Evolutionary Multi-Objective Design of Autoencoder for Compact Representation of Histopathology Images

by

Davood Zaman Farsa

A thesis submitted to the
School of Graduate and Postdoctoral Studies in partial
fulfillment of the requirements for the degree of

Master of Science in Computer Science

Faculty of Science

University of Ontario Institute of Technology (Ontario Tech University)

Oshawa, Ontario, Canada

August 2021

© Davood Zaman Farsa, 2021

THESIS EXAMINATION INFORMATION

Submitted by: **Davood Zaman Farsa**

Master of Science in **Computer Science**

Thesis title:
Evolutionary Multi-Objective Design of Autoencoder for Compact Representation of Histopathology Images

An oral defense of this thesis took place on August 16, 2021 in front of the following examining committee:

Examining Committee:

Chair of Examining Committee	Dr. Patrick Hung
Research Supervisor	Dr. Shahryar Rahnamayan
Examining Committee Member	Dr. Mehran Ebrahimi
Thesis Examiner	Dr. Khalid Elgazzar

The above committee determined that the thesis is acceptable in form and content and that a satisfactory knowledge of the field covered by the thesis was demonstrated by the candidate during an oral examination. A signed copy of the Certificate of Approval is available from the School of Graduate and Postdoctoral Studies.

ABSTRACT

Deep learning can cope with complex tasks; however, it suffers from the crucial problem of high dimensionality which makes it intractable to learn the patterns and classify the data. Importantly, having a fixed model cannot achieve high accuracy on every issue and the optimal solution is embodied in the architecture of the model tailored, specifically, for that issue. Motivated by these challenges, we propose a multi-objective evolutionary scheme that evolves autoencoders for the goal of dimension reduction. Throughout the evolution of each model, multiple minimization objectives are considered. These objectives can obtain compressed models that extract significant features while having the highest classification accuracy. In our case study, we use the extracted features of Histopathology images from a DenseNet trained to classify 30 subtypes of carcinomas from TCGA repository. As a result, we increased classification accuracy over 8% and compressed the representation of gigapixel images above 46,000 times, simultaneously.

Keywords: Evolving autoencoder; Digital pathology; Dimension reduction; Neural architecture search; Multi-objective optimization

AUTHOR'S DECLARATION

I hereby declare that this thesis consists of original work of which I have authored. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize the University of Ontario Institute of Technology (Ontario Tech University) to lend this thesis to other institutions or individuals for the purpose of scholarly research. I further authorize University of Ontario Institute of Technology (Ontario Tech University) to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research. I understand that my thesis will be made electronically available to the public.

_____ Davood Zaman Farsa

STATEMENT OF CONTRIBUTIONS

The following contributions are presented in this thesis:

- Automating the architecture design of autoencoders tailored specifically for each dataset without any human expert intervention.
- Proposing a simple yet robust postprocessing optimizer to compress the representation of gigapixel WSIs and increase the classification accuracy per tumor type, simultaneously.
- Managing the issue of curse of dimensionality by designing compact autoencoders to extract the significant features and provide a compact representation.
- Showing the potential of the evolving autoencoder as a dimension reduction technique for designing accurate models per tumor type with small amount of data.
- Utilizing a bilevel multi-objective evolutionary optimization algorithm to optimally design the architecture of each autoencoder based on objectives such as FLOPs, reconstruction error etc. at the same time.
- Providing compact representation of each WSI to facilitate image search and retrieval.
- Suggesting a variety of autoencoders as trade-off solutions based on the objective functions on a Pareto-Front so that expert user will have a set of choice to select from.
- Proposed scheme can be easily attached to any DNN for obtaining a compact and accurate representation of the input data without requiring any GPU or expensive computation power.

Sole authorship

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication. I have used standard referencing practices to acknowledge ideas, research techniques, or other materials that belong to others. Furthermore, I hereby certify that I am the sole source of the creative works and/or inventive knowledge described in this thesis.

ACKNOWLEDGEMENTS

I would like to express my gratitude and acknowledge everyone who supported me during my research and in times of need.

