Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Master in Artificial Intelligence

Master Thesis

Estimation of Distribution Algorithms for Generative Adversarial and Convolutional Neural Network Hyper-Parameter Optimization

Author: Janire Amesti Soler
Supervisors: Concha Bielza and Pedro Larrañaga

Madrid, July, 2024

Este Trabajo Fin de Máster se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Master Thesis
Master in Artificial Intelligence

Title: Estimation of Distribution Algorithms for Generative Adversarial and Convolutional Neural Network Hyper-Parameter Optimization

July, 2024

Author: Janire Amesti Soler
Supervisors: Concha Bielza and Pedro Larrañaga
Computational Intelligence Group
ETSI Informáticos
Universidad Politécnica de Madrid

Acknowledgments

I would like to thank my mentors Concha Bielza and Pedro Larrañaga for the opportunity to be part of this project and for all the advice, support and confidence they have placed on me these months. It has been a pleasure and a rewarding experience to collaborate with them on the research.

I would like to express my gratitude and affection to my family for the unconditional support and to my colleagues at the Computational Intelligence Group: Clara García, Daniel Zaragoza, Jaime Jiménez, Juan Fernández del Pozo, Mario Rubio, Mehsun Ihtiyan, Pedro Mejías and Victor Salvador, for promoting a collaborative environment. I also want to extend my thanks to Kevin Paniagua and Laura González, as well as to all the members of the CeSViMa building, for their dedication and hard work.

Additionally, I want to thank Universidad Politécnica de Madrid for providing computing resources on the Magerit Supercomputer, which was essential for the completion of this work.

This work has been partially supported by the Spanish Ministry of Science and Innovation through the PID2022-139977NB-I00, PLEC2023-010252 and TED2021-131310B-I00 projects, and by the Autonomous Community of Madrid within the ELLIS Unit Madrid framework.

Resumen

Esta tesis explora la optimización de hiperparámetros en modelos de Aprendizaje Profundo, centrándose en Redes Neuronales Convolucionales (CNNs en su acrónimo en inglés) y Redes Generativas Adversariales Convolucionales Profundas (DCGANs).

El estudio investiga el uso de Algoritmos de Estimación de Distribución (EDAs), específicamente el Algoritmo de Estimación de Redes Bayesianas (EBNA), para optimizar tanto la arquitectura de la red como el proceso de entrenamiento en CNNs (EDA-CNN) y en DCGANs (EDA-DCGAN). El objetivo es mejorar el rendimiento en tareas de visión por computador, especialmente en clasificación de imágenes y generación de imágenes.

Dado que el enfoque EDA-CNN implica un alto coste computacional, se implementa un modelo sustituto para aproximar la función de ajuste, reduciendo así el coste computacional del proceso de optimización. Además, se combinan los métodos de evaluación basados en el entrenamiento de CNNs y en el modelo sustituto con el objetivo de reducir el coste computacional y alcanzar resultados más precisos. Este enfoque, EDA-CNN-Surrogate, aplica el método de evaluación basado en el entrenamiento de CNNs en soluciones prometedoras, evitando el entrenamiento en soluciones cuyas aproximaciones basadas en el modelo sustituto son peores que un umbral determinado. Este umbral se considera fijo o dinámico, reduciendose a medida que avanzan las generaciones, siendo más selectivo en la determinación de soluciones prometedoras. Además, se presenta un EDA multiobjetivo que optimiza tanto la precisión como el coste computacional requerido para el entrenamiento.

Se comparan diferentes enfoques analizando la convergencia y el coste computacional durante el proceso de entrenamiento utilizando el conjunto de datos CIFAR-10. El enfoque EDA-CNN-Sustituto con umbral dinámico devuelve el mejor óptimo local encontrado, alcanzando una precisión del 94,60% en el conjunto de validación y reduciendo en un 38,82% el coste computacional en comparación con el enfoque EDA-CNN.

También se analiza el tamaño óptimo de la población, concluyendo que cuanto mayor es el tamaño de la población, mejor y más rápida es la convergencia. Sin embargo, para la tarea de clasificación de CIFAR-10, un tamaño de población de 30 individuos es suficiente para lograr resultados prometedores.

En cuanto al algoritmo de optimización EDA-DCGAN, se analiza la convergencia y los resultados del algoritmo EBNA para la optimización de hiperparámetros en arquitecturas simples de DCGAN para la generación de imágenes del dataset MNIST, conluyendo que el algoritmo es prometedor en la generación de imágenes de alta calidad en arquitecturas más complejas.

Abstract

This Master's Thesis explores the optimization of hyper-parameters in Deep Learning models, with a focus on Convolutional Neural Networks (CNNs) and Deep Convolutional Generative Adversarial Networks (DCGANs).

The study investigates the use of Estimation of Distribution Algorithms (EDAs), concretely Estimation of Bayesian Networks Algorithm (EBNA), to optimize both the network architecture and training process in CNNs (EDA-DCGAN) and DCGANs (EDA-DCGAN), aiming to enhance performance in computer vision tasks, specially in image classification and image generation tasks.

Since EDA-CNN approach requires high computational cost, a surrogate model is implemented in order to approximate the fitness function, thereby reducing the computational cost of the optimization process. CNN and surrogate based evaluation methods are combined in order to both reduce computational cost and reach more precise results. This approach, EDA-CNN-Surrogate, applies CNN training evaluation method in promising solutions, avoiding the training in solutions whose surrogate based approximations are poorer than a determined threshold. This threshold is also considered dynamic, reducing it as generations go by, being thus more selective in determining the promising solutions. Moreover, a multi-objective EDA is also presented optimizing both the accuracy and the computational cost the training require.

Different approaches are compared analysing the convergence and the computational cost during the training process using CIFAR-10 dataset. EDA-CNN-Surrogate approach with dynamic threshold achieves the best found local optima, reaching an accuracy of 94.60% in the validation set and reducing in 38.82% the computational cost in comparison with EDA-CNN approach.

Optimum population size is also analysed, concluding the higher the population size the better and faster is the convergence. Nevertheless, for CIFAR-10 classification task a population size of 30 individuals is enough to achieve promising results.

Regarding the EDA-DCGAN optimization algorithm, the project examines the convergence and results of the EBNA algorithm for optimizing hyper-parameters in basic DCGAN architectures aimed to generate images from the MNIST dataset. The algorithm results promising in producing high-quality images in more complex architectures.

Contents

1 Introduction **1**
- 1.1 Deep Learning Hyper-parameter Optimization . . . 1
- 1.2 Deep Learning in Computer Vision . . . 2
- 1.3 Evolutionary Computation for Hyper-parameter Optimization . . . 2
- 1.4 Objectives . . . 3
  - 1.4.1 Dependencies among hyper-parameters . . . 3
  - 1.4.2 Surrogate based evaluation . . . 4
  - 1.4.3 CNN structure optimization . . . 4
  - 1.4.4 Multi-objective optimization . . . 5
  - 1.4.5 Summary . . . 5
- 1.5 Structure . . . 5

2 State of The Art **7**
- 2.1 Optimization techniques . . . 7
  - 2.1.1 Grid Search . . . 7
  - 2.1.2 Random Search . . . 7
  - 2.1.3 Bayesian Optimization . . . 8
  - 2.1.4 Evolutionary Computation . . . 9
- 2.2 Hyper-parameter Optimization in Computer Vision . . . 9
  - 2.2.1 Convolutional Neural Networks . . . 9
  - 2.2.2 Generative models . . . 11
- 2.3 Estimation of Distribution Algorithms . . . 12
  - 2.3.1 Discrete variables . . . 12
  - 2.3.2 Continuous variables . . . 14
  - 2.3.3 Applications in Computer Vision . . . 15

3 Methodology **17**
- 3.1 Convolutional Neural Networks . . . 17
  - 3.1.1 Convolutional layers . . . 17
  - 3.1.2 Pooling layers . . . 19
  - 3.1.3 Fully connected layers . . . 19
  - 3.1.4 Residual blocks . . . 20
  - 3.1.5 CNN training . . . 20
- 3.2 EDA encoding scheme . . . 22
- 3.3 EDA-CNN . . . 24
  - 3.3.1 Training approaches . . . 25
  - 3.3.2 Surrogate model . . . 27
  - 3.3.3 Multi-objective EDA-CNN . . . 28

3.4 EDA-DCGAN 29
3.4.1 Deep Convolutional Generative Adversarial Networks 29
3.4.2 Hyper-parameter Optimization 29

4 Results **31**
4.1 EDA-CNN approaches 31
4.1.1 EDA-CNN evaluation 32
4.1.2 EDA-Surrogate model evaluation 33
4.1.3 EDA-CNN-Surrogate model evaluation 33
4.1.4 Multi-objective EDA-CNN 38
4.1.5 Comparisons 39
4.2 EDA-DCGAN 41

5 Conclusions and future lines **43**
5.1 Conclusions 43
5.2 Future lines 44

Bibliography **45**

Chapter 1

Introduction

Deep learning (DL) is a subset of machine learning that uses multi-layered neural networks known as Deep Neural Networks (DNN) to simulate the complex decision-making power of the human brain. These models are capable of handling large amount of complex data and are particularly effective at recognizing patterns and making predictions. This approach has led to breakthroughs in fields such as computer vision (Khan et al., 2018), natural language processing (Deng and Liu, 2018), and speech recognition (Abdel-Hamid et al., 2014).

1.1 Deep Learning Hyper-parameter Optimization

DNN hyper-parameter optimization (HPO) is crucial since its performance and effectiveness for specific tasks depends highly on an appropriate setting of hyper-parameters, which control both learning process and network architecture.

When it comes to the learning process, the optimal hyper-parameters can speed up the convergence of the training process, reducing time and computational resources required to train the model. A well-chosen learning rate, for instance, ensures an efficient learning without oscillating or diverging.

Hyper-parameter optimization also involves making architectural decisions such as the number of layers, the number of neurons per layer, and the type of activation functions, leading to more efficient and effective models. Additionally, it involves balancing trade-offs between different aspects of the model, such as complexity versus performance, speed versus accuracy, and bias versus variance. Common hyper-parameters in DL include the learning rate, which controls the step size during the weight update process; batch size, which determines the number of training examples used in one iteration to update the model weights; the number of layers and neurons, which mean the depth and width of the neural network; dropout rate, which is the fraction of neurons to drop during training to prevent over-fitting; regularization parameters, which penalize large weights to prevent over-fitting; and activation functions, such as ReLU, sigmoid, and tanh, which introduce non-linearity into the model.

1.2 Deep Learning in Computer Vision

The computer vision field focuses on developing algorithms and techniques to extract meaningful insights from images and videos enabling machines to interpret and understand visual information. Computer vision has numerous applications across various industries, including healthcare (Esteva et al., 2021; Gao et al., 2018; Khang et al., 2024), manufacturing (Zhou et al., 2022), automotive (Janai et al., 2020), surveillance (Thai et al., 2022) and construction (Xu et al., 2021).

Computer vision algorithms (Szeliski, 2022) process visual data to perform tasks such as image classification, object detection, facial recognition and image segmentation. By analyzing visual inputs, computer vision systems can make decisions, recognize patterns, and extract valuable information for further processing or decision-making. DL techniques (Hassaballah and Awad, 2020; Chai et al., 2021), particularly Convolutional Neural Networks (CNNs) (Indolia et al., 2018), have significantly improved the accuracy and efficiency of computer vision tasks.

Furthermore, computer vision systems become more effective, versatile, and capable of handling complex and real world tasks by taking advantage of data augmentation and transformation capabilities of generative AI. Deep Convolutional Generative Adversarial Networks (DCGAN) and Variational Autoencoders, for instance, generate synthetic data leading to more robust and accurate models where real-world labeled data is scarce, expensive, or difficult to obtain. In addition, DCGANs, for instance, not only are able to generate images, but they can also enhance image resolution, producing high-resolution images from low-resolution inputs, and fill missing parts of images.

Hyper-parameter optimization in models like CNNs and DCGANs is also relevant for identifying the optimal network topology and learning process.

1.3 Evolutionary Computation for Hyper-parameter Optimization

Hyper-parameters on various DL architectures have complex relationships. While certain hyper-parameters may significantly improve performance in simpler networks, their effects may differ in more complex architectures. Additionally, findings from one dataset may not directly apply to another dataset with different characteristics such as image properties, class distributions, or sample sizes. Selecting the right set of hyper-parameters lacks a definitive formula and often relies on a combination of prior experience and trial-and-error.

However, given the computationally intensive nature of DL algorithms, which can take days to train on conventional hardware, relying solely on trial-and-error for hyper-parameter tuning is inefficient and not exhaustive. Therefore, Evolutionary Computation (EC) is an alternative to efficiently explore the hyper-parameter space and discover configurations that lead to improved model performance across diverse datasets and architectures. The implementation of EC algorithms to optimize DL algorithms is called Evolutionary Deep Learning (Zhan et al., 2022).

EC algorithms mainly include Evolutionary Algorithms (EA) (Yu and Gen, 2010) and Swarm Intelligence (SI) (Kennedy, 2006) algorithms. Among EA, Genetic Algorithm

(GA) (Holland, 1992), Evolutionary Strategy (ES) (Beyer and Schwefel, 2002), Genetic Programming (GP) (Koza, 1990), Evolutionary Programming (EP) (Yao et al., 1999), Differential Evolution (DE) (Storn and Price, 1997), and Estimation of Distribution Algorithms (EDA) (Larrañaga and Lozano, 2001) are included. Whereas SI algorithms include Ant Colony Optimization (ACO) (Dorigo and Gambardella, 1997) and Particle Swarm Optimization (PSO) (Kennedy and Eberhart, 1995).

1.4 Objectives

This project focuses on the implementation of EDAs in Deep Neural Networks, which consist on learning and sampling from the probability distribution of the best individuals of the population at each iteration of the algorithm. The main objectives resides on looking for the best configuration of discrete hyper-parameters for (i) CNNs in classification problem tasks based on state of the art SHEDA algorithm (Li et al., 2021) using CIFAR-10 dataset and (i) for DCGANs in image generation tasks that maximize the quality of generated images from the generator network using MNIST dataset, since there is no previous work of DCGAN hyper-parameter optimization with EDAs.

In order to analyze deeply the different objectives involved in CNN hyper-parameter optimization, SHEDA algorithm and the contributions of the project are explained in detail.

Li et al. (2021) presented SHEDA algorithm, which consists on the first EDA implementation in hyper-parameter optimization problems in CNNs. The algorithm deals with the following challenges: (i) mixed of continuous and discrete hyper-parameters, (ii) the large-scale search space and (iii) expensive computational cost for individual evaluation.

1.4.1 Dependencies among hyper-parameters

Dealing with mixed type hyper-parameters, SHEDA algorithm uses fitness weighted methods to learn probabilistic models for continuous and discrete variables separately and then combines them to get the hybrid model for sampling new individuals. For continuous variables, weights are combined with Gaussian distribution to learn the probabilistic model and then to sample according to the Gaussian distribution. For discrete hyper-parameters, probabilistic models are also calculated based on weighted values, however, new individuals are sampled by the roulette selection method based on calculated probabilities. Although algorithm deals with mixed type hyper-parameters, it is assumed that there are no dependencies among the hyper-parameters.

The implementation of EDAs in CNNs in this project optimizes hyper-parameters encoded in discrete way, nevertheless, multivariate model is implemented capable of capturing multivariate interactions among hyper-parameters unlike SHEDA algorithm. In particular, Estimation of Bayesian Network Algorithm (EBNA) is implemented, which learns Bayesian Networks to sample the promising solutions. It uses the Bayesian Information Criterion (BIC) (Schwarz, 1978) to evaluate Bayesian Network structures in the greedy network construction algorithm.

1.4.2 Surrogate based evaluation

In order to reduce the computational costs required to evaluate each individual, surrogate-assisted multi-level evaluation method (SME) is implemented in SHEDA. Taking into consideration the evaluation of each individual consists on a CNN training, the evolutionary algorithm until its convergence is computationally very expensive. Therefore, it employs a surrogate model instead avoiding CNN training. SHEDA adopts Gaussian process model with default settings, which receives individuals as input and outputs the predicted fitness of the individual.

In particular, SME combines both evaluation methods, training based evaluation (CNN) and surrogate based evaluation. Firstly, surrogate based evaluation is used to evaluate the individuals, and then, the promising ones are evaluated with the training based evaluation. Individuals are considered promising when the surrogate based fitness evaluation is better than the average fitness of the individuals in the arch, being the arch the set of solutions that have been trained with training based evaluation.

This project also combines two evaluation methods, CNN training based evaluation and surrogate based evaluation method. However, what differs from SHEDA algorithm is (i) the surrogate model and (ii) the criteria of selecting the evaluation method to use.

Random Forest Regressor is the implemented surrogate model whose hyper parameters are optimized using grid search optimization method. With respect to evaluation method criteria, fixed and dynamic criteria are compared. In fixed criterion, the promising individuals are the individuals whose surrogate based fitness is better than a percentage of the previous generation fitnesses, a threshold. The considered promising ones are therefore evaluated with CNN training phase. Dynamic criterion, otherwise, is more selective as generations go by, reducing the percentage of the promising individuals and consequently the amount of CNN trainings, resulting on a reduction in computational cost.

