UNIVERSIDAD DE INVESTIGACIÓN DE TECNOLOGÍA EXPERIMENTAL YACHAY

Escuela de Ciencias Matemáticas y Computacionales

TÍTULO: Genetic Algorithms for Hyperparameter Tuning of a DC-UNet Model for Medical Image Segmentation

Trabajo de integración curricular presentado como requisito para la obtención del título de Ingeniero en Tecnologías de la Información

Autor:

Román Eras Krishna Gautama

Tutor:

Ph.D. - Armas Andrade Tito Rolando

Co-tutor:

Ph.D. - Morocho Cayamcela Manuel Eugenio

Urcuquí, Noviembre de 2023

Autoría

Yo, **Krishna Gautama Román Eras**, con cédula de identidad 2300776594, declaro que las ideas, juicios, valoraciones, interpretaciones, consultas bibliográficas, definiciones y conceptualizaciones expuestas en el presente trabajo; así cómo, los procedimientos y herramientas utilizadas en la investigación, son de absoluta responsabilidad de el/la autor/a del trabajo de integración curricular. Así mismo, me acojo a los reglamentos internos de la Universidad de Investigación de Tecnología Experimental Yachay.

Urcuquí, Noviembre de 2023.

Krishna Gautama Román Eras

CI: 2300776594

Autorización de publicación

Yo, **Krishna Gautama Román Eras**, con cédula de identidad 2300776594, cedo a la Universidad de Investigación de Tecnología Experimental Yachay, los derechos de publicación de la presente obra, sin que deba haber un reconocimiento económico por este concepto. Declaro además que el texto del presente trabajo de titulación no podrá ser cedido a ninguna empresa editorial para su publicación u otros fines, sin contar previamente con la autorización escrita de la Universidad.

Asimismo, autorizo a la Universidad que realice la digitalización y publicación de este trabajo de integración curricular en el repositorio virtual, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación

Urcuquí, Noviembre de 2023.

Firmado electrónicamente por:
KRISHNA GAUTAMA
ROMAN ERAS

Krishna Gautama Román Eras

CI: 2300776594

Dedication

Dedicado con mucho cariño para mi familia, que me ha brindado su apoyo incondicional en todas las etapas de mi vida. Cada logro alcanzado ha sido y será el suyo también.

Krishna Gautama Román Eras

Acknowledgment

Quiero comenzar agradeciendo a mis padres, la educación y los valiosos consejos de vida que he recibido de ellos han sido fundamentales en mi crecimiento y desarrollo personal. Les debo ser la persona que soy en la actualidad. Asimismo, quiero extender mi gratitud a mis hermanos mayores, quienes han desempeñado un papel crucial al mantener unida a nuestra familia durante los momentos más difíciles.

Por otro lado, no puedo dejar de agradecer a las personas que me han acompañado en esta etapa universitaria. A mis amigos y compañeros de largas jornadas de estudio, como David, Jerry, Gilda, Stadyn, Carlos, Stalyn y Andy, les agradezco su constante apoyo, no solo en lo académico. También, quiero destacar a mis hermanos Horus y Osiris, quienes siempre han estado a mi lado.

Así mismo a personas que han compartido conmigo grandes momentos durante mis últimos años de estancia en la universidad, especialmente Nelly y Kenia, mi vida en el campus no hubiese sido la misma sin su valiosa compañía.

Agradezco a mis tutores Rolando Armas, Manuel Morocho y Paola Quiloango quienes han aportado mucho en este trabajo con sus valiosos consejos y retroalimentación. Además, quiero reconocer la labor de los profesores que han dejado una huella significativa en mi crecimiento académico. Entre ellos puedo mencionar a Rasha Mohamed, Eusebio Ariza, Franklin Camacho, Josephine Javens, Isidro Amaro, Francisco Hidrobo, Cédric Campos, Israel Pineda, Fredy Cuenca, Saba Infante y Erick Cuenca.

Una vez más, gracias a todos por su contribución en mi camino por este mundo de la investigación, ciencia y tecnología. Somos Yachay Tech!

Krishna Gautama Román Eras

Resumen

La visión por computador es una rama de la inteligencia artificial que permite a las máquinas extraer información de imágenes y realizar tareas como la segmentación de imágenes, que implica dividir una imagen en múltiples regiones para identificar diferentes elementos. La segmentación de imágenes se aplica en el área médica para ayudar al personal de la salud en el diagnóstico de enfermedades de los pacientes basándose en información visual, y se requiere que tengan la mayor precisión posible. En este trabajo, se utiliza un modelo de visión por computador llamado dual channel U-Net (DC-UNet) para la segmentación de imágenes médicas. Específicamente nos enfocamos en el área de la detección de pólipos que son lesiones que pueden variar en tamaño desde unos pocos milímetros hasta varios centímetros, y la importancia de esta aplicación radica en la identificación temprana para la prevención del cáncer colorrectal. Para entrenar el modelo de segmentación se empleó uno de los conjuntos de datos públicos más desafiantes en este campo, llamado CVC-ClinicDB. Estas imágenes médicas corresponden a fotogramas extraídos de vídeos de colonoscopia, cuyas imágenes de referencia consisten en una segmentación binaria entre el pólipo y el fondo. Además, para aumentar el rendimiento del modelo DC-UNet en este desafiante conjunto de datos, proponemos un algoritmo genético que encuentra la combinación de hiperparámetros óptima para esta aplicación en específico. Finalmente, utilizamos diferentes configuraciones genéticas para estudiar el rendimiento de algunos optimizadores en el estado del arte basados en el gradiente con respecto a esta tarea.

Palabras Clave:
Segmentación de imágenes médicas, ajuste de hiperparámetros, algoritmos genéticos, visión por computador, optimización.

Abstract

Computer vision is a branch of artificial intelligence that enables computers to extract information from images and perform tasks such as image segmentation, which involves identifying multiple elements as image regions. Then, the application of image segmentation in the medical area is used to assist physicians in disease diagnosis from patients based on visual information, and they are required to have the best possible accuracy. In this work, a computer vision model called dual channel U-Net (DC-UNet) is used for medical image segmentation. Specifically, we focus on the area of polyp detection which are lesions that can vary in their size from a few millimeters to several centimeters, and the importance of this application relies on the early identification for colorectal cancer prevention. One of the most challenging public datasets in this field called CVC-ClinicDB was employed to train the segmentation model. These medical images correspond to colonoscopy video frames, whose ground truth images consist of a fully annotated binary segmentation between polyp and background. Furthermore, to increase the performance of the DC-UNet model on this challenging dataset, we propose a genetic algorithm that finds the optimal hyperparameter combination for this specific application. Finally, we use different genetic configurations to study the performance of some state of the art gradient-based optimizers regarding this task.

Keywords:
Medical image segmentation, hyperparameter tuning, genetic algorithms, computer vision, optimization.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

Image segmentation is a branch of computer vision that consists of extracting information and minimizing the region of interest to recognize particular areas that need to be analyzed for any kind of purpose. One advantage is that it can remove undesired features and isolate them from the image. The fact of doing it manually would be time-consuming in addition to the need of an expert in the area where it is applied. However, the use of artificial intelligence allows these kinds of repetitive tasks to be done automatically by models that are getting more accurate. Most of the first segmentation models were based on machine learning. Region-based methods find groups of pixels with similar properties while edge segmentation consists of finding abrupt property changes in the image, which will be the boundaries between regions, and segmentation based on the threshold is based on big contrasts between the background and the object [1]. However, these are traditional methods that can not overcome challenging problems like the CVC-ClinicDB dataset. So, that is why currently, it is usually done by deep learning as it has been at the center of attention since some years ago. The segmentation could be simply to classify each single pixel of the image. Then, one particular application of image segmentation in the realm of medical imaging is medical image segmentation. The analysis of medical images has been playing an important role in supporting medical staff to diagnose diseases efficiently. In the field of medical imaging, there are a variety of ways it can be obtained such as x-ray, computed tomography (CT), magnetic resonance imaging (MRI), etc. This kind of

data is usually used to identify and segment different organs or anomalies such as tumors, liver, prostate, knee cartilage, and the brain [2]. Its main purpose is to assist radiologists and physicians in the diagnosis or treatment of illnesses or diseases from their patients based on the visual information presented in the images [3]. For instance, having a more thorough understanding of the anatomy and physiology of a patient's body can help medical practitioners plan and carry out therapies more successfully. Additionally, medical image segmentation can be used to highlight certain structures or areas of interest in real-time images, providing real-time assistance during procedures like minimally invasive surgeries.

Due to its importance, researchers have been dealing with a lot of challenges regarding medical image segmentation, and some of them are related to deep learning models such as overfitting and training time [4]. Overfitting occurs when a model is trained to memorize the patterns and regularities in the training data, but fails to generalize to new unseen data, resulting in poor performance compared with the training data [5]. Furthermore, reducing the training time is another topic of study because has a lot of benefits regarding efficiency.

Some deep learning models like U-Net [6] and its variants are the most popular convolutional neural network architectures related to medical image segmentation. However, this kind of model has some parameters that can not be learned during the training process, and those are called hyperparameters because they are fixed previously by the experimenter. It is important to select the correct set of hyperparameters as the impact they have on the performance of the model is significant. The process of finding the best hyperparameter configuration could be difficult to approximate. There are some brute force approach methods to solve this task such as grid search or random search. They are usually used because of their ease of implementation, but these do not help either to converge to a locally optimal solution.

On the other hand, genetic algorithms are optimization algorithms inspired by the process of natural selection. In genetic algorithms, potential solutions—a collection of hyperparameters—are produced at random and iteratively improved by using genetic operators like crossover and mutation to create children with better characteristics [7]. Therefore, genetic algorithms are a good choice to do hyperparameter tuning of a deep learning model. The population eventually evolves towards a set of hyperparameters that perform well on

the task. The fitness of each solution is determined by evaluating the performance of the deep learning model on a validation set. The process continues until a desirable solution is found or a stopping criterion is reached.

Genetic algorithms are an optimization method well suited for hyperparameter tunning in deep learning because they can handle complex and high-dimensional search spaces and are easy to parallelize [8]. It is helpful because the search space in this case is not finite as it has some real values such as dropout and learning rate. Also, as hyperparameter tunning is a time-consuming task it could be tackled by using high-performance computing techniques to considerably reduce the training time.

1.2 Problem statement

Some challenges regarding medical image segmentation are scarcity of data, class imbalance in the ground truth, high memory demand, and datasets with a limited number of images [9]. The problem to solve in this work is the image segmentation of the dataset CVC-ClinicDB which consists basically of endoscopy images to detect polyps. In total, 612 images correspond to 29 video frames. The architecture to be used in this work is DC-UNet [1] which is a variant of the known U-Net [6]. Then, genetic algorithms will be used to find the best configuration of the hyperparameters in this case batch size, dropout, and optimizer learning rate. Grid search is not considered because there are a lot of values to try in every hyperparameter, and that would be computationally expensive as it requires evaluating every single combination to know which is the best. Genetic algorithms mimic the process of evolution and select the solution based on natural selection. Soon, there will be a performance analysis of some optimizers to figure out which performs better for this specific model and dataset. The optimizers used in this work are RMSProp, Adam, Nadam, and AMSGrad.

1.3 Objectives

1.3.1 General Objective

Find the best hyperparameter combination in medical image segmentation with DC-UNet model and CVC-ClinicDB dataset using optimization based on genetic algorithms.

1.3.2 Specific Objectives

1. Find the optimal optimizer algorithm which outcomes the best performance for this specific task.

2. Provide great segmentation results using a state of the art medical image segmentation model with a challenging dataset.

3. Analyze the effectiveness of genetic algorithms for the hyperparameter tuning challenge.

Chapter 2

Theoretical Framework

In this chapter, key concepts of deep learning, medical image segmentation, and genetic algorithms are introduced. These topics provide a better understanding of how the hyper-parameter tuning works with the use of genetic algorithms.

2.1 Artificial Neural Networks and Optimization

Artificial neural networks (ANNs) are inspired by the human brain which is composed of interconnected neurons that receive and transmit electrochemical signals from other neurons across a synapse. With some attempts to give computers the capacity of learning, McCulloch and Pitts place the idea that some computational elements inspired in biological neurons can perform complex computations, simplifying them to binary on/off activation based input signals [10]. After that, Frank Rosenblatt made a significant contribution by designing the perceptron, which was developed to learn and identify patterns through supervised learning [11]. The perceptron algorithm updated the weights to make accurate predictions based on the input patterns, however, one of its limitations was the inability to solve non-linearly separable problems. After that, Minsky and Papert propose the idea of multilayer perceptrons, introducing the concept of multiple layers of neurons between the input and output layers, and they call them hidden layers [12]. Although they discussed the potential of multilayer perceptrons, they did not provide an algorithm for training this kind of network. Some years later, the idea of the backpropagation algorithm came to the scene by multiple contributions of several researchers in the 1980s using the chain rule of calculus to adjust weights in a neural network by calculating the gradient.