TABLE OF CONTENTS

Abstract ..... **iii**
Table of Contents ..... **vi**
List of Tables ..... **vii**
List of Figures ..... **xii**
List of Abbreviations and Symbols ..... **xiv**
Chapter 1: Introduction **1**
1.1 Optimization 1
1.2 Multi-objective Evolutionary Algorithms 5
1.3 Autoencoders (AEs) 11
1.4 Neural Architecture Search (NAS) 17
1.5 Digital Pathology 27
1.6 Motivations and Contributions 29
1.7 Thesis Outline and Organization 31

Chapter 2: Proposed Evolving Autoencoder **32**
2.1 Optimization Objectives and Enhancements 32
2.2 Encoding of the Search Space 34
2.3 Proposed Framework 36

Chapter 3: Designing Autoencoders for Fashion MNIST **42**
3.1 Investigation on Evolving Autoencoder 44
3.2 Sensitivity Analysis of Hyperparameters 45
3.2.1 Npop, Data Size, Neurons 46
3.2.2 Number of Generations 47
3.2.3 Box constraint for Increasing Number of Neurons 48
3.2.4 Training Epochs 49
3.2.5 Number of Layers 50
3.3 Designing Undercomplete Autoencoders 53
3.4 Box Constraint Handling 55
3.5 Investigation on Reconstruction of Mean Image 57
3.6 Investigation on Dataset Size Reduction 58
3.7 Investigation on One Training Epoch 60
3.8 Discussion 61

Chapter 4: Designing Autoencoders for Histopathology Features **63**
4.1 Results on Bi-objective Schemes 68
4.1.1 RMSE and FLOPs 68
4.1.2 RMSE and Classification Error 69
4.1.3 Classification Error and FLOPs 70
4.1.4 Sum of losses (Classification Error and RMSE) and FLOPs 74
4.2 Results on Many-objective Schemes 79
4.3 Results on Three-Objectives Schemes 82
4.3.1 FLOPs, Classification Error, Code Size 83

4.3.2 Supervised AE 84
4.4 Comprehensive Analysis 90
4.5 Comparative Studies 106
4.5.1 Evolving Autoencoders Versus Human-designed Models 106
4.5.2 Evolving Autoencoders Versus PCA 111

Chapter 5: Conclusion Remarks and Future Works 114

References 116

Appendices 122
Appendix A 122
A.1 Detailed PF solutions for all Histopathology Schemes.....122
A.2 EAE vs. Human-designed AEs.....145

LIST OF TABLES

CHAPTER 3

Table 3.1: Investigation on Evolving Autoencoder …….....45

Table 3.2: Investigation on Reduced Npop, Data Size, Neurons 47

Table 3.3: Investigation on Number of Generations 48

Table 3.4: Investigation on Box-constraint for Increasing Number of Neurons.....49

Table 3.5: Investigation on Training Epochs 50

Table 3.6: Investigation on Deep Autoencoders 52

Table 3.7: Investigation on Undercomplete Autoencoders 54

Table 3.8: Box Constraint Handling 56

Table 3.9: Investigation on Reconstruction of Mean Image 58

Table 3.10: Investigation on Dataset Size Reduction 59

Table 3.11: Investigation on Dataset Size Reduction 60

Table 3.12 : Investigation on One Training Epoch 61

CHAPTER 4

Table 4.1: Investigation on the 2 objectives of RMSE and FLOPs vs. 2 objectives of RMSE and Classification Error vs. 2 objectives of Classification Error and FLOPs 73

Table 4.2: Investigation on the single loss vs. multiple summed losses for fully training while RMSE is only used during evolution 77

Table 4.3: Investigation on the single loss vs. multiple summed losses for fully training while summed losses is use during evolution 78

Table 4.4: Investigation on the 4-objectives schemes 82

Table 4.5: Investigation on the 3-objectives schemes 88

Table 4.6: Investigation on the Supervise AE (SAE) scheme with population size of 100 where only RMSE is used as the loss function during both evolution and fully training….89

Table 4.7: Investigation on the two, three and four objectives experiments on Brain tumor where only RMSE is used as the loss function during both evolution and fully training….95

