Leveraging Gram Matrix Representation in Shallow DNN for GI Tract MRI Image Segmentation

EEE 400
Project And Thesis

A Thesis submitted in partial fulfillment of the requirement for the degree of Bachelor of Science in Electrical and Electronic Engineering

By
Fariza Siddiqua; Student ID: 1706043
Tamim Ahmed; Student ID: 1706063

Under the Supervision of
Dr. Mohammed Imamul Hassan Bhuiyan
Professor,EEE,BUET

Department of Electrical and Electronic Engineering
Bangladesh University of Engineering and Technology

Date of Submission: May , 2023

DECLEARATION

We declare that the thesis “Leveraging Gram Matrix Representation in Shallow DNN for GI Tract MRI Image Segmentation” is our own, unaided work, except where otherwise acknowledged. It is being submitted for the degree of Bachelor of Science in Electrical and Electronic Engineering in the University of Bangladesh University of Engineering and Technology. It has not been submitted before for any degree or examination in any other university.

AUTHORS

1. Fariza Siddiqua
ID:1706043

2. Tamim Ahmed
ID: 1706063

SUPERVISOR:

Dr. Mohammed Imamul Hassan Bhuiyan

Professor

Department of Electrical and Electronic Engineering Bangladesh University of Engineering and Technology

ACKNOWLEDGEMENT

We would like to place our ardent gratefulness to Almighty Allah for blessing us to complete this thesis. We would like to express our deepest sense of gratitude to our supervisor, Dr. Mohammed Imamul Hassan Bhuiyan, PhD, Professor, Department of Electrical and Electronic Engineering, BUET for his relentless support, guidance and suggestions throughout the whole year. He has helped us in every way a student can be helped. He arranged for all the facilities and necessary support needed for the successful completion of the thesis.

Dhaka Fariza Siddiqa
May, 2023 Tamim Ahmed

Table of Contents

List of Figures

List of Tables

ABSTRACT

Medical image segmentation plays a vital role in case of diagnosis of many diseases and their proper treatment.

In case of GI tract cancer or tumor treatment, radio oncologists must apply X-ray beams pointing towards the tumor cell and avoiding the other organs. From MRI scan they can visualize the inner parts of the GI tract but if they must manually segment the organs then it becomes a time consuming and labor-intensive process. Therefore, a computer aided automatic, fast and accurate method is required for better treatment.

The U-Net architecture, one of the pioneering models in this domain, utilizes an encoder-decoder structure with skip connections to effectively capture and integrate contextual information at different levels. This allows for precise localization and segmentation of various structures within GI tract images. Recently many deep learning architectures which are based on U-Net such as Attention U-Net, TransU-Net, U-Net++ have been proposed for GI tract image segmentation with a great segmentation result. Attention U-Net inherits quadratic complexity. Transformer U-Net patchifies images which causes degradation in resolution. U-Net and U-Net++ inherits a bulk architecture which causes a large amount of trainable parameter. More parameter means more computational power and a huge time will need to train the model for a large dataset. Therefore, it becomes difficult to embed these models in less expensive medical devices.

In our thesis, a gram matrix oriented shallow by pass attention mechanism having a shallow U-shaped network is proposed. This architecture gives equivalent result as U-Net with drastically reduced number of parameters. The computational complexity of the proposed model is almost linear and the trainable parameter is drastically reduced as compared to U-Net. To implement the model, in this paper, GI tract MRI scanned image dataset provided by UW Madison has been used and the results have been analyzed.

Chapter 1

Introduction

1.1 Background

Deep learning algorithms have revolutionized medical image segmentation, playing a pivotal role in the diagnosis and treatment of various diseases and abnormalities. Segmentation of different organs or structures is a fundamental task in medical imaging, enabling accurate localization and analysis of specific regions of interest. However, manual segmentation of medical images, such as MRI scans of the GI tract, can be time-consuming, taking anywhere from 15 minutes to hours. This manual process hinders treatment progress and efficiency.

To address this challenge, there is a high demand for automated image segmentation techniques that can save time and improve accuracy. Deep learning architectures have emerged as powerful solutions for this task. Several notable models have been proposed specifically for GI tract image segmentation, employing multi-level semantic segmentation approaches.

U-Net [1], one of the pioneering architectures, employs an encoder-decoder structure with skip connections. This design allows for effective feature extraction and integration of contextual information at different levels, resulting in precise segmentation of stomach, intestines, and tumor cells.

Attention U-Net [2] enhances the U-Net framework by incorporating attention mechanisms. These mechanisms enable the model to dynamically focus on informative image regions, improving its ability to handle complex structures and further enhancing segmentation accuracy in the GI tract.

Transformer-based U-Net [3], inspired by transformer networks used in natural language processing, leverages the power of self-attention mechanisms to capture long-range dependencies and global contextual information. This allows for more accurate segmentation of the GI tract, aiding in the precise localization of tumor cells.

U-Net++ [4] is an extension of the original U-Net architecture that introduces nested skip connections. By incorporating multiple levels of skip connections, U-Net++ captures multi-scale features and preserves spatial details, leading to improved segmentation performance in the GI tract.

These deep learning architectures have shown promising results in GI tract image segmentation, offering automated solutions that significantly reduce the time required for segmentation while maintaining high accuracy. Their ability to leverage complex feature extraction, attention mechanisms, and hierarchical representations enables more efficient and reliable medical image analysis.

1.2 Related Works

1.2.1 Medical Image Segmentation Based on CNN

U-net is the first encoder-decoder CNN-based method demonstrating great image segmentation performance. The remarkable performance of the Unet resulted in many variants of U-shaped structures. For instance, in weighted Res-UNet [8], a weighted attention mechanism is used to segment small regions. U-net++ is a U-shaped structure with a series of nested, dense skip pathways leading to less semantic gap between the feature maps of the encoder and decoder. U-net3+ [12] applies a full-scale skip connection to combine low-level and high-level details. Dense-UNet [13] introduced a hybrid densely connected UNet in order to optimize both intra-slice and inter-slice features. ENS-UNet [14] is another U-shaped architecture with no need for massive pre-processing and post-processing. C-UNet [15] takes advantage of Inception-like convolutional block, recurrent convolutional block, and dilated convolutional layers for skin lesion segmentation. In general, CNN-based methods are widely used in image segmentation tasks due to the ability to extract features effectively by paying attention to adjacent pixels.

1.2.2 Medical Image Processing Based on Transformers

CNN-based methods cannot model long-range dependencies due to the intrinsic locality of convolutional operation. After the success of transformers in Natural Language Processing (NLP) fields [16], a vision transformer (ViT) [9] has been proposed as a transformer in image processing tasks. Since transformers can capture long-range dependencies of input sequences using a multi-head self-attention (MSA) mechanism, ViT demonstrated high performance in image segmentation, image recognition, and object-detection tasks. DeiT [17] utilized ViT as its backbone and reduced the data dependency of ViT using data-efficient training strategies.

One of the drawbacks of ViT is its computational complexity which is quadratic to the image size, so using ViT for high-resolution images is challenging. A new network called the Swin transformer [10] has been introduced to address the computational complexity issue.

The focal transformer [18] employs a focal self-attention mechanism incorporating fine grained local and coarse-grained global interactions. Focal transformer utilized both short and long-range dependencies. The focal self-attention mechanism has larger receptive fields with less time and memory cost than the standard self-attention mechanism. A focal modulation network (focal nets) is proposed as a new structure that employs focal modulation instead of self-attention. Focal nets outperform state-of-the-art methods such as the Swin transformer and focal transformer in image classification, object detection, and semantic segmentation.

1.2.3 Medical Image Segmentation Based on Transformers

TransU-Net [5] is the first structure that combines transformer and CNN in medical image segmentation tasks to use the advantages of both networks. Afterward, many other researchers attempted to apply a transformer in a CNN-based architecture for high-performance image segmentation. For instance, IT-U-Net [16] combines CNN and transformer for organs-at-risk segmentation. FcTC-U-Net [17] is a hybrid method based on CNN and transformers to thoracic segmentation. UCATR [18] is another example of this hybrid structure used for lesion segmentation. Medical transformer [7] and transfuse [11] are examples of utilizing both CNN and transformer. Despite the above models, some methods are purely transformer based to segment medical images. Swin-U-Net [6] is an instance of this category. Swin-U-Net is a U-shaped pure transformer-based structure that uses a hierarchical Swin transformer in both encoder and decoder for local and global semantic feature learning. Experimental results have shown that these convolutional free architectures can result in more accurate segmentation in some cases.

1.3 Motivation

The mentioned models, U-Net, Attention U-Net, Transformer-based U-Net, and U-Net++, have demonstrated strong capabilities in GI tract image segmentation. However, they do have specific limitations:

(i) While U-Net performs well in medical image segmentation, it requires a large filter size, resulting in a high number of trainable parameters. This can lead to increased computational complexity and memory requirements, making it less efficient in certain scenarios.

(ii) Although Attention U-Net achieves good segmentation results, it inherits quadratic computational complexity due to the attention mechanisms. This quadratic complexity can limit its scalability and make it computationally expensive, particularly when dealing with larger images or datasets.

(iii) The pacification of images in Transformer-based U-Net can cause degradation in segmented image resolution. This is because the patch-based approach may not capture fine-grained details and spatial relationships as effectively as pixel-level analysis, potentially leading to a loss of information in the segmentation results.

(iv) U-Net++ has a complex architecture with a large number of layers and skip connections. While this architecture contributes to improved segmentation performance by capturing multi-scale features, it can also require significant computational resources and memory, making it challenging to deploy on less powerful or resource-constrained hardware.