1.4.3 CNN structure optimization

In addition, Li et al. (2021) present some limitations and future work related to SHEDA algorithm. SHEDA showed to be efficient tool for optimizing the hyper-parameters, which adopts ResNet model with depth 20 as the basic networks for hyper-parameter optimization. Although ResNet is a poweful CNN in DL that can lead to a promising solutions with limited time and resources, it may not be suitable for different real world problems. Since choosing a suitable basic CNN for different tasks is a difficult problem, the evolution of network depth and topology is considered for SHEDA as future research.

Taking the proposed future research into account, this project not only optimizes the hyper-parameters of a basic CNN, but it also searches for the most promising depth and topology, allowing to construct the CNN structure that performs best on different problems. So, this work optimizes the structure of the network and the hyper-parameters of the considered layers in the structure at once.

1.4.4 Multi-objective optimization

The implementation of a multi-objective EDA for hyper-parameter optimization is another contribution to previous work. It consist on minimizing the classification task error and the computational cost the network require, looking for the best performing network with the topology that requires the lowest computational cost. In order to carry it out, non-dominating sorting and crowding distance (Deb et al., 2002) are introduced into EDAs to rank individuals providing both intensity and diversity on the solutions over the search space.

This approach allows to select in each generation the solutions with lower classification error penalized by the domination counts, penalizing more the more dominated solutions and without penalizing the non dominated solutions. Moreover, it also allows the decision-maker to determine among the non-dominated solutions his preferences, whether to choose the hyper-parameter configuration that leads to lowest classification error with high computational cost, or to choose a solution that seeks a trade-off between both objectives, the classification error and the computational cost.

1.4.5 Summary

Firstly, the project extends the SHEDA algorithm by integrating a multivariate model that captures interactions between hyper-parameters, implementing EBNA algorithm for this purpose.

Secondly, it compares the convergence and the computational cost of different approaches that differ on the evaluation methods of individuals. Random Forest Regressor is employed as the surrogate model in some approaches to approximate and evaluate the fitness of the individuals. The combined approach, CNN training based and surrogate based evaluation approach, also explores both fixed and dynamic criteria for selecting the promising results, enhancing the efficiency and selectivity of the optimization process. Different population sizes are also compared analyzing both efficiency and required computational costs.

Thirdly, the project not only optimizes hyper-parameters but also searches for the most suitable network depth and topology, addressing the challenge of selecting an appropriate CNN structure for different tasks.

Additionally, the project introduces a multi-objective EDA approach to balance classification error and computational cost, providing flexibility in choosing optimal configurations.

Lastly, it is the first implementation of EDAs in the hyper-parameter optimization of DCGANs, aiming to maximize the quality of generated images.

1.5 Structure

State of The Art chapter, Chapter 2, reviews existing literature and methodologies relevant to the research topic, beginning with an overview of various optimization techniques, including Grid Search, Random Search, Bayesian Optimization, and Evolutionary Computation. It then delves into hyper-parameter optimization in computer vision, discussing CNNs and generative models, and concludes with an exploration of EDAs.

Chapter 3 details the methods used in the research, starting with an in-depth look at CNNs, explaining their components and the training process. It then describes the encoding scheme to be able to implement EDA to CNNs and continues with a discussion on EDA-CNN, outlining various training approaches, surrogate models, and multi-objective EDA-CNN. Finally EDA-DCGAN is introduced, providing insights into the corresponding networks and hyper-parameter optimization.

Chapter 4 presents the findings of the research, beginning with the results of EDA-CNN approaches as well as comparisons of the explained approaches, and the results related to EDA-DCGAN.

Finally, Chapter 5 summarizes the conclusions drawn from the research and the proposed future lines. The work concludes with a bibliography, providing references for the sources cited throughout the research.

Chapter 2

State of The Art

Since performance of DL algorithms depends on many factors such as the architecture and hyper-parameters, DL algorithm optimization has become a research topic in the field. Different techniques are analysed in literature aimed to optimize hyper-parameters.

2.1 Optimization techniques

2.1.1 Grid Search

Grid search is a technique that involves evaluating all possible combinations within a predefined set of hyper-parameters to identify the combination that optimizes the objective function, such as maximizing accuracy on a validation set. However, as the number of hyper-parameters increases, the number of network configurations that need to be trained grows exponentially. This makes the computational cost of grid search potentially prohibitive for high-dimensional hyper-parameter spaces. Priyadarshini and Cotton (2021), for instance, implemented grid-search to locate optimal hyper-parameters for a Long-Short-Term-Memory (LSTM)-CNN networks in order to classify more accurately polarity of sentiments. However, different optimization technique implementation is a mentioned future line.

2.1.2 Random Search

In order to avoid the computational costs grid-search involves, random search randomly samples hyper-parameter combinations from predefined sets. In high dimensional spaces, it results more efficient than grid search since not every combination is evaluated. Bergstra and Bengio (2012) demonstrated that random search over the same domain is able to find models that are as good or better within a small fraction of the computation time than grid-search since it searches in a larger configuration space. When it comes to its applications, Huber et al. (2021) implemented random search for CNN hyper-parameter optimization for noise reduction tasks in computed tomography and Torres et al. (2019) optimizes Feed-Forward Neural Network (FFNN) hyper-parameters for power consumption forecasting.

2.1.3 Bayesian Optimization

Bayesian Optimization, sequential model-based optimization, differs from previous techniques in that it improves the speed of searching using previous results, while the other two methods are independent of previous evaluations. Thus, the performance of past hyper-parameters affects the future decision. Bayesian Optimization builds a probability model of the objective function and uses the model to select following hyper-parameters to evaluate (Wu et al., 2019).

Probability model consists on a surrogate model that approximates the true objective function given a configuration of hyper-parameters. The selection of the hyper-parameters is made by maximizing the expected improvement, which consist on the difference between the minimum observed true objective function score and true objective function scores, being true objective function score the prediction of the surrogate model given a configuration of hyper-parameters. Surrogate model is constantly updating across iterations adding latest history samples.

Bergstra et al. (2011) and Wu et al. (2019) presented multiple algorithms that are common for training surrogate models, such as Gaussian Process Model (GP) (Williams and Rasmussen, 2006) also known as Kriging model and Tree-structured Parzen Estimators (TPE) (Bergstra et al., 2011). Many surrogate-based optimization algorithms that are successful in real-world applications utilize GP as their surrogate model, such as PESMO (Hernández-Lobato et al., 2016) and K-RVEA evolutionary algorithms (Chugh et al., 2017).

GP algorithm is a non-parametric and interpretable Bayesian model that predicts expected values and uncertainties of objectives, which are essential for balancing exploration and exploitation during optimization. However, standard GP has limitations: it is not well-suited for non-continuous search spaces and suffers from high computational complexity relative to data size. These issues are problematic since real-world problems often involve complex search spaces. There are different approaches in order to address these challenges: (i) incorporating advanced techniques into GP to enhance its performance, or (ii) using a surrogate model other than GP.

For the first approach, Liu et al. (2020) reviewed recent advances for improving the scalability and capability of GP models. For the second approach, non-GP-based optimization algorithms have been proposed, such as mentioned TPE algorithm. TPE is known for the standard solver of Hyperopt (Bergstra et al., 2015) and Optuna (Akiba et al., 2019), open source software for HPO that are widely used in machine learning. This surrogate model can naturally handle complex search spaces and scale to a greater amount of both variables and observations.

When it comes to applications, Victoria and Maragatham (2021) proposed Bayesian hyper-parameter optimization algorithm to enhance the performance of the CNN model in 10 classes classification task and Shin et al. (2020) developed a DNN model, optimizing its hyper-parameters using the Bayesian Optimization method to predict engine-out NOx emissions by using the worldwide harmonized light vehicles test procedure of diesel engines.

2.1.4 Evolutionary Computation

EC uses principles of natural selection to evolve a population of candidate solutions over generations. Population-based optimization algorithms (POAs) start by creating and updating a population with each generation, evaluating each individual until the global optimum is identified. The main differences between various POAs lie in their methods for generating and selecting populations. POAs can be easily parallelized, as a population of N individuals can be evaluated on up to N threads or machines in parallel. GAs, PSO and EDAs are some common POAs for HPO problems.

These evolutionary algorithms have been implemented for hyper-parameter optimization in different DL tasks: regression problems (Luo et al., 2020), classification problems (De Falco et al., 2019), time series forecasting (Nakisa et al., 2018; Erden, 2023), image classification (Aszemi and Dominic, 2019) and generative models (Lin et al., 2022) for instance.

Next section presents a more exhaustive analysis of the hyper-parameter optimization methods, such as evolutionary algorithms like GAs and EDAs, in computer vision tasks, including CNNs and generative models.

2.2 Hyper-parameter Optimization in Computer Vision

2.2.1 Convolutional Neural Networks

According to HPO in CNNs for image classification tasks, there are CNNs whose architectures are hand-crafted with extensive domain expertise: DenseNet (Huang et al., 2017), which includes dense blocks, ResNet (He et al., 2016), which includes residual blocks, VGG (Simonyan and Zisserman, 2014) and All-CNN (Springenberg et al., 2014).

However, when optimizing a CNN architecture for a specific task, it is impossible to know in advance the optimal architecture of the network, such as the optimal number of layers to create. Moreover, manually designing CNNs requires extensive expertise in both CNN architectures and the problem domain. These are often unavailable in practice. Therefore, there is a huge need for algorithms that can effectively and efficiently design CNN architectures without requiring such expertise.

Xie and Yuille (2017) proposed a genetic-CNN that aims to automatically learn deep network structures. Since the number of possible network structures grows exponentially with the number of network layers, they used a genetic algorithm to efficiently explore the large search space.

Liu et al. (2017) presented an effective evolutionary method (Hierarchical evolution) that identifies high-performance neural architectures based on a novel hierarchical representation scheme, in which smaller operations are used as building blocks to form larger ones.

Moreover, reinforcement learning is also implemented to look for an economical and efficient architecture search although they require more extensive computational resources. In Efficient Architecture Search (EAS) (Cai et al., 2018) algorithm, the reinforcement learning agent learns to take actions for network transformation to explore the architecture space efficiently, using knowledge stores in previously trained

networks. Zhong et al. (2018) also used Q-Learning paradigm with epsilon-greedy exploration strategy to choose component layers, known as Block-QNN-S. Zoph and Le (2016) also proposed Neural Architecture Search (NAS) with reinforcement learning.

Some previous algorithms presented limitations such as the fixed-length encoding without knowing the best network depth and the restriction of not using crossover operators in variable-length encoding schemes. Therefore, Sun et al. (2019) proposed a CNN structure evolution method based on residual blocks and GA (CNN-GA), where each block can have different configurations and the total number of blocks is also evolved based on a variable-length encoding scheme. Moreover, CNN-GA algorithm automatically discovers the best CNN architectures, and does not require any manual and human intervention.

Taking advantage of mentioned residual blocks, Song et al. (2020) presented an improved EA to optimize the block types in the network and the hyper-parameters of each block.

Some presented algorithms, such as CNN-GA and NAS are completely automatic algorithms. Additionally, Cartesian Genetic Programming (CGP-CNN) algorithm (Suganuma et al., 2017) and Large Scale Evolution algorithm (Real et al., 2017) are also completely automatic evolutionary algorithms aimed to optimize hyper-parameters.

When it comes to PSO swarm intelligence algorithm implementation in HPO problems for CNN algorithms, Wang et al. (2019b) presented cPSO-CNN algorithm which brings three mechanisms: (i) vectorizing the acceleration coefficients to adapt for variant ranges of CNN hyper-parameters, (ii) enhancing exploration capability with compound normal confidence distribution, and (iii) linear-estimation based scheme for fast fitness evaluation.

Singh et al. (2021) implemented multi-level PSO for CNN architecture and hyper-parameter optimization, training the initial swarm at first level optimizing architecture and multiple swarms at level two optimizing hyper-parameters used in each layer.

Wang et al. (2018) proposed variable-length PSO, IPPSO-CNN, for optimizing CNN architecture in which three improvements are made based on traditional PSO: (i) new design of encoding scheme in order to efficiently encode CNN architecture inspired by how the network IP address works, (ii) deal with the constraint of the fixed length encoding of traditional PSO in order to learn variable-length architectures of CNNs and (iii) use of partial dataset to accelerate fitness evaluation and therefore evolutionary process.

Multi-objective evolution are also analysed in literature. Vidnerová and Neruda (2020) proposed a novel approach to neural architecture search for deep neural networks. The proposed algorithms are based on multi-objective genetic algorithms: NSGA-II (Deb et al., 2002) and NSGA-III (Deb and Jain, 2013). The objectives consist on optimizing simultaneously both network architecture and network size performance in order to generate efficient networks of reasonable size. NSGA-II-CNN and NSGA-III-CNN are two implemented algorithms for CNN optimization.

When it comes to Estimation of Distribution Algorithms, Li et al. (2021) proposed SHEDA algorithm, which consists on EDA implementation to find suitable hyper-parameter configuration. They identified three limitations in HPO problem in CNN: (i)

mixed of continous and discrete hyper-parameters, (ii) the large-scale search space and (iii) expensive computational cost for individual evaluation. In order to tackle these limitations, SHEDA algorithm contributions are (i) hybrid-model EDA which uses a mixed-variable encoding scheme, (ii) orthogonal initialization to help initialize solutions to cover all possible choices of each variable and (iii) surrogate-assisted multi-level evaluation method to reduce the expensive computational cost. SHEDA outperforms other state of the art algorithms.

2.2.2 Generative models

Since the optimization of network architecture and hyper-parameters can significantly improve the generation performance, Wang et al. (2019a) firstly implemented evolutionary computing methods in DCGANs, known as evolutionary GAN (E-GAN). In E-GAN, generators are regarded as an evolutionary population and discriminator acts as an environment. However, network architecture is static and fixed.

Taking into account convolutions cannot model perplexing geometric shapes and that they are difficult to learn long-range-dependencies, self-attention mechanism is introduced into each generator and discriminator in Attentive E-GAN (AEGAN) algorithm (Wu et al., 2021). By using normalized self-attention mechanism, the generator can draw images based on the importance of the features.

Gong et al. (2019) and Wang and Huan (2019) attempted to implement NAS into GANs. In AutoGAN (Gong et al., 2019) a Recurrent Neural Network controller is used to conduct the search for the generator architectures. AGAN (Wang and Huan, 2019), however, used reinforcement learning to automatically design the generator in GANs.

Lin et al. (2022) firstly attempted to introduce evolutionary algorithms into GANs to optimize the architecture and its associated hyper-parameters at once, Evolutionary Architecture Search GAN (EAS-GAN).

An evolutionary algorithm assisted GAN framework, EvoGAN, was also proposed (Liu et al., 2022) to generate various compound expressions. They transfered the synthesis of compound expression to an optimisation problem and use genetic algorithms to search for the optimum.

With regard to Variational Autoencoders (VAE), Chen et al. (2020) proposed EvoVAE algorithm in which VAE is generalized to a more general and asymmetric VAE with four blocks. Moreover, a mechanism for encoding genes with a variable-length genetic algorithm is proposed in the algorithm with an adapted efficient genetic operator to find the optimal network depth.

In order to determine the performance metric of generative models, Inception Score (IS) (Salimans et al., 2016) and Fréchet Inception Distance (FID) (Heusel et al., 2017) are metrics to quantitatively evaluate the quality of images synthesis. IS calculates the Kullback-Leibler divergence between the conditional class distribution and the marginal class distribution. Higher IS means better image synthesis quality.

On the other hand, FID calculates the Wasserstein-2 distance between the generated samples and the real images in the feature space of the Inception-v3 network. Lower FID values indicate a smaller distance between the distributions of generated and real data.

2.3 Estimation of Distribution Algorithms

Taking into consideration the different hyper-parameter optimization techniques analysed in computer vision tasks, and the project objective of implementing an EDA to this purpose, a definition and an analysis of the state of the art of EDAs is developed in this section.

EDAs are evolutionary algorithms that explore the solution space in order to achieve promising solutions. In a generation, a population of solutions is taken into account, considering EDAs, therefore, population based algorithms. Individuals in population evolve over generations to converge towards local or global optimum.

In order to evaluate individuals at each generation, fitness function is designed, which determines the quality of individuals in population. This quantitative evaluation allow to rank solutions so as to select the most promising solutions minimizing or maximizing fitness value.

When it comes to general algorithm procedure, Algorithm 1, first initial population is randomly sampled. From this initial population, algorithm learns a model that attempts to capture the probability distribution of the promising solutions. Once the model is constructed, new solutions are generated by sampling the distribution encoded by the learned model. What differ EDAs from other evolutionary algorithms are the mentioned learning and sampling steps. The sampled solutions are evaluated using fitness function and ranked to select the promising solutions with a selection criterion. These promising individuals form the next generation population. The process is repeated until termination criterion is met. An example of EDA procedure of first generation is illustrated in Figure 2.1.

Algorithm 1 EDA Pseudocode

$g \leftarrow 0$
Generate initial population $P(0)$
while not done **do**
  Select promising solutions $S(g)$ from $P(g)$
  Build probabilistic model $M(g)$ from $S(g)$
  Sample $M(g)$ to generate new candidate solutions $O(g)$
  Incorporate $O(g)$ into $P(g)$
  $g \leftarrow g + 1$
end while

Depending on the types of variables to optimize and the dependencies among variables, different models are analysed in literature.