Table 4.8: Investigation on the two, three and four objectives experiments on Pulmonary tumor where only RMSE is used as the loss function during both evolution and fully training.....96

Table 4.9: Detailed description of all the proposed schemes labeled from 1 to 9 for ease of reference in the comparison plots 98

Table 4.10: Human-designed vs. EAE for the Supervised AE (SAE) scheme on All Tumors.....108

Table 4.11: PCA vs. EAE for the Supervised AE (SAE) scheme on All Tumors 113

Table 4.12: Investigation on the 2 objectives of RMSE and FLOPs on Brain Tumor 123

Table 4.13: Investigation on the 2 objectives of RMSE and FLOPs on Pulmonary Tumor.....124

Table 4.14: Investigation on the 2 objectives of RMSE and Classification Error on Brain Tumor.....125

Table 4.15: Investigation on the 2 objectives of RMSE and Classification Error on Pulmonary Tumor.....126

Table 4.16: Investigation on the 2 objectives of Classification Error and FLOPs on Brain Tumor.....127

Table 4.17: Investigation on the 2 objectives of Classification Error and FLOPs on Pulmonary Tumor.....128

Table 4.18: Investigation on the 2 objectives of Classification Error and FLOPs with Summation of Losses vs. single RMSE Loss on Brain Tumor 129

Table 4.19: Investigation on the 2 objectives of Classification Error and FLOPs with Summation of Losses vs. single RMSE Loss on Pulmonary Tumor 130

Table 4.20: Investigation on the 4 objectives of RMSE, Classification Error, Code Size and FLOPs on Brain Tumor with 100 Generations 131

Table 4.21: Investigation on the 4 objectives of RMSE, Classification Error, Code Size and FLOPs on Pulmonary Tumor with 100 Generations 132

Table 4.22: Investigation on the 4 objectives of RMSE, Classification Error, Code Size and FLOPs on Brain Tumor with 300 Generations 133

Table 4.23: Investigation on the 4 objectives of RMSE, Classification Error, Code Size and FLOPs on Pulmonary Tumor with 300 Generations 134

Table 4.24: Investigation on the 3 objectives of Classification Error, Code Size and FLOPs on Brain Tumor 135

Table 4.25: Investigation on the 3 objectives of Classification Error, Code Size and FLOPs on Pulmonary Tumor 136

Table 4.26: Investigation on the 3 objectives of Classification Error, Code Size and FLOPs with the SAE Method on Brain Tumor 137

Table 4.27: Investigation on the 3 objectives of Classification Error, Code Size and FLOPs with the SAE Method on Endocrine Tumor 138

Table 4.28: Investigation on the 3 objectives of Classification Error, Code Size and FLOPs with the SAE Method on Gastro Tumor 139

Table 4.29: Investigation on the 3 objectives of Classification Error, Code Size and FLOPs with the SAE Method on Gynaeco Tumor 140

Table 4.30: Investigation on the 3 objectives of Classification Error, Code Size and FLOPs with the SAE Method on Liver Tumor 141

Table 4.31: Investigation on the 3 objectives of Classification Error, Code Size and FLOPs with the SAE Method on Melanocytic Tumor 142

Table 4.32: Investigation on the 3 objectives of Classification Error, Code Size and FLOPs with the SAE Method on Prostate Tumor 142

Table 4.33: Investigation on the 3 objectives of Classification Error, Code Size and FLOPs with the SAE Method on Pulmonary Tumor 143

Table 4.34: Investigation on the 3 objectives of Classification Error, Code Size and FLOPs with the SAE Method on Urinary Tract Tumor 144

Table 4.35: Human-designed vs. EAE for the Supervised AE (SAE) scheme on Brain ...146

Table 4.36: Human-designed vs. EAE for the Supervised AE (SAE) scheme on Endocrine.....147

Table 4.37: Human-designed vs. EAE for the Supervised AE (SAE) scheme on Gastro.....148

Table 4.38: Human-designed vs. EAE for the Supervised AE (SAE) scheme on Gynaeco.....149