To overcome these limitations, we propose a Gram matrix-oriented bypass attention mechanism integrated into a U-shaped architecture. This mechanism reduces the computational complexity by employing fewer multiplication operations and more concatenation operations. Consequently, the computational complexity becomes almost linear, addressing the scalability issue associated with quadratic complexity. Additionally, the implementation of the Gram matrix results in a more developed feature matrix at each step, allowing for a reduction in the number of layers and filter size. As a result, the trainable parameters are significantly reduced, enabling more efficient model training and deployment.

For training and validation, we utilized an MRI scanned GI tract image dataset provided by UW-Madison. The dataset focuses on a three-level semantic segmentation problem, specifically involving the segmentation of the stomach, large bowel, and small bowel from the MRI images. Through our proposed model, we achieved segmentation performance comparable to that of U-Net and U-Net++, showcasing the effectiveness of our approach in addressing the limitations of existing models.

1.4 Objectives

The fundamental objective of this thesis work is to propose a model which will be as efficient as U-Net and other complex architectures but with the help of less computational power. So, in order to design a model, our main objectives are

- Create an architecture where we integrate, Gram Matrix oriented by-pass attention U-Net
- Which inherits almost linear computational complexity
- The trainable parameter should be drastically reduced
- At the same time, the performance will be equivalent to U-Net or other models

1.5 Thesis Review

There are four chapters to describe our thesis. The first chapter is about background, objectives and motivation of our thesis. In the next chapter firstly, we discuss methodology and materials where the description of dataset, preprocessing, the entire architecture of our proposed model and the performance parameters to evaluate the model. In Chapter 3, the model is evaluated and is compared to the previous works. Finally, we conclude this thesis by summarizing our works and introducing some scopes for future researches.

Chapter 2

Materials and Methodology

This chapter consists of four sections: database, pre-processing, model architecture and performance metric. The first section describes the resources used for the segmentation task and the later three sections explain the total work flow of this work and the measures used for evaluating the model.

2.1 Dataset

In 2019, an estimated 5 million people were diagnosed with a cancer of the gastro-intestinal tract worldwide. Of these patients, about half are eligible for radiation therapy, usually delivered over 10-15 minutes a day for 1-6 weeks. Radiation oncologists try to deliver high doses of radiation using X-ray beams pointed to tumors while avoiding the stomach and intestines. With newer technology such as integrated magnetic resonance imaging and linear accelerator systems, also known as MR-Linacs, oncologists are able to visualize the daily position of the tumor and intestines, which can vary day to day. In these scans, radiation oncologists must manually outline the position of the stomach and intestines in order to adjust the direction of the x-ray beams to increase the dose delivery to the tumor and avoid the stomach and intestines. This is a time-consuming and labor intensive process that can prolong treatments from 15 minutes a day to an hour a day, which can be difficult for patients to tolerate—unless deep learning could help automate the segmentation process. A method to segment the stomach and intestines would make treatments much faster and would allow more patients to get more effective treatment.

- The train.csv has 115,488 total rows and 3 columns
- There are 38,96 unique ids - or cases
- Each unique id appears within the dataset 3 times, depending on the class of the image (large_bowel, small_bowel, stomach
- The class should be treated like a **flag** that shows where is the healthy organs are actually located within one image
- The segmentation category flags precisely (not with bounding box, but using pixels) the organs - if nothing is found in neither class, it will be marked as none (or missing)

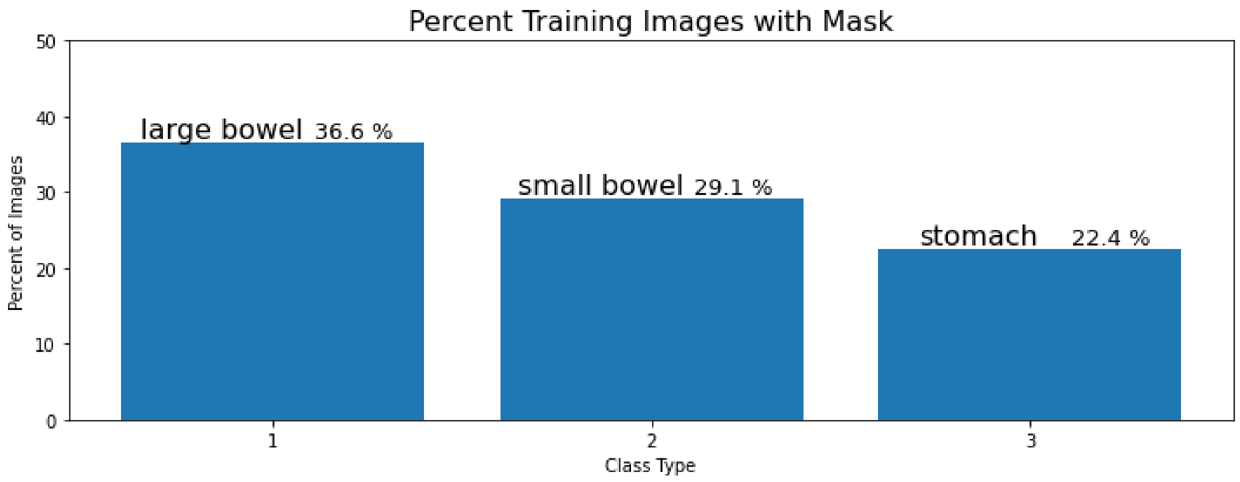
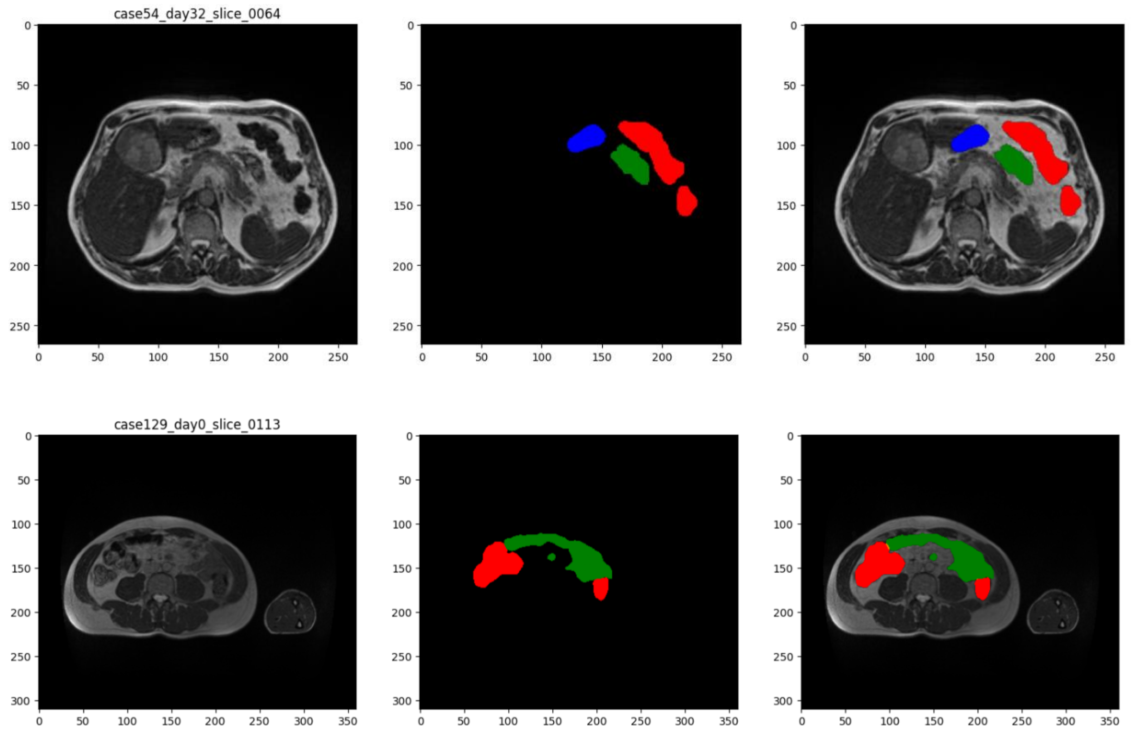


Figure 2. 1: Percent of images with training masks for each class

2.2 Preprocessing

Before training the model with the dataset, all the images were transformed into 160x160 sizes cropping out the unnecessary parts. The RLE masks were decoded. No augmentation is used as the end output is quite promising.



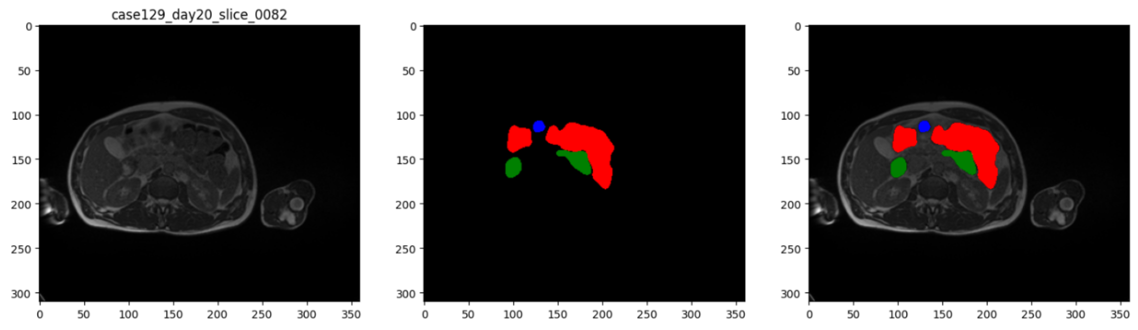


Figure 2. 2: Random slices of MRI image and their masks

In fig.2.2, we see MRI images and it RLE masks are decoded accordingly. Here red denotes large bowel, green denotes small bowel and blue denotes the stomach.