2.3.1 Discrete variables

For discrete type variables without dependencies, Univariate Marginal Distribution Algorithm (UMDA) (Mühlenbein and Paass, 1996), Population Based Incremental learning (PBIL) (Baluja, 1994) and compact Genetic Algorithm (cGA) (Harik et al., 1999) models are some common EDAs.

UMDA uses a probability vector $p = (p_1, p_2, \ldots, p_n)$ as the probabilistic model, where p_i denotes the probability of a 1 in position i of solution. PBIL and cGA also uses probability vectors, however, unlike UMDA, PBIL and cGA are incremental models,

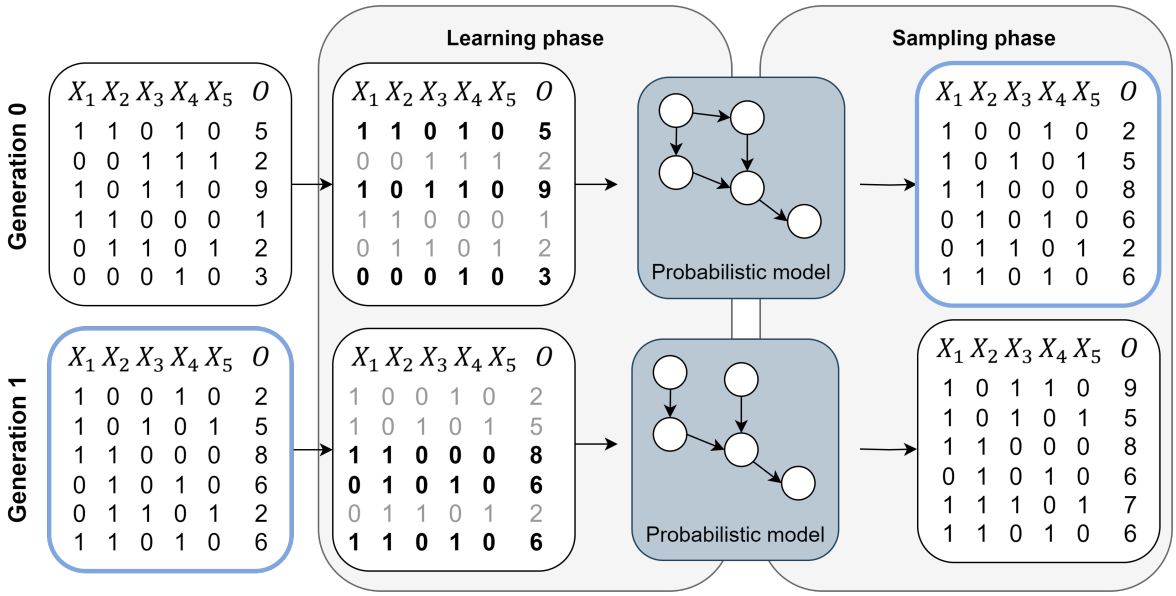


Figure 2.1: *Estimation of Distribution Algorithm procedure.*

what means they use a probabilistic model to represent the distribution of potential solutions without storing large populations of solutions. The probabilistic model is updated incrementally over time.

The Mutual Information Maximizing Input Clustering (MIMIC) (De Bonet et al., 1996) is a bivariate EDA, capable to capture pair-wise interactions between variables. MIMIC calculates the mutual information between all pairs of variables. Variables with highest mutual information with the last added variable are added to a chain until all variables are included. So, final model consists of a tree of a single chain of dependencies, where each parent has one child. The probability distribution at generation l, factorizes as

$$p_{\pi_l}(\mathbf{x}) = p_l(x_{i_1} \mid x_{i_2}) p_l(x_{i_2} \mid x_{i_3}) \cdots p_l(x_{i_{n-1}} \mid x_{i_n}) p_l(x_{i_n})$$

where $p(X_{i_j} \mid X_{i_{j+1}})$ denotes the conditional probability of X_{i_j} given $X_{i_{j+1}}$.

In order to model multivariate interactions between the variables, there are several EDAs that are based on probabilistic models: Extended Compact Genetic Algorithm (ECGA) (Harik et al., 2006), Bayesian Optimization Algorithm (BOA) (Pelikan et al., 1999), Estimation of Bayesian Network Algorithm (EBNA) (Etxeberria and Larrañaga, 1999), Learning Factorized Distribution Algorithm (LFDA) (Mühlenbein et al., 1999) and Markovianity based Optimization Algorithm (MOA) (Shakya and Santana, 2008).

The Extended Compact Genetic Algorithm (ECGA) groups variables into clusters, treating each as a single variable. Starting with all variables independent, it iteratively merges clusters to improve the model's Minimum Description Length (MDL) until no further improvement is possible. A probability table for each cluster is then computed to generate new solutions. This process repeats every generation, potentially forming different clusters each time.

BOA, EBNA and LFDA use Bayesian Networks to model candidate solutions. A Bayesian network (Pearl, 1988) is a probabilistic graphical model (G, P) over vari-

ables $\mathbf{X}$, where G is a qualitative component in the form of a directed acyclic graph (DAG) composed of vertices and edges $G = (V, E)$, and P is a quantitative component representing a set of conditional probabilities (CPTs). The conditional probabilities factorize the joint probability distribution (JPD) using the chain rule, and the Bayesian network can be viewed as an encoding of the local Markov condition, meaning that each variable X_i is conditionally independent of its non-descendants given its parents $\text{Pa}(X_i)$.

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i \mid \text{Pa}(X_i))$$

Mentioned algorithms start with a network without edges and a greedy algorithm is then employed to enhance the network by iteratively adding the edge that provides the greatest improvement according to a metric. BOA employs the Bayesian-Dirichlet (BD) metric, whereas EBNA and LFDA use Bayesian Information Criterion (BIC) (Schwarz, 1978).

MOA algorithm encode multivariate interactions with Markov Networks. Markov Network connections, unlike Bayesian Network connections, are undirected. In MOA, Gibbs sampling is used to generate new solutions from learnt model.

This project focuses on mainly EBNA model since it allows modelling multivariate interactions between CNN discrete hyper-parameters. Therefore, Algorithm 2 explains in detail EBNA model procedure.

Algorithm 2 starts with an initial model M_0 composed of an initial structure S_0 and parameters θ_0. By sampling N individuals from the initial model M_0, generates a dataset D_0. Then, following steps are repeated until a stopping criterion is met:

1. Selection: Select S_e individuals from the dataset D_{l-1} to form D_{l-1}^{Se}.
2. Structure learning: Find the structure S_l' that maximizes the Bayesian Information Criterion (BIC) score based on the selected data D_{l-1}^{Se}.
3. Parameter calculation: Calculate the parameters θ_l using the formula $\theta_{ijk} = \frac{N_{ijk}^l + 1}{N_{ij}^l + r_i}$, where D_{l-1}^{Se} is used as the dataset. θ_{ijk} represents the conditional probability of variable X_i being in its k-th value, given that the set of its parent variables is in its j-th value. N_{ijk}^l refers to the number of cases where a variable X_i takes its k-th value and its parent variables are in its j-th configuration and N_{ij}^l the number of cases where the parents of the variable X_i take the j-th configuration. r_i represents all possible values of X_i.
4. Model update: Update the model to M_l with the new structure S_l' and parameters θ_l.
5. Sampling: Generate a new dataset D_l by sampling N individuals from the updated model M_l using the Probabilistic Logic Sampling (PLS) method.

2.3.2 Continuous variables

For continuous type variables, continuous UMDA (UMDA$_C$) (Mühlenbein and Paass, 1996) and continuous PBIL (PBIL$_C$) (Baluja, 1994) are univariate EDAs that use inde-

Algorithm 2 EBNA$_{BIC}$

1: $M_0 \leftarrow (S_0, \theta_0)$
2: $D_0 \leftarrow$ Sample N individuals from M_0
3: **for** $l = 1, 2, \ldots$ until a stop criterion is met **do**
4: $\quad D_{l-1}^{Se} \leftarrow$ Select S_e individuals from D_{l-1}
5: $\quad S'_l \leftarrow$ Find the structure which maximizes $BIC(S_l, D_{l-1}^{Se})$
6: $\quad \theta^l \leftarrow$ Calculate $\{\theta_{ijk} = \frac{N_{ijk}^l + 1}{N_{ij}^l + r_i}\}$ using D_{l-1}^{Se} as the data set
7: $\quad M_l \leftarrow (S'_l, \theta^l)$
8: $\quad D_l \leftarrow$ Sample N individuals from M_l using PLS
9: **end for**

pendent Gaussian distributions. MIMIC$_C$ is a continuous bivariate EDA which learns a chain structured probabilistic model by adapting the concept of conditional entropy for Gaussian distributions.

Estimation of Gaussian Networks Algorithm (EGNA) (Larrañaga et al., 2000), Continuous Iterated Density Estimation Algorithm (IDEA) (Bosman and Thierens, 2000), Estimation of Multivariate Normal Distribution Algorithm (EMNA) (Larrañaga and Lozano, 2001) and Distribution Estimation Using the Markov network algorithm (DEUM) (Shakya et al., 2009) are some multivariate EDAs for continuous variables.

EGNA works by creating a Gaussian Bayesian Network to model the interactions between continuous variables in the population of current generation. The network structure is learned greedily using a continuous version of the Bayesian Dirichlet (BDe) metric (Heckerman et al., 1995), with a penalty term to prefer simpler models. As EGNA, IDEA also models a Gaussian Bayesian Network. In EMNA a multivariate Gaussian distribution is estimated based on the best solution of the previous iteration. DEUM algorithm, on the other hand, uses Markov Networks to model and sample the distribution.

Soloviev et al. (2023) proposed a semiparametric EDA (SPEDA) that overcomes the limitations of traditional EGNAs by relaxing the Gaussianity assumption. The algorithm uses semiparametric Bayesian networks where kernel-estimated nodes coexist with Gaussian nodes, allowing the algorithm to choose the most suitable type for each variable. Additionally, it leverages information from multiple past iterations to build the network, improving robustness and reducing solution variance.

2.3.3 Applications in Computer Vision

Although EDAs have been applied in various deep learning techniques, minimal research exists on their use in hyper-parameter optimization for computer vision tasks. In contrast, GA and PSO algorithms have been more extensively analyzed.

SHEDA algorithm (Li et al., 2021), as mentioned, is the first implementation of EDA in hyper-parameter optimization problem in CNNs. This project focuses its motivation in improving the limitations SHEDA presents and in developing presented future lines.

Moreover, there is no work in which EDAs are implemented to hyper-parameter optimization for any generative model, such as DCGANs.

Chapter 3

Methodology

Methodology chapter begins by detailing the performance of CNNs, including the structure and the training hyper-parameters necessary for optimization. This is followed by an encoding scheme designed for the execution of the EDA. The main characteristics of the EDA-CNN algorithm are then presented, highlighting various training approaches and the role of a surrogate model in addressing certain training scenarios. Additionally, the implementation of a multi-objective EDA-CNN is explained. Finally, the EDA-DCGAN section is introduced, which analyzes the application of EDA for optimizing the hyper-parameters of DCGANs. The code is available in Github[1].

3.1 Convolutional Neural Networks

A CNN is a type of deep learning model specifically designed for processing data with a grid-like topology, such as images. CNNs are designed to automatically and adaptively learn spatial hierarchies of features, ranging from low-level to high-level patterns. This mathematical framework typically consists of three main types of layers: convolutional, pooling, and fully connected layers. The convolutional and pooling layers are responsible for feature extraction, while the fully connected layer maps these extracted features to the final output, such as classification results. Figure 3.1 illustrates CNN structure. Moreover, residual blocks can be also added to CNNs, which consist of architectural components that help address the vanishing gradient problem, enabling the training of very DNN by incorporating skip connections.

3.1.1 Convolutional layers

A convolution layer is a core component of the CNN architecture, responsible for feature extraction through a combination of linear and nonlinear operations, specifically the convolution operation and activation function.

- Convolution operation

This operation involves a small array of numbers, called a kernel, applied across the input tensor. At each location of the tensor, an element-wise product between the kernel and the input tensor is calculated and summed to produce an output value in the feature map. Multiple kernels are applied to generate various feature maps,

[1] https://github.com/JanireAmestiS/EDA_HPO.git

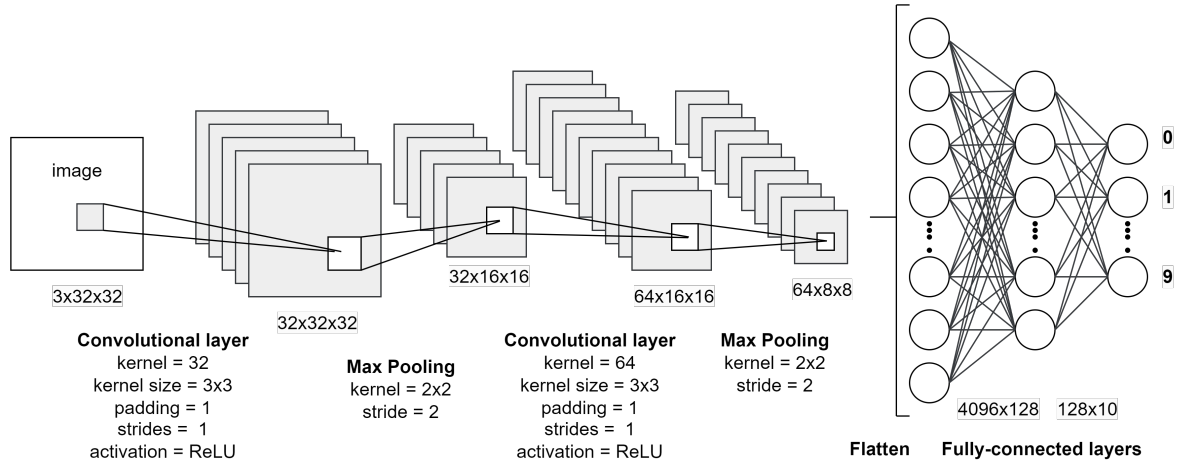


Figure 3.1: *Convolutional Neural Network structure*

each representing different characteristics of the input. Key hyper-parameters for convolution are the size and number of kernels; common sizes are 3×3, 5×5, or 7×7, while the number of kernels determines the depth of the feature maps.

Kernels are shared across all image positions, making the extracted features translation invariant, allowing the model to learn spatial hierarchies of features, and enhancing model efficiency by reducing the number of parameters compared to fully connected networks.

The convolution operation reduces the height and width of the output feature map compared to the input tensor, as the center of each kernel cannot overlap the outermost element of the input tensor. To address this, padding, typically zero padding, adds rows and columns of zeros around the input tensor to ensure the convolution does not shrink the output feature map compared to the input tensor, maintaining the same dimensions. Zero padding also avoids feature maps getting smaller after convolution operation, allowing, therefore, adding more layers.

The stride in the convolution operation refers to the number of pixels the filter matrix moves across the input matrix. A stride of 1 means the filter moves one pixel at a time, while a stride of 2 means it jumps two pixels, and so on. The value of the stride impacts the model in several ways. Larger stride values result in smaller output dimensions, effectively performing dimensionality reduction. They also speed up computation since the filter is applied fewer times. However, larger strides can cause the model to lose detailed information because the filter skips over some pixels, potentially reducing accuracy. So, choosing the appropriate stride value involves balancing information preservation with computational efficiency and dimensionality reduction.

During training, the CNN model learns the optimal kernels for a given task, while the kernel size, number of kernels, padding, and stride are predetermined hyper-parameters.

- Activation functions

The activation function introduces non-linearities to CNN, which are desirable for multi-layer networks to detect nonlinear features

The sigmoid function transforms the values in the range 0 to 1 and it is continuously differentiable. Also, the sigmoid function is not symmetric about zero, which means that the signs of all output values of neurons will be the same.

Hyperbolic Tangent function (Tanh) is similar to the sigmoid function but it is symmetric to around the origin. It is continuous and differentiable and the values lies in the range -1 to 1. It has gradients which are not restricted to vary in a certain direction and also, it is zero centered.

Rectified Linear Unit (ReLU) widely used non-linear activation function in neural network. The function is linear for values greater than zero, but it is a nonlinear function as negative values are always output as zero, deactivating neurons in this case. The inactivity of neurons during training and consistently outputting zero, leads to loss the learning capability. This problem is known as dying ReLU problem.

In order to deal with dying ReLU problem, Learky ReLU instead of defining the negative values as zero, it defines a small linear component of negative values and Exponential Linear Unit (ELU) introduces a parameter slope for the negative values, using a curve for defining the negative values.

3.1.2 Pooling layers

Pooling layers perform downsampling operations along the spatial dimensions (width and height) of the input, reducing the dimensionality of the feature maps. This process helps in reducing both the computational complexity and the amount of trainable parameters.

The most popular form of pooling operation is Max pooling, which extracts patches from the input feature maps, outputs the maximum value in each patch, and discards all the other values. A max pooling with a filter of size 2×2 is commonly used in practice. It downsamples the high and width dimension of feature maps by a factor of 2. Depth dimension of feature maps, however, remains unchanged. Average pooling works same but it performs average value in each patch.