ANNs have many architectures based on connection patterns, and the most common is the feed-forward neural network. In this architecture, its neurons and connections in the hidden layers can be seen as graphs with no loops or cycles, so the information is transferred in a unidirectional way. Each neuron receives multiple inputs that have associated weights. The weighted sum of inputs is passed to an activation function that will be the output for that neuron. The sigmoid activation function is often used because this function and its derivative are continuous. So, the output of each neuron is calculated by:

$$O = f\left(\sum_{i=1}^{n} w_i x_i\right) \tag{2.1}$$

where f is an arbitrary activation function while x and w are the input and weight vectors respectively [13].

The problem to solve by applying supervised learning through artificial neural networks is to optimize a loss function by adjusting the weights for all neurons. To formalize the inference problem, it can be mathematically described as follows:

$$w_* = \underset{w \in \mathbb{R}^n}{argmin}\left\{\frac{1}{|X|}\sum_{x \in X} L(x, w)\right\}, \tag{2.2}$$

where w_* is the optimum value, w is the weight vector to be optimized with dimension n, and $L(x, w)$ is the computed loss from samples in the training set $x \in X$ [14].

In deep learning, the optimization problems are usually defined over a large training set, which requires high computational resources. Several optimizing algorithms allow us to solve this task, and the most popular ones are stochastic gradient descent (SGD) and its variants. These gradient-based algorithms select a random mini-batch of the training set at each iteration to evaluate the loss function and its gradient. Small batch sizes tend to be untrustworthy, while calculating gradients for full-batch computations is frequently challenging, resulting in a balance between stability and effectiveness. Then, if we take a random mini-batch B_t, the weights are updated at each iteration t by the following rule:

$$w_t = w_{t-1} - \alpha_t\left(\frac{1}{|B_t|}\sum_{x \in B_t} \nabla L(x, w_t)\right), \tag{2.3}$$

where α_t and $|B_t|$ are the learning rate and batch size hyperparameters [15]. In gradient-

based optimization algorithms, the learning rate is the step size when adjusting the weights at each iteration. For a better understanding of how neural networks are optimized, let's talk about all the optimization algorithms we use in this work.

2.1.1 RMSProp

The root mean squared propagation algorithm, which is best known as the RMSProp algorithm, is an effective method for optimizing deep neural networks. Inspired by SGD, it is designed in such a way it easily converges in non-convex settings. This optimizer uses an exponentially weighted moving average of the squared gradient to update the weights. During training, the exponential decaying average ignores history from far away past to quickly converge after identifying a convex area [16]. This process considers all different weights separately accelerating the step sizes in small gradients while slowing down the big ones. So, this algorithm speeds up the optimization at the same time avoids overshooting the minima. The RMSProp algorithm is illustrated in Algorithm 1 in its standard form.

Algorithm 1: RMSProp optimizer algorithm

Require: Global learning rate α, discounting factor for past gradients β
Require: Initial parameter w
Require: A small constant ϵ for numerical stability, by default 10^{-7}

$r \leftarrow 0$ `/* Initialize accumulative squared gradient variable */`;
while *stopping criterion not met* **do**
  Sample a mini-batch of size n from the training set $\{x_1, ..., x_n\}$;
  $g \leftarrow \frac{1}{n}\nabla_w \sum_i L(x_i, w)$ `/* Compute gradient */`;
  $r \leftarrow \beta r + (1-\beta) g \odot g$ `/* Accumulate squared gradient */`;
  $\nabla w \leftarrow -\frac{\alpha}{\sqrt{\epsilon + r}} \odot g$ `/* Compute update parameter */`;
  $w \leftarrow w + \nabla w$ `/* Apply update */`;
end while

2.1.2 Adam

The Adam optimizer, whose name comes from adaptive moments, is considered an upgraded algorithm that combines the advantages of RMSProp and SGD with momentum [16]. This fact, uses the exponential moving average of the gradient and the squared gradient to update the weights, making the method more robust [17]. Additionally, it introduces

a correction method for the first and second order moments estimates to deal with the moving average variables initialized as zero vectors. By rescaling the gradient, the start of the process is not limited by the exponential decay B_1 and B_2. Finally, Adam optimizer is presented in Algorithm 2.

Algorithm 2: Adam optimizer algorithm

Require: Global learning rate α
Require: Exponential decay rates for moment estimates B_1 and B_2 in $[0, 1)$, by default 0.9 and 0.999 respectively
Require: A small constant ϵ for numerical stability, by default 10^{-8}

$m \leftarrow 0$, $v \leftarrow 0$ `/* Initialize 1st and 2nd moment vectors */`;
$t \leftarrow 0$ `/* Initialize time-step */`;
while *stopping criterion not met* **do**
  Sample a mini-batch of size n from the training set $\{x_1, ..., x_n\}$;
  $g \leftarrow \frac{1}{n} \nabla_w \sum_i L(x_i, w)$ `/* Compute gradient */`;
  $t \leftarrow t + 1$;
  $m \leftarrow \beta_1 m + (1 - \beta_1) g$ `/* Update biased 1st moment estimate */`;
  $v \leftarrow \beta_2 v + (1 - \beta_2) g \odot g$ `/* Update biased 2nd moment estimate */`;
  $\hat{m} \leftarrow \frac{m}{1 - \beta_1^t}$ `/* Correct bias in 1st moment estimate */`;
  $\hat{v} \leftarrow \frac{v}{1 - \beta_2^t}$ `/* Correct bias 2nd moment estimate */`;
  $\nabla w \leftarrow -\frac{\alpha \hat{m}}{\sqrt{\hat{v}} + \epsilon}$ `/* Compute update parameter */`;
  $w \leftarrow w + \nabla w$ `/* Apply update */`;
end while

2.1.3 Nadam

Nadam optimizer can be seen as Adam with Nesterov momentum, known as Nesterov accelerated gradient. What the Nesterov momentum does is evaluate the gradient after the current velocity of the past iterations is applied. This type of momentum is sometimes superior to the classical momentum used in SGD based optimizers. However, when trying Nesterov momentum to be combined with Adam, the mathematical operations behind this are not intuitive. Additionally, they establish the notion of the variable μ (in our notation β_1) to be indexed by time-step $\mu_1, ..., \mu_T$ because it often helps to gradually increase or decrease this value over iterations [18]. The pseudocode for the Nadam optimizer algorithm is included in Algorithm 3, considering a constant value for β_1 for simplicity.

Algorithm 3: Nadam optimizer algorithm

Require: Global learning rate α

Require: Exponential decay rates for moment estimates B_1 and B_2 in $[0, 1)$, by default 0.9 and 0.999 respectively

Require: A small constant ϵ for numerical stability, by default 10^{-8}

$m \leftarrow 0$, $v \leftarrow 0$ `/* Initialize 1st and 2nd moment vectors */`;

$t \leftarrow 0$ `/* Initialize time-step */`;

while *stopping criterion not met* **do**

  Sample a mini-batch of size n from the training set $\{x_1, ..., x_n\}$;

  $g \leftarrow \frac{1}{n} \nabla_w \sum_i L(x_i, w)$ `/* Compute gradient */`;

  $t \leftarrow t + 1$;

  $m \leftarrow \beta_1 m + (1 - \beta_1) g$ `/* Update biased 1st moment estimate */`;

  $v \leftarrow \beta_2 v + (1 - \beta_2) g \odot g$ `/* Update biased 2nd moment estimate */`;

  $\hat{m} \leftarrow \frac{m}{1 - \beta_1^{t+1}}$ `/* Correct bias in 1st moment estimate */`;

  $\hat{v} \leftarrow \frac{v}{1 - \beta_2^t}$ `/* Correct bias in 2nd moment estimate */`;

  $\bar{m} \leftarrow \frac{(1 - \beta_1)}{(1 - \beta_1^t)} g + \beta_1 \hat{m}$ `/* Nesterov trick */`;

  $\nabla w \leftarrow -\frac{\alpha \bar{m}}{\sqrt{\hat{v}} + \epsilon}$ `/* Compute update parameter */`;

  $w \leftarrow w + \nabla w$ `/* Apply update */`;

end while

2.1.4 AMSGrad

AMSGrad is considered a variant of Adam that incorporates the property of non-increasing step size which is missing in some optimizers, and it improves the convergence properties in some scenarios. Specifically, those optimizers are the exponential moving average based, like RMSProp and Adam. In AMSGrad work, researchers show that when the variance of the gradients concerning time is large, Adam can not converge to an optimal solution [19]. So, they propose an algorithm that does not lose the advantages of an exponential moving average optimizer relying on long-term memory for the past gradients. Then, this optimizer requires a smaller learning rate because of the slow decay concerning the previous gradients. The AMSGrad optimizer is illustrated in Algorithm 4, considering fixed values for hyperparameters α and β_1.

Algorithm 4: AMSGrad optimizer algorithm

Require: Global learning rate α

Require: Exponential decay rates for moment estimates B_1 and B_2 in $[0, 1)$, by default 0.9 and 0.999 respectively

Require: A small constant ϵ for numerical stability, by default 10^{-8}

$m \leftarrow 0$, $v \leftarrow 0$ `/* Initialize 1st and 2nd moment vectors */`;

$\hat{v} \leftarrow 0$ `/* Initialize the maximum 2nd moment vector */`;

$t \leftarrow 0$ `/* Initialize time-step */`;

while *stopping criterion not met* **do**

  Sample a mini-batch of size n from the training set $\{x_1, ..., x_n\}$;

  $g \leftarrow \frac{1}{n} \nabla_w \sum_i L(x_i, w)$ `/* Compute gradient */`;

  $t \leftarrow t + 1$;

  $m \leftarrow \beta_1 m + (1 - \beta_1) g$ `/* Update 1st moment estimate */`;

  $v \leftarrow \beta_2 v + (1 - \beta_2) g \odot g$ `/* Update 2nd moment estimate */`;

  $\hat{v} \leftarrow max(\hat{v}, v)$ `/* Applied element-wise */`;

  $\hat{V} \leftarrow diag(\hat{v})$;

  $\nabla w \leftarrow -\frac{\alpha \bar{m}}{\sqrt{\hat{v}} + \epsilon}$ `/* Compute update parameter */`;

  $w \leftarrow \underset{w \in \mathbb{R}^n}{argmin} ||\hat{V}^{1/2}(w + \nabla w)||$ `/* Update via projection operation */`;

end while

2.2 Medical Image Segmentation

In computer vision, image segmentation involves to group pixels of an image into different and separate classes called sections, and these sections are meant to depict different elements of the image such as the foreground, background, or an item [20]. There are numerous ways to complete this task that come from manual segmentation by radiologists or other clinicians with the appropriate training to automated segmentation using various computational techniques. In recent years, machine learning as well as deep learning methods, have become effective ways for medical image segmentation, and they have produced state-of-the-art outcomes in numerous applications. As these techniques lay the groundwork for understanding the evolutionary algorithm approach to hyperparameter tuning that we are applying in this research, we will explore these approaches in this section, along with their benefits and drawbacks. Before explaining some image segmentation techniques, let's introduce convolutional neural networks as it is one of the most popular methods used in computer vision.

2.2.1 Convolutional Neural Networks

Convolutional neural networks (CNNs) have demonstrated outstanding results in a variety of computer vision and machine learning challenges. They were born as an improvement of how ANNs deal with images. As we know, ANN models receive one-dimensional vectors as input, so images need to pass through a flattening process. This process destroys the spatial distribution of pixels in the image. To preserve that information after the flattening process, CNNs use convolutional and pooling layers which increment the receptive field in the subsequent feature maps.

Convolutional layer

The convolutional layer applies multiple feature detectors to the input data. A feature detector is a convolution matrix that computes an element-wise product with a region of the same size in the input, and it results in a single number by summing these products together [21]. The previous process is repeated for every position from the top left to the bottom right corner of the input sliding a predetermined amount of pixels named as stride. Single units are connected with local receptive fields on the input, that extract elementary visual features such as edges, corners, or textures [22]. Then, the resulting output of applying a feature detector is called a feature map, and it is typically followed by a rectified linear unit (ReLU) function to add non-linearity to learn complex relationships. Finally, the convolution operation for a 3×3 input and 2×2 feature detector is shown in Figure 2.1 with a stride of 1.