2.3 Model Architecture

In this chapter, our proposed deep learning architecture is explained in details.

2.3.1 Overview

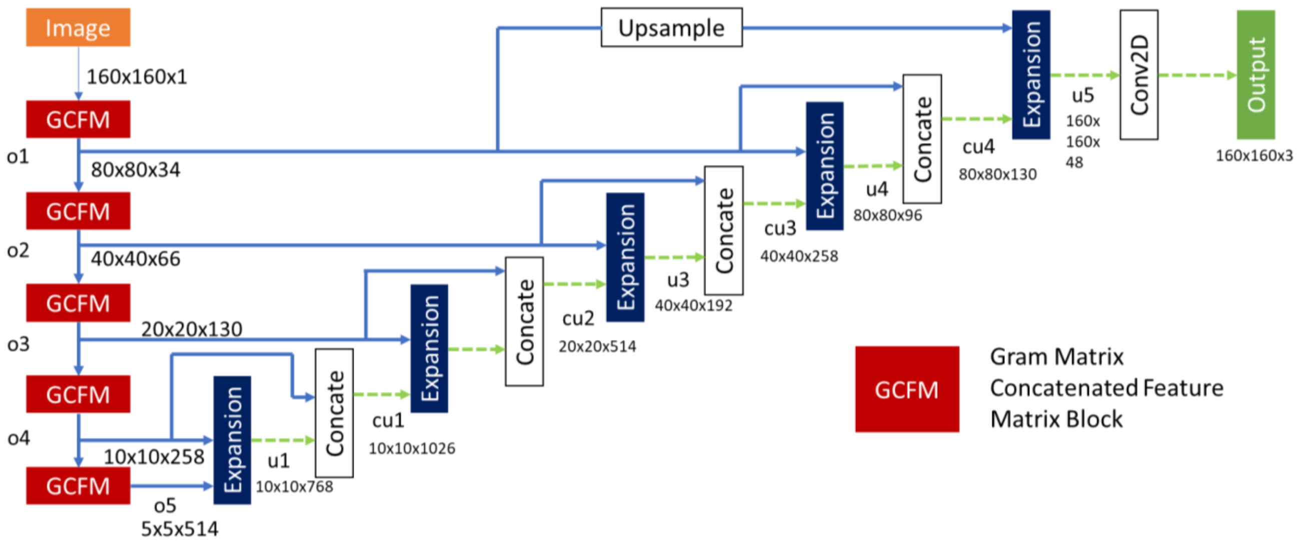


Figure 2. 3:Complete Diagram of the Architecture

In our proposed model, as shown in fig.2.3, we have redesigned the encoder and decoder part of the U-shape architecture. Each layer of the encoder is named as GCFM (Gram matrix concatenated feature matrix) block. There is total 5 GCFM blocks. Each layer in the decoder is named as Expansion block. There is total 5 Expansion blocks. In each GCFM block, a 3-D feature matrix is passed through 4 parallel convolution layers with different kernel size. In each of these convolutional layers, a 1-D vector corresponding to the feature matrix is produced. From four parallel convolutional layers, we get four 1-D vectors. Later these 1-D vectors are concatenated to form the one vector V corresponding to the primary feature matrix. This vector is then transposed. Gram matrix is defined as,

Gram matrix= $V \times V^t$ --------------------------------------- (1)

Gram matrix represents the correlation coefficient among the components of the vector V.

This 2-D Gram Matrix is then concatenated in the front and end of the primary 3-D feature matrix. If the size of the feature matrix is 80x80x32, the size of the Gram matrix concatenated feature matrix will be 80x80x34. Gram Matrix represents the correlation of one element of V_1 with the other elements of the same matrix. Therefore, it gives an insight of importance of different parts of the feature matrix. The basic block diagram of the proposed model is given below. Here, five GCFM blocks are used in the encoder side and five Expansion blocks are used in the decoder side.

2.3.2 GFMC Block

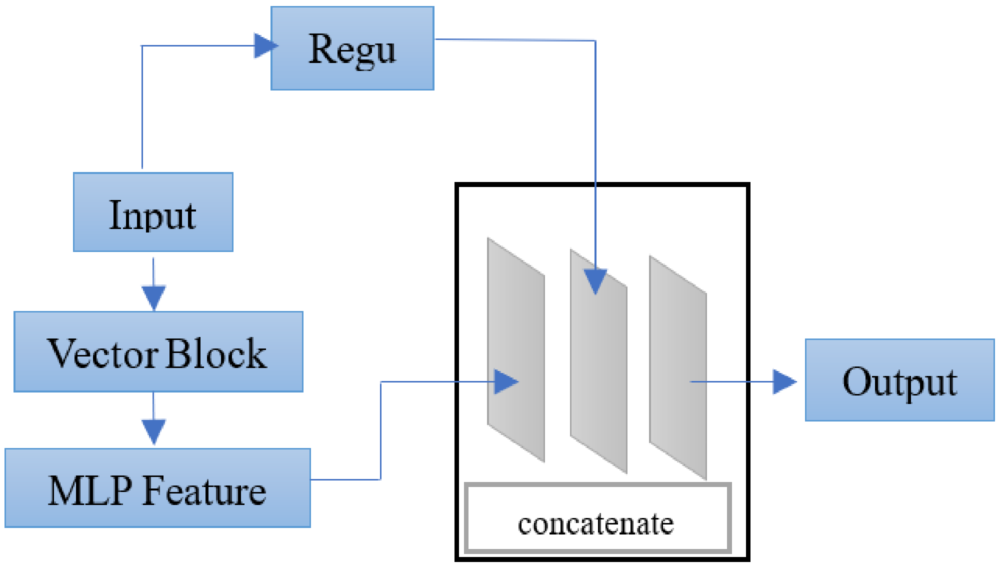


Figure 2. 4: GFMC Block Diagram

In fig.2.4 we can see, GCFM block takes a 3-D tensor as input and sends it to the vector block. The vector block produces the 1-D component vector. MLP feature block takes this vector as input and generates Gram matrix. After that, the Gram Matrix is concatenated with the feature matrix which is shown fig.2.5.

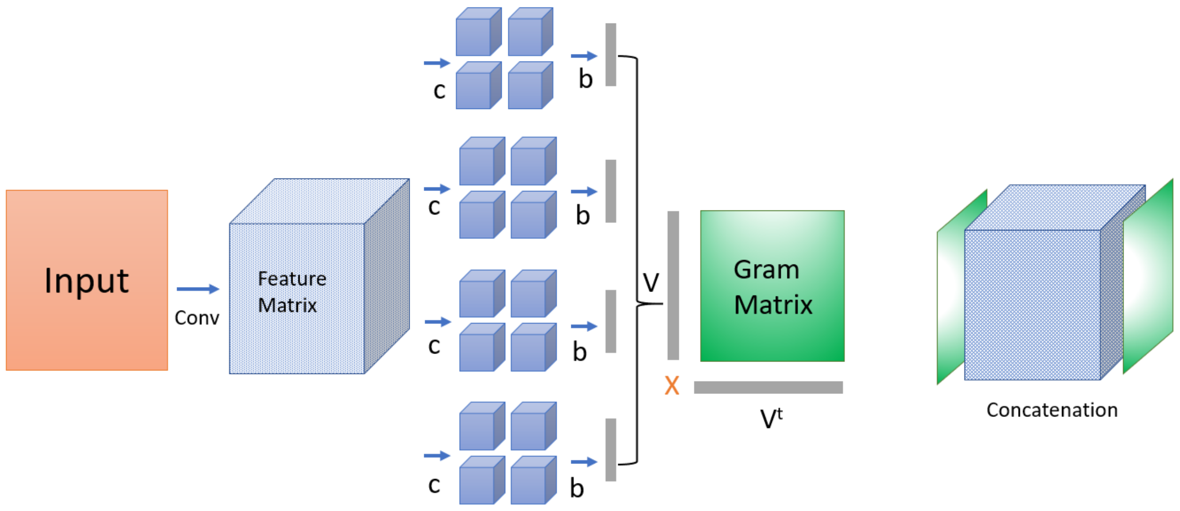


Figure 2. 5: Gram Matrix concatenation with feature matrix

2.3.3 Vector Block

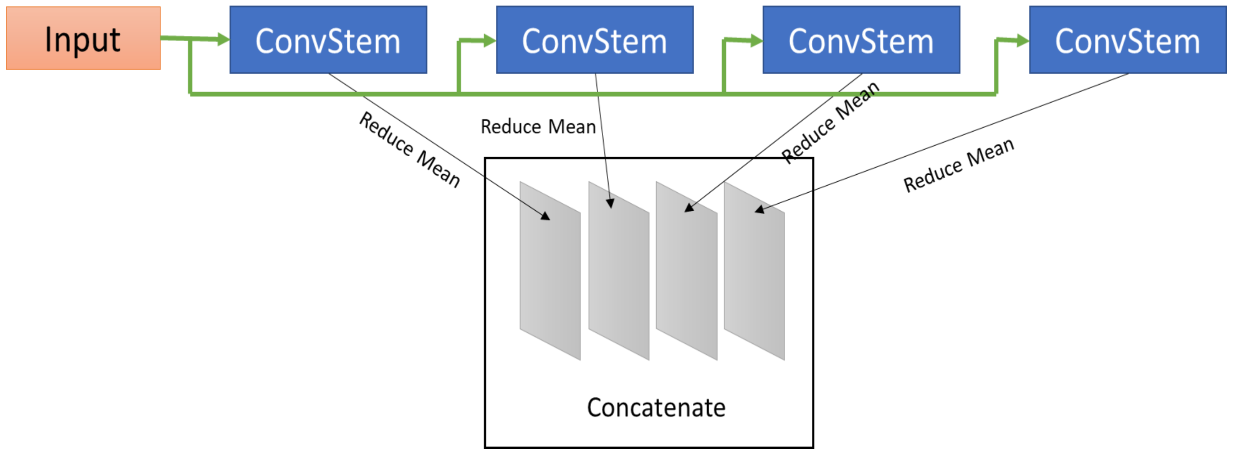


Figure 2. 6: Block diagram of Vector Block

In fig.2.6 the vector block takes the 3-D tensor and sends it to four parallel convstem layers. In each convstem layer, convolution, gelu activation and normalization take place. The output of the convstem layer is channel-wise averaged. After passing through the convstem layer, a 1-D vector is generated. From 4 convstem layers, 4 1-D vectors are concatenated and the final 1D component vector is generated.