Table 4.39: Human-designed vs. EAE for the Supervised AE (SAE) scheme on Liver.....150

Table 4.40: Human-designed vs. EAE for the Supervised AE (SAE) scheme on Melanocytic.....151

Table 4.41: Human-designed vs. EAE for the Supervised AE (SAE) scheme on Prostate.....152

Table 4.42: Human-designed vs. EAE for the Supervised AE (SAE) scheme on Pulmonary.....153

Table 4.43: Human-designed vs. EAE for the Supervised AE (SAE) scheme on Urinary Tract 154

LIST OF FIGURES

CHAPTER 1

Figure 1.1: Schematic of the Bilevel Optimization for NAS…….....5

Figure 1.2: Pareto Fronts of a sample population ….....9

Figure 1.3: Framework of NSGA-II 10

Figure 1.4: General categories of AEs 14

Figure 1.5: Sparse AE 16

Figure 1.6: NAS Framework 21

CHAPTER 2

Figure 2.1: A visual illustration of the 7-dimensional chromosome encoding.....34

Figure 2.2: A sample example of the 7-dimensional chromosome encoding 35

Figure 2.3: A visual illustration of the 4-dimensional chromosome encoding.....36

Figure 2.4: A sample example of the 4-dimensional chromosome encoding 36

Figure 2.5: Overall schematic of our proposed framework 37

Figure 2.6: Schematic of the proposed method 41

CHAPTER 3

Figure 3.1: A sample of 25 images from all classes of Fashion MNIST dataset 43

Figure 3.2: Mean Image of each Fashion MNIST Class 57

CHAPTER 4

Figure 4.1: DenseNet architecture of KimiaNet 63

Figure 4.2: The schematic view of the AE architectures utilized in Histopathology application 65

Figure 4.3: Schematic of our proposed framework 67

Figure 4.4: The schematic of the SAE scheme utilized in histopathology application.....85

Figure 4.5: Two objectives of RMSE and FLOPs on Brain 99

Figure 4.6: Two objectives of RMSE and FLOPs on Pulmonary 99

Figure 4.7: Two objectives of RMSE and F1-score on Brain 100

Figure 4.8: Two objectives of RMSE and F1-score on Pulmonary 101

Figure 4.9: Two objectives of Classification Error and FLOPs on Brain 102

Figure 4.10: Two objectives of Classification Error and FLOPs on Pulmonary 102

Figure 4.11: Three objectives of Classification Error, FLOPs, and Code Size on Brain ...103

Figure 4.12: Three objectives of Classification Error, FLOPs, and Code Size on Pulmonary.....103

Figure 4.13: Two objectives of Classification Error and Code Size on Brain 105

Figure 4.14: Two objectives of Classification Error and Code Size on Pulmonary 105

Figure 4.15: Human-designed vs. EAE for the Supervised AE (SAE) scheme on Brain...109

Figure 4.16: Human-designed vs. EAE for the Supervised AE (SAE) scheme on Endocrine.....109

Figure 4.17: Human-designed vs. EAE for the Supervised AE (SAE) scheme on Gastro.....109

Figure 4.18: Human-designed vs. EAE for the Supervised AE (SAE) scheme on Gynaeco..... 109

Figure 4.19: Human-designed vs. EAE for the Supervised AE (SAE) scheme on Liver...110

Figure 4.20: Human-designed vs. EAE for the Supervised AE (SAE) scheme on Melanocytic.....110

Figure 4.21: Human-designed vs. EAE for the Supervised AE (SAE) scheme on Prostate.....110

Figure 4.22: Human-designed vs. EAE for the Supervised AE (SAE) scheme on Pulmonary.....110

Figure 4.23: Human-designed vs. EAE for the Supervised AE (SAE) scheme on Urinary Tract.....111