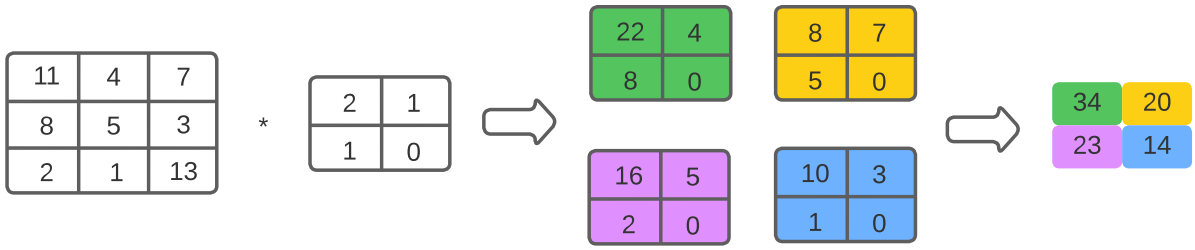


Figure 2.1: Convolution in a 3×3 image with 2×2 feature detector and stride of 1.

Pooling layer

In pooling layers, the most common operations are max pooling and average pooling. These operations transform a subregion into an nxn window into a pixel which is represented by a number. Similar to convolutions, this process is done by covering all regions of the image with steps predefined by the stride, but this operation needs no parameters. In max pooling the resulting number is the maximum value from that subregion while average pooling takes the average. With these layers, spatial invariance is achieved in the feature maps with a lower resolution, ignoring its exact position [23]. Additionally, features can be detected even if they are not the same, detecting small changes. We can see the max pooling operation in Figure 2.2 for a 6×4 image with a 2×2 sliding widow and stride of 2.

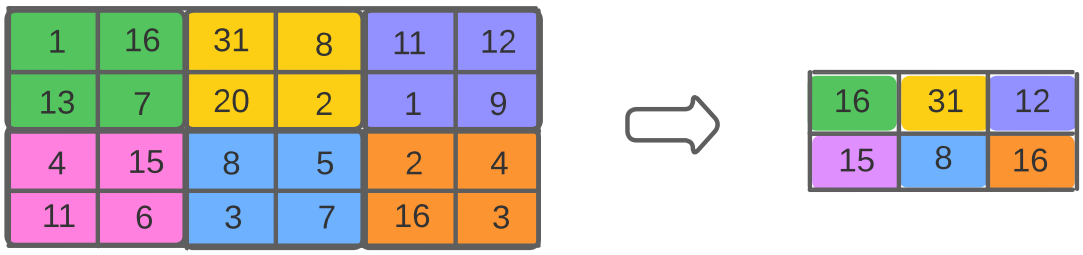


Figure 2.2: Max pooling in a 6×4 image with a 2×2 sliding window and stride of 2.

2.2.2 CNN-based Semantic Segmentation

Segmentation models based on deep learning are a good option to solve this task because they considerably increase the accuracy of results with CNN structures. The backbone of a CNN-based image segmentation model is the CNN architecture. The architecture should be selected by the particular task and dataset, and it can be modified for better performance. Some approaches like [24] construct a classifier by patches in the input image through a sliding window, so that each pixel is classified by the context of surrounding pixels. Their final prediction is based on fully connected layers after convolutional blocks. However, the limitation of this method is the restriction of learning global features, especially when the patch size is relatively small. In [25], fully convolutional networks were proposed by replacing fully connected layers with 1×1 convolutions. Their 1×1 convolutions have one

channel per class so that each one contains a class score prediction. This model applies deconvolutions as a method to backward-strided convolutions and pooling layers. Finally, it returns the full segmented image at once.

2.2.3 U-Net Architecture

U-Net [6] is a neural network architecture that uses convolutional layers to perform the segmentation. Indeed, this architecture is based on the fully convolutional network [25] mentioned before, so it does not have fully connected layers. The structure of U-Net is almost symmetric where on one side we have the encoder and on the other the decoder, and it is depicted in Figure 2.3. The encoder is used to extract spatial features. It has four convolution blocks each of them representing 3×3 convolutions and 2×2 max pooling with a stride of 2. After each convolution block, the down-sampling operation performed by max pooling reduces the size of the feature maps by half, so the number of filters is doubled in the next block. Then, the decoder is in charge of obtaining the segmentation map from the features extracted by the encoder. It performs up-sampling operations to the feature maps with 2×2 transposed convolutions increasing the size of the feature maps by double. Then, the number of feature channels is reduced to half after each up-sampling operation. The decoder uses 3×3 convolutions as the encoder.

Next, the final segmentation map is generated by a 1×1 convolution. This final convolution uses the sigmoid activation function in contrast to the others which use ReLU activation. Last but not least, this architecture uses skip connections which directly connect the encoder with the decoder in order not to lose the spatial features in the pooling operations. It is concatenated with the output of the transposed convolutions and is propagated to the next layers. This concatenation is accomplished to by truncating the encoder feature maps by centering them as they are slightly greater than the encoder ones.

2.2.4 DC-UNet

Some challenging datasets require segmenting objects into regions which can vary a lot in size for the same class. So, approaches explained before poorly segment them because they have to deal with a trade-off between reducing kernel sizes and learning higher resolution

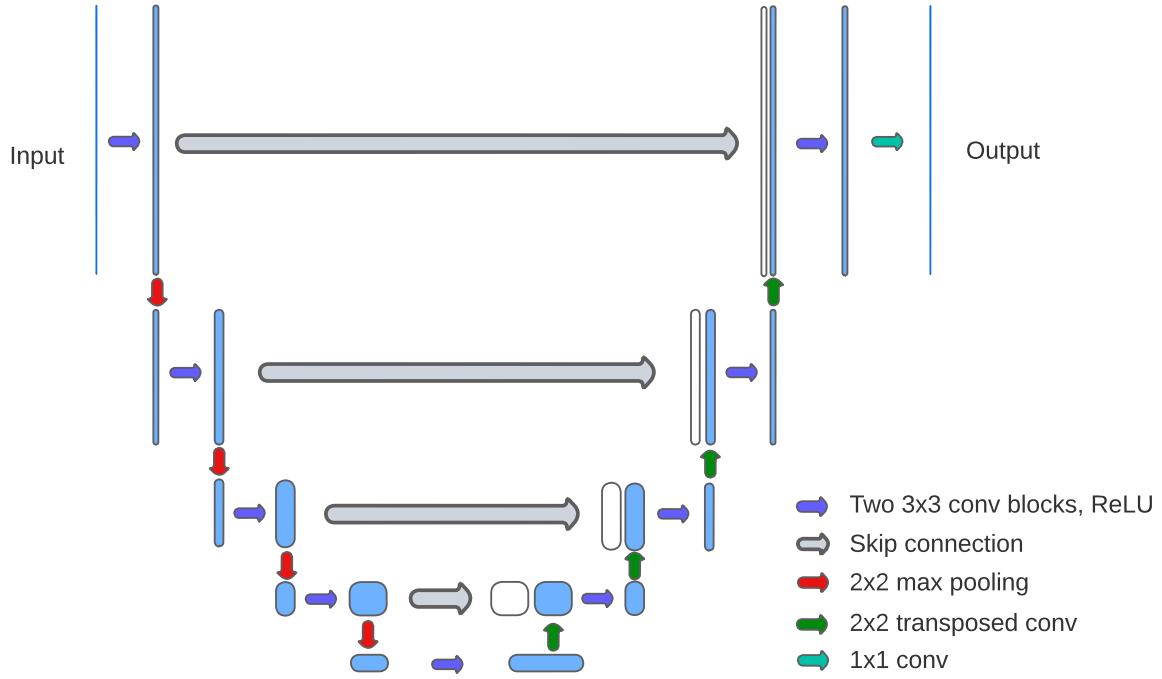


Figure 2.3: U-Net architecture representation, based on fully convolutional networks.

features. This problem can be solved by the use of convolutions with the use of different kernel sizes in parallel so that feature maps can learn features at different scales. Then, the model for this section, called dual channel U-Net (DC-UNet) [1], is inspired by U-Net and MultiResUNet [26] architectures, introducing the idea of dual channel blocks. A dual channel block is an effective way to obtain a considerable number of different scaled spatial features, and its representation can be seen in Figure 2.4. This block creates multiple feature maps whose receptive fields have sizes of 3×3, 5×5, and 7×7, where the two last are built from consecutive 3×3 convolutions. An addition operation between the resulting matrices from each channel is then carried out to provide more spatial features. Moreover, this model uses batch normalization layers after each DC-Block, which provides faster and better convergence [27].

Another important detail in this architecture is the replacement of skip connections in U-Net [1] with Res-Paths proposed in [26]. It consists of successive 3×3 convolutions which are added to 1×1 convolutions called residual connections [28]. It concatenates encoder features with the decoder as skip connections, but its additional operations reduce the semantic gap between them. At each level of depth, the number of 3×3 convolutions with residual connections is reduced by one, starting from 4 to 1. An illustration of a Res-Path

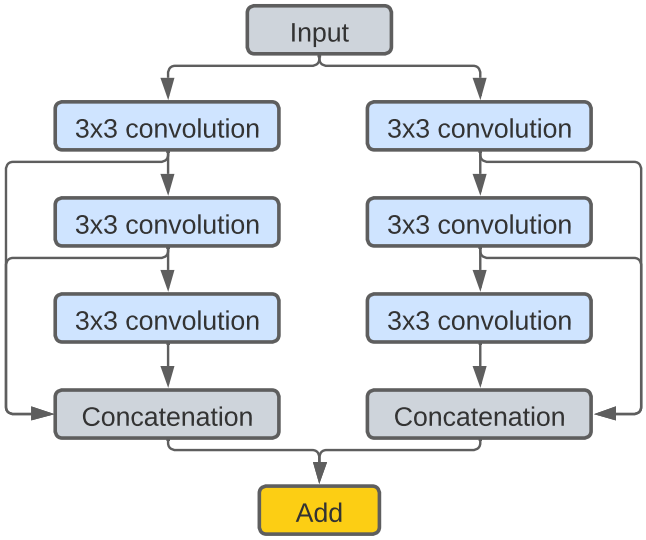


Figure 2.4: Dual channel block graphical representation, which captures multiple scale features.

in the first level is shown in Figure 2.5. It is important to mention that the number of filters used in DC-UNet for each DC-Block is reduced by half in comparison with MultiResUNet [26]. This drastically reduces the number of trainable parameters, making the model less time-consuming during training. In Res-Paths, the number of filters for each 3×3 and 1×1 convolution are {32, 64, 128, 256} for Res-Path {1, 2, 3 4} respectively. At last, DC-UNet architecture is shown in Figure 2.6.

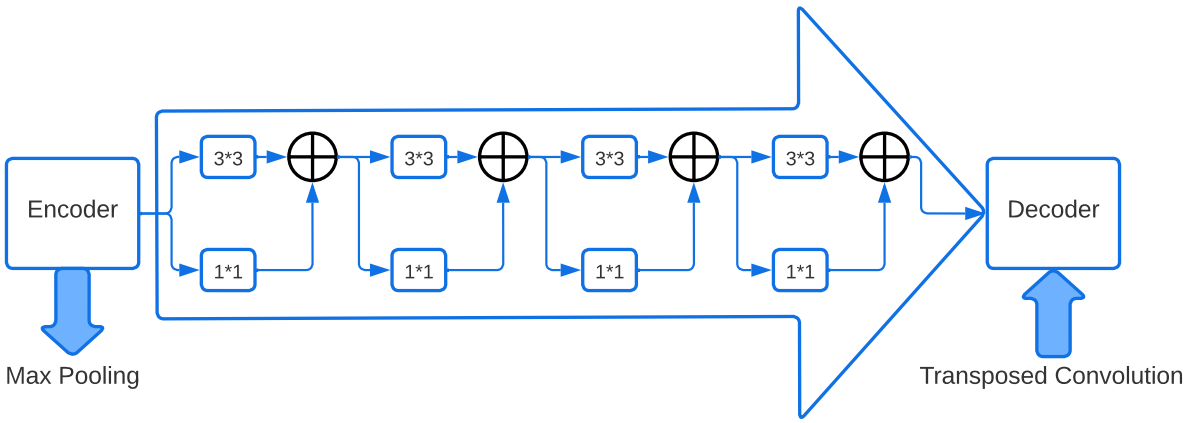


Figure 2.5: Res-Path connections with 4 successive 3×3 convolutions and its conrresponding residual connections.