2.3.4 MLP Feature Block

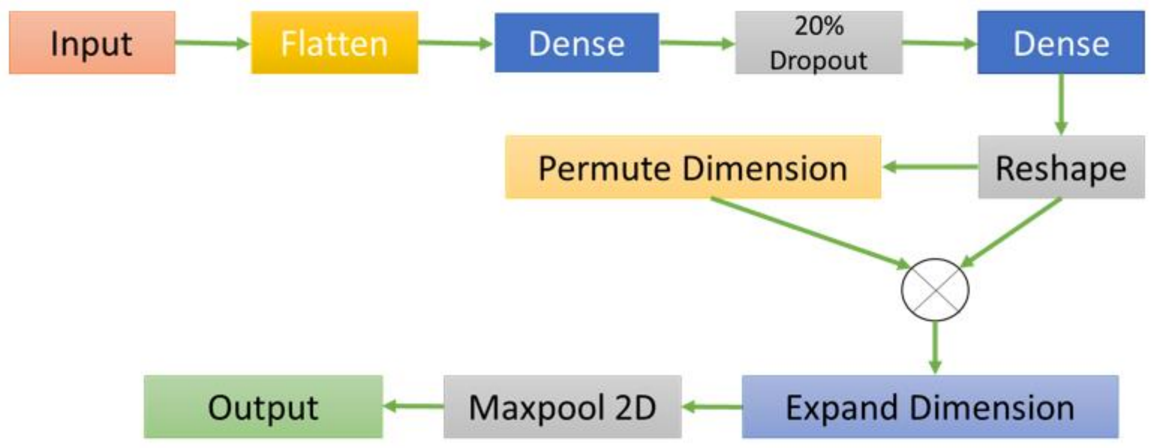


Figure 2. 7: MLP Feature Block

In fig.2.7 we can see, MLP feature block takes the 1-D component vector as input. Then this vector is flattened. After that it is passed through two successive dense convolution layers. The inner product of the vector is determined and the Gram matrix is generated.

2.3.5 Expansion Block

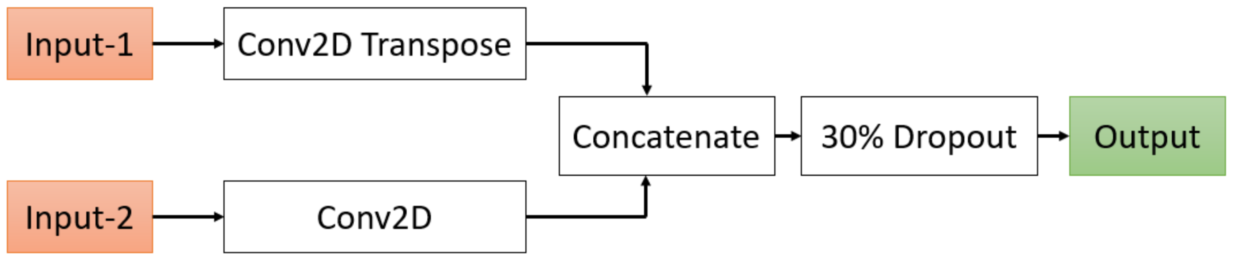


Figure 2. 8: Block diagram of Expansion Block

In fig.2.8 the expansion block takes two inputs. Considering the first expansion block, the first input is the output of the last GCFM block. And the 2nd input is the output of the 2nd last GCFM block. These two inputs and concatenated after passing through the convolution and transpose convolution layer. 30% dropout is used to avoid overfitting. Then the generated output is again concatenated with the 2nd input. This is operation in one expansion block. This repeats 5 times in out model.