Global Average pooling, performs an extreme form of downsampling by converting a feature map into a 1×1 array. This is achieved by averaging all the elements in each feature map, while retaining the depth of the feature maps. Global Maximum pooling operation, calculates the maximum instead. Global pooling is typically applied only once, just before the fully connected layers.

3.1.3 Fully connected layers

The fully connected layer serves as the final stage in the CNN process, integrating all the extracted features to make a prediction.

The output feature maps from the final convolution or pooling layer are typically flattened into a one-dimensional vector. This vector is then fed into one or more fully connected layers, also known as dense layers, where each input is connected to every output through a learnable weight.

Each fully connected layer is followed by a non-linear activation function, such as ReLU, which introduces non-linearity into the model, aiding in learning complex patterns. This combination of dense connections and non-linear activation functions

allows the network to effectively translate high-level features into accurate predictions, enabling successful tasks like image classification. The last fully connected layer usually has the same number of output nodes as there are classes in the task.

This enables the network to accurately classify or predict outcomes based on the complex patterns and relationships present in the input data.

The number of dense layers, the amount of neurons in each dense layer and activation functions are predetermined hyper-parameters, weights, however, are trainable parameters.

3.1.4 Residual blocks

A residual block consist on an architecture designed to improve neural network training and performance.

It consists mainly of the following components: (i) two convolutional layers, each with a kernel size of 3x3 and padding of 1, (ii) batch normalization after each convolutional layer normalizing the input to the following layer stabilizing and accelerating training, (iii) non-linear activation functions such as ReLU applied after first batch normalization and again after skip connection and (iv) skip connection which adds the input of the block to the output of the last batch normalization layer.

This connection facilitates gradient flow, reducing the vanishing gradient problem and performance degradation problem, enabling the training of deeper networks.

In summary, the input to a residual block is added to the output of its layers before the final activation function is applied. This allows the network to learn both the identity function and any necessary transformation, improving generalization and the ability to learn complex patterns. Learning the identity function refers to the ability to learn to output the same input as received. This is interesting since it ensures that if learning transformations in some layers do not improve the model, those layers can at least return the same output, without causing any harm.

3.1.5 CNN training

In CNN training procedure, there are other hyper-parameter to optimize such as the batch size, the learning rate and the optimizer to use for weight updating.

- Batch size

The batch size is a hyper-parameter that determines the number of samples to work through before updating the internal model parameters. This constitutes a single iteration.

The choice of batch size can have a significant impact on the learning process. A smaller batch size can lead to faster convergence and can help the model escape from local minima. It also reduce memory requirements. However, it may lead to noisy gradient estimates, which can lead to instability in the learning process. A larger batch size can reduce the variance of the gradient estimates and improve the stability of the training. However, it also increases the memory requirements and may lead to slower convergence. Therefore, batch size involves trade-off between stability and speed.

- Learning rate

Learning rate determines the size of the steps taken during the optimization process and determines how quickly or slowly a model converges to the optimal solution. A large learning rate can lead to rapid convergence but may result in unstable and oscillating training. A small learning rate, however, can ensure stable and smooth training but may result in slower convergence. Therefore, it is important to choose a learning rate that balance both training speed and stability.

Learning rate decay methods have been implemented, which consist on start training the network with a large learning rate and then slowly reduce it until local minima is obtained, with the aim to avoid oscillation and to obtain faster convergence. Larger learning rates at first training epochs accelerate training and help the network escape local minima. As epochs goes by, decaying the learning rate helps the network converge to a local minimum and avoid oscillation.

Step decay schedule, for instance, drops the learning rate by a factor every few epochs and exponential decay schedule decays the learning rate using an exponential function.

- Optimization algorithms

Gradient descent is an optimization algorithm used to minimize the loss function by iteratively adjusting the model parameters. The algorithm calculates the gradient of the loss function with respect to each parameter, indicating the direction of the steepest increase in loss. By updating the parameters in the opposite direction of the gradient, the algorithm seeks to reduce the loss and find the optimal values that minimize prediction error. There are three variants of gradient descent, which differ in how much data we use to compute the gradient of the objective function: batch gradient descent, stochastic gradient descent (SGD) and mini-batch gradient descent.

Batch gradient descent calculates the gradient of the loss function using the entire training dataset. This ensures stable and accurate parameter updates but can be computationally expensive and slow, especially for large datasets.

SGD updates the model parameters using one training example at a time. This introduces noise in the updates, which can help escape local minima but results in less stable convergence.

Mini-batch gradient descent divides the training dataset into small batches and uses these to compute the gradient and update the model parameters. This approach balances the efficiency of SGD with the stability of batch gradient descent, reducing computational burden and speeding up training while maintaining more stable and accurate updates.

Momentum is an optimization technique that accelerates gradient descent by adding a fraction of the previous update to the current update. This helps to smooth out the path of the parameter updates, reducing oscillations and speeding up convergence, especially in directions with consistent gradients.

Nesterov Accelerated Gradient improves momentum by looking ahead at the future position of the parameters. This leads to more informed and potentially faster updates and also helps to correct the overshooting problem associated with momentum.

Adaptive Gradient (Adagrad) adapts the learning rate for each parameter individually

based on the historical gradient information. It adapts the learning rate to the parameters, performing larger updates for infrequent parameters and smaller updates for frequent parameters. For this reason, it is well-suited for dealing with sparse data.

AdaDelta is an extension of Adagrad that tries to deal with decaying learning rate problem that Adagrad may present. It restricts the window of gradient accumulation to a fixed size, allowing the learning rate to adapt over time without decaying to near-zero. This makes it more robust for long training sessions.

Root Mean Square Propagation (RMSProp) is another adaptive learning rate method. It is an extension of gradient descent and AdaGrad algorithms. The algorithm works by exponentially decaying the learning rate every time the squared gradient is less than a certain threshold. It helps to avoid the problem of the learning rate being too small or too large. If the gradients are small, the learning rate will be increased to speed up convergence, and if the gradients are large, the learning rate will be decreased to avoid overshooting the minimum of the loss function.

Adaptive Moment Estimation (Adam) combines the advantages of both RMSProp and Momentum. Adam keeps track of an exponentially decaying average of past gradients. This helps to smooth out the updates and navigate noisy gradients. Adam also computes an exponentially decaying average of past squared gradients. This is used to scale the learning rate for each parameter, allowing for larger updates for infrequent features and smaller updates for frequent ones. It incorporates a bias correction step to counteract that the means calculated in the initial iterations are biased towards zero. AdamW is a variant of the Adam optimizer that separates weight decay from the gradient update.

3.2 EDA encoding scheme

Individuals in an EDA have a fixed length, however, an objective of the project consists of looking for the best promising network topology for classification tasks, so a fixed length solution may not be suitable for this purpose. Therefore, a criteria has been develop in order to encode a not fixed network structure into a fixed length solution.

Firstly, the hyper-parameters to optimize are the following: amount of convolutional layers, amount of filters of each layer, convolutional layers kernel size, convolutional layer strides, whether or not to include a residual block, whether or not to include a pooling layer, amount of fully connected layers, amount of neurons in each fully connected layer, batch size, learning rate and optimizer. Therefore, an individual is an array composed with 11 real number values.

Each hyper-parameter contains multiple options and the encoding of the individuals will be encoded based on these options.

Batch size and learning rate, for instance, are discretized, being batch size options powers of 2, such as 16, 32, 64, 128, 256; and learning rate options 0.01, 0.001, 0.0001 or 0.00001. Optimizer options, on other hand, are Adam, SGD, AdamW and RMSProp.

Encoding of convolutional layer characteristics such as amount of convolutional layers, amount of filters of each layer, convolutional layers kernel size, convolutional

layer strides, residual blocks and pooling layers is limited to a maximum number of layers, 10 layers in the project, establishing a maximum length of 10 values.

For these variables an option does not consist of a real value number, but of a list of maximum 10 values. For instance, an option in amount of filter hyper-parameter would be [32, 32, 64, 64, 128, 128, 64, 64, 32, 32]. This vector determines a unique solution, a combination of kernels of different convolutional layers. In this case, therefore, the amount of convolutional layers would be 10. However if a combination contains 0s, for example [32, 32, 64, 64, 128, 128, 0, 0, 0, 0], the amount of layers would be 6 and will contain 32, 32, 64, 64, 128 and 128 kernels each.

Residual blocks and pooling layers options are also arrays of maximum length of 10, in this case binary value array, and each option indicate where to add the residual block or pooling layer. For example option [0,0,0,0,0,0,0,0,0,0] for residual block hyper-parameter means there is no residual blocks in the topology of the individual network. [0,0,1,0,0,1,0,0,0,0] option, however, means there are 2 residual blocks in network topology. First residual block will start in third convolutional layer and second one in sixth convolutional layer. In order to know how many kernels add to these residual blocks, third and sixth position of the amount of kernel hyper-parameter will determine the kernels of these residual blocks. In this case, two more convolutional layers will be added since a single residual block is formed by two convolutional layer structure.

Pooling layer hyper-parameter options work same, a pooling layer is added after the convolutional layer position in which a 1 appears. [0,0,0,1,0,0,0,1,0,0] option indicates 2 pooling layers after fourth and eighth convolutional layers. Since pooling layers perform downsampling along spatial dimensions, width and height, there is a limited amount of pooling layers to include, so the addition of pooling layers is restricted to a determined number. Depending on the size of the original images, this amount changes.

The hyper-parameter that indicates the depth of the network is the amount of kernels in each convolution layer. So, if variable indicates a depth of 8 layers, the length of the other hyper-parameters will be shorted to a length of 8. For instance, if amount of kernel option is [32, 32, 64, 64, 128, 128, 0, 0, 0, 0], pooling layer hyper-parameter [0,0,0,1,0,0,0,1,0,0] option will be taken into consideration only up to its sixth position.

Fully-connected network encoding works with same criterion. Number of neurons of each layer hyper-parameter will determine the amount of fully connected layers and its amount of neurons. [64] option determines a fully connected layer with 64 neurons followed by a final fixed layer with the same amount of neurons as classes of the classification task.

In previous paragraphs the options and the encoding of each hyper-parameter have been explained. The encoding of an individual of EDA is based on these options. Values of the individual array consist of the index of the selected option within a set of options for each hyper-parameter. Since options are predetermined, an individual will include 9 values since there are 9 hyper-parameters excluding amount of convolutional layers and fully-connected layers and each value will be in range from 0 to number of options or combinations in the set of each hyper-parameter. Then, the hyper-parameters for the number of convolutional layers and the number

of fully connected layers are added to the solution, which are determined from the already defined hyper-parameters: number of neurons of fully-connected layers and the number of filters of each convolutional layer. So, individual, finally will contain 11 values, being 11 the amount of hyper-parameters to optimize. Figure 3.2 illustrates an example of the encoding of an individual.

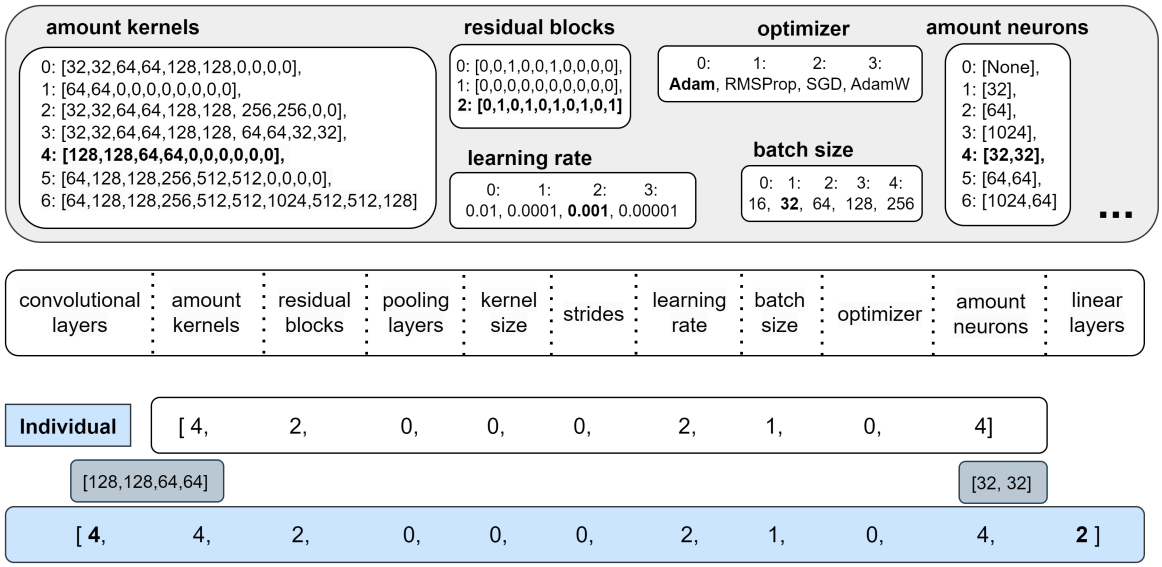


Figure 3.2: *Individual encoding scheme.*

3.3 EDA-CNN

Taking into consideration CNN hyper-parameters to optimize are discrete hyper-parameters, the implemented EDA is the multivariate EBNA model, allowing also dependencies between CNN hyper-parameters.

In order to analyse the performance of the algorithm, a 10 category classification problem is used, CIFAR-10 (Krizhevsky et al., 2009) dataset. Dataset contains 50000 training images and 10000 test images, where each image height and width are 32 × 32 pixels and each pixel contains three channels.

When it comes to EDA main characteristics in the proposed implementation, a generation is composed by 50 individuals that allow to learn the Bayesian Network. The structure learning phase is carried out implementing a score+search algorithm that looks for the Bayesian Network that maximizes the BIC score, Hill Climb search algorithm. The algorithm starts with an initial direct acyclic graph (DAG) and then iteratively makes small changes to the DAG in order to improve BIC of the solution. It proceeds with small changes until it reaches a local maximum, meaning that no further improvement can be made.

Although a population size of 50 is set for experiments, an analysis will also be conducted in order to understand the effect of population size and the generations required to converge to a local minimum.

Related to CNN, topology and training characteristics are defined by the selected hyper-parameters. However, all individuals have some common characteristics: (i)

loss function, (ii) number of training epochs and (iii) learning rate scheduler.

CNN training aims to minimize the cross-entropy loss, which measures the difference between the discovered probability distribution of a classification model and the predicted values. It allows to find the optimal solution by adjusting the weights of CNN model during training. A measure closer to 0 is a sign of a good model, whereas a measure closer to 1 is a sign of a poor-performing model. Taking it into consideration, EDA focuses on minimizing the cross-entropy loss.

Learning rate scheduler is implemented, OneCycle learning rate concretely, which anneals the learning rate from an initial value to some maximum learning rate and then from that maximum to some minimum learning rate much lower than the initial one. The OneCycle learning rate policy changes the learning rate after every batch, this is every step. Thus, learning rate hyper-parameter will determine the maximum learning rate the scheduler can reach.

Number of epochs needed to determine the quality of a CNN architecture in CIFAR-10 problem is set to 25 epochs providing a trade-off between quality and computational cost. Experiments showed training less epochs do not allow to distinguish between the solutions that get stuck in local minimum and the solutions that keep improving in following epochs. On other hand, although training more epochs result on a more precise performance metric, computational costs also increase. Therefore, it has been analysed 25 epochs is enough to identify the solutions that keep on improving and performing better. Since evolutionary algorithm aims to identify the best solutions in a population, it is enough to identify which ones keep on improving and performing well.

Following sections study (i) different training approaches with different evaluation methods for individuals in a population of the EDA, (ii) the training of a necessary surrogate model to carry some training approaches out and (iii) the multi-objective approach aiming to look for best hyper-parameters taking into consideration its computational training costs.

3.3.1 Training approaches

An individual in each generation is evaluated with a fitness function. The fitness value of each individual, in this case, corresponds to the cross-entropy loss function achieved in the CNN training with the hyper-parameters the solution impose. Being f the fitness function which corresponds to CNN training, EDA-CNN procedure is explained in Algorithm 3.

As each generation contains 50 individuals, each generation requires 50 CNN trainings. However, each training requires high computational costs and resources. In order to deal with this limitation, a surrogate model is considered simplifying evaluations of complex CNN models. Surrogate model maps input data to outputs when its relationship is computationally expensive to evaluate, avoiding CNN trainings for each individual of the generation.

Therefore, a surrogate model must be trained to be able to approximate fitness value given the hyper-parameters of the solution. Section 3.3.2 explains in detail the surrogate training procedure.

CNN-Surrogate approach consists of replacing f CNN training function with surrogate

Algorithm 3 EDA-CNN

1: $M_0 \leftarrow (S_0, \theta_0)$
2: $D_0 \leftarrow$ Sample N individuals from M_0
3: **for** $l = 1, 2, \ldots$ until a stop criterion is met **do**
4: $\quad D_{l-1}^{Se} \leftarrow$ Select S_e individuals from D_{l-1} depending on F_{l-1}
5: $\quad S_l' \leftarrow$ Find the structure which maximizes $BIC(S_l, D_{l-1}^{Se})$
6: $\quad \theta^l \leftarrow$ Calculate $\{\theta_{ijk} = \frac{N_{ijk}^l + 1}{N_{ij}^l + r_i}\}$ using D_{l-1}^{Se} as the data set
7: $\quad M_l \leftarrow (S_l', \theta^l)$
8: $\quad D_l \leftarrow$ Sample N individuals from M_l using PLS
9: $\quad F_l \leftarrow f(D_l)$ CNN training
10: **end for**

model prediction step. This step does not require computational resources, since it does not work with images and no CNN training is necessary. A surrogate model g must be build with initial solutions O.