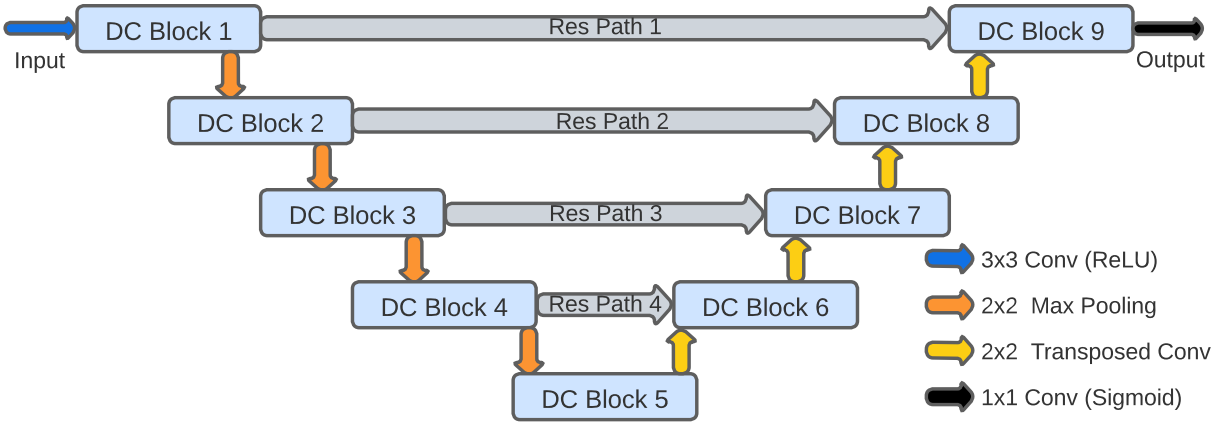


Figure 2.6: DC-UNet architecture based on U-Net to capture multiple scale features for medical image segmentation.

2.3 Genetic algorithms

Genetic algorithms are heuristic search algorithms inspired by biological evolution, and their purpose is to optimize a fitness function which usually is multivariate. Each solution candidate is called an individual, and the process of evolution makes a population adapt better to the environment to have more chances to pass their features to the next generation. The solution representation in an individual is called chromosome [29], which contains the values for each variable usually as an array. In this section, we introduce some important genetic operators which are sampling, selection, crossover, and mutation that are explained next.

2.3.1 Sampling

Sampling is the first step in genetic algorithms, it is the initialization of the population. Is important to make a good initialization as it will define the variability of the population. A bad initialization could lead to increased time towards a solution to prevent convergence to the global optimum solution [30].

2.3.2 Selection

This operation simulates the process of natural selection, or in other words the survival of the fittest. Almost all selection techniques are based on the fitness function, where the fittest solutions have more chances to survive and reproduce. This is an important step

for evolution where the new generation is likely to be better than previous ones. Some selection algorithms are based on random probabilities, giving the fittest individuals more chance of being selected, and one classic algorithm with this approach is the Roulette wheel selection explained in [31].

2.3.3 Crossover

It is an operator that is in charge of combining the chromosomes from two or more parent solutions. This operation mimics the mating between parents to generate offspring solutions that inherit similar features from both. The purpose is to try to outperform their parents which would have successful parts of chromosomes. For bit string representation, one of the most known crossover algorithms is the n-point crossover, which splits parents in n random positions and mixes them alternatively by their splitting points [29]. We can see an example of one-point crossover in Figure 2.7.

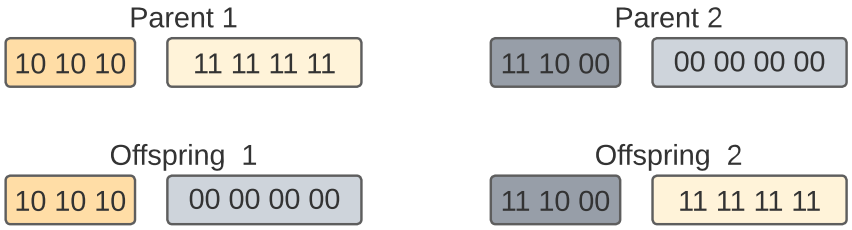


Figure 2.7: One-point crossover for bit string representation, where two parental solutions generate two offsprings by mixing their chromosomes.

2.3.4 Mutation

Mutation operators alter a solution by disrupting it, and random alterations in the chromosome are what cause mutation. In this operation, we have a parameter that indicates the probability of individuals mutating, and it is called mutation rate [31]. The value of this parameter is that generates a disturbance among the population. Additionally, every point in the solution space must be accessible from an arbitrary point because there must be at least a minimum chance to access to every part of the solution space. If not, there is a low likelihood that the best solution will be discovered.

2.3.5 Fitness

The fitness function evaluates the phenotype of the solution, measuring the quality of individuals produced by the reproduction of the previous generation. The choice of the fitness function is an important part of the process, and it can guide the search for the genetic algorithm. Thus, this function is the objective to be optimized by the genetic operators. Many times, the bottleneck is in evaluating the fitness function, so the genetic algorithm must be designed in such a way as to minimize the number of function calls [29]. That is our case in which the fitness function envelops a deep learning model training.

Chapter 3

State of the Art

In the field of medical image segmentation, there are some public challenges and datasets that envelop the segmentation of brain tumors, ischemic stroke lesions, and multiple sclerosis, among others. In Table 3.1, we summarize some recent medical image segmentation related works with metrics and datasets used. Most of them consist of CT and MRI, commonly used in brain analysis. Brain imaging studies some metrics, where some well known ones are true positive rate, dice similarity coefficient, Hausdorff distance, and average symmetric surface distance [32]. Additionally, other works also include intersection over union as well to evaluate performance [33]. In some research regarding lung segmentation, one of the best known datasets is the lung image database consortium. Specifically, in recent research using this dataset [34], authors consider precision, sensitivity, and mean intersection over union as evaluation metrics. Finally, concerning this work in polyp segmentation, some well known datasets are the Kvasir-SEG (1000 images), ETIS (196 images), CVC-ColonDB (380 images) and Endotec (1000 images), which were used in [35].

Apart from the deep learning optimization methods used in this work, there are several optimizers to adjust the weights in these models, and the most influential ones are shown in Table 3.2. After the backpropagation algorithm became well known, several gradient-based algorithms were developed to optimize those models. The regular SGD was the inspiration for most gradient-based algorithms today, which update weights in the opposite direction of the gradient. One of them is SDG with momentum, which was introduced in [36]. This optimizer adds the notion of updating the weights based on past weight changes to speed up the SGD optimization. Then, Adagrad establishes the idea of using the historical

squared values of the gradient, which is helpful to converge after falling in a convex area [16]. As those approaches, some breakthroughs have developed state of the art optimization methods such as Adadelta, Adam, Adamax, and others.

In the history of hyperparameter tuning methods, the first ones are grid search and random search. Those methods are still very popular because of their ease of implementation. Nowadays some medical applications use these methods to optimize deep learning models such as emboli detection [37], heart disease prediction [38], medical image segmentation [39], and others. Nevertheless, we can consider them as brute force approaches as they do not consider the performance of previous evaluations to predict better configurations, so these methods need to evaluate the model several times to find a good hyperparameter configuration.

Other methods outperform grid search and random search like Bayesian optimization. This is one of the state of the art methods which constructs a probabilistic model for the objective function, considering previous evaluations to compute the next points to try [40]. This approach helps to reduce the number of evaluations, and it can even optimize non-convex functions. Genetic algorithms based optimization is another approach that is on the state of the art methods which perform well. Additionally, both approaches can be parallelized to run several function evaluations asynchronously. This allows the efficient use of the hardware to reduce the optimization time, as the bottleneck is the function evaluation most of the time.

According to research works with genetic algorithms to optimize deep learning models in medical imaging problems, there exist some that are focused on tuning hyperparameters that change the network structure as proposed by [41]. They use a single objective genetic algorithm whose genes represent components of a CNN such as the number of layers, the number of neurons for each layer, activation functions, optimizers, and loss functions. The optimized model is used for medical image denoising, so their fitness function is based on the restored image quality measure for each individual. However, other works use single objective genetic algorithms to tune only hyperparameters of the model instead of changing the network structure itself. It is the case of the work developed by [42] which compares Bayesian optimization algorithms with the covariance matrix adaptation evolution strategy (CMA-ES). This genetic algorithm is friendly to parallel the evaluation of the solutions

to speed up the optimization, as the number of evaluations reaches up to 2000. In this case, the hyperparameters they use are batch size, alpha & epsilon for batch normalization, dropout rates in different layers, and optimizer's learning rates, among others.

Additionally, multi-objective algorithms have been used to tune hyperparameters in machine learning models because in some cases is necessary to consider optimizing a model through more than one metric at a time. That is the case of [43], they used a multi-objective genetic algorithm to optimize the parameters of a support vector machine classifier to deal with imbalanced data. As accuracy is not a suitable measure for imbalanced data, they also use the G-mean and average cost metrics with a 10-fold cross-validation. Thus, they have 3 fitness functions to be used in the NSGA-II algorithm. The NSGA-II algorithm that is based on crowding distance sorting has been used in hyperparameter tunning in classifiers for disease diagnosis by [44]. In this work, the models are based on decision trees, and the fitness functions used are sensitivity and specificity. They define 20 generations as stopping criteria and select the best non-dominated solutions as the set of hyperparameter configurations.

According to the literature review, medical image segmentation datasets are characterized by having very few images with unbalanced data. In addition, there are a lot of evaluation metrics that can be considered when performing this task. Besides that, the choice of optimizers has an important role for deep learning where it is important to consider them according to the current state of the art. Finally, related works have demonstrated the potential use of different kinds of genetic algorithms for medical purposes, so we would study its performance on a specific case in medical image segmentation with one of the most challenging datasets.

Authors	Title	Datasets	Evaluation metrics
Karimi, D. & Salcudean, E. (2019)	Reducing the Hausdorff distance in medical image segmentation with convolutional neural networks [39]	2D prostate ultrasound, 3D prostate MRI, 3D liver CT, and 3D pancreas CT	Dice similarity coefficient, Hausdorff distance, and symmetric surface distance
Li, G. et al. (2023)	IB-TransUNet: Combining Information Bottleneck and Transformer for Medical Image Segmentation [33]	Synapse multi-organ segmentation, and breast ultrasound image	Hausdorff distance, dice similarity coefficient, and intersection over union
Zhi, L. et al (2023)	Deep neural network pulmonary nodule segmentation methods for CT images: Literature review and experimental comparisons [34]	Lung image database consortium, image database resource initiative, and lung nodule analysis 16	Dice similarity coefficient, precision, sensitivity, and mean intersection over union
Nachmani, R. et al (2023)	Segmentation of polyps based on pyramid vision transformers and residual block for real-time endoscopy imaging [35]	Kvasir-SEG, ETIS, CVC-ColonDB, CVC-ClinicDB, Endotec	Intersection over union, dice similarity coefficient, precision, recall, and F2

Table 3.1: Medical image segmentation related works

Optimizer	Authors	Title	Year
SGD	Robins, H., & Monro, S.	A stochastic approximation method [45]	1951
SGD whit momentum	Rumelhart, D. et al	Learning internal representations by error propagation [36]	1985
Adagrad	Duchi, J., et al.	Adaptive subgradient methods for online learning and stochastic optimization [46]	2011
RMSProp	Tieleman, T., & Hinton, G.	Rmsprop: Divide the gradient by a running average of its recent magnitude. coursera: Neural networks for machine learning [47]	2012
Adadelta	Zeiler, M.	Adadelta: an adaptive learning rate method [48]	2012
Adam & Adamax	Kingma, D., and Ba, J.	Adam: A method for stochastic optimization [17]	2014
Nadam	Dozat, T.	Incorporating nesterov momentum into adam [18]	2016
AMSGrad	Reddi, S. et al.	On the convergence of adam and beyond [19]	2019

Table 3.2: Optimizers' related works

Chapter 4

Methodology

4.1 Description of the Problem

4.1.1 Dropout Layers

In this approach, we slightly modify the DC-UNet [1] architecture by simply adding dropout layers in the model, as the unique regularizer used in their work is batch normalization. Dropout [49] is a technique used in neural networks that randomly deactivate units with a probability of p. This deactivation is temporary, so new units are sampled at each iteration during training. Dropout helps the model to generalize some features avoiding the model from doing too well on training data. We use these layers just in between DC-Blocks and Res-Paths to generalize encoding features.

4.1.2 Loss Function

This problem can be seen as a binary classification for each pixel in the medical images as we predict if they belong to a polyp or not. So, the loss function used is the binary cross-entropy loss, and its mathematical representation is:

$$CrossEntropy(y, \hat{y}) = \sum_{x \in X} -(y \times log(\hat{y}) + (1 - y) \times log(1 - \hat{y})) \tag{4.1}$$

where $x \in X$ are pixels in the input image, $\hat{y}$ is the model prediction and y is the ground truth [50]. This function is optimized with respect to the model weights w as seen in section 2.1 during training, and it is represented as $L(x_i, w)$ for image x_i on those optimizing

algorithms.