2.3.6 Overall Model Summary and Calculation of trainable parameters

```
Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 160, 160, 1  0           []
                                )]

 conv2d (Conv2D)                (None, 160, 160, 32  320         ['input_1[0][0]']
                                )

 max_pooling2d (MaxPooling2D)   (None, 80, 80, 32)   0           ['conv2d[0][0]']

 conv2d_1 (Conv2D)              (None, 16, 16, 80)   64080       ['max_pooling2d[0][0]']

 conv2d_2 (Conv2D)              (None, 16, 16, 80)   64080       ['max_pooling2d[0][0]']

 conv2d_3 (Conv2D)              (None, 16, 16, 80)   64080       ['max_pooling2d[0][0]']

 conv2d_4 (Conv2D)              (None, 16, 16, 80)   64080       ['max_pooling2d[0][0]']

 activation (Activation)        (None, 16, 16, 80)   0           ['conv2d_1[0][0]']

 activation_1 (Activation)      (None, 16, 16, 80)   0           ['conv2d_2[0][0]']

 activation_2 (Activation)      (None, 16, 16, 80)   0           ['conv2d_3[0][0]']

 activation_3 (Activation)      (None, 16, 16, 80)   0           ['conv2d_4[0][0]']

 layer_normalization (LayerNorm  (None, 16, 16, 80)  160         ['activation[0][0]']
 alization)

 layer_normalization_1 (LayerNo  (None, 16, 16, 80)  160         ['activation_1[0][0]']
 rmalization)

 layer_normalization_2 (LayerNo  (None, 16, 16, 80)  160         ['activation_2[0][0]']
 rmalization)

 layer_normalization_3 (LayerNo  (None, 16, 16, 80)  160         ['activation_3[0][0]']
 rmalization)

 tf.math.reduce_mean (TFOpLambd  (None, 80)          0           ['layer_normalization[0][0]']
 a)

 tf.math.reduce_mean_1 (TFOpLam  (None, 80)          0           ['layer_normalization_1[0][0]']
 bda)

 tf.math.reduce_mean_2 (TFOpLam  (None, 80)          0           ['layer_normalization_2[0][0]']
 bda)

 tf.math.reduce_mean_3 (TFOpLam  (None, 80)          0           ['layer_normalization_3[0][0]']
 bda)

 concatenate (Concatenate)      (None, 320)          0           ['tf.math.reduce_mean[0][0]',
                                                                  'tf.math.reduce_mean_1[0][0]',
                                                                  'tf.math.reduce_mean_2[0][0]',
                                                                  'tf.math.reduce_mean_3[0][0]']

 flatten (Flatten)              (None, 320)          0           ['concatenate[0][0]']

 dense (Dense)                  (None, 160)          51360       ['flatten[0][0]']
```

```
dropout (Dropout)              (None, 160)          0           ['dense[0][0]']

dense_1 (Dense)                (None, 320)          51520       ['dropout[0][0]']

reshape (Reshape)              (None, 1, 320)       0           ['dense_1[0][0]']

tf.compat.v1.transpose (TFOpLa  (None, 320, 1)      0           ['reshape[0][0]']
mbda)

multiply (Multiply)            (None, 320, 320)     0           ['reshape[0][0]',
                                                                 'tf.compat.v1.transpose[0][0]']

tf.expand_dims (TFOpLambda)    (None, 320, 320, 1)  0           ['multiply[0][0]']

max_pooling2d_1 (MaxPooling2D)  (None, 80, 80, 1)   0           ['tf.expand_dims[0][0]']

concatenate_1 (Concatenate)    (None, 80, 80, 34)   0           ['max_pooling2d_1[0][0]',
                                                                 'max_pooling2d[0][0]',
                                                                 'max_pooling2d_1[0][0]']

conv2d_5 (Conv2D)              (None, 80, 80, 64)   19648       ['concatenate_1[0][0]']

max_pooling2d_2 (MaxPooling2D)  (None, 40, 40, 64)  0           ['conv2d_5[0][0]']

conv2d_6 (Conv2D)              (None, 8, 8, 40)     64040       ['max_pooling2d_2[0][0]']

conv2d_7 (Conv2D)              (None, 8, 8, 40)     64040       ['max_pooling2d_2[0][0]']

conv2d_8 (Conv2D)              (None, 8, 8, 40)     64040       ['max_pooling2d_2[0][0]']

conv2d_9 (Conv2D)              (None, 8, 8, 40)     64040       ['max_pooling2d_2[0][0]']

activation_4 (Activation)      (None, 8, 8, 40)     0           ['conv2d_6[0][0]']

activation_5 (Activation)      (None, 8, 8, 40)     0           ['conv2d_7[0][0]']

activation_6 (Activation)      (None, 8, 8, 40)     0           ['conv2d_8[0][0]']

activation_7 (Activation)      (None, 8, 8, 40)     0           ['conv2d_9[0][0]']

layer_normalization_4 (LayerNo  (None, 8, 8, 40)    80          ['activation_4[0][0]']
rmalization)

layer_normalization_5 (LayerNo  (None, 8, 8, 40)    80          ['activation_5[0][0]']
rmalization)

layer_normalization_6 (LayerNo  (None, 8, 8, 40)    80          ['activation_6[0][0]']
rmalization)

layer_normalization_7 (LayerNo  (None, 8, 8, 40)    80          ['activation_7[0][0]']
rmalization)

tf.math.reduce_mean_4 (TFOpLam  (None, 40)          0           ['layer_normalization_4[0][0]']
bda)

tf.math.reduce_mean_5 (TFOpLam  (None, 40)          0           ['layer_normalization_5[0][0]']
bda)

tf.math.reduce_mean_6 (TFOpLam  (None, 40)          0           ['layer_normalization_6[0][0]']
bda)

tf.math.reduce_mean_7 (TFOpLam  (None, 40)          0           ['layer_normalization_7[0][0]']
bda)

concatenate_2 (Concatenate)    (None, 160)          0           ['tf.math.reduce_mean_4[0][0]',
                                                                 'tf.math.reduce_mean_5[0][0]',
                                                                 'tf.math.reduce_mean_6[0][0]',
                                                                 'tf.math.reduce_mean_7[0][0]']
```

```
flatten_1 (Flatten)            (None, 160)          0           ['concatenate_2[0][0]']

dense_2 (Dense)                (None, 80)           12880       ['flatten_1[0][0]']

dropout_1 (Dropout)            (None, 80)           0           ['dense_2[0][0]']

dense_3 (Dense)                (None, 160)          12960       ['dropout_1[0][0]']

reshape_1 (Reshape)            (None, 1, 160)       0           ['dense_3[0][0]']

tf.compat.v1.transpose_1 (TFOp  (None, 160, 1)      0           ['reshape_1[0][0]']
Lambda)

multiply_1 (Multiply)          (None, 160, 160)     0           ['reshape_1[0][0]',
                                                                 'tf.compat.v1.transpose_1[0][0]'
                                                                ]

tf.expand_dims_1 (TFOpLambda)  (None, 160, 160, 1)  0           ['multiply_1[0][0]']

max_pooling2d_3 (MaxPooling2D)  (None, 40, 40, 1)   0           ['tf.expand_dims_1[0][0]']

concatenate_3 (Concatenate)    (None, 40, 40, 66)   0           ['max_pooling2d_3[0][0]',
                                                                 'max_pooling2d_2[0][0]',
                                                                 'max_pooling2d_3[0][0]']

conv2d_10 (Conv2D)             (None, 40, 40, 128)  76160       ['concatenate_3[0][0]']

max_pooling2d_4 (MaxPooling2D)  (None, 20, 20, 128)  0          ['conv2d_10[0][0]']

conv2d_11 (Conv2D)             (None, 4, 4, 20)     64020       ['max_pooling2d_4[0][0]']

conv2d_12 (Conv2D)             (None, 4, 4, 20)     64020       ['max_pooling2d_4[0][0]']

conv2d_13 (Conv2D)             (None, 4, 4, 20)     64020       ['max_pooling2d_4[0][0]']

conv2d_14 (Conv2D)             (None, 4, 4, 20)     64020       ['max_pooling2d_4[0][0]']

activation_8 (Activation)      (None, 4, 4, 20)     0           ['conv2d_11[0][0]']

activation_9 (Activation)      (None, 4, 4, 20)     0           ['conv2d_12[0][0]']

activation_10 (Activation)     (None, 4, 4, 20)     0           ['conv2d_13[0][0]']

activation_11 (Activation)     (None, 4, 4, 20)     0           ['conv2d_14[0][0]']

layer_normalization_8 (LayerNo  (None, 4, 4, 20)    40          ['activation_8[0][0]']
rmalization)

layer_normalization_9 (LayerNo  (None, 4, 4, 20)    40          ['activation_9[0][0]']
rmalization)

layer_normalization_10 (LayerN  (None, 4, 4, 20)    40          ['activation_10[0][0]']
ormalization)

layer_normalization_11 (LayerN  (None, 4, 4, 20)    40          ['activation_11[0][0]']
ormalization)

tf.math.reduce_mean_8 (TFOpLam  (None, 20)          0           ['layer_normalization_8[0][0]']
bda)

tf.math.reduce_mean_9 (TFOpLam  (None, 20)          0           ['layer_normalization_9[0][0]']
bda)

tf.math.reduce_mean_10 (TFOpLa  (None, 20)          0           ['layer_normalization_10[0][0]']
mbda)

tf.math.reduce_mean_11 (TFOpLa  (None, 20)          0           ['layer_normalization_11[0][0]']
mbda)
```

```
concatenate_4 (Concatenate)    (None, 80)           0           ['tf.math.reduce_mean_8[0][0]',
                                                                 'tf.math.reduce_mean_9[0][0]',
                                                                 'tf.math.reduce_mean_10[0][0]',
                                                                 'tf.math.reduce_mean_11[0][0]']

flatten_2 (Flatten)            (None, 80)           0           ['concatenate_4[0][0]']

dense_4 (Dense)                (None, 40)           3240        ['flatten_2[0][0]']

dropout_2 (Dropout)            (None, 40)           0           ['dense_4[0][0]']

dense_5 (Dense)                (None, 80)           3280        ['dropout_2[0][0]']

reshape_2 (Reshape)            (None, 1, 80)        0           ['dense_5[0][0]']

tf.compat.v1.transpose_2 (TFOp  (None, 80, 1)       0           ['reshape_2[0][0]']
Lambda)

multiply_2 (Multiply)          (None, 80, 80)       0           ['reshape_2[0][0]',
                                                                 'tf.compat.v1.transpose_2[0][0]'
                                                                ]

tf.expand_dims_2 (TFOpLambda)  (None, 80, 80, 1)    0           ['multiply_2[0][0]']

max_pooling2d_5 (MaxPooling2D)  (None, 20, 20, 1)   0           ['tf.expand_dims_2[0][0]']

concatenate_5 (Concatenate)    (None, 20, 20, 130)  0           ['max_pooling2d_5[0][0]',
                                                                 'max_pooling2d_4[0][0]',
                                                                 'max_pooling2d_5[0][0]']

conv2d_15 (Conv2D)             (None, 20, 20, 256)  299776      ['concatenate_5[0][0]']

max_pooling2d_6 (MaxPooling2D)  (None, 10, 10, 256)  0          ['conv2d_15[0][0]']

conv2d_16 (Conv2D)             (None, 2, 2, 10)     64010       ['max_pooling2d_6[0][0]']

conv2d_17 (Conv2D)             (None, 2, 2, 10)     64010       ['max_pooling2d_6[0][0]']

conv2d_18 (Conv2D)             (None, 2, 2, 10)     64010       ['max_pooling2d_6[0][0]']

conv2d_19 (Conv2D)             (None, 2, 2, 10)     64010       ['max_pooling2d_6[0][0]']

activation_12 (Activation)     (None, 2, 2, 10)     0           ['conv2d_16[0][0]']

activation_13 (Activation)     (None, 2, 2, 10)     0           ['conv2d_17[0][0]']

activation_14 (Activation)     (None, 2, 2, 10)     0           ['conv2d_18[0][0]']

activation_15 (Activation)     (None, 2, 2, 10)     0           ['conv2d_19[0][0]']

layer_normalization_12 (LayerN  (None, 2, 2, 10)    20          ['activation_12[0][0]']
ormalization)

layer_normalization_13 (LayerN  (None, 2, 2, 10)    20          ['activation_13[0][0]']
ormalization)

layer_normalization_14 (LayerN  (None, 2, 2, 10)    20          ['activation_14[0][0]']
ormalization)

layer_normalization_15 (LayerN  (None, 2, 2, 10)    20          ['activation_15[0][0]']
ormalization)

tf.math.reduce_mean_12 (TFOpLa  (None, 10)          0           ['layer_normalization_12[0][0]']
mbda)

tf.math.reduce_mean_13 (TFOpLa  (None, 10)          0           ['layer_normalization_13[0][0]']
mbda)

tf.math.reduce_mean_14 (TFOpLa  (None, 10)          0           ['layer_normalization_14[0][0]']
mbda)

tf.math.reduce_mean_15 (TFOpLa  (None, 10)          0           ['layer_normalization_15[0][0]']
mbda)

concatenate_6 (Concatenate)    (None, 40)          0           ['tf.math.reduce_mean_12[0][0]',
                                                                 'tf.math.reduce_mean_13[0][0]',
                                                                 'tf.math.reduce_mean_14[0][0]',
                                                                 'tf.math.reduce_mean_15[0][0]']

flatten_3 (Flatten)            (None, 40)          0           ['concatenate_6[0][0]']

dense_6 (Dense)                (None, 20)          820         ['flatten_3[0][0]']

dropout_3 (Dropout)            (None, 20)          0           ['dense_6[0][0]']

dense_7 (Dense)                (None, 40)          840         ['dropout_3[0][0]']

reshape_3 (Reshape)            (None, 1, 40)       0           ['dense_7[0][0]']

tf.compat.v1.transpose_3 (TFOp  (None, 40, 1)      0           ['reshape_3[0][0]']
Lambda)

multiply_3 (Multiply)          (None, 40, 40)      0           ['reshape_3[0][0]',
                                                                 'tf.compat.v1.transpose_3[0][0]'
                                                                ]

tf.expand_dims_3 (TFOpLambda)  (None, 40, 40, 1)   0           ['multiply_3[0][0]']

max_pooling2d_7 (MaxPooling2D)  (None, 10, 10, 1)  0           ['tf.expand_dims_3[0][0]']

concatenate_7 (Concatenate)    (None, 10, 10, 258)  0          ['max_pooling2d_7[0][0]',
                                                                 'max_pooling2d_6[0][0]',
                                                                 'max_pooling2d_7[0][0]']

conv2d_20 (Conv2D)             (None, 10, 10, 512)  1189376    ['concatenate_7[0][0]']

max_pooling2d_8 (MaxPooling2D)  (None, 5, 5, 512)  0           ['conv2d_20[0][0]']

conv2d_21 (Conv2D)             (None, 1, 1, 5)     64005       ['max_pooling2d_8[0][0]']

conv2d_22 (Conv2D)             (None, 1, 1, 5)     64005       ['max_pooling2d_8[0][0]']

conv2d_23 (Conv2D)             (None, 1, 1, 5)     64005       ['max_pooling2d_8[0][0]']

conv2d_24 (Conv2D)             (None, 1, 1, 5)     64005       ['max_pooling2d_8[0][0]']

activation_16 (Activation)     (None, 1, 1, 5)     0           ['conv2d_21[0][0]']

activation_17 (Activation)     (None, 1, 1, 5)     0           ['conv2d_22[0][0]']

activation_18 (Activation)     (None, 1, 1, 5)     0           ['conv2d_23[0][0]']

activation_19 (Activation)     (None, 1, 1, 5)     0           ['conv2d_24[0][0]']

layer_normalization_16 (LayerN  (None, 1, 1, 5)    10          ['activation_16[0][0]']
ormalization)

layer_normalization_17 (LayerN  (None, 1, 1, 5)    10          ['activation_17[0][0]']
ormalization)

layer_normalization_18 (LayerN  (None, 1, 1, 5)    10          ['activation_18[0][0]']
ormalization)

layer_normalization_19 (LayerN  (None, 1, 1, 5)    10          ['activation_19[0][0]']
ormalization)

tf.math.reduce_mean_16 (TFOpLa  (None, 5)          0           ['layer_normalization_16[0][0]']
mbda)
```

```
tf.math.reduce_mean_17 (TFOpLa  (None, 5)           0           ['layer_normalization_17[0][0]']
mbda)

tf.math.reduce_mean_18 (TFOpLa  (None, 5)           0           ['layer_normalization_18[0][0]']
mbda)

tf.math.reduce_mean_19 (TFOpLa  (None, 5)           0           ['layer_normalization_19[0][0]']
mbda)

concatenate_8 (Concatenate)    (None, 20)           0           ['tf.math.reduce_mean_16[0][0]',
                                                                 'tf.math.reduce_mean_17[0][0]',
                                                                 'tf.math.reduce_mean_18[0][0]',
                                                                 'tf.math.reduce_mean_19[0][0]']

flatten_4 (Flatten)            (None, 20)           0           ['concatenate_8[0][0]']

dense_8 (Dense)                (None, 10)           210         ['flatten_4[0][0]']

dropout_4 (Dropout)            (None, 10)           0           ['dense_8[0][0]']

dense_9 (Dense)                (None, 20)           220         ['dropout_4[0][0]']

reshape_4 (Reshape)            (None, 1, 20)        0           ['dense_9[0][0]']

tf.compat.v1.transpose_4 (TFOp  (None, 20, 1)       0           ['reshape_4[0][0]']
Lambda)

multiply_4 (Multiply)          (None, 20, 20)       0           ['reshape_4[0][0]',
                                                                 'tf.compat.v1.transpose_4[0][0]'
                                                                ]

tf.expand_dims_4 (TFOpLambda)  (None, 20, 20, 1)    0           ['multiply_4[0][0]']

max_pooling2d_9 (MaxPooling2D)  (None, 5, 5, 1)     0           ['tf.expand_dims_4[0][0]']

concatenate_9 (Concatenate)    (None, 5, 5, 514)    0           ['max_pooling2d_9[0][0]',
                                                                 'max_pooling2d_8[0][0]',
                                                                 'max_pooling2d_9[0][0]']

conv2d_transpose (Conv2DTransp  (None, 10, 10, 512)  2369024    ['concatenate_9[0][0]']
ose)

conv2d_25 (Conv2D)             (None, 10, 10, 256)  594688      ['concatenate_7[0][0]']

concatenate_10 (Concatenate)   (None, 10, 10, 768)  0           ['conv2d_transpose[0][0]',
                                                                 'conv2d_25[0][0]']

dropout_5 (Dropout)            (None, 10, 10, 768)  0           ['concatenate_10[0][0]']

concatenate_11 (Concatenate)   (None, 10, 10, 1026  0           ['dropout_5[0][0]',
                               )                                 'concatenate_7[0][0]']

conv2d_transpose_1 (Conv2DTran  (None, 20, 20, 256)  2364160    ['concatenate_11[0][0]']
spose)

conv2d_26 (Conv2D)             (None, 20, 20, 128)  149888      ['concatenate_5[0][0]']

concatenate_12 (Concatenate)   (None, 20, 20, 384)  0           ['conv2d_transpose_1[0][0]',
                                                                 'conv2d_26[0][0]']

dropout_6 (Dropout)            (None, 20, 20, 384)  0           ['concatenate_12[0][0]']

concatenate_13 (Concatenate)   (None, 20, 20, 514)  0           ['dropout_6[0][0]',
                                                                 'concatenate_5[0][0]']

conv2d_transpose_2 (Conv2DTran  (None, 40, 40, 128)  592256     ['concatenate_13[0][0]']
spose)
```

```
 conv2d_27 (Conv2D)             (None, 40, 40, 64)   38080       ['concatenate_3[0][0]']

 concatenate_14 (Concatenate)   (None, 40, 40, 192)  0           ['conv2d_transpose_2[0][0]',
                                                                  'conv2d_27[0][0]']

 dropout_7 (Dropout)            (None, 40, 40, 192)  0           ['concatenate_14[0][0]']

 concatenate_15 (Concatenate)   (None, 40, 40, 258)  0           ['dropout_7[0][0]',
                                                                  'concatenate_3[0][0]']

 conv2d_transpose_3 (Conv2DTran  (None, 80, 80, 64)  148672      ['concatenate_15[0][0]']
 spose)

 conv2d_28 (Conv2D)             (None, 80, 80, 32)   9824        ['concatenate_1[0][0]']

 concatenate_16 (Concatenate)   (None, 80, 80, 96)   0           ['conv2d_transpose_3[0][0]',
                                                                  'conv2d_28[0][0]']

 dropout_8 (Dropout)            (None, 80, 80, 96)   0           ['concatenate_16[0][0]']

 concatenate_17 (Concatenate)   (None, 80, 80, 130)  0           ['dropout_8[0][0]',
                                                                  'concatenate_1[0][0]']

 up_sampling2d (UpSampling2D)   (None, 160, 160, 34  0           ['concatenate_1[0][0]']
                                )

 conv2d_transpose_4 (Conv2DTran  (None, 160, 160, 32  37472      ['concatenate_17[0][0]']
 spose)                         )

 conv2d_29 (Conv2D)             (None, 160, 160, 16  4912        ['up_sampling2d[0][0]']
                                )

 concatenate_18 (Concatenate)   (None, 160, 160, 48  0           ['conv2d_transpose_4[0][0]',
                                )                                 'conv2d_29[0][0]']

 dropout_9 (Dropout)            (None, 160, 160, 48  0           ['concatenate_18[0][0]']
                                )

 conv2d_30 (Conv2D)             (None, 160, 160, 3)  1299        ['dropout_9[0][0]']

==================================================================================================
Total params: 9,314,745
Trainable params: 9,314,745
Non-trainable params: 0
```