LIST OF ABBREVIATIONS AND SYMBOLS

ABC	Artificial Bee Colony
ACC	Adrenocortical Carcinoma
ACO	Ant Colony Optimization
AE	Autoencoder
AF1	Averaged F1-score
BA	Bat Algorithm
BBO	Black-Box Optimization
BLCA	Bladder Urothelial Carcinoma
CESC	Cervical Squamous Cell Carcinoma and Endocervical Adenocarcinoma
CHOL	Cholangiocarcinoma
CNN	Convolutional Neural Network
COAD	Colon Adenocarcinoma
DE	Differential Evolution
DenseNet	Densely Connected Convolutional Networks
DL	Deep Learning
DNN	Deep Neural Network
EAE	Evolutionary Autoencoder
EMO	Evolutionary Multi-objective Optimization
ENAS	Evolutionary Neural Architecture Search
EA	Evolutionary Algorithm
ESCA	Esophageal Carcinoma
FLOPs	Floating-point Operations per second
GA	Genetic Algorithm
GBM	Glioblastoma Multiforme
KICH	Kidney Chromophobe
KIRC	Kidney Renal Clear Cell Carcinoma

KIRP	Kidney Renal Papillary Cell Carcinoma
K-NN	k-Nearest Neighbor Algorithm
LGG	Brain Lower Grade Glioma
LIHC	Liver Hepatocellular Carcinoma
LUAD	Lung Adenocarcinoma
LUSC	Lung Squamous Cell Carcinoma
MESO	Mesothelioma
NAS	Neural Architecture Search
NN	Neural Network
Npop	Population Size
NSGA-II	Non-dominated Sorting Genetic Algorithm version II
OV	Ovarian Serous Cystadenocarcinoma
PAAD	Pancreatic Adenocarcinoma
PCA	Principal Component Analysis
PCPG	Pheochromocytoma and Paraganglioma
PRAD	Prostate Adenocarcinoma
PF	Pareto Front
PSO	Particle Swarm Optimization
READ	Rectum Adenocarcinoma
RL	Reinforcement Learning
RMSE	Reconstruction Mean Squared Error
SAE	Supervised Autoencoder
SKCM	Skin Cutaneous Melanoma
STAD	Stomach Adenocarcinoma
TCGA	The Cancer Genome Atlas
TGCT	Testicular Germ Cell Tumors
THCA	Thyroid Carcinoma
UCS	Uterine Carcinosarcoma

UVM	Uveal Melanoma
VAE	Variational Autoencoder
WSI	Whole Slide Image

Chapter 1. Introduction

In the first chapter we will have an introduction to the major parts of our work to give a roadmap of our proposed multi-objective evolutionary framework that evolves autoencoders for the purpose of having autoencoders as a dimension reduction approach to cope with curse of dimensionality. And help with finding the optimal architecture designed, specifically, for that issue. First, we discuss optimization and classify the group of our optimization problem. Then, we consider multi-objective evolutionary algorithms and describe the scenario of having multiple objectives that are optimized, simultaneously. Afterwards, we will have an introduction to Deep Neural Networks (DNNs) and go in depth over the autoencoders, as an unsupervised learning model, and mention some of their common kinds. Next, Neural Architecture Search (NAS) methods are mentioned in detail and how they can help in automation of Neural Networks (NNs) architecture design. In addition, the topic of digital pathology is discussed next, and some of the key challenges that is commonly faced in medical domain when applying Machine Learning (ML) algorithms to such applications. Finally, the main motivation of this work is explained briefly.

1.1 Optimization

It is generally accepted that most industrial and research works ranging from engineering to medical domains require optimal decision variables when applied to a problem in order to cope with it, accurately and inexpensively. To move the problem that we are tackling towards its optimal state we need to formulate these efforts into an optimization problem by considering an objective or a set of objectives

to be minimized or maximized [1]. The problem needs to be defined based on a single or multiple objective (fitness) function so that we can solve the optimization problem. However, the process of modeling the problem and defining the objective function(s) is a challenging task. Because the accuracy, performance of the optimizer and the quality of the results are highly connected with the defined model and its objective function(s). The general formulation of single objective optimization problem is given in Eq. (1.1) in which f(x) represents the objective function and g(x) and h(x) stand for inequality and equality constraints.