However, surrogate based evaluations are approximations of the real fitness value and depends on the quality of the surrogate model. In order to have a more precise results, EDA-CNN-Surrogate approach is proposed that combines both previous evaluation methods, CNN training evaluations and surrogate predictions. This approach provides a trade-off between the precision of training a CNN and the required computational cost.

Algorithm changes the evaluation method in comparison to previous ones. The evaluation method involve predicting each individual fitness value using surrogate based evaluation and comparing each prediction with a percentage, a threshold, which is calculated at each generation. As it is a minimizing problem, if surrogate based evaluation is below the threshold, it will be considered a promising solution, replacing its evaluation with a CNN training, to achieve a more precise evaluation of the promising solutions. This allow to prevent from evaluating with a CNN no promising solutions, reducing therefore computational costs. At first generation, surrogate based evaluation is applied since there are no other previous individuals evaluated to compare with.

The threshold indicates the 30th quantile of the fitness values on previous generation, being this value the point at which 30% percent of the data fall below that value. If the surrogate based evaluation is below this value, it is classified as a promising solution, since it is below the 30% percent of previous generation fitness values.

Algorithm 4 displays the pseudo-code of EDA-CNN-Surrogate approach.

Moreover, based on EDA-CNN-Surrogate approach, another criterion have been compared, which stands on reducing the threshold as generations go by, being more critical on classifying a solution as promising. Initial threshold is quantile 30 and it is reduced to quantile 5. As generations goes by, solutions convergence towards local minima, so more solutions may result promising. That is why a more critical threshold is applied in higher generations, reducing more required computational resources and costs.

Algorithm 4 EDA-CNN-Surrogate

$O \leftarrow$ Run initial solutions
Build surrogate model $g(O)$
$M_0 \leftarrow (S_0, \theta_0)$
$D_0 \leftarrow$ Sample N individuals from M_0
for $l = 1, 2, \ldots$ until a stop criterion is met **do**
    $D_{l-1}^{Se} \leftarrow$ Select S_e individuals from D_{l-1} depending on F_{l-1}
    $S'_l \leftarrow$ Find the structure which maximizes $BIC(S_l, D_{l-1}^{Se})$
    $\theta^l \leftarrow$ Calculate $\{\theta_{ijk} = \frac{N_{ijk}^l + 1}{N_{ij}^l + r_i}\}$ using D_{l-1}^{Se} as the data set
    $M_l \leftarrow (S'_l, \theta^l)$
    $D_l \leftarrow$ Sample N individuals from M_l using PLS
    $F_l \leftarrow g(D_l)$ surrogate based evaluations
    if l > 0 **then**
        $q30 \leftarrow quantile30(F_{l-1})$ Get previous generation fitness quantile 30
        for individual D_i in D_l **do**
            if F_l < q30 **then**
                $F_i \leftarrow f(D_i)$ CNN training
            end if
        end for
    end if
end for

3.3.2 Surrogate model

An initial population is essential so as to build the surrogate model. Given the solutions and its fitness values, surrogate model can be trained to understand the relationships between the predictive variables which include the hyper-parameters encoded in the solution and the target variable, this is the fitness value achieved after the CNN training of each solution.

Surrogate model precision highly depends on this initial population, so a diverse initial population is interesting in order to cover the objective search space, providing more predictive capabilities to the surrogate model. If initial population does not cover all search space, model predictions may be poor approximations of the fitness value. If the solution space is not too big, it is more likely to cover the entire search space and build a precise and efficient surrogate model. However, in problems in which the solution space is huge, initial population would not cover the entire search space due to the computational costs required for these initial evaluations. Therefore, surrogate model efficiency would decrease, reducing the fitness approximation capabilities. Mentioned EDA-CNN-Surrogate approach tackle the mentioned situation, using surrogate model to detect the promising solutions and then training CNN to replace the surrogate approximation with the training results of the CNN.

Different models have been trained to output fitness value given the hyper-parameters of the solution: Gaussian Process Regressor, Decision Tree Regressor and Random Forest Regressor. Implementing grid search to optimize hyper-parameter of these models trying to reduce the mean squared error between predicted and real fitness values, Random Forest Regressor shows more robust and efficient performance, achieving, in addition, the minimum mean squared error between real and predicted

values.

A Random Forest Regressor is an ensemble learning method that merges numerous decision trees to produce a single outcome. It operates by constructing multiple decision trees during training and outputting the average prediction of the individual trees. In Random Forest, bootstrapping is a fundamental technique used to create multiple subsets of the original dataset. Each bootstrap sample is used to train an individual decision tree within the Random Forest. The use of bootstrapping in Random Forest introduces diversity among the decision trees because each tree is trained on a different subset of data, providing effectiveness of the ensemble method. When predictions are made, it aggregates the predictions from all the individual trees, typically by averaging them in regression tasks, to produce a final, more accurate and stable prediction.

Surrogate Random Forest model contains 50 decision trees with 15 leaves depth, achieving a root mean squared error of 0.2159. The more important features according to regressor model are the learning rate, the amount of convolutional layers, the optimizer and the addition of residual blocks.

This Random Forest surrogate model is implemented in EDA-Surrogate, EDA-CNN-Surrogate and EDA-CNN-Surrogate with threshold decay approaches.

3.3.3 Multi-objective EDA-CNN

Multi-objective EDA-CNN approach aims to optimize hyper-parameters so as to get a solution that minimizes its cross entropy loss in CIFAR-10 classification task while taking into consideration required computational costs, trying to minimize it. In this situation, there is no a single solution that is best with respect to all objectives. Instead, a set of trade-off solutions exists, where improving one objective would lead to a deterioration in the other objective.

Therefore, the result of this approach consist of a set of solutions known as non-dominated solutions. A solution is considered non-dominated if there is no other solution that is better in at least one objective without being worse in another. This set of all non-dominated solutions is known as the Pareto front and represents the best trade-off between the objectives.

Multi-objective EDA-CNN combines non-dominating sorting and crowding distance approaches to select the individuals of next generation. Non-dominating sorting is used to sort the solutions in population according to the Pareto dominance principle. Non-dominated sorting first selects all the non-dominated solutions from population and assigns them to first Pareto front; then, it selects all the non-dominated solutions from the remaining solutions and assigns them to second Pareto front and it repeats the above process until all individuals have been assigned to a front. Crowding distance, on the other hand, measures the density of a solution in the objective space calculating the average distance of its two neighboring solutions. During the selection process, solutions with lower Pareto front assignation and larger crowding distances are preferred in order to maintain good and dispersed solutions, promoting both intensity and diversity.

Taking both techniques into consideration, multi-objective EDA-CNN approach uses EBNA algorithm trying to minimize the CNN training loss function, penalizing solu-

tions by the corresponding Pareto front assigned. The more dominated solutions, this is, the solutions in last Pareto front, are more penalized, whereas, non-dominated solution fitness values do not have any negative effect. For this purpose, the value of the Pareto front to which a solution is assigned is added to the fitness value of the individual. Added values are normalised within a range established between the minimum and maximum fitness value so that penalizations do not affect more than the fitness value of solutions.

This approach allow the decision maker to analyse all non-dominated solutions and establish its preferences between the fitness or computational cost objectives. Taking fitness value as a preference, the non-dominated solution that minimizes the individual fitness value would be considered, whereas prioritizing computational cost, the non-dominated solution that minimizes the computation costs would be considered. A promising solution also would be a non-dominated solution that provides a trade-off between both objectives.

3.4 EDA-DCGAN

3.4.1 Deep Convolutional Generative Adversarial Networks

Deep Convolutional Generative Adversarial Networks (DCGANs) are an advanced type of GAN that incorporate CNNs to improve the quality and stability of generated images. GANs consist of two neural networks, a generator and a discriminator, which compete against each other in a zero-sum game. The generator creates fake images from random noise, while the discriminator attempts to distinguish between real and fake images. In a DCGAN, both the generator and discriminator are built using deep convolutional layers, which allows the model to effectively capture spatial hierarchies.

The generator network employs transposed convolutions to upsample the input noise into a full-sized image, while the discriminator uses standard convolutions to classify the images as real or fake. Latent space refers to a lower-dimensional vector space from which the generator network creates new data samples.

This architecture enhances the generator's ability to produce high-resolution and realistic images, and the discriminator's capacity to effectively distinguish between real and generated images.

3.4.2 Hyper-parameter Optimization

Hyper-parameter optimization has been applied to the DCGAN architecture for generating MNIST images with EDAs, aiming to produce highly accurate and visually convincing images of handwritten digits from the MNIST dataset, demonstrating the effectiveness of hyper-parameter optimization in enhancing the generative capabilities of the model. Dataset contains 60000 training images and 10000 test images, where each image height and width are 28×28 pixels and a unique channel.

Taking into consideration generated images must have a determined size, 28x28 with one channel in MNIST images, there are combinations of hyper-parameters that construct a DCGAN that does not generate desired size images. In order to deal with this limitation, an architecture has been established for both networks, not allowing to change the topology and the depth of both networks. Moreover, generator and dis-

criminator networks are designed to have similar or mirrored architectures, resulting on a symmetric DCGAN.

The architecture of the discriminator includes two convolutional layers, incorporating the batch normalization technique to normalize the inputs to each layer. This normalization helps to stabilize and accelerate the training process. Additionally, the architecture employs the LeakyReLU non-linear activation function after each layer. Fully-connected layer of the discriminator also contains 2 layers.

Generator network starts with a vector of random noise which is passed through a fully connected layer, followed by a reshape operation to form an initial feature map. This helps in transitioning from the noise vector to a more structured representation. Then, two transposed convolutional layers are applied, performing upsampling, gradually increasing the spatial dimensions of the feature maps while reducing the number of channels. Batch normalization and LeakyReLU activation functions are applied after each transposed layer.

Model weights of both networks are updated minimizing binary cross entropy with logits loss, since both perform binary classification tasks. It combines the sigmoid activation function and the binary cross-entropy loss into a single function, making it more efficient and numerically stable.

Therefore, the hyper-parameters to optimize include amount of kernels of two convolutional and transposed convolutional layers, kernel size, strides, latent dimension size, discriminator learning rate, generator learning rate, batch size and optimizer, all encoded using explained encoding scheme.

Chapter 4

Results

This section presents the results of implementing EDAs for hyper-parameter optimization in two tasks: image classification using CNNs on the CIFAR-10 dataset, and image generation using DCGANs on MNIST digits dataset.

The experiments were conducted using the EDAspy Python library (Soloviev et al., 2024). In order to run neural networks to evaluate each individual in population, a high-performance graphics processing unit, NVIDIA A100-PCIE-40GB GPU, has been used which provides exceptional processing power and efficiency in handling large-scale computations.

4.1 EDA-CNN approaches

This section provides the results of the different training approaches mentioned in previous sections: EDA-CNN, EDA-Surrogate, EDA-CNN-Surrogate and multi-objective EDA-CNN.

All approaches have in common that as generations goes by, the amount of duplicated solutions increases. Figure 4.1 illustrates the number of duplicated individuals in each generation, being the population size of 50 individuals. The analysis shows that in the 4th generation, duplicated solutions already exist, and in the 7th generation, more than half of the population is duplicated.

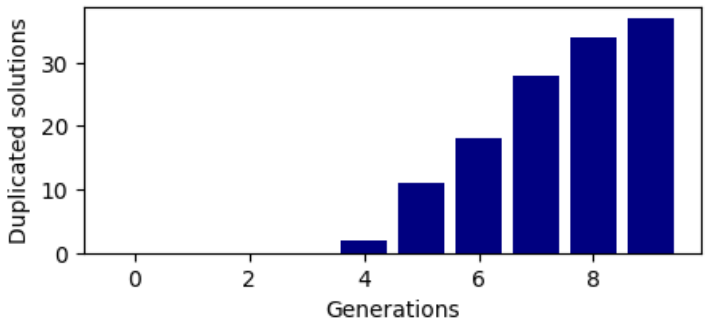


Figure 4.1: *EDA-CNN: amount of duplicated individuals (copies) in each generation.*

Taking this analysis into consideration, termination criteria of the different training

approaches is established to the 6th generation, limiting the amount of duplicated individuals to the half of the population.

Therefore, the convergence and the computational costs each approach require is analysed and visualized up to the 6th generation. Cross-entropy loss in validation data set is visualized of each individual in a generation to analyze the convergence of the algorithm. For computational cost analysis, mean execution time of each generation in seconds is visualized. Moreover, computational costs interquartile range is also displayed to show the spread of the middle half of the distribution.

4.1.1 EDA-CNN evaluation

This approach require a CNN training for all individuals, so taking into account each generation include 50 individuals, the algorithm consists of 300 CNN trainings and the evolutionary procedure.

Figure 4.2 shows the convergence of the algorithm, reaching a population of promising solutions that minimize the cross-entropy loss in the validation set. Analysing each generation boxplot, this algorithm seems to have already convergence in 4th and 5th generations. Although the solutions seems to be better as generations go by, it has been analysed that from 4th generation on, the amount of duplicated solutions increase, taking as a conclusion that this algorithm requires around 4 or 5 generations to convergence to local minimum.

However, the mean execution time is high, requiring 1000 seconds by mean to train an individual. In total, each generation requires 50000 seconds to evaluate the 50 individuals, around 14 hours of execution time per generation. Therefore, whole algorithm execution is around 85 hours.

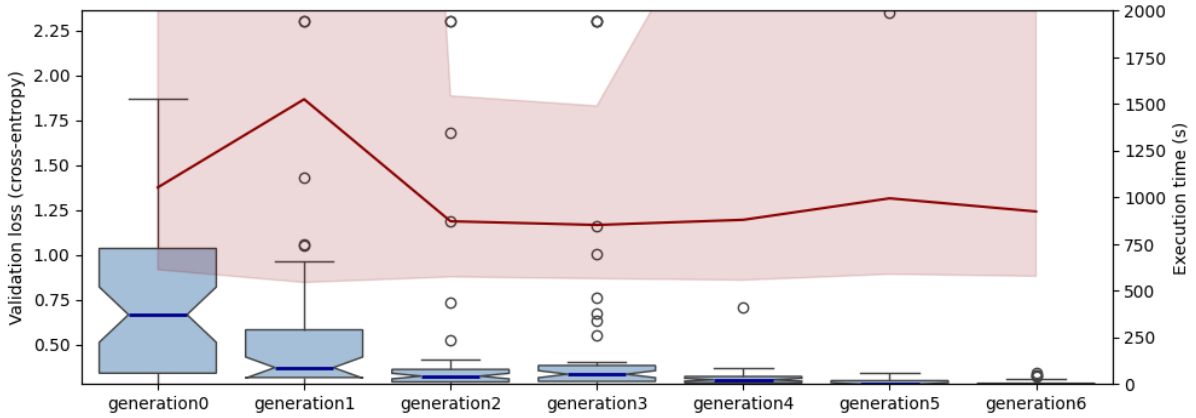


Figure 4.2: *EDA-CNN algorithm performance: convergence and computational cost.*

The algorithm outputs the hyper-parameter configuration that reaches the minimum validation cross-entropy. The behaviour, including the cross-entropy loss and the accuracy, of the best achieved CNN training is analysed during 50 epochs in training and validation sets in Figure 4.3. Optimized CNN reaches 99.9% of accuracy and 0.005 cross-entropy loss in CIFAR-10 training set, and 93.6% of accuracy and a cross-entropy loss of 0.263 in validation set. Therefore, there is low bias and high variance, since algorithm perform better in training set than in validation set, overfitting the training data.

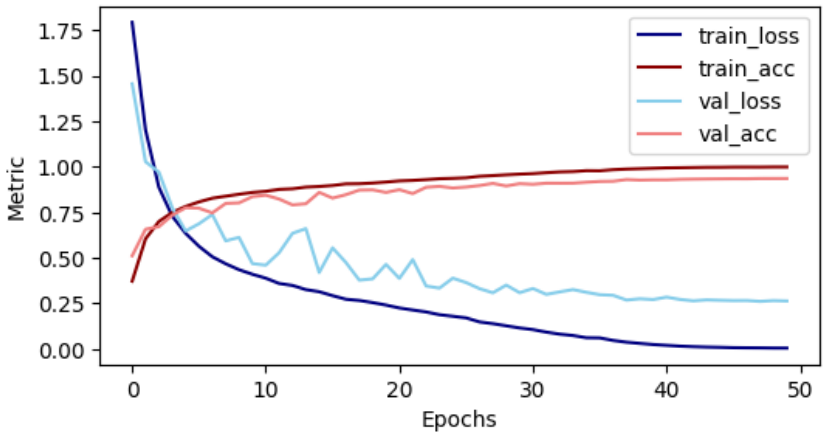


Figure 4.3: *Optimized EDA-CNN algorithm performance: loss and accuracy.*

4.1.2 EDA-Surrogate model evaluation

EDA-Surrogate approach, needs more generations to reach a convergence comparing to previous EDA-CNN algorithm, since 2nd and 3th generations do not improve previous generations. 4th and subsequent generations, however, enhance population including individuals with lower cross-entropy loss. This performance is illustrated in Figure 4.4.