2.4 Performance Metric

The performance parameters are briefly discussed below:

The Dice coefficient is defined as the ratio of twice the intersection of the predicted and ground truth regions to the sum of the sizes of the predicted and ground truth regions. The Dice coefficient ranges from 0 to 1, where a value of 1 indicates a perfect overlap between the predicted and ground truth segmentations, and a value of 0 indicates no overlap at all.

$$\text{Dice coefficient} = \frac{2 * |\text{A intersect B}|}{(|\text{A}|+|\text{B}|)} \ldots\ldots\ldots\ldots\ldots\ldots (2.2)$$

The IOU coefficient is defined as the ratio of the intersection of the predicted and ground truth regions to the union of the predicted and ground truth regions. The IOU coefficient ranges from 0 to 1, where a value of 1 indicates a perfect overlap between the predicted and ground truth segmentations, and a value of 0 indicates no overlap at all.

IoU(A, B) = $\frac{A \cap B}{|A| + |B|}$……………………………...(2.3)

The Dice loss is defined as the complement of the Dice coefficient.

Dice Loss = 1 - Dice Coefficient……………..…(2.4)

The BCE loss component focuses on individual pixel or voxel classification, penalizing the model for incorrect predictions of foreground and background classes. It measures the similarity between the predicted probabilities and the ground truth labels using the binary cross-entropy formula.

BCE_loss = BCE + Dice Loss ……………..….(2.5)

The Tversky loss is defined based on the Tversky index, which is a similarity measure similar to the Dice coefficient. The Tversky index allows for a trade-off between false positives and false negatives by incorporating two parameters: alpha (α) and beta (β).

Tversky_loss α,β(y,ˆy) = $\frac{\mathrm{TP}}{\mathrm{TP}+\alpha\mathrm{FP}+\beta\mathrm{FN}}$ ………..…..(2. 6)

Where BCE means Binary Cross Entropy and for Tversky loss TP, FP and FN is the number of true positives, false positives and false negatives respectively and also α, β are the weighting factors for false negatives and false positives, respectively.

Chapter 3

Result Analysis

3.1 Simulation environment

All of the experimentations were implemented in Kaggle using NVIDIA P100 GPU with max 12.68 GB RAM, which is available for free. GPU specification changed sporadically due to Kaggle's GPU allocation policy. Sklearn.model_selection is used to split the dataset where 80% of the data is used to train the model and 20% for validation. 56 epochs are used to train the entire dataset and it costs about 11 hours. Inference of each 16 images using our model required only 25 ms.

