Model Architecture

25/25  10s 230ms/step - accuracy: 0.7746 - loss: 0.8712

Test accuracy: 0.77

#### A2.3 Test Accuracy of Resnet with BAM model

25/25 - 12s - 491ms/step - accuracy: 0.8438 - loss: 0.5640

Test accuracy: 0.84375

#### A2.4 Test Accuracy of Resnet with CBAM model

## REFERENCES

- [1] Noor, M.N.; Nazir, M.; Khan, S.A.; Song, O.-Y.; Ashraf, I. Efficient Gastrointestinal Disease Classification Using Pretrained Deep Convolutional Neural Network. *Electronics* 2023, 12, 1557.
- [2] Bhardwaj, S.; Jha, D.; Ali, S.; Riegler, M.A.; Halvorsen, P.; Johansen, D.; Johansen, H.D. Predicting Gastrointestinal Disease Using Deep Transfer Learning and Multiclass Endoscopy Images. *Diagnostics* 2023, 13, 1057.
- [3] Sivari, E. Bostanci, E. Guzel, M.S. Acici, K. Asuroglu, T. Ayyildiz, T.E. (2023). A New Approach for Gastrointestinal Tract Findings Detection and Classification: Deep Learning-Based Hybrid Stacking Ensemble Models. *Diagnostics*, 13(720).
- [4] Gunasekaran, H., Ramalakshmi, K., Swaminathan, D.K., J, A., & Mazzara, M. (2023). GIT-Net: An Ensemble Deep Learning-Based GI Tract Classification pf Endoscopic Images. *Bioengineering* 10(7),809.
- [5] Sujatha, R., Mahalakshmi, K., & Yoosuf, M.S. (2023). Colorectal cancer prediction via histopathology segmentation using DC-GAN and VAE-GAN. *EAI Endorsed Transactions on Pervasive Health and Technology*, 10, 101942.
- [6] Mohapatra, S., Pati, G.K., Mishra, M., & Swarnkar, T. (2023). Gastrointestinal Abnormality Detection and Classification Using Empirical Wavelet Transform and and Deep Convolutional Neural Network from Endoscopic Images. *Ain Shams Engineering Journal*,14,101942.
- [7] Tang, S., Yu, X., Cheang, C.F., Liang, Y., Zhao, P., Yu, H.H., & Choi, I.C. (2023). Transformer-based multi-task learning for classification and segmentation of gastrointestinal tract endoscopic images. *Computers in Biology and Medicine*, 157, 106723.