4.1.3 Performance Metric

After the optimization for the loss function is done, we need to evaluate the model predicted solutions. Nevertheless, all pixel values in the output are related to the probabilities of belonging to a specific object. Consequently, it can be seen as a grayscale image instead of a binary one. According to [1], binarizing the predicted images to compute the performance binary vs. binary would result in a loss of information. So, the performance metric used to evaluate predicted results in the validation set is Tanimoto similarity [51], that is for grayscale images. It is considered an extension of Jacard similarity because Tanimoto similarity obtains the same results when pixel values are in $\{0, 1\}$. Considering images A and B to be two sets, Tanimoto similarity $T(A, B)$ can be described as:

$$|A \cap B| = \sum a_i b_i \tag{4.2}$$

$$|A \cup B| = |A| + |B| - |A \cap B| = \sum (a_i^2 + b_i^2 - a_i b_1) \tag{4.3}$$

$$T(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{4.4}$$

where $a_i \in A$ and $b_i \in B$.

4.1.4 Dataset Description

The dataset used in this work is the CVC-ClinicDB that was introduced in [52]. This dataset contains several images corresponding to frames from colonoscopy videos, and it is one of the most challenging datasets as the objective may have different sizes. The segmenting objective in the dataset is polyps, they are lesions that can vary in size from a few millimeters to several centimeters [53]. Additionally, it is fully annotated where the ground truth images consist of a segmentation of the polyp and the background. In Figure 4.1, there are some examples of how the dataset looks like, where the first and second rows stand for original images and annotated images respectively.

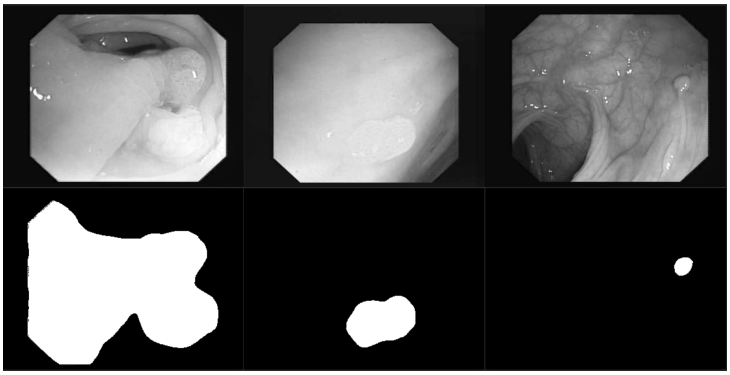


Figure 4.1: CVC-ClinicDB dataset examples, with original images (first row) and their corresponding ground truths (second row).

4.1.5 Data Preprocessing

Here, we use the same rescaling procedure as in [1] where images are resized from 384×288 to 128×96 pixels. This process considerably reduces memory usage and computational cost during the training of the model.

4.2 Model Proposal

4.2.1 Sampling

To sample the first generation, we use discrete and continuous uniform distribution as we have integer and real variables to optimize. For discrete variables, all possible values have the same probability of occurrence. Then, the probability mass function for them is illustrated in equation 4.5 [54]. Besides, the continuous distribution is used to model the occurrence of events with constant probabilities over intervals of the same size. Further, the probability density function for real variables is shown in equation 4.6 [54], where a and b are the low and up variable boundaries respectively.

$$P(X = x_i) = \frac{1}{n}, i = 1, 2, .., n \tag{4.5}$$

$$f(x) = \begin{cases} \dfrac{1}{b-a} & \text{, if } a \leq x \leq b \\ 0 & \text{, otherwise} \end{cases} \tag{4.6}$$

4.2.2 Selection

This operator is quite straightforwardly implemented, as we use a fitness-based truncated selection approach which ensures the survival of the most adapted solutions. In this approach, parent and offspring solutions are sorted by their fitness value, then the truncation operation selects the n-th fittest individuals for the next generation, and each generation will have the same population size [55]. After that, the algorithm performs a permutation among all individuals in the current population so that the arrangement is not the same for the next generation, and then we form pairs with all of them. Therefore, all individuals in that generation take part in the mating process once. For example, if we have the permutation of the n-th fittest individuals $[a, b, c, d, e, f]$, the parent selection will be the pairs $[(a, b), (c, d), (e, f)]$.

4.2.3 Crossover

For mating, the simulated binary crossover [56] (SBX) operator is used, which is a technique used for real parameter values. However, in our implementation, we use this operator also for integer values which are rounded after the crossover. As we know, each parent solution is a vector, so the SBX operator is applied variable by variable. This operator uses a parameter η_c which defines the spread of offspring solutions. Over a probability distribution around parent solutions, large values for η_c tend offsprings to be close to their parents while small values tend to create them away. Considering a problem with the i-th variable to be real $x_i \in [0, 1]$, and the parent solutions to be $x_i^{(1,t)} = 0.2$ and $x_i^{(2,t)} = 0.8$ for that variable. Then, probability density functions follow a polynomial distribution, and they can be seen in Figures 4.2 and 4.3 for offspring solutions with $\eta_c = 1$ and $\eta_c = 10$ respectively.

The procedure to compute two offspring solutions from parents is described next. Suppose we have $x_i^{(1,t)} \leq x_i^{(2,t)}$, then the spread factor $\beta_i^{(1,t)}$ and $\beta_i^{(2,t)}$ are calculated as in equations 4.7 and 4.8.

$$\beta_i^{(1,t)} = 1 + \frac{2(x_i^{(1,t)} - x_i^l)}{x_i^{(2,t)} - x_i^{(1,t)}} \tag{4.7}$$

$$\beta_i^{(2,t)} = 1 + \frac{2(x_i^u - x_i^{(2,t)})}{x_i^{(2,t)} - x_i^{(1,t)}} \tag{4.8}$$

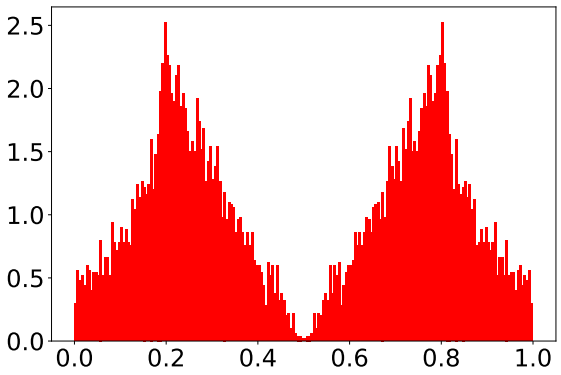


Figure 4.2: Probability density function for $x_i^{(1,t+1)}$ and $x_i^{(2,t+1)}$ with $\eta_c = 1$.

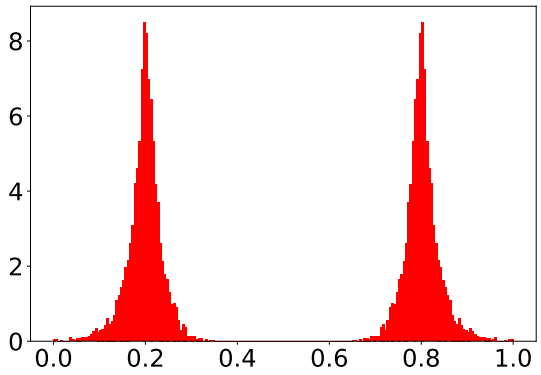


Figure 4.3: Probability density function for $x_i^{(1,t+1)}$ and $x_i^{(2,t+1)}$ with $\eta_c = 10$.

where x_i^l and x_i^u are the lower and upper bounds for that variable. After that, we sample u_i for the uniform distribution explained before between 0 and 1, and we compute the values $\beta_{qi}^{(1,t)}$ and $\beta_{qi}^{(2,t)}$ as:

$$\beta_{qi}^{(k,t)} = \begin{cases} (u_i\alpha)^{\frac{1}{\eta_c+1}} & \text{, if } u_i \leq 1/\alpha \\ \dfrac{1}{(2-u_i\alpha)^{\frac{1}{\eta_c+1}}} & \text{, otherwise} \end{cases} \tag{4.9}$$

where $\alpha = 2 - \left(\beta_i^{(k,t)}\right)^{-(\eta_c+1)}$ and $k \in \{1, 2\}$. In our implementation, we choose $\eta_c = 15$ fixed to generate offspfing variable solutions close to their parents, as they are meant to have successful chromosomes. Finally, the offspring solutions $x_i^{(1,t+1)}$ and $x_i^{(2,t+1)}$ are computed (see equation 4.10).

$$x_i^{(k,t+1)} = 0.5 * \left(x_i^{(1,t)} + x_i^{(2,t)} + \beta_i^{(k,t)} * \left(x_i^{(1,t)} + x_i^{(2,t)}\right)\right) \tag{4.10}$$

Soon, we sample b_i from a uniform distribution, then if $b_i > 0.5$ we swap values from $x_i^{(1,t+1)}$ and $x_i^{(2,t+1)}$. So, we have a 50% probability of being from the first or second parent. Finally, there is a repair operation in case $x_i^{(k,t+1)}$ were out of boundaries.

$$x_i^{(k,t+1)} = x_i^l \qquad \text{, if } x_i^{(k,t+1)} < x_i^l \tag{4.11}$$

$$x_i^{(k,t+1)} = x_i^u \qquad \text{, if } x_i^{(k,t+1)} > x_i^u \tag{4.12}$$

4.2.4 Mutation

In this case, we use a polynomial mutation analyzed in [57], with an index parameter $\eta = 20$ to perturb a solution in a parent's vicinity. This operator behaves very similarly to crossover as both of them have a polynomial distribution function. The polynomial mutation is for real variables, however, for integer values we applied the mutation with a rounding repair. As in [57], we set the mutation parameters as follows. Operator probability is set $p_c = 0.9$ so that 90% of the current population is likely to mutate. Mutation probability for each variable is $p_m = 1/n$, where n is the number of variables of the genetic algorithm. Thus, on average, one variable gets mutated per mutating individual. With our approach, the probability density function for a mutating variable with $\eta_c = 20$ is shown in Figure 4.4.

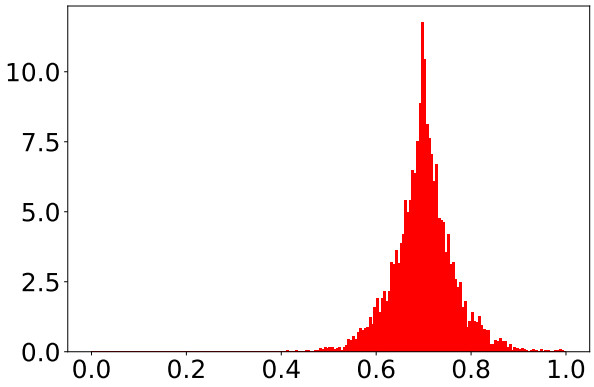


Figure 4.4: Polynomial mutation with $\eta_c = 20$.

Next, the computation of the polynomial mutation we use is explained. First, we sample a value b_p from a uniform distribution between 0 and 1 per individual such that if $b_p < p_c$, the correspondent individual is considered for mutation. In the same way, we sample a value b_m for each variable in mutating individuals so that if $b_m < p_m$, the correspondent variable in that individual mutates. Now, lets consider δ_{1i} and δ_{2i} to be as follows:

$$\delta_{1i} = \frac{x_i - x_i^l}{x_i^u - x_i^l}, \qquad \delta_{2i} = \frac{x_i^u - x_i}{x_i^u - x_i^l} \tag{4.13}$$

Then, a variable u_i is also sampled in uniform distribution between 0 and 1 for i-th variable that is mutating. So, we compute the parameters δ_{q1i} and δ_{q2i} as in equations 4.14 and

4.15.

$$\delta_{q1i} = \left[2u_i + (1-2u_i)(1-\delta_{1i})^{\eta_c+1}\right]^{\frac{1}{\eta_c+1}} - 1 \tag{4.14}$$

$$\delta_{q2i} = 1 - \left[2(1-u_i) + 2(u_i-0.5)(1-\delta_{2i})^{\eta_c+1}\right]^{\frac{1}{\eta_c+1}} \tag{4.15}$$

So, the mutation rule for the i-th variable is in equation 4.16. Finally, there is a repair operation if x_i were outside boundaries just like in the crossover operation at equations 4.11 and 4.12.

$$P_m(x_i) = \begin{cases} x_i + \delta_{q1i}(x^u - x^l) & \text{, if } u_i \leq 0.5 \\ x_i + \delta_{q2i}(x^u - x^l) & \text{, otherwise} \end{cases} \tag{4.16}$$

4.2.5 Fitness function

The fitness function for the genetic algorithm will be based on the Tanimoto similarity introduced in section 4.1.3. This metric is applied to a test set that the model has not seen during training. Additionally, we use the k-fold cross-validation technique, where the dataset X is randomly shuffled and split into mutually exclusive subsets $X_1, X_2, ..., X_k$ of approximately the same size [58]. Then, the training is carried out k times such that at time t the training set is $X \backslash X_t$ and the test set on X_t. Next, the model cross-validation accuracy (Acc_{cv}) is estimated as the mean accuracy of all k-folds. This approach gives the model parameters more reliability as the resulting accuracy is the average evaluation of every single fold on the dataset. In our implementation, we set $k = 5$, and we can see how the 5-fold cross-validation looks like in Figure 4.5.