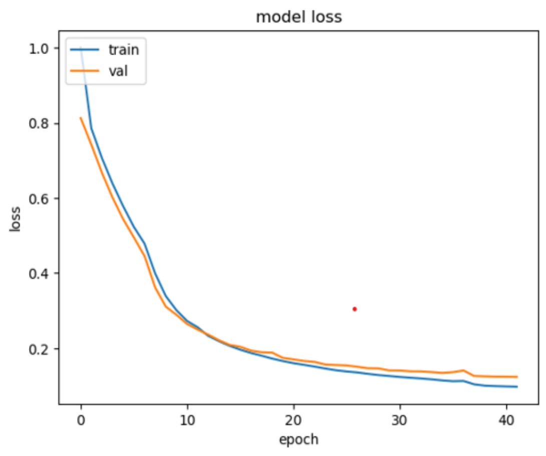
3.2 Optimization Algorithms

For image segmentation, the gradient has varying scale. To encounter this sparse gradient, we need to adapt the learning rate for each parameter individually. We use Adam Optimizer with initial learning rate as 1e-4.

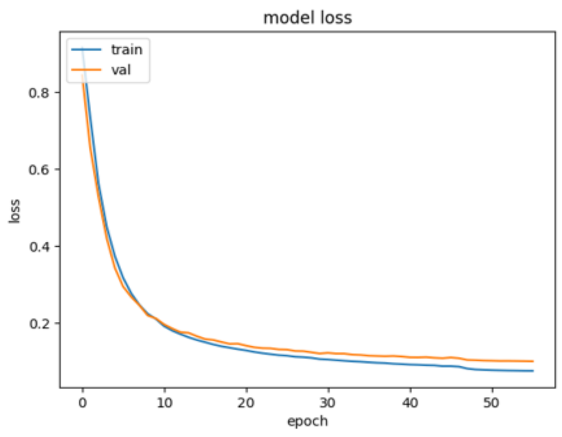
Early Stopping is used to prevent over fitting the model and also ReduceLROnPlateau is used to dynamically adjust the learning rate during model training.

3.3 Performance Comparison for different Loss functions

Loss functions provide a measure of how well the model is optimizing it's parameter during training. To evaluate our model, we use both BCE loss and Tversky loss.



Using BCE Loss



Using Tversky Loss

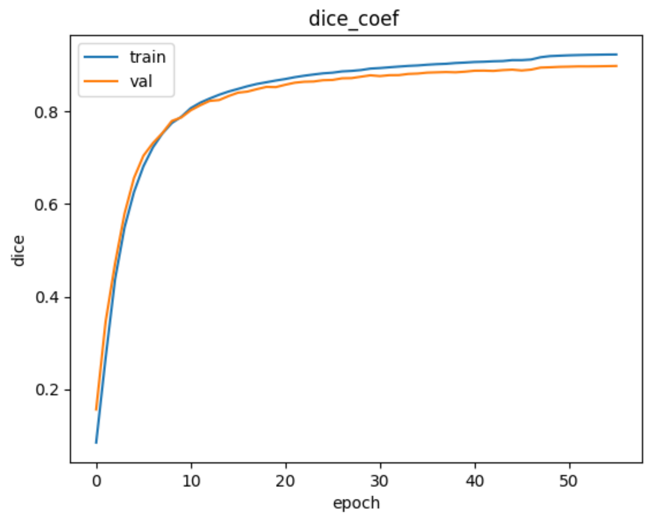
Figure 3. 1: Comparison between BCE loss and Tversky loss

Table 3.1: Loss Comparison

Loss	Train	Validation
BCE Loss	0.0975	0.1237
Tversky Loss	0.0765	0.1013

It is seen from table.3.1 that Tversky loss function works better in our model.

3.4 Performance Measure



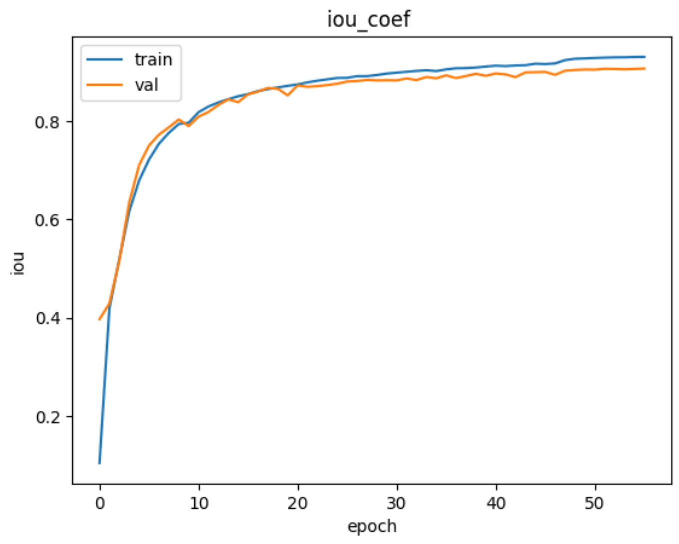


Figure 3. 2: Dice and IoU Coefficient Plot

In fig.3.2 we can see, the dice coefficient values for training and validation becomes stable after 40 epochs. Same thing can said in case of IoU coefficient values as shown in fig.8. The results are summarized in table.3.2

Table 3.2: Performance metrices for both train and validation

	IoU	Dice coefficient	Loss
Training	0.9313	0.9235	0.0765
Validation	0.912	0.9008	0.1073

3.4 Comparison with existing architecture

To compare our proposed model with existing models, we also train the same dataset for U-Net, U-Net++ where same number of epochs, same type of loss.

Table 3.3: Comparison with different models

Validation accuracy				
Models	IoU	Dice coefficient	Loss	Parameter Count
Unet	0.9107	0.9032	0.1032	36,893,154
Unet++	0.9023	0.9012	0.1062	10,365,568
Proposed Model	0.912	0.9008	0.1073	9,314,745

From table 3.3, we can say that the performance parameters are quite the same. U-Net needs over 36 million and U-Net++ needs over 10 million trainable parameters. Our proposed model only needs 9.3 million parameters which is about 74% less than U-Net and 11% less than U-Net++ . We able to reduce the parameter drastically. But they all have same efficiency in segmentation performance.

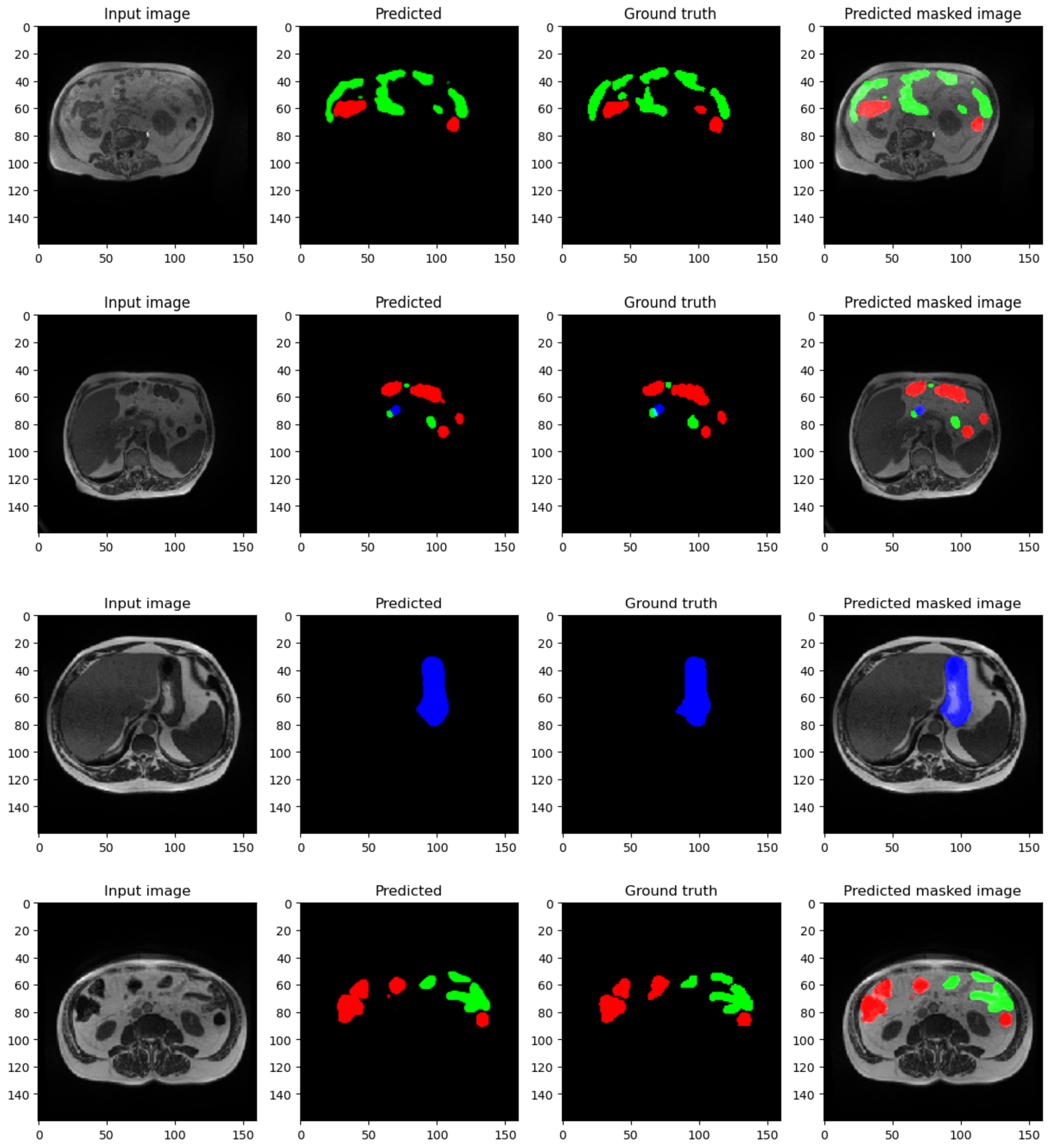


Figure 3. 3: Visual representation of predicted mask and comparison with ground truth

In fig.3.3, the predicted masks by our model are shown for 3 random images. Here, red denotes large bowel, green denotes small bowel and blue denotes stomach. It can be claimed that our model gives quite a good segmentation result.