Yet algorithm performance improves EDA-CNN mainly in the required execution time. The scale of the execution time in the Figure 4.4 goes from 0 to 0.2 seconds, what indicates that the evaluation of an individual does not exceed the 0.2 seconds. Each generation execution time, therefore, is by mean of 1.45 seconds, needing for whole algorithm 10.2 seconds.

Nevertheless, this approach requires additional time to train the initial samples used for surrogate model training. The more initial samples, the more likely it is to train a surrogate model that approximates better the fitness function. But taking into consideration each individual by mean needs 15 minutes to be trained, a trade-off must be found in order to balance the quality and computational cost of the surrogate model. For instance, training 100 individuals for the surrogate model training, the total computation cost of the algorithm increase 24 hours approximately,

On the other hand, since all the evaluations in this approach consist of surrogate evaluations, fitness value of individuals are approximations of the cross-entropy loss given the hyper-parameter configuration. Therefore, the output result given by the algorithm, this is, the best achieved hyper-parameter configuration, is evaluated with the CNN training, whose performance is shown in Figure 4.5. This model also overfits the training data, since it reaches a 99.10% of accuracy and 0.028 cross-entropy loss in the training data set, while a 92.00% of accuracy and 0.337 cross-entropy loss in the validation set.

4.1.3 EDA-CNN-Surrogate model evaluation

Combining surrogate and CNN evaluation methods, algorithm achieves an improvement in each generation, converging to a local optimum while reducing computational costs in comparison with EDA-CNN approach.

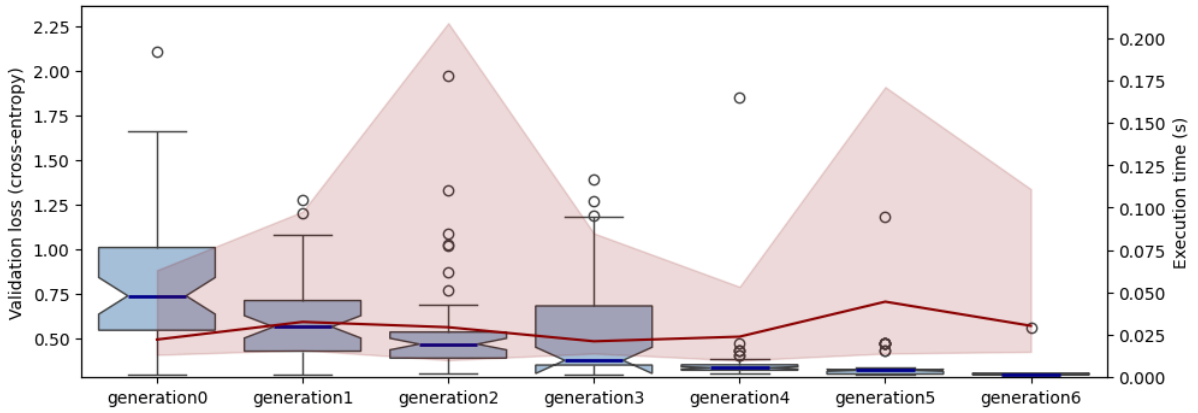


Figure 4.4: *EDA-Surrogate algorithm performance: convergence and computational cost.*

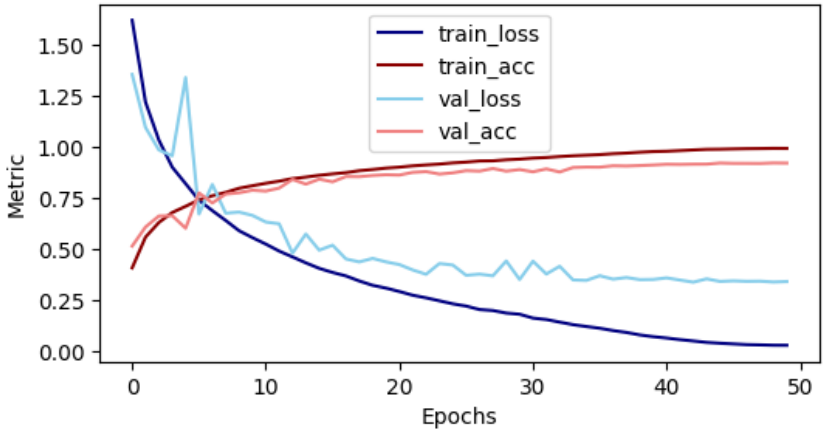


Figure 4.5: *Optimized EDA-Surrogate algorithm performance: loss and accuracy.*

Taking the percentile 30 as the threshold to determine the promising solutions for the CNN evaluation method, the performance of the algorithm, Figure 4.6, illustrates that solutions in 5th generation have already converged. The best solution reaches after 50 epochs an accuracy of 0.997 in training data and 0.936 in validation data and a cross-entropy loss of 0.010 in training data and 0.274 in validation data.

Related to the execution time, as in each generation individuals are evaluated with both evaluation methods, the mean by generation decrease to 365 seconds approximately. Analysing the interquartile range of the execution time to know the spread of the middle half of the distribution (Figure 4.6), the execution time reduces over the generations, indicating there are less solutions with a fitness lower than the fitness corresponding to the established threshold, this is, there are less promising solutions. Early generations contain more promising solutions since the algorithm has not already converged and the fitness threshold is high. However, as solutions improve in each generation, the threshold decrease, making it more difficult to achieve the promising solutions capable of reducing the threshold.

Each generation, containing 50 individuals, require from 12371 to 36766 seconds, this is, from 3.5 hours to 10 hours. In total, the whole algorithm lasts 127787 sec-

onds, 35 hours. Nevertheless, as surrogate model is used for solution evaluation, the surrogate model training computational cost must also be taken into consideration. Therefore, 24 hours must be added to the 35 hours, requiring in total 59 hours.

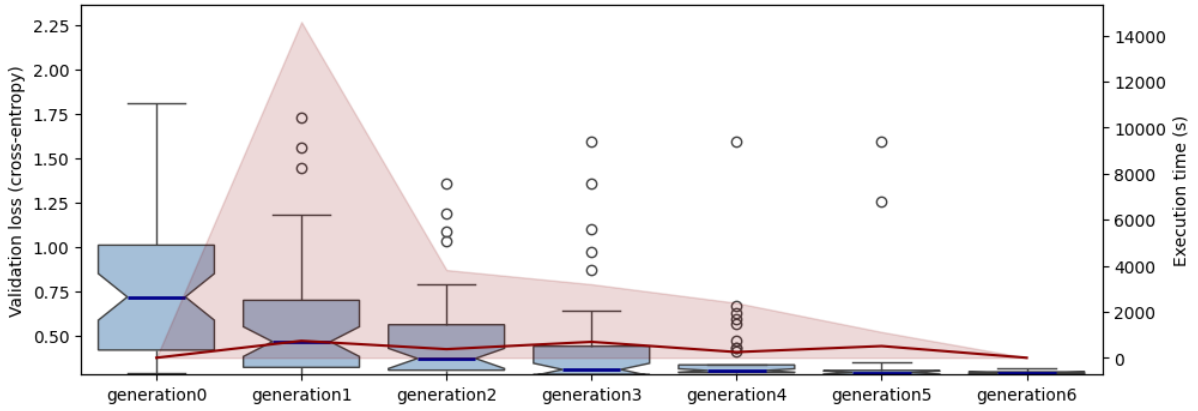


Figure 4.6: *EDA-CNN-Surrogate algorithm performance: convergence and computational cost.*

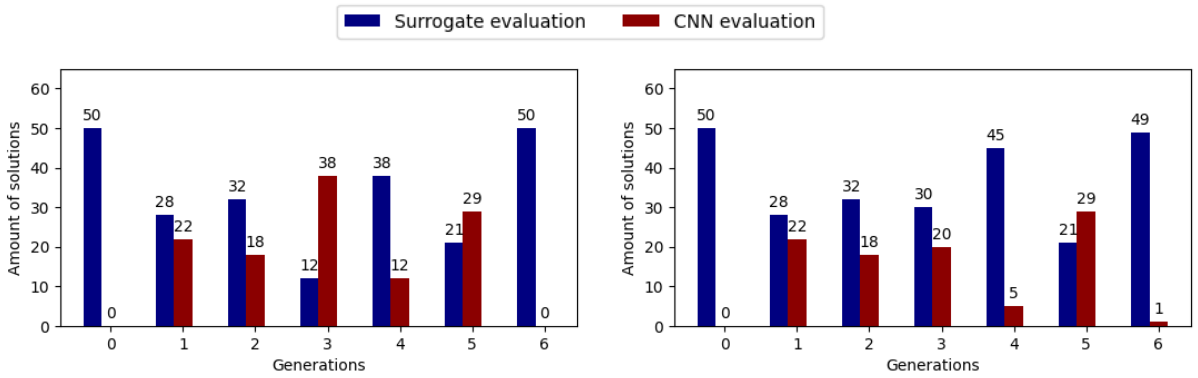
Related to the proposed approach of reducing the threshold as generations goes by in order to be more selective in determining the promising solutions, the algorithm convergence seems similar while reducing even more the computational cost. Achieved best individual reaches an accuracy of 0.983 and 0.929 and a cross-entropy loss of 0.052 and 0.250 in training and validation data respectively.

Evaluation time of an individual in EDA-CNN-Surrogate with dynamic decaying threshold goes from 80 to 750 seconds by mean in each generation. First generation, for instance, requires in total 37128 seconds (10 hours) to evaluate all individuals, whereas 4th generation requires 4025 seconds (1.1 hours) since almost all individuals are evaluated with the surrogate model without being classified as promising solutions. Total execution time of all generations is approximately of 102571 seconds, this is, of 28 hours. Adding the training execution time of the surrogate model, the total time reaches 52 hours.

In order to analyse deeply the evaluation methods in each generation taking into consideration the different approaches of establishing the threshold, this is fixed or dynamic threshold, Figure 4.7 illustrates the amount of individuals that have been evaluated with the corresponding evaluation method in the two approaches.

Decaying the threshold reduces the amount of individuals that are evaluated with CNN training method. In 3rd generation, for instance, a fixed threshold classify 76% of individuals as promising, running a CNN to evaluate the individual. However, a dynamic threshold reduces the percentage to a 60%, which in this case consists of 30 individuals. In 4th generation the promising solution percentage also reduces from 24% to 10%.

Taken the optimized configuration of hyper-parameters of EDA-CNN-Surrogate with dynamic threshold approach, accuracy metric and loss function of the CNN training is shown in Figure 4.8 during the training epochs, finally achieving the already mentioned results, this is, 92.90% of accuracy in the validation set. As already analyzed optimized models from other approaches, an over-fitting is analysed comparing train



(a) Fixed threshold (b) Dynamic threshold

Figure 4.7: *Individual evaluation methods in (a) fixed threshold and (b) dynamic threshold EDA-CNN-Surrogate approaches.*

and validation sets.

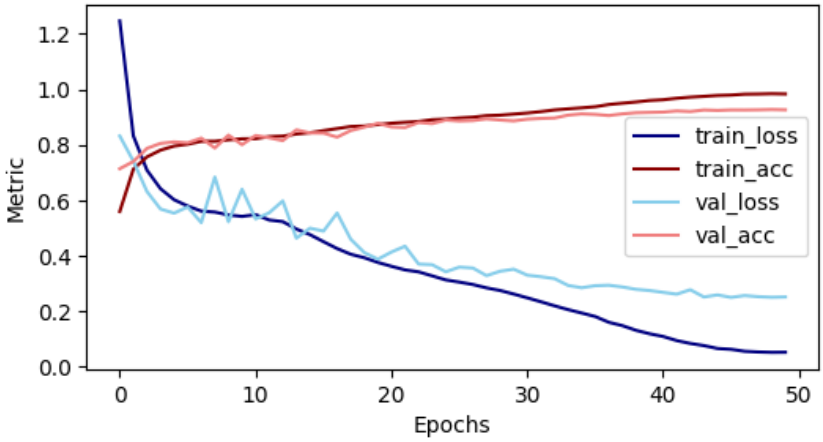


Figure 4.8: *Optimized EDA-CNN-Surrogate algorithm performance with decaying threshold approach: loss and accuracy.*

Furthermore, as EBNA algorithm evolve population by sampling from a learnt Bayesian Network, the EDA-CNN-Surrogate algorithm also returns the learnt Bayesian Network and the conditional independencies between variables, in this case, between the CNN hyper-parameters.

On the other hand, a fixed population size has been determined to carry the experiments out, however, the behaviour of the EDA-CNN-Surrogate algorithm has also been tested with different population sizes in order to determine the minimum amount of individuals needed to achieve a good behaviour and, therefore to reduce the computational cost.

Figure 4.9 shows both cross-entropy validation loss and computation cost with 20, 30, 40 and 50 individuals in each generation. It is remarkable the greater the amount of individuals, the faster algorithm reaches a convergence, achieving moreover, a population including solutions with lower fitness values taking into consideration the loss function. 20 and 30 individuals in a generation, however, need more generations

to converge.

Related to execution time needed in each generation, 50 individuals require more time to evaluate all the solutions, as it is reflected in first generation. However, the amount of CNN training evaluations highly depends on the amount of promising classified solutions, which consists of solutions lower than the calculated fitness threshold. With 50 individuals, the search space may be more covered, and more interesting spaces can be reached more easily, being the threshold of 30% of the fitness value lower due to the quality of individuals. This way, individuals in following generation need to be lower than this fitness threshold in order to be classified as promising and since this threshold may be lower due to the good quality of solutions in previous generation, less individuals may be classified as promising, resulting on less CNN trainings.

Therefore, although more individuals require more evaluations, a larger population size does not require a proportional computational cost increase, since surrogate model evaluation helps to determine those promising solutions and use the CNN evaluation method only in promising solutions, avoiding CNN training of poorer hyperparameter configurations.

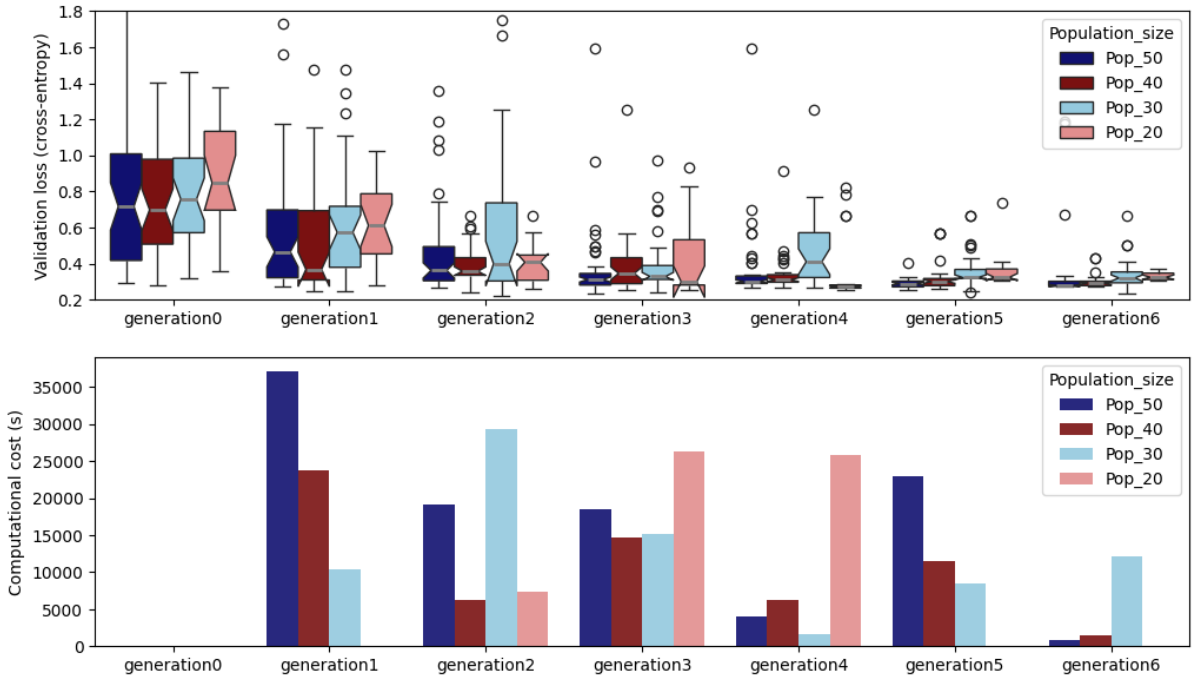


Figure 4.9: *Comparison of cross-entropy loss and computational cost with different population sizes.*

Analysing the numerical results in Table 4.1, it is remarkable the EDA-CNN-Surrogate algorithm with 30 individuals reach a validation accuracy of 94.60%, reaching the best found local optima. However, in previous figure is analysed that population size of 30 do not convergence as well as a population size of 50 or 40. A population size of 40 come up with a similar solution to a population size of 50 when it comes to the metrics, nevertheless, it requires 10 hours less in this optimization problem task. Additionally, the surrogate model training time must be added to all the approaches showed in the table, which is the same in all cases.