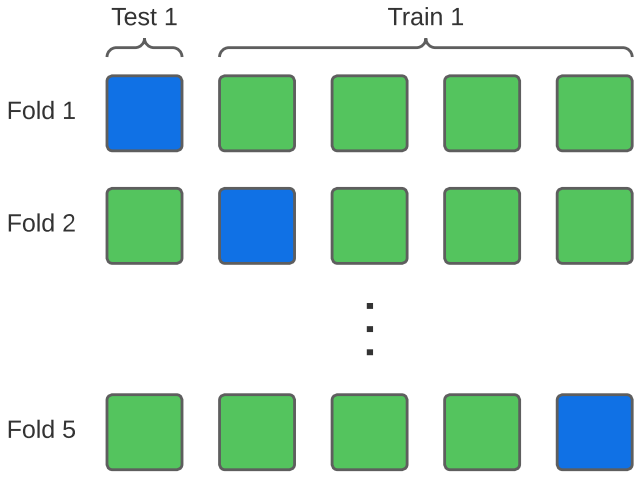


Figure 4.5: 5-Fold cross validation.

Finally, as the software used for genetic algorithms is specifically for minimization, we need to transform our fitness value Acc_{cv} into $1 - Acc_{cv}$ to maximize Acc_{cv}.

4.3 Experimental setup

4.3.1 Hardware and Software

As hyperparameter tuning requires high computational capabilities, the research was carried out in an HPC cluster provided by CEDIA where the hardware we used envelops AMD EPYC 7742 64-Core Processor (2.25 GHz), 128GB RAM, with 4 NVIDIA A100 SXM4 40GB GPUs. Additionally, the implementation was done in Python 3 and, we use Tensorflow [59] as the main deep learning framework. Therefore, the framework used for genetic algorithms is Pymoo introduced in [60]. This is a powerful tool that provides the use of several multi-objective and single-objective genetic algorithms and customizes them to the user's needs. Last but not least, frameworks used for parallelization are Dask [61] and Dask-cuda which are compatible with Tensorflow and Pymoo. With these frameworks together, the evaluation function can be distributed into a couple of workers, where each GPU is a worker itself. In this way, the evaluation function for each individual in the population can be performed 4 at a time, paralleling the training over available GPUs.

4.3.2 Hyperparameter Tuning

In the genetic configuration for hyperparameter tuning, we considered 8 experiments. We used population sizes of 25 and 50, and we ran 500 fitness evaluations for each configuration. Thus, there will be 20 and 10 generations as the termination rule for the population size of 25 and 50 respectively. With both configurations, we try to figure out which is the best in our case because there is a trade-off between population size and the number of generations in genetic algorithm optimization. On one hand, large population sizes give the algorithm more diversity of solutions, so the algorithm is more likely to avoid local minima. On the other hand, small numbers of generations sometimes are not enough for the algorithm to converge.

Each pair of configurations uses a different optimizer which are RMSProp, Adam, Nadam, and AMSGrad (see section 2.1) to figure out which of them performs better for

this specific model and dataset. The configuration for each experiment can be seen in Table 4.1. Additionally, the hyperparameters taken into account for optimization are learning rate, dropout, and batch size whose upper and lower bounds are shown in Table 4.2. Thus, we have a mixed variable problem with 1 integer and 2 real variables. Learning rate and dropout are real-valued variables while batch size is an integer value which in our implementation is transformed into 2^x, similar as in [42].

Id	Population size	Generations	Optimizer
1	25	20	RMSProp
2	50	10	RMSProp
3	25	20	Adam
4	50	10	Adam
5	25	20	Nadam
6	50	10	Nadam
7	25	20	AMSGrad
8	50	10	AMSGrad

Table 4.1: Experiment configurations.

Parameter	x^l	x^u
Learning rate	10^{-4}	10^{-2}
Dropout	0	1
Batch size (2^x)	1	3

Table 4.2: Parameters' lower and upper bounds.

The configuration for each optimizer algorithm was their default recommended by Tensorflow [59]. Additionally, boundaries for the learning rate hyperparameter were based to be around their recommended value, that is 10^{-3}. In the case of batch size, the boundaries were based on the configuration used in [1], which is equal to 2^2. It is important to mention that batch size values are too short for this model because of the memory usage that medical images require.

Chapter 5

Results and Discussion

In this section, we will show the performance of different genetic configurations, and we will provide the best hyperparameter configuration found for the DC-UNet model. Additionally, we will show which is the optimal optimizer for this specific task. Our results will demonstrate the power of genetic algorithms for the hyperparameter tuning of a deep learning model. Then, we present time and memory consumption for the genetic optimization, and we show some segmentations for the best configuration to see how well this model behaves. We analyzed our results dividing them into two groups of genetic configurations: a population size of 25 with 20 generations, and a population size of 50 with 10 generations.

5.1 Population size of 25 with 20 generations

In these configurations, the Adam optimizer had the best optimal solution. After 20 generations, the best Adam individual reached 80.80% accuracy while the average accuracy for the last generation was 80.29%. Nadam also had a really good performance, with maximum accuracy very close to Adam as its best result reached 80.73%. In the last generation's average accuracy, Nadam got 80.32% which is superior to Adam. Relatively far from being the best, AMSGrad got its best individual with 80.11% accuracy and an average accuracy at the last generation of 78.81%. Further, RMSProp had the worst performance with 79.59% and 79.00% maximum and average accuracies respectively.

In Figure 5.1, we can see the fitness values of the fittest individuals after a certain number of generations. The maximum values of optimizers at first generation are 77.47% 77.52%, 79.16%, and 79.32% for AMSGrad, RMSProp, Adam, and Nadam respectively. In

the case of AMSGrad, the improvement of the maximum value generation by generation is the largest in comparison with the others. This could be because populations do not lose diversity too quickly in this configuration. In the same way, Adam has better growth in comparison with Nadam as it starts with a smaller value and finishes with the fittest individual over this configuration.

AMSGrad and Adam seem to converge first because their maximum accuracy stops growing earlier than the others. Unfortunately, we can not assert they found their global optimum solution, in fact, all of them probably get stacked in a local minima. This is because this configuration has only 25 individuals in each generation which may not be enough to cover all solution space, and adding more generations for them could be unworthy. We can see the best hyperparameters found for each optimizer regarding this kind of configuration in Table 5.1, where the batch size of 2^2 was the optimum value for all of them.

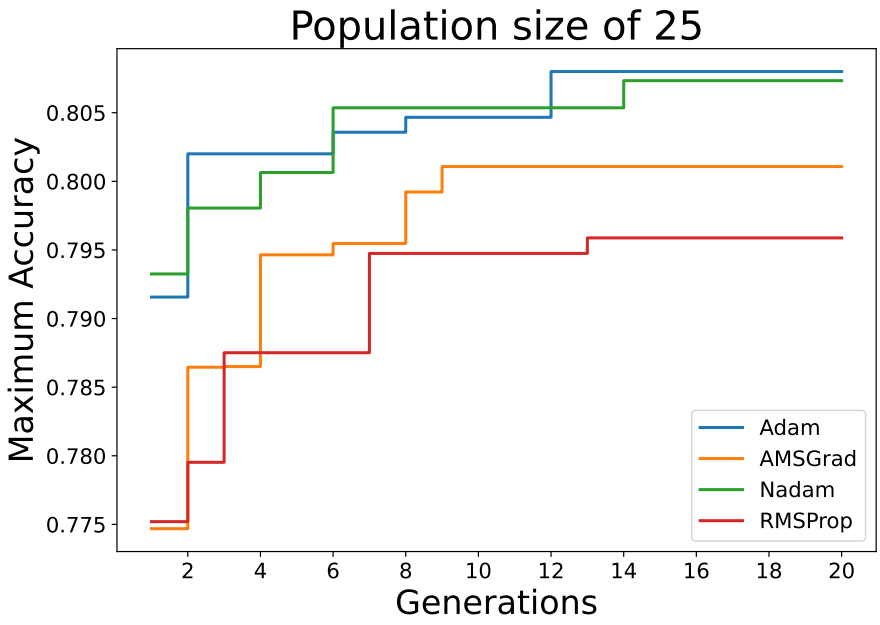


Figure 5.1: Accuracy values of fittest individuals per generation for each optimizer configuration with a population size of 25.

Additionally, we can see how populations evolve over generations by their average accuracy in Figure 5.2. Despite we saw Adam had the fittest individual over all optimizers, Nadam has on average better populations at every single generation which means the configuration for this optimizer may not finish to converge at all. Besides that, AMS-

Optimizer	Batch size (2^x)	Dropout	Learning rate
RMSProp	2	0.5748	7.06×10^{-4}
Adam	2	0.6050	4.51×10^{-4}
Nadam	2	0.7051	6.21×10^{-4}
AMSGrad	2	0.7755	6.97×10^{-4}

Table 5.1: Optimum hyperparameters for population size of 25.

Grad populations evolve better than RMSProp at each generation, finishing with better average accuracy even when its initial population has the worst mean accuracy among all optimizers.

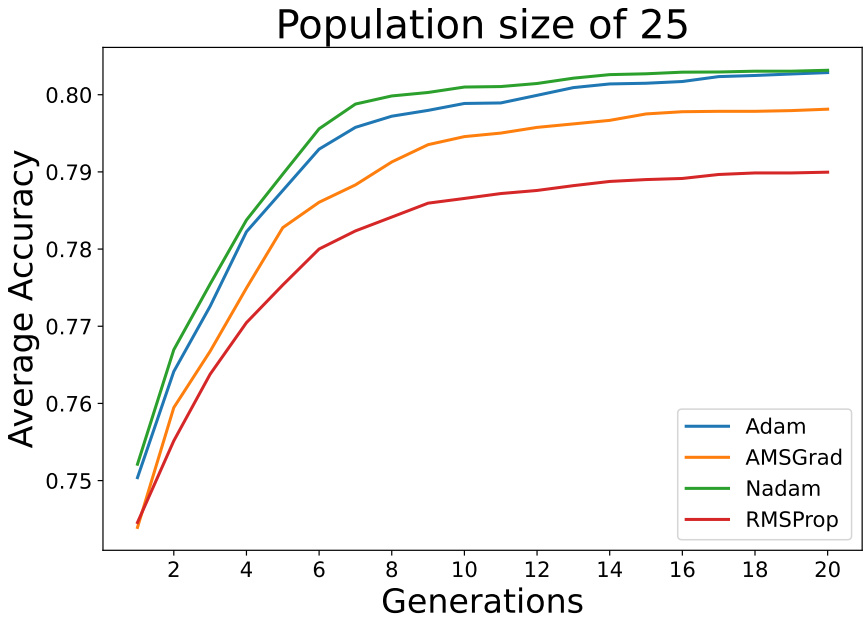


Figure 5.2: Populations' average accuracy values per generation for each optimizer configuration with a population size of 25.

For all configurations in this section, the initial population was the same, where the standard deviations for each variable were 0.8158, 0.2814, and 2.80×10^{-3} for batch size, dropout, and learning rate respectively. Next, can see the standard deviation of each variable for the last generation in Table 5.2. The standard deviation for each variable was reduced significantly in comparison with the first generation, which means the genetic algorithm succeeded in convergence, especially with the learning rate where the standard deviation was reduced more than 10 times. In batch size, those configurations discarded batch size of 2^1, which means this value does not perform well.

Optimizer	Batch size (2^x)	Dropout	Learning rate
RMSProp	0.3666	0.0717	1.81×10^{-4}
Adam	0	0.0846	2.17×10^{-4}
Nadam	0.3666	0.1078	1.63×10^{-4}
AMSGrad	0.4963	0.1083	1.83×10^{-4}

Table 5.2: Hyperparameters' standard deviation for population size of 25.

Finally, we can see the mean values for each population at the last generation in Table 5.3. All configurations converged to similar values in every single hyperparameter. In the case of learning rate, their values are relatively near to the default configuration in Tensorflow which is 10^{-3}. Furthermore, for all optimizers, batch size of 2^2 seems to have the best performance as individuals in the last generation tended to this value. The configuration for Adam optimizer completely converged to this value. In the case of dropout, all configurations converged around 0.65.