$$\begin{aligned} &\text{Min/Max:} && f(x) && \\ &\text{Subject to} && g_i(x) \leq 0, && i = 1, 2, 3, \ldots, m; \\ &&& h_j(x) = 0, && j = 1, 2, 3, \ldots, p; \end{aligned} \tag{1.1}$$

Optimization algorithms can be divided into two types of continuous and combinatorial. In continuous algorithms we can further divide them into linear and non-linear. Linear methods are consisted of interior-point search [2] and simplex method [3], while non-linear approaches are mainly categorized as global and local search methods in which local methods can further described in two types of gradient-based and gradient-free algorithms. On the other side, in combinatorial optimization we can have exact methods such as branch-and-bound [4][5] and dynamic programming [6][7]. Also, we have approximate methods which generalize to a similar set of algorithms as global search methods like heuristic [8], meta-heuristic [9], and random search [10]. For the purpose of our work, we handle continuous nonlinear methods that are under the category of global search methods in meta-heuristic methods. Now, meta-heuristic algorithms are classified as either

single-solution based or population-based search methods. For single-solution based methods we can name tabu search [11], simulated annealing [12], while for population-based methods we can mention two sub types of evolutionary and swarm intelligence algorithms. Precisely, evolutionary algorithms are subset of evolutionary computation technique based on the biological evolution and are using reproduction, selection, recombination, and mutation. They incorporate variety of algorithms, including but not limited to, Genetic Algorithm (GA) [13], Differential Evolution (DE) [14], evolutionary programming [15], evolutionary strategies [16], etc. While on the other hand, swarm intelligence methods are originated from the collective behaviour of an organized group of animals, as they struggle to survive. It is consisted of algorithms such as Particle Swarm Optimization (PSO) [17], Ant Colony Optimization (ACO) [18], Artificial Bee Colony (ABC) [19], Bat Algorithm (BA) [20], etc.

Nowadays, many academic and industrial challenges belong to the class of large-scale optimization problems and in most organizations the lower-level sections make impact on the decisions of the upper-level sections in optimizing their goals which constitutes a hierarchical bilevel structure; so, the usage of bilevel optimization algorithms has got way too much attraction in tackling such problems [21]. In bilevel optimization we have a nesting of two problems. The upper level is called the leader which makes some decision based on the input variables that it takes and once it is done, the lower level reacts with respect to the decision by tackling an independent optimization problem. In other words, the lower-level optimizer only sees the decisions of the leader and then optimizes its particular strategies [22][23].

Interestingly, some other researchers have utilized evolutionary algorithms to mitigate the challenges such as non-convexity, being extremely hard to manage it mathematically, and belonging to the class of NP-hard problems [30], that are faced in bilevel optimization [24][25].

Since our framework is following the concept of Neural Architecture Search (NAS), which will be discussed in great detail later, we are addressing a bilevel optimization problem in which our upper-level optimizer or the leader is a search strategy that designs various neural networks, specifically autoencoders, and just focuses on the architecture of the models. While the lower-level optimizer takes the designed architectures from the upper-level and optimizes the weights of each architecture. So, the learning happens at the lower-level which then defines the performance of the designed architecture as depicted in Figure 1.1. Besides, the mathematical model of bilevel optimization is given in Eq. (1.2) where F shows the upper-level objective function and f is for the lower-level objective function and x represents the upper-level decision vector while y is the lower-level decision vector. Moreover, G_i and g_j are the inequality constraints for the upper and lower-levels of the optimizer, accordingly. Finally, we can have equality constraint of h(x), h(y) as well.

$$
\begin{aligned}
&\text{Min:} && F(x, y) \\
&\text{Subject to:} && G_i(x, y) \leq 0, \qquad i = 1, 2, 3, \ldots, I; \\
& && y \in \arg\min \{f(x, z) : g_j(x, z) \leq 0, j = 1, 2, 3, \ldots, J\} \\
& && \text{where: } F, f : R^{nx} \times R^{ny} \rightarrow R; \\
& && \quad G_i, g_j : R^{nx} \times R^{ny} \rightarrow R; \\
& && \quad X \subseteq R^{nx}, \; Y \subseteq R^{ny};
\end{aligned}
\tag{1.2}
$$