Model	Train		Validation		Computational cost
	Loss	Accuracy	Loss	Accuracy	
Population 50	0.010	0.997	0.274	0.936	28.5h
Population 40	0.003	0.999	0.305	0.938	17.5h
Population 30	0.001	0.999	0.278	0.946	21.3h
Population 20	0.032	0.990	0.288	0.928	16.5h

Table 4.1: Training and validation metrics and computational cost for EDA-CNN-Surrogate algorithm with different population sizes.

4.1.4 Multi-objective EDA-CNN

Multi-objective EDA-CNN evaluates each individual with CNN training, since although a surrogate model that approximates the loss function is already trained, there is no model that approximates the second objective, the computational cost.

Multi-objective EDA-CNN optimization process is carried out with 50 individuals in each generation, as in previous approaches to enable the comparison between approaches. However, in order to visualize the evolution of the solutions and the different fronts they form, an execution of 20 individuals is implemented.

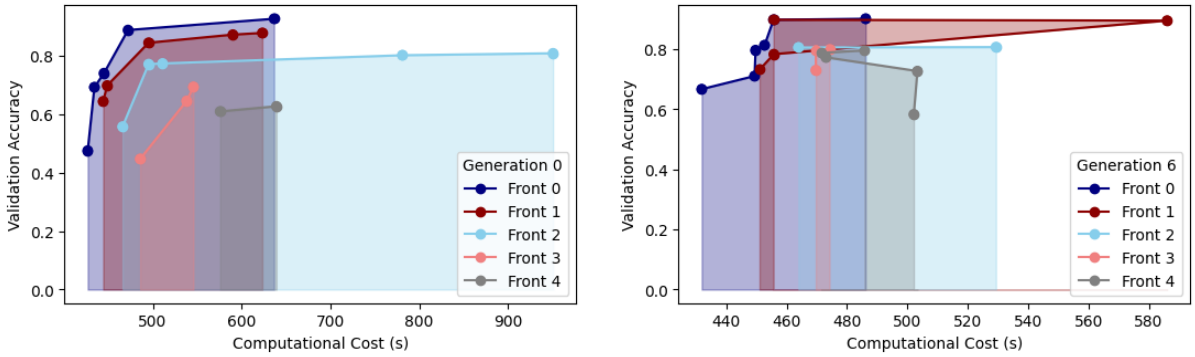
Figure 4.10 shows the difference between the 20 individuals of the first generation and the last generation in the different objectives. Although objectives consist of minimizing both individual validation cross-entropy loss and computational cost, Figure 4.10 shows the validation accuracy instead of the loss to visualize and interpret better the results. Since the objective is to reach a high validation accuracy metric with low computational cost, the optimal solution would be situated in top left position. Figure 4.10 illustrates all the individuals on the generation, being the non-dominated solutions the ones corresponding to front 0.

Individuals in the initial generation, Figure 4.10a, exhibit a wide range of computational costs, with many individuals showing low validation accuracy metric. This is typical in early stages where the algorithm is exploring a broad solution space.

Compared to the first generation, the last generation, Figure 4.10b, shows solutions with improved validation accuracy, since multiple individuals overcome 92% of accuracy and all individuals are above 60% of accuracy. The computational costs are generally lower, the range goes from 440 to 580 seconds, while in first generation reaches 900 seconds. The result, therefore, indicates a more efficient set of solutions.

Behaviour with a population size of 50 is similar. The computational cost ranges from 500 to 7000 in first generation and from 400 to 1200 in last generation. The quality of the solutions also increase as generations go by.

Taking the non-dominated solutions from the set of solutions of the last generation with a population size of 50 individuals, there is a possibility to establish the preferences between the objectives and determine the configuration of hyper-parameters that best fits the preferences.



(a) First generation

(b) Last generation

Figure 4.10: (a) First generation. (b) Last generation. Pareto fronts.

Establishing the preference on the solution that maximizes the validation accuracy, the best achieved solution with a population size of 50 reaches an accuracy of 0.998 and 0.940 and cross-entropy loss of 0.004 and 0.314 in training and validation sets. Therefore, multi-objective algorithm also provides hyper-parameter configurations of high quality. With 20 individuals, however, it reaches an accuracy of 0.998 and 0.938 and loss of 0.005 and 0.339 in training and validation sets respectively.

Related to computational cost of the algorithm with a population size of 50, each generation needs by mean 36030 seconds, this is 10 hours. In total, algorithm requires 259816 seconds approximately, 72 hours.

With a population size of 20 individuals, however, each generation needs by mean 11000 seconds, this is 3 hours. In total, algorithm requires 76080 seconds approximately, 21 hours. This approach is faster than the other mentioned approaches, since the population size is 66% lower.

4.1.5 Comparisons

Comparing the computational cost of EDA-CNN, EDA-Surrogate and EDA-CNN-Surrogate with fixed and dynamic thresholds all with a population size of 50, Figure 4.11 clearly illustrates how surrogate model reduces the computational cost of the algorithm in each generation. It is also analysed that decaying the threshold as generations go by, it reduces the computational cost, being more selective in determining the promising solutions.

Table 4.2 summarizes the previous 4 approaches and the multi-objective CNN-EDA results. EDA-Surrogate total computational cost is reduced in 71.76% in comparison with EDA-CNN. The combined method, EDA-CNN-Surrogate reduces in 30.59%, however, it provides a more precise results by training CNN with promising solutions. Moreover, reducing the threshold in generations reaches a reduction of a 38.82%. In multi-objective approach, as there is no surrogate model that approximates both objectives, the computation cost is high, about 72 hours.

Related to the achieved results, EDA-CNN-Surrogate with a fixed threshold provides interesting results, achieving 93.60% of accuracy in the validation set. Moreover, although it is not reflected in the table, EDA-CNN-Surrogate with a dynamic threshold and a population size of 30 has reach the best local optimum, reaching 94.60% of

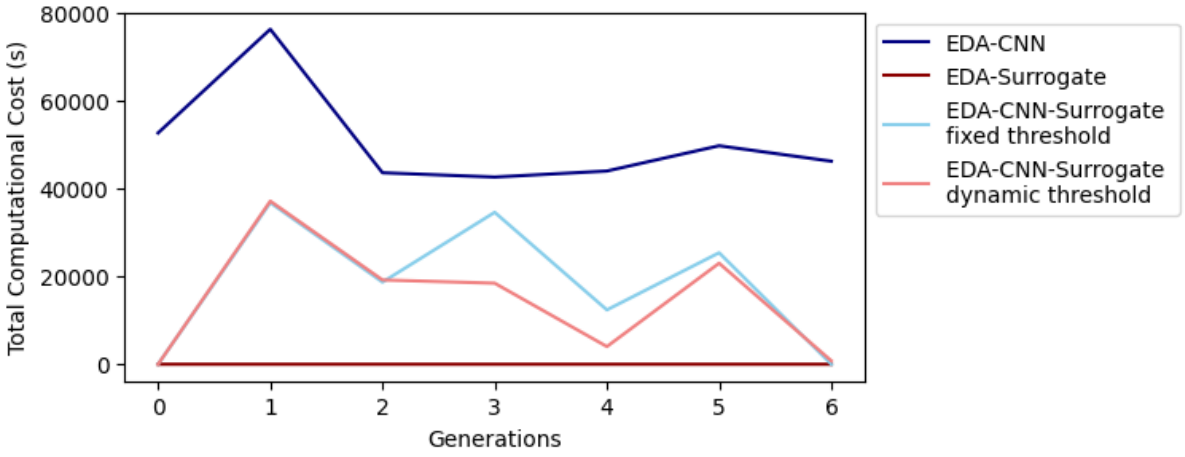


Figure 4.11: *Comparison between differente approaches: computational cost.*

accuracy and reducing the computational cost to 45 hours. Multi-objective approach, in addition, reaches 94.00% of accuracy.

Model	**Train**		**Validation**		**Computational cost**
	Loss	**Accuracy**	**Loss**	**Accuracy**	
EDA-CNN	0.005	0.999	0.263	0.936	85h
EDA-Surrogate	0.028	0.991	0.337	0.920	24h
EDA-CNN-Surrogate fixed threshold	0.010	0.997	0.274	0.936	59h
EDA-CNN-Surrogate dynamic threshold	0.052	0.983	0.250	0.929	52h
Multi-objective EDA-CNN	0.004	0.998	0.314	0.940	72h

Table 4.2: Performance metrics and computational cost of different analysed approaches with a population size of 50 individuals.

Taking the best achieved result, EDA-CNN-Surrogate with dynamic threshold and with a population size of 30 individuals, which reaches a 94.96% of accuracy in validation set in CIFAR-10 classification task, it is compared to other state of the art algorithm results in Table 4.3.

CNN-GA and SHEDA algorithms results to be better algorithms since they reach a validation accuracy greater than 96.00%. Nevertheless, EDA-CNN-Surrogate model improves the manually designed CNNs and other evolutionary computation based CNNs. Moreover, the best hyper-parameter configuration of EDA-CNN-Surrogate algorithm consists of 10 convolutional layers with two residual blocks and it is able to improve manual ResNet with a depth of 20 and 100 layers.

Method	Model	Accuracy %
Manually designed CNNs	ResNet(depth = 20) (He et al., 2016)	91.25
	ReaNet(depth = 110)(He et al., 2016)	93.17
EC based CNNs	Genetic CNN (Xie and Yuille, 2017)	92.90
	CNN-GA (Sun et al., 2019)	96.78
	SHEDA (Li et al., 2021)	96.36
Proposed EC algorithm	EDA-CNN-Surrogate	94.64

Table 4.3: Comparison between best achieved EDA-CNN-Surrogate model and other state of the art algorithms.

4.2 EDA-DCGAN

Analysing EDA-DCGAN evolution during the training, Figure 4.12 shows the evolution of the FID score in each generation. Minimum, maximum, median and the interquartile range of FID score of a population is visualized. There is a significant improvement until 4th generation, where it seems to already have converged, since the median reduces significantly. However, 7th generation is remarkable since the interquartile range is around median, meaning the half of the population is around the median. The reason can be based on the amount of duplicated solutions in the generation. As shown in Figure 4.13, in 7th generation the amount of duplicated solutions reach almost the half of the population, thus explaining the behaviour of the interquartile range in 7th generation.

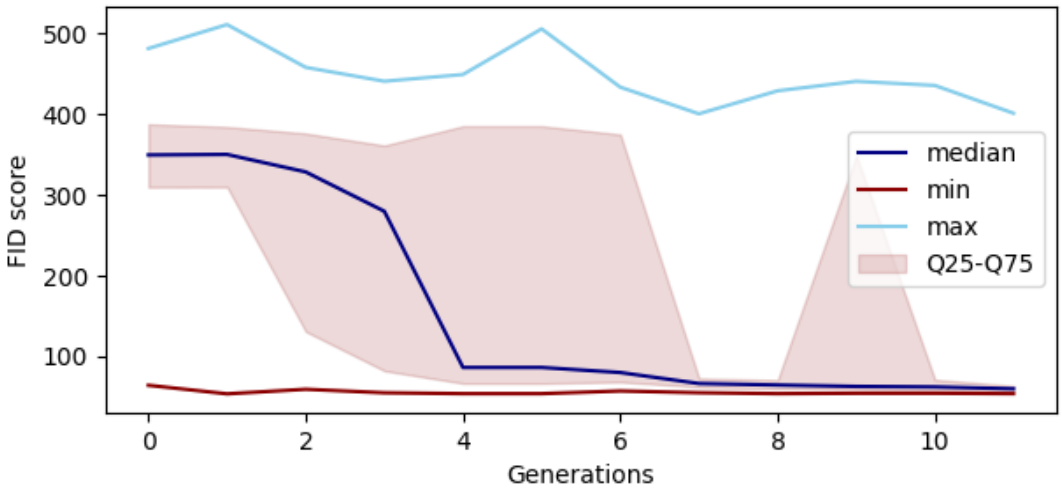


Figure 4.12: *EDA-DCGAN algorithm performance.*

The hyper-parameter configuration that lower FID score has achieved in the algorithm execution achieves a 52.99 FID score in 754 seconds of execution time. This DCGAN generates the images illustrated in Figure 4.14b.

Although generated images may be improved generating more complex DCGANs, the topology of the algorithm is limited to an amount of convolutional layers due to the

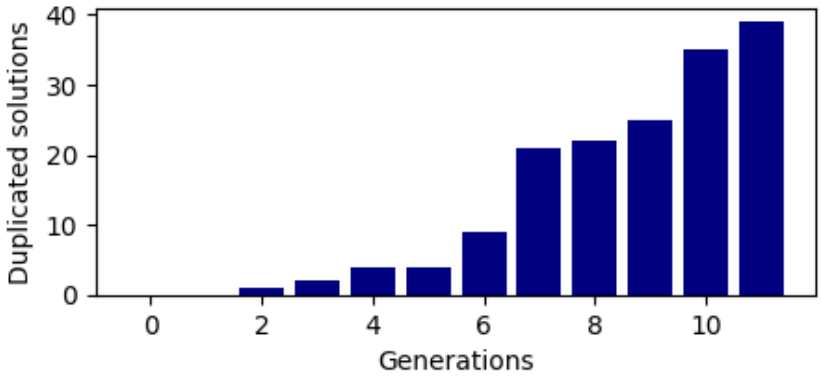
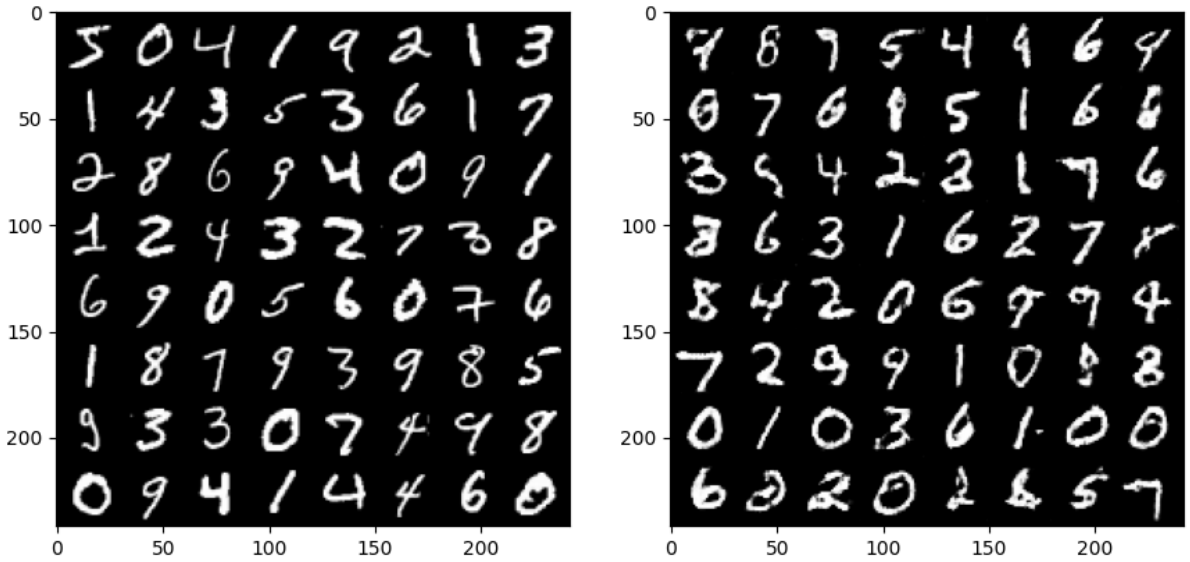


Figure 4.13: *EDA-DCGAN duplicated solutions in each generation.*



(a) Original images

(b) Generated images.

Figure 4.14: (a) Original images. (b) Generated images.

required computational costs and available resources.

Nevertheless, the performance of the EDA-DCGAN algorithm proves the EDAs help to find the optimum hyper-parameters for the DCGAN network layers.

Chapter 5

Conclusions and future lines

5.1 Conclusions

In order to implement a CNN hyper-parameter optimization on CIFAR-10 classification task, EDAs have resulted on a successful behaviour in looking for a optimum solution. EBNA estimation of distribution algorithm is implemented which learns a Bayesian network allowing dependencies between the hyper-parameters to sample from.

Moreover, the presented encoding scheme allows not only to optimize the hyper-parameters of a basic CNN, but also to look for the most promising depth and topology of the network. It allows to create a CNN from 2 to 10 convolutional layers, and it also allows to add residual blocks that result successful in CNN classification tasks.

With EBNA algorithm and the proposed encoding scheme, different approaches are compared: EDA-CNN, EDA-Surrogate, EDA-CNN-Surrogate with a fixed and dynamic threshold and multi-objective EDA-CNN, last one providing the possibility to establish the preferences between quality and computational cost objectives.

Regarding to the best found hyper-parameter configurations reached by mentioned approaches with a population size of 50 individuals, the quality metrics ranges from 92.00% to 94.00% of accuracy in the validation set.

EDA-CNN requires high computational cost. CNN-Surrogate model, however, reduces the computational cost in 71.76%. However, the results and fitness evaluations are approximations of the quality of the models. EDA-CNN-Surrogate approach, presents more precise results since promising solutions are evaluated with CNNs and the computational cost reduces in 30% or 38%, depending on the criteria on selecting the promising solutions.