Optimizer	Batch size (2^x)	Dropout	Learning rate
RMSProp	2.16	0.6266	7.09×10^{-4}
Adam	2	0.6547	7.22×10^{-4}
Nadam	2.16	0.6416	7.20×10^{-4}
AMSGrad	2.44	0.6109	7.65×10^{-4}

Table 5.3: Hyperparameters' mean for population size of 25.

5.2 Population size of 50 with 10 generations

Regarding this type of configuration, Nadam reached the best accuracy result in the last generation. Nadam configuration got 80.89% and 80.01% for maximum and average accuracies respectively. Although Adam's optimizer beat Nadam in average accuracy with 80.07%, its maximum accuracy was 80.82%, with no significant difference between these numbers. They are followed by AMSGrad with 80.19% maximum accuracy and 79.60% average accuracy. RMSProp is last again in both maximum and average accuracy with 79.34% and 78.68% respectively.

The maximum values for each optimizer per generation are shown in Figure 5.3. Nadam and Adam do not improve so much in comparison with the initial population's maximum values which were 80.19% and 80.46% respectively. They may already fall near their

optimum values since the first generation as the population size in this case is bigger. AMSGrad and RMSProp had better improvement starting from 79.19% and 78.39%. These optimum values may not finish converging, as the number of generations is very low here, and adding some generations to this configuration would improve the results even more. Additionally, we can see optimum values for each optimizer after 10 generations in Table 5.4. Same as previous configurations, all optimizer's maximum values have batch size equals 2^2, which means this is a good default value and is preferred over 2^1 and 2^3. Also, there are no similarities in dropout values between them.

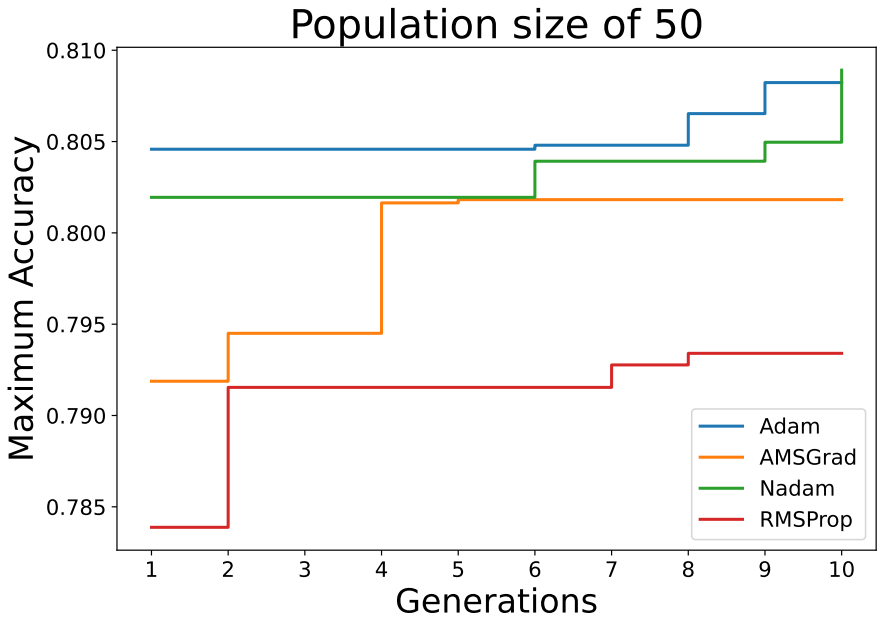


Figure 5.3: Accuracy values of fittest individuals per generation for each optimizer configuration with a population size of 50.

Optimizer	Batch size (2^x)	Dropout	Learning rate
RMSProp	2	0.2433	9.70×10^{-4}
Adam	2	0.5735	7.95×10^{-4}
Nadam	2	0.6377	6.85×10^{-4}
AMSGrad	2	0.3510	6.14×10^{-4}

Table 5.4: Optimum hyperparameters for population size of 50.

The average accuracy per generation is shown in Figure 5.4, where we can see similar behavior in the evolutions in Adam and Nadam populations. RMSProp population poorly improves their fitness values, being by far the worst optimizer in this kind of configuration.

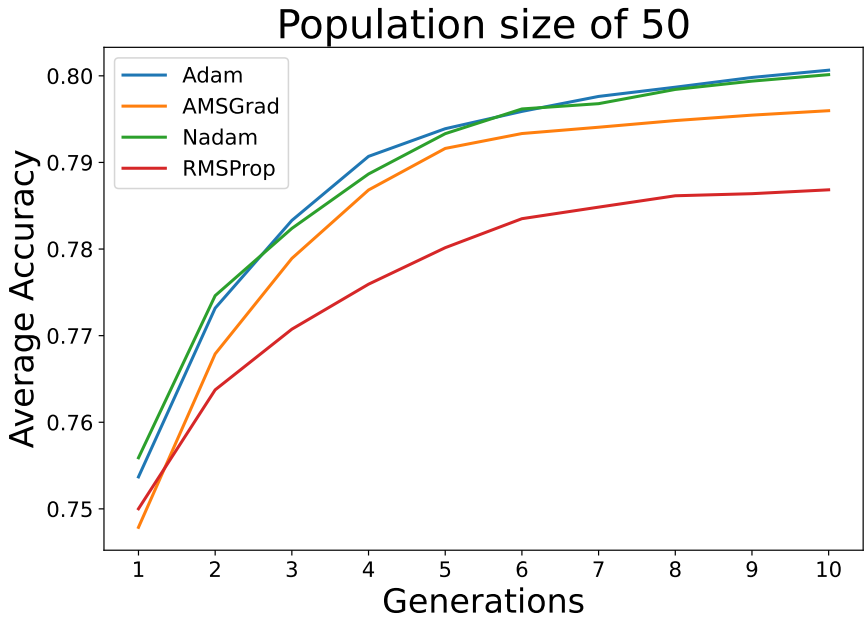


Figure 5.4: Populations' average accuracy values per generation for each optimizer configuration with a population size of 50.

The first 50 individuals were the same for all optimizers regarding this configuration. The initial standard deviations were: 0.7705 for batch size, 0.2752 for dropout, and 2.68×10^{-3} for learning rate. After 10 generations, the standard deviation is very similar for each parameter except RMSProp batch size and dropout. Despite not converging at all in dropout, RMSProp converged better than others at batch size. Standard deviations for the last generation are listed in Table 5.5 for every optimizer.

In addition, average values for each optimizer are shown in Table 5.6. Again, all populations in the last generation converged to similar values at all hyperparameters. None of them even had one configuration with 2^1 batch size. AMSGrad population tended to 2^3 batch size while the others to 2^2. Dropout values are around 0.5 while learning rates are around 7.5×10^{-4}

Optimizer	Batch size (2^x)	Dropout	Learning rate
RMSProp	0.2375	0.2072	1.83×10^{-4}
Adam	0.4854	0.1253	1.58×10^{-4}
Nadam	0.4964	0.1339	2.24×10^{-4}
AMSGrad	0.4271	0.1348	2.24×10^{-4}

Table 5.5: Hyperparameters' standard deviation for population size of 50.

Optimizer	Batch size (2^x)	Dropout	Learning rate
RMSProp	2.06	0.4708	7.21×10^{-4}
Adam	2.38	0.4960	7.16×10^{-4}
Nadam	2.44	0.5338	8.14×10^{-4}
AMSGrad	2.76	0.5363	8.44×10^{-4}

Table 5.6: Hyperparameters' mean for population size of 50.

5.3 Overall analysis

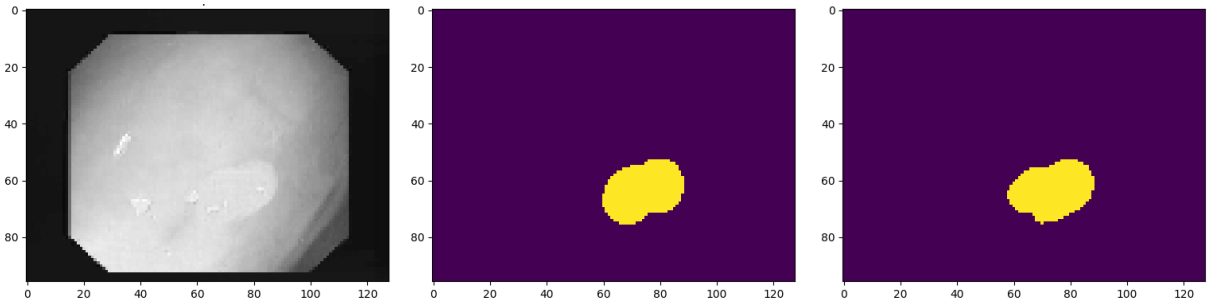
The computation time of fitness values per generation was high even with a parallelization with 4 GPUs. This is because every evaluation includes 5-fold cross-validation which implies training the model 5 times. On average, it took about 7.5 hours for a population size of 25, and 14 hours for a population size of 50. Initial populations were most time-consuming because they also included some individuals with a batch size of 2^1 that need more computations as they make the model upgrade parameters most frequently during training. The total RAM used for the genetic algorithms in this research was about 80 GB. This is because Tensorflow does not manage memory very well, and the memory used to store graphs for each model can not be released completely when collecting garbage memory. Moreover, medical images require a lot of memory, especially when batch size becomes larger.

The configurations with a population size of 50 found the best optimum values, with the only exception of RMSProp. This means more diversity is needed for this specific problem. In both kinds of configurations, Adam behaves similarly to Nadam, so Nesterov's momentum does not make a great difference in this case. Regarding the optimizers we use in this work, RMSProp is the unique optimizer that is not based on first order moments, and it had the worst performance. Therefore, combining first and second order moments may contribute a lot to finding the optimum solution for this specific task. Then, the property of the non-increasing step size of AMSGrad worsens the results as it is behind Nadam and Adam in accuracy. So, the variance between loss function and time is far from being large as it is the kind of scenario where AMSGrad outperforms Adam.

Dropout mean values at the last generation differ from both configuration types. This means the dropout layers for encoding features may not be enough to influence the model significantly. It would be better to add more dropout layers inside the model, for instance,

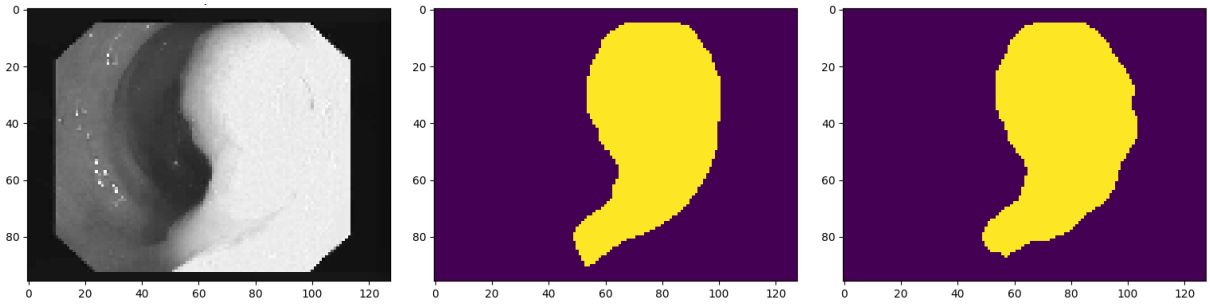
before each down-sampling and up-sampling operation.

The best hyperparameter configuration reached 80.89% accuracy which is slightly behind the value tested with the same model in [1] which was 80.94%. However, they used 150 epochs in their optimization while in this research only 100 epochs were used, obtaining almost the same performance. A smaller number of epochs is beneficial to reduce the training time for the model, but sometimes it reduces the accuracy as it is more likely not to converge. Finally, we can see some examples of the segmented medical images through the use of semantic segmentation in Figures 5.5, 5.6, and 5.7. Here, the parameters of the best model were used, with the model performing a great job recognizing even small regions.