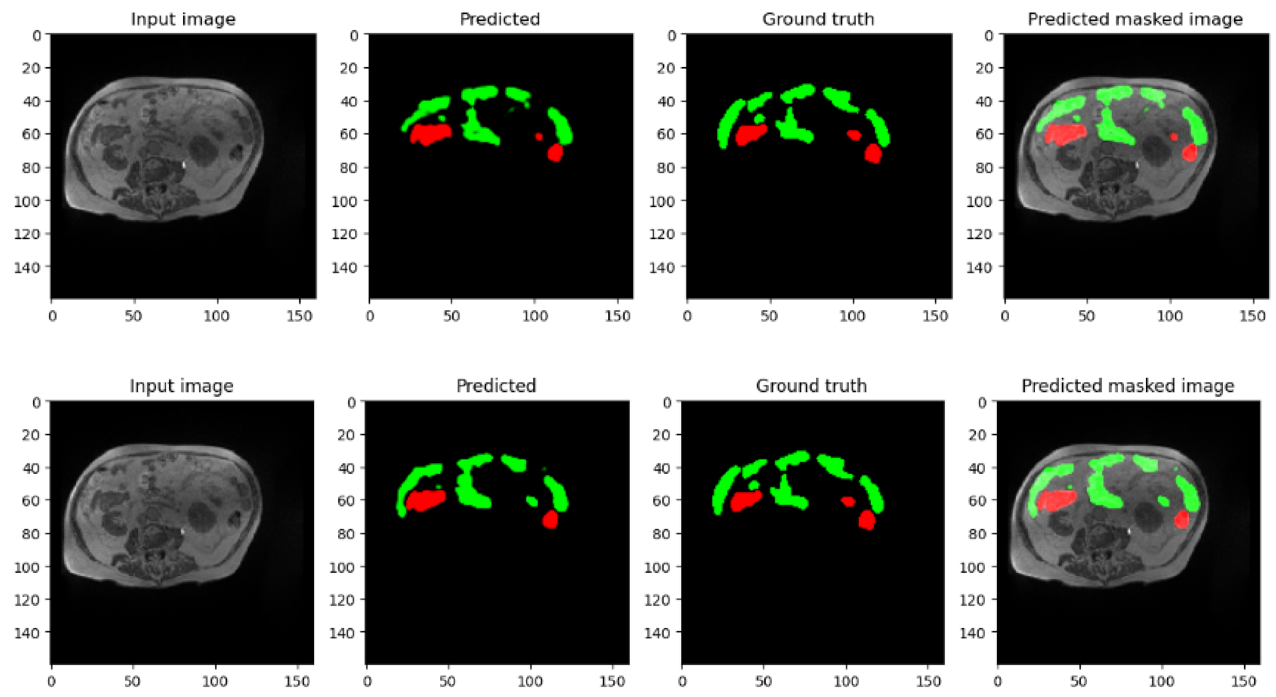


Figure 3. 4::Comparison of predicted masks (a) Our proposed model (b)U-Net

In fig.3.4, segmentation results for proposed model and U-Net are shown for the same image. We can claim that, in some region, our proposed architecture performs well, while in some region, U-Net performs well. On average the performance of our model is equivalent to that of U-Net. In table.2 the validation results for U-Net, U-Net++ and the proposed model are shown. In this table it is evident that, the values of coefficients and loss are almost same as U-Net and U-Net++. But the number of trainable parameters is drastically lower than the other two models.

Conclusion

From the description of the model and result analysis, it can be said that, the proposed model in this paper has successfully bypassed a well performing attention mechanism. It can maintain almost linear complexity while achieving reasonable performance. Above all, it can maintain all these requiring a very low number of trainable parameters. This shallow DNN can be embedded in less expensive medical devices and make better treatment more accessible. But we can’t claim the model is better than the other mentioned models in overall image segmentation implementing on one dataset. We look forward to comparing the performance of the model in case of large medical image datasets as well as natural image dataset. We will also try to integrate it with pretrained model and create a better architecture which will be more efficient.

Reference

[1] Ronneberger, Olaf & Fischer, Philipp & Brox, Thomas. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. LNCS. 9351. 234-241. 10.1007/978-3-319-24574-4_28.

[2] Vaswani, Ashish & Shazeer, Noam & Parmar, Niki & Uszkoreit, Jakob & Jones, Llion & Gomez, Aidan & Kaiser, Lukasz & Polosukhin, Illia. (2017). Attention Is All You Need.

[3] Petit, Olivier & Thome, Nicolas & Rambour, Clément & Themyr, Loic & Collins, Toby & Soler, Luc. (2021). U-Net Transformer: Self and Cross Attention for Medical Image Segmentation. 10.1007/978-3-030-87589-3_28.

[4] Zhou, Zongwei & Rahman Siddiquee, Md Mahfuzur & Tajbakhsh, Nima & Liang, Jianming. (2018). UNet++: A Nested U-Net Architecture for Medical Image Segmentation: 4th International Workshop, DLMIA 2018, and 8th International Workshop, ML-CDS 2018, Held in Conjunction with MICCAI 2018, Granada, Spain, September 20, 2018, Proceedings. 10.1007/978-3-030-00889-5_1.

[5] [UW-Madison]. ([2022, July]).[UW-Madison GI Tract Image Segmentation]. Retrived from July, 2022 from https://www.kaggle.com/competitions/uw-madison-gi-tract-image-segmentation/data

[6] Li, Yanghao & Wang, Naiyan & Liu, Jiaying & Hou, Xiaodi. (2017). Demystifying Neural Style Transfer. 2230-2236. 10.24963/ijcai.2017/310.

[7] Shamshad, Fahad & Khan, Salman & Zamir, Syed Waqas & Khan, Muhammad Haris & Hayat, Munawar & Khan, Fahad & Fu, Huazhu. (2022). Transformers in Medical Imaging: A Survey.

[8] Liu, Xiangbin & Song, Liping & Liu, Shuai & Zhang, Yudong. (2021). A Review of Deep-Learning-Based Medical Image Segmentation Methods. Sustainability. 13. 1224. 10.3390/su13031224.

[9] Dosovitskiy, Alexey & Beyer, Lucas & Kolesnikov, Alexander & Weissenborn, Dirk & Zhai, Xiaohua & Unterthiner, Thomas & Dehghani, Mostafa & Minderer, Matthias & Heigold, Georg & Gelly, Sylvain & Uszkoreit, Jakob & Houlsby, Neil. (2020). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale.

[10] Liu, Ze & Lin, Yutong & Cao, Yue & Hu, Han & Wei, Yixuan & Zhang, Zheng & Lin, Stephen & Guo, Baining. (2021). Swin Transformer: Hierarchical Vision Transformer using Shifted Windows.

[11] Zhang, Yundong & Liu, Huiye & Hu, Qiang. (2021). TransFuse: Fusing Transformers and CNNs for Medical Image Segmentation. 10.1007/978-3-030-87193-2_2.

[12] Huang, Huimin & Lin, Lanfen & Tong, Ruofeng & Hu, Hongjie & Qiaowei, Zhang & Iwamoto, Yutaro & Han, Xian-Hua & Chen, Yen-Wei & Wu, Jian. (2020). UNet 3+: A Full-Scale Connected UNet for Medical Image Segmentation.

[13] Najeeb, Suhail & Bhuiyan, M.. (2022). Spatial feature fusion in 3D convolutional autoencoders for lung tumor segmentation from 3D CT images. Biomedical Signal Processing and Control. 78. 103996. 10.1016/j.bspc.2022.103996..

[14] Meng, Zhu & Fan, Zhongyue & Zhao, Zhicheng & Su, Fei. (2018). ENS-Unet: End-to-End Noise Suppression U-Net for Brain Tumor Segmentation. Conference proceedings: ... Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Conference. 2018. 5886-5889. 10.1109/EMBC.2018.8513676.

[15] Wu, Junyan & Chen, Eric & Rong, Ruichen & Li, Xiaoxiao & Xu, Dong & Jiang, Hongda. (2019). Skin Lesion Segmentation with C-UNet. Conference proceedings: ... Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Conference. 2019. 2785-2788. 10.1109/EMBC.2019.8857773.

[16] H. Kan et al., "ITUnet: Integration Of Transformers And Unet For Organs-At-Risk Segmentation," 2022 44th Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC), Glasgow, Scotland, United Kingdom, 2022, pp. 2123-2127, doi: 10.1109/EMBC48229.2022.9871945.

[17] Touvron, Hugo & Cord, Matthieu & Douze, Matthijs & Massa, Francisco & Sablayrolles, Alexandre & Jégou, Hervé. (2020). Training data-efficient image transformers & distillation through attention.

[18] Yang, Jianwei et al. “Focal Self-attention for Local-Global Interactions in Vision Transformers.” *ArXiv* abs/2107.00641 (2021): n. pag.