The best found local optima that reaches the highest accuracy in validation set is EDA-CNN-Surrogate with dynamic threshold and with a population size of 30, reaching 94.60% of accuracy.

Related to population size, although the approaches were compared with 50 individuals in the population, EDA-CNN-Surrogate approach is compared with 20, 30, 40 and 50 individuals. In CIFAR-10 classification task all population sizes reach solutions of high quality, in fact, the best solution achieved is optimized with EDA-CNN-Surrogate

with 30 individuals. Multi-objective approach is also analysed with 20 individuals, reaching a solution of high quality.

However, analysing convergence with different population sizes, the higher the population size the better and faster is the convergence. In more complex problems with a more complex and bigger search space, a higher population size would be needed in order to cover all the search space, taking also into consideration the computational cost.

Furthermore, a larger population size does not necessarily imply a higher computational cost, but depends largely on the quality of the individuals. The threshold determines which individuals should be evaluated with CNN training, so if there are many high quality individuals in a population, the threshold would be low, making it more difficult to come up with solutions that minimize this threshold.

On the other hand, EBNA algorithm is successful in hyper-parameter optimization of DCGANs for image generation tasks, however, since the proposed architecture is simple due the available resources and computational cost, the quality of the generated images may improve with more complex architectures.

In summary, the behaviour of EDAs converges to a local optimum, achieving a hyper-parameter configuration that maximises precision or accuracy in both image classification and image generation tasks.

5.2 Future lines

By applying parallelization to a population of CNNs in each generation, the efficiency and speed of the evolutionary process would significantly improve, as multiple solutions could be evaluated and optimized concurrently.

Another future line is to test the different approaches on more complex problems, such as CIFAR-100. Applying the approaches to datasets with higher complexity and more categories would help to evaluate the robustness and scalability of the techniques. This will help in understanding how well the approach performs under more demanding conditions and in more diverse scenarios, providing valuable insights for further improvement.

Another future direction is to test DCGANs with more complex architectures. Experimenting with advanced and deeper network structures, could lead to improve the model performance and, therefore, the quality of the generated images. This will help in understanding the capabilities and limitations of DCGANs when scaled up, providing valuable information for optimizing and enhancing their design.

Bibliography

Ossama Abdel-Hamid, Abdel-Rahman Mohamed, Hui Jiang, Li Deng, Gerald Penn, and Dong Yu. Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 22(10):1533–1545, 2014.

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2623–2631, 2019.

Nurshazlyn Mohd Aszemi and PDD Dominic. Hyperparameter optimization in convolutional neural network using genetic algorithms. *International Journal of Advanced Computer Science and Applications*, 10(6), 2019.

Shumeet Baluja. *Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning*. School of Computer Science, Carnegie Mellon University Pittsburgh, PA, 1994.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(2), 2012.

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in Neural Information Processing Systems*, 24, 2011.

James Bergstra, Brent Komer, Chris Eliasmith, Dan Yamins, and David D Cox. Hyperopt: a python library for model selection and hyperparameter optimization. *Computational Science & Discovery*, 8(1):014008, 2015.

Hans-Georg Beyer and Hans-Paul Schwefel. Evolution strategies–a comprehensive introduction. *Natural Computing*, 1:3–52, 2002.

Peter AN Bosman and Dirk Thierens. Expanding from discrete to continuous estimation of distribution algorithms: the IDEA. In *International Conference on Parallel Problem Solving from Nature*, pages 767–776. Springer, 2000.

Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Efficient architecture search by network transformation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Junyi Chai, Hao Zeng, Anming Li, and Eric WT Ngai. Deep learning in computer vision: A critical review of emerging techniques and application scenarios. *Machine Learning with Applications*, 6:100134, 2021.

Xiangru Chen, Yanan Sun, Mengjie Zhang, and Dezhong Peng. Evolving deep convolutional variational autoencoders for image classification. *IEEE Transactions on Evolutionary Computation*, 25(5):815–829, 2020.

Tinkle Chugh, Karthik Sindhya, Kaisa Miettinen, Yaochu Jin, Tomas Kratky, and Pekka Makkonen. Surrogate-assisted evolutionary multiobjective shape optimization of an air intake ventilation system. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 1541–1548. IEEE, 2017.

Jeremy De Bonet, Charles Isbell, and Paul Viola. Mimic: Finding optima by estimating probability densities. *Advances in Neural Information Processing Systems*, 9, 1996.

Ivanoe De Falco, Giuseppe De Pietro, Antonio Della Cioppa, Giovanna Sannino, Umberto Scafuri, and Ernesto Tarantino. Evolution-based configuration optimization of a deep neural network for the classification of obstructive sleep apnea episodes. *Future Generation Computer Systems*, 98:377–391, 2019.

Kalyanmoy Deb and Himanshu Jain. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, Part I: solving problems with box constraints. *IEEE Transactions on Evolutionary Computation*, 18(4):577–601, 2013.

Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

Li Deng and Yang Liu. *Deep Learning in Natural Language Processing*. Springer, 2018.

Marco Dorigo and Luca Maria Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

C Erden. Genetic algorithm-based hyperparameter optimization of deep learning models for pm2. 5 time-series prediction. *International Journal of Environmental Science and Technology*, 20(3):2959–2982, 2023.

Andre Esteva, Katherine Chou, Serena Yeung, Nikhil Naik, Ali Madani, Ali Mottaghi, Yun Liu, Eric Topol, Jeff Dean, and Richard Socher. Deep learning-enabled medical computer vision. *NPJ digital medicine*, 4(1):5, 2021.

Ramon Etxeberria and Pedro Larrañaga. Global optimization using Bayesian networks. In *Second International Symposium on Artificial Intelligence*, pages 332–339, 1999.

Junfeng Gao, Yong Yang, Pan Lin, and Dong Sun Park. Computer vision in healthcare applications. *Journal of Healthcare Engineering*, 2018, 2018.

Xinyu Gong, Shiyu Chang, Yifan Jiang, and Zhangyang Wang. Autogan: Neural architecture search for generative adversarial networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3224–3234, 2019.

Georges R Harik, Fernando G Lobo, and David E Goldberg. The compact genetic algorithm. *IEEE Transactions on Evolutionary Computation*, 3(4):287–297, 1999.

Georges R Harik, Fernando G Lobo, and Kumara Sastry. Linkage learning via prob-

abilistic modeling in the extended compact genetic algorithm (ecga). In *Scalable Optimization Via Probabilistic Modeling*, pages 39–61. Springer, 2006.

Mahmoud Hassaballah and Ali Ismail Awad. *Deep Learning in Computer Vision: Principles and Applications*. CRC Press, 2020.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

David Heckerman, Dan Geiger, and David M Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20: 197–243, 1995.

Daniel Hernández-Lobato, Jose Hernandez-Lobato, Amar Shah, and Ryan Adams. Predictive entropy search for multi-objective Bayesian optimization. In *International Conference on Machine Learning*, pages 1492–1501. PMLR, 2016.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in Neural Information Processing Systems*, 30, 2017.

John H Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. The MIT press, 1992.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4700–4708, 2017.

Nathan R Huber, Andrew D Missert, Hao Gong, Scott S Hsieh, Shuai Leng, Lifeng Yu, and Cynthia H McCollough. Random search as a neural network optimization strategy for convolutional-neural-network (CNN)-based noise reduction in CT. In *Medical Imaging 2021: Image Processing*, volume 11596, pages 509–515. SPIE, 2021.

Sakshi Indolia, Anil Kumar Goswami, Surya Prakesh Mishra, and Pooja Asopa. Conceptual understanding of convolutional neural network-a deep learning approach. *Procedia Computer Science*, 132:679–688, 2018.

Joel Janai, Fatma Güney, Aseem Behl, Andreas Geiger, et al. Computer vision for autonomous vehicles: Problems, datasets and state of the art. *Foundations and Trends® in Computer Graphics and Vision*, 12(1–3):1–308, 2020.

James Kennedy. Swarm intelligence. In *Handbook of Nature-Inspired and Innovative Computing: Integrating Classical Models with Emerging Technologies*, pages 187–219. Springer, 2006.

James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-International Conference on Neural Networks*, volume 4, pages 1942–1948. ieee, 1995.

Salman Khan, Hossein Rahmani, Syed Afaq Ali Shah, Mohammed Bennamoun, Gerard Medioni, and Sven Dickinson. *A Guide to Convolutional Neural Networks for Computer Vision*. Springer, 2018.

Alex Khang, Vugar Abdullayev, Eugenia Litvinova, Svetlana Chumachenko, Abuzarova Vusala Alyar, and PTN Anh. Application of computer vision (CV) in the healthcare ecosystem. In *Computer Vision and AI-Integrated IoT Technologies in the Medical Ecosystem*, pages 1–16. CRC Press, 2024.

John R Koza. *Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems*, volume 34. Stanford University, Department of Computer Science Stanford, CA, 1990.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

P. Larrañaga, R. Etxeberria, J. A. Lozano, and J. M. Peña. Optimization in continuous domains by learning and simulation of gaussian networks. In *Proceedings Genetic and Evolutionary Computation Congress*, pages 201–204, 2000.

Pedro Larrañaga and Jose A Lozano. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*, volume 2. Springer Science & Business Media, 2001.

Jian-Yu Li, Zhi-Hui Zhan, Jin Xu, Sam Kwong, and Jun Zhang. Surrogate-assisted hybrid-model estimation of distribution algorithm for mixed-variable hyperparameters optimization in convolutional neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 34(5):2338–2352, 2021.

Qiuzhen Lin, Zhixiong Fang, Yi Chen, Kay Chen Tan, and Yun Li. Evolutionary architectural search for generative adversarial networks. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 6(4):783–794, 2022.

Feng Liu, Hanyang Wang, Jiahao Zhang, Ziwang Fu, Aimin Zhou, Jiayin Qi, and Zhibin Li. EvoGAN: An evolutionary computation assisted GAN. *Neurocomputing*, 469:81–90, 2022.

Haitao Liu, Yew-Soon Ong, Xiaobo Shen, and Jianfei Cai. When Gaussian process meets big data: A review of scalable GPs. *IEEE Transactions on Neural Networks and Learning Systems*, 31(11):4405–4423, 2020.

Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. *arXiv preprint arXiv:1711.00436*, 2017.

XJ Luo, Lukumon O Oyedele, Anuoluwapo O Ajayi, Olugbenga O Akinade, Hakeem A Owolabi, and Ashraf Ahmed. Feature extraction and genetic algorithm enhanced adaptive deep neural network for energy consumption prediction in buildings. *Renewable and Sustainable Energy Reviews*, 131:109980, 2020.

Heinz Mühlenbein and Gerhard Paass. From recombination of genes to the estimation of distributions I. Binary parameters. In *International Conference on Parallel Problem Solving from Nature*, pages 178–187. Springer, 1996.

Heinz Mühlenbein, Thilo Mahnig, and Alberto Ochoa Rodriguez. Schemata, distributions and graphical models in evolutionary optimization. *Journal of Heuristics*, 5 (2):215–247, 1999.

Bahareh Nakisa, Mohammad Naim Rastgoo, Andry Rakotonirainy, Frederic Maire, and Vinod Chandran. Long short term memory hyperparameter optimization for

a neural network based emotion recognition framework. *IEEE Access*, 6:49325–49338, 2018.

Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan kaufmann, 1988.

M. Pelikan, D. E. Goldberg, and E. Cantú-Paz. BOA: The Bayesian optimization algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 525–532, 1999.

Ishaani Priyadarshini and Chase Cotton. A novel LSTM–CNN–grid search-based deep neural network for sentiment analysis. *The Journal of Supercomputing*, 77(12): 13911–13932, 2021.

Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *International Conference on Machine Learning*, pages 2902–2911. PMLR, 2017.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training GANs. *Advances in Neural Information Processing Systems*, 29, 2016.

Gideon Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, pages 461–464, 1978.

Siddhartha Shakya and Roberto Santana. An EDA based on local Markov property and Gibbs sampling. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, pages 475–476, 2008.

Siddhartha Shakya, Alexander Brownlee, John McCall, François Fournier, and Gilbert Owusu. A fully multivariate DEUM algorithm. In *2009 IEEE Congress on Evolutionary Computation*, pages 479–486. IEEE, 2009.

Seunghyup Shin, Youngbok Lee, Minjae Kim, Jihwan Park, Sangyul Lee, and Kyoungdoug Min. Deep neural network model with Bayesian hyperparameter optimization for prediction of nox at transient conditions in a diesel engine. *Engineering Applications of Artificial Intelligence*, 94:103761, 2020.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Pratibha Singh, Santanu Chaudhury, and Bijaya Ketan Panigrahi. Hybrid MPSO-CNN: Multi-level particle swarm optimized hyperparameters of convolutional neural network. *Swarm and Evolutionary Computation*, 63:100863, 2021.

Vicente P Soloviev, Concha Bielza, and Pedro Larrañaga. Semiparametric estimation of distribution algorithms for continuous optimization. *IEEE Transactions on Evolutionary Computation*, 2023.

Vicente P Soloviev, Pedro Larrañaga, and Concha Bielza. Edaspy: An extensible python package for estimation of distribution algorithms. *Neurocomputing*, 2024.

Dehua Song, Chang Xu, Xu Jia, Yiyi Chen, Chunjing Xu, and Yunhe Wang. Efficient residual dense block search for image super-resolution. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 12007–12014, 2020.

Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

Rainer Storn and Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11:341–359, 1997.

Masanori Suganuma, Shinichi Shirakawa, and Tomoharu Nagao. A genetic programming approach to designing convolutional neural network architectures. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 497–504, 2017.

Yanan Sun, Bing Xue, Mengjie Zhang, and Gary G Yen. Completely automated cnn architecture design based on blocks. *IEEE Transactions on Neural Networks and Learning Systems*, 31(4):1242–1254, 2019.

Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer Nature, 2022.

Phat Thai, Sameer Alam, Nimrod Lilith, and Binh T Nguyen. A computer vision framework using convolutional neural networks for airport-airside surveillance. *Transportation Research Part C: Emerging Technologies*, 137:103590, 2022.

José F Torres, David Gutiérrez-Avilés, Alicia Troncoso, and Francisco Martínez-Álvarez. Random hyper-parameter search-based deep neural network for power consumption forecasting. In *Advances in Computational Intelligence: 15th International Work-Conference on Artificial Neural Networks, IWANN, 2019, Proceedings, Part I 15*, pages 259–269. Springer, 2019.

A Helen Victoria and Ganesh Maragatham. Automatic tuning of hyperparameters using bayesian optimization. *Evolving Systems*, 12(1):217–223, 2021.

Petra Vidnerová and Roman Neruda. Multi-objective evolution for deep neural network architecture search. In *International Conference on Neural Information Processing*, pages 270–281. Springer, 2020.

Bin Wang, Yanan Sun, Bing Xue, and Mengjie Zhang. Evolving deep convolutional neural networks by variable-length particle swarm optimization for image classification. In *2018 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2018.

Chaoyue Wang, Chang Xu, Xin Yao, and Dacheng Tao. Evolutionary generative adversarial networks. *IEEE Transactions on Evolutionary Computation*, 23(6):921–934, 2019a.

Hanchao Wang and Jun Huan. Agan: Towards automated design of generative adversarial networks. *arXiv preprint arXiv:1906.11080*, 2019.

Yulong Wang, Haoxin Zhang, and Guangwei Zhang. cPSO-CNN: An efficient PSO-based algorithm for fine-tuning hyper-parameters of convolutional neural networks. *Swarm and Evolutionary Computation*, 49:114–123, 2019b.

Christopher KI Williams and Carl Edward Rasmussen. *Gaussian Processes for Machine Learning*, volume 2. The MIT press Cambridge, MA, 2006.

Jia Wu, Xiu-Yun Chen, Hao Zhang, Li-Dong Xiong, Hang Lei, and Si-Hao Deng. Hyperparameter optimization for machine learning models based on Bayesian optimization. *Journal of Electronic Science and Technology*, 17(1):26–40, 2019.

Zhongze Wu, Chunmei He, Liwen Yang, and Fangjun Kuang. Attentive evolutionary generative adversarial network. *Applied intelligence*, 51:1747–1761, 2021.

Lingxi Xie and Alan Yuille. Genetic CNN. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1379–1388, 2017.

Shuyuan Xu, Jun Wang, Wenchi Shou, Tuan Ngo, Abdul-Manan Sadick, and Xiangyu Wang. Computer vision techniques in construction: a critical review. *Archives of Computational Methods in Engineering*, 28:3383–3397, 2021.

Xin Yao, Yong Liu, and Guangming Lin. Evolutionary programming made faster. *IEEE Transactions on Evolutionary computation*, 3(2):82–102, 1999.

Xinjie Yu and Mitsuo Gen. *Introduction to Evolutionary Algorithms*. Springer Science & Business Media, 2010.

Zhi-Hui Zhan, Jian-Yu Li, and Jun Zhang. Evolutionary deep learning: A survey. *Neurocomputing*, 483:42–58, 2022.

Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu. Practical block-wise neural network architecture generation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2423–2432, 2018.

Longfei Zhou, Lin Zhang, and Nicholas Konz. Computer vision techniques in manufacturing. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 53(1): 105–117, 2022.

Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.