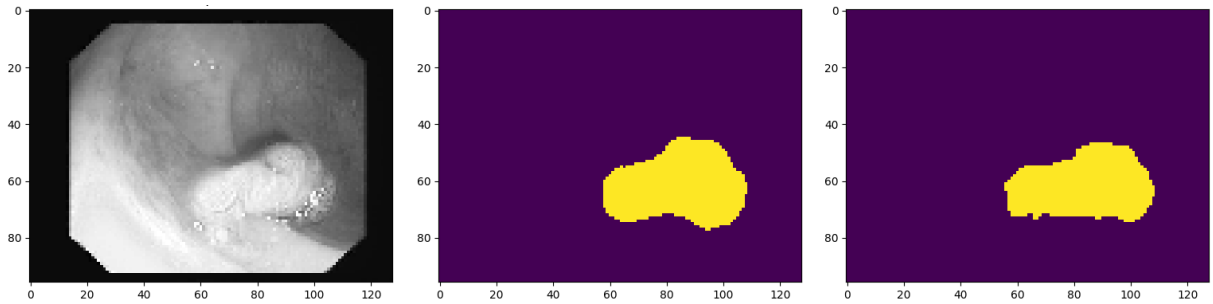
(a) Input image (b) Ground truth segmentation (c) Predicted segmentation

Figure 5.5: Segmentation with DC-UNet model on image 29: (a) Original image, (b) Ground truth image, (c) Predicted image.



(a) Input image (b) Ground truth segmentation (c) Predicted segmentation

Figure 5.6: Segmentation with DC-UNet model on image 385: (a) Original image, (b) Ground truth image, (c) Predicted image.



(a) Input image (b) Ground truth segmentation (c) Predicted segmentation

Figure 5.7: Segmentation with DC-UNet model on image 514: (a) Original image, (b) Ground truth image, (c) Predicted image.

Chapter 6

Conclusions

1. Nadam optimizer reached the most optimum value with 80.89% accuracy, batch size of 2^2, dropout of 0.6377, and learning rate equals 6.85×10^{-4}. In addition, Adam performed very similarly to Nadam in both configurations showing that Nesterov momentum slightly improved the model optimization.

2. We found that the combination of first and second order moments is a good option to optimize the DC-UNet model. That is the case of Adam, Nadam, and AMSGrad who use this approach. On the contrary, RMSProp was the worst optimizer regarding this task as it does not use first order moments for weight updates.

3. In most cases, a bigger number of individuals per generation performed better even with fewer generations because it gives populations greater diversity and scope in the whole search space. Thus, we recommend at least 50 individuals per generation for hyperparameter tuning of deep learning models.

4. Although batch size is one of the most influential hyperparameters in deep learning models, for this specific case we recommend setting its value at 2^2 unless we have enough memory to try with higher ones. As in [1], this batch size has shown to be a good default value most of the time.

5. The optimization of deep learning models in medical image segmentation usually is a time-consuming task that also uses a lot of memory. So, we suggest the use of sophisticated hardware for the hyperparameter tuning of this kind of model and parallelization techniques as well.

6.1 Future works

Further research can improve the implementation of dropout in the DC-UNet model as we found dropout layers before Res-Paths were not enough to make significant improvements on its performance at testing data, so this hyperparameter did not converge to similar values for different genetic configurations. Additionally, researchers can consider improving the selection operator by adding a kind of tournament selection (e.g. binary) to help the model to converge faster. Also, they could evaluate the performance of multi-objective genetic algorithms for the CVC-ClinicDB dataset by evaluating the model on different metrics at a time. Furthermore, it would be useful to add more hyperparameters for tuning such as β_1 and β_2 for the first and second order moments that may improve the results even more. Finally, pre-training the DC-UNet model with annotated medical images containing polyps would be beneficial in optimizing this model for the CVC-ClinicDB dataset, and it would improve its performance through the use of transfer learning and fine-tuning techniques.

Bibliography

[1] A. Lou, S. Guan, and M. Loew, "Dc-unet: rethinking the u-net architecture with dual channel efficient cnn for medical image segmentation," in *Medical Imaging 2021: Image Processing*, vol. 11596. SPIE, 2021, pp. 758–768.

[2] J. Ker, L. Wang, J. Rao, and T. Lim, "Deep learning applications in medical image analysis," *Ieee Access*, vol. 6, pp. 9375–9389, 2017.

[3] J. Gao, Q. Jiang, B. Zhou, and D. Chen, "Convolutional neural networks for computer-aided detection or diagnosis in medical image analysis: an overview," *Mathematical Biosciences and Engineering*, vol. 16, no. 6, pp. 6536–6561, 2019.

[4] M. H. Hesamian, W. Jia, X. He, and P. Kennedy, "Deep learning techniques for medical image segmentation: achievements and challenges," *Journal of digital imaging*, vol. 32, pp. 582–596, 2019.

[5] R. Golan, C. Jacob, and J. Denzinger, "Lung nodule detection in ct images using deep convolutional neural networks," in *2016 international joint conference on neural networks (IJCNN)*. IEEE, 2016, pp. 243–250.

[6] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.

[7] M. Kumar, D. Husain, N. Upreti, D. Gupta *et al.*, "Genetic algorithm: Review and application," *Available at SSRN 3529843*, 2010.

[8] S. Sivanandam, S. Deepa, S. Sivanandam, and S. Deepa, *Genetic algorithms*. Springer, 2008.

[9] B. Kayalibay, G. Jensen, and P. van der Smagt, "Cnn-based segmentation of medical imaging data," *arXiv preprint arXiv:1701.03056*, 2017.

[10] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, pp. 115–133, 1943.

[11] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[12] M. Minsky and S. Papert, "An introduction to computational geometry," *Cambridge tiass., HIT*, vol. 479, p. 480, 1969.

[13] A. Abraham, "Artificial neural networks," *Handbook of measuring system design*, 2005.

[14] P. M. Radiuk, "Impact of training set batch size on the performance of convolutional neural networks for diverse datasets," 2017.

[15] M. Li, T. Zhang, Y. Chen, and A. J. Smola, "Efficient mini-batch training for stochastic optimization," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 661–670.

[16] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[17] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[18] T. Dozat, "Incorporating nesterov momentum into adam," 2016.

[19] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of adam and beyond," *arXiv preprint arXiv:1904.09237*, 2019.

[20] J. Baskauf, G. Brookman, T. Eidmann, M. Gorra, H. Pearson, and B. Richter, "A comparison of image segmentation algorithms."

[21] J. Wu, "Introduction to convolutional neural networks," 2017.

[22] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[23] D. Scherer, A. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition," in *International conference on artificial neural networks*. Springer, 2010, pp. 92–101.

[24] D. Ciresan, A. Giusti, L. Gambardella, and J. Schmidhuber, "Deep neural networks segment neuronal membranes in electron microscopy images," *Advances in neural information processing systems*, vol. 25, 2012.

[25] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[26] N. Ibtehaz and M. S. Rahman, "Multiresunet: Rethinking the u-net architecture for multimodal biomedical image segmentation," *Neural networks*, vol. 121, pp. 74–87, 2020.

[27] S. C. Pereira, J. Rocha, A. Campilho, P. Sousa, and A. M. Mendonça, "Lightweight multi-scale classification of chest radiographs via size-specific batch normalization," *Computer Methods and Programs in Biomedicine*, vol. 236, p. 107558, 2023.

[28] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 31, no. 1, 2017.

[29] O. Kramer and O. Kramer, *Genetic algorithms*. Springer, 2017.

[30] L. Kallel and M. Schoenauer, "Alternative random initialization in genetic algorithms." in *ICGA*. Citeseer, 1997, pp. 268–275.

[31] X. Yu and M. Gen, *Introduction to evolutionary algorithms*. Springer Science & Business Media, 2010.

[32] Z. Akkus, A. Galimzianova, A. Hoogi, D. L. Rubin, and B. J. Erickson, "Deep learning for brain mri segmentation: state of the art and future directions," *Journal of digital imaging*, vol. 30, pp. 449–459, 2017.

[33] G. Li, D. Jin, Q. Yu, and M. Qi, "Ib-transunet: Combining information bottleneck and transformer for medical image segmentation," *Journal of King Saud University-Computer and Information Sciences*, vol. 35, no. 3, pp. 249–258, 2023.

[34] L. Zhi, W. Jiang, S. Zhang, and T. Zhou, "Deep neural network pulmonary nodule segmentation methods for ct images: Literature review and experimental comparisons," *Computers in Biology and Medicine*, p. 107321, 2023.

[35] R. Nachmani, I. Nidal, D. Robinson, M. Yassin, and D. Abookasis, "Segmentation of polyps based on pyramid vision transformers and residual block for real-time endoscopy imaging," *Journal of Pathology Informatics*, vol. 14, p. 100197, 2023.

[36] D. E. Rumelhart, G. E. Hinton, R. J. Williams *et al.*, "Learning internal representations by error propagation," 1985.

[37] B. E. Sakar, G. Serbes, and N. Aydin, "Emboli detection using a wrapper-based feature selection algorithm with multiple classifiers," *Biomedical Signal Processing and Control*, vol. 71, p. 103080, 2022.

[38] R. Valarmathi and T. Sheela, "Heart disease prediction using hyper parameter optimization (hpo) tuning," *Biomedical Signal Processing and Control*, vol. 70, p. 103033, 2021.

[39] D. Karimi and S. E. Salcudean, "Reducing the hausdorff distance in medical image segmentation with convolutional neural networks," *IEEE Transactions on medical imaging*, vol. 39, no. 2, pp. 499–513, 2019.

[40] V. Nguyen, "Bayesian optimization for accelerating hyper-parameter tuning," in *2019 IEEE second international conference on artificial intelligence and knowledge engineering (AIKE)*. IEEE, 2019, pp. 302–305.

[41] P. Liu, M. D. El Basha, Y. Li, Y. Xiao, P. C. Sanelli, and R. Fang, "Deep evolutionary networks with expedited genetic algorithms for medical image denoising," *Medical image analysis*, vol. 54, pp. 306–315, 2019.

[42] I. Loshchilov and F. Hutter, "Cma-es for hyperparameter optimization of deep neural networks," *arXiv preprint arXiv:1604.07269*, 2016.

[43] R. Guido, M. C. Groccia, and D. Conforti, "A hyper-parameter tuning approach for cost-sensitive support vector machine classifiers," *Soft Computing*, vol. 27, no. 18, pp. 12 863–12 881, 2023.

[44] S. Kumar and S. Ratnoo, "Multi-objective hyperparameter tuning of classifiers for disease diagnosis."

[45] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

[46] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization." *Journal of machine learning research*, vol. 12, no. 7, 2011.

[47] T. Tieleman and G. Hinton, "Rmsprop: Divide the gradient by a running average of its recent magnitude. coursera: Neural networks for machine learning," *COURSERA Neural Networks Mach. Learn*, vol. 17, 2012.

[48] M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[49] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[50] C. Guo, X. Chen, Y. Chen, and C. Yu, "Multi-stage attentive network for motion deblurring via binary cross-entropy loss," *Entropy*, vol. 24, no. 10, p. 1414, 2022.

[51] D. J. Rogers and T. T. Tanimoto, "A computer program for classifying plants: The computer is programmed to simulate the taxonomic process of comparing each case with every other case." *Science*, vol. 132, no. 3434, pp. 1115–1118, 1960.

[52] J. Bernal, F. J. Sánchez, G. Fernández-Esparrach, D. Gil, C. Rodríguez, and F. Vilariño, "Wm-dova maps for accurate polyp highlighting in colonoscopy: Validation vs. saliency maps from physicians," *Computerized medical imaging and graphics*, vol. 43, pp. 99–111, 2015.

[53] R. Jain and R. Chetty, "Gastric hyperplastic polyps: a review," *Digestive diseases and sciences*, vol. 54, pp. 1839–1846, 2009.

[54] L. P. Fávero and P. Belfiore, "Chapter 6 - random variables and probability distributions," in *Data Science for Business and Decision Making*, L. P. Fávero and P. Belfiore, Eds. Academic Press, 2019, pp. 137–165. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128112168000069

[55] R. Lange, T. Schaul, Y. Chen, C. Lu, T. Zahavy, V. Dalibard, and S. Flennerhag, "Discovering attention-based genetic algorithms via meta-black-box optimization," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2023, pp. 929–937.

[56] K. Deb, K. Sindhya, and T. Okabe, "Self-adaptive simulated binary crossover for real-parameter optimization," in *Proceedings of the 9th annual conference on genetic and evolutionary computation*, 2007, pp. 1187–1194.

[57] K. Deb and D. Deb, "Analysing mutation schemes for real-parameter genetic algorithms," *International Journal of Artificial Intelligence and Soft Computing*, vol. 4, no. 1, pp. 1–28, 2014.

[58] M. Stone, "Cross-validation: A review," *Statistics: A Journal of Theoretical and Applied Statistics*, vol. 9, no. 1, pp. 127–139, 1978.

[59] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving,

M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[60] J. Blank and K. Deb, "pymoo: Multi-objective optimization in python," *IEEE Access*, vol. 8, pp. 89 497–89 509, 2020.

[61] M. Rocklin, "Dask: Parallel computation with blocked algorithms and task scheduling," in *Proceedings of the 14th Python in Science Conference*, K. Huff and J. Bergstra, Eds., 2015, pp. 130 – 136.