$$h(x) = 0, h(y) = 0.$$

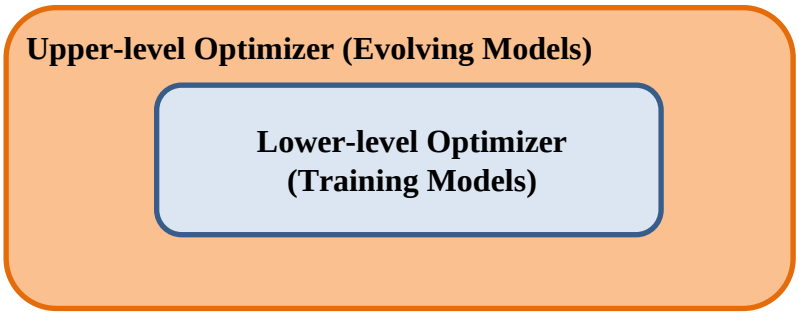

Figure 1.1: Schematic of the Bilevel Optimization for NAS. The upper-level optimizer designs various architectures for NN, and the lower-level optimizer tunes their weight connections

1.2 Multi-objective Evolutionary Algorithms

As Evolutionary Multi-objective Optimization (EMO) has projected itself into the eyes of researchers and has acquired popularity in industry, in our framework, for the upper-level optimizer we concentrate on population-based evolutionary multi-objective algorithms. More specifically, we use multi-objective version of GA called Non-dominated Sorting Genetic Algorithm, almost known as NSGA-II, proposed in [26], which incorporates multiple objectives when solving the optimization problem, simultaneously. In multi-objective optimization problems, we have various objectives that we need to optimize at the same time and these objectives could be both minimization or maximization objectives; however, sometimes we might have in conflict objectives that one should be minimized while the other maximized like complexity and accuracy of a model, in which we wish to minimize the complexity but maximize the accuracy. Thus, it gives more importance to a set of optimal solutions that is referred to as Pareto-optimal solutions on the Pareto Front (PF)[27]. In general, Evolutionary Algorithms (EAs) are based on a single objective

optimization, and they are popular as they are flexible, easily implemented and applied to Black-Box Optimization (BBO) problems where no information of the derivative of the objective function is available. So, at each generation of the algorithm they use a population of candidate solutions to optimize a single objective function. Consequently, they begin working by randomly initializing a population of individuals and this population enters an iterative process of selection, crossover, and mutation which is called generation and it stops when the termination criteria are met. Each of the operators in a generation are designed for a purpose. After fitness evaluation of every individual, the selection operator is applied to select above-average individuals by means of stochastic selection methods such as tournament selection [28] based on their fitness value. Then the crossover operator is applied to exchange information among the selected individuals and form new set of individuals and afterwards, the mutation operator is applied to mutate one or some the genes of the individuals in order to allow EA to explore the search space better. In addition to these common operators, we have elitism operator as well. It saves some of the better quality (i.e., better fitness value) individuals among the combined old and newly generated population to be passed to the next generations and is used to prevent performance degrading. Now, in EMO we have the same procedure except that now we have more than one objective (fitness) function to optimize, and its general formulation based on [29] is shown in Eq. (1.3).

$$\begin{aligned} &\text{Min/Max:} && [f_1(x), f_2(x), \ldots, f_M(x)] \\ &\text{Subject to:} && g_b(x) \geq 0, && b = 1, 2, 3, \ldots, J; \\ & && h_c(x) = 0, && c = 1, 2, 3, \ldots, K; \end{aligned} \tag{1.3}$$

$$x_d^{(L)} \leq x_d \leq x_d^{(U)}, \quad d = 1, 2, 3, \ldots, n.$$

where x is a candidate solution vector of n variables (i.e., a chromosome of n genes), M is the objective space dimension, L and U stand for upper and lower bound of each variable (i.e., gene of the chromosome). And g(x) represents inequality constraint and h(x) is known as equality constraint that can be applied to the optimization problem. It is important to note that the dimension of the problem is defined based on the number of variables that contribute to the objective function. By increasing the number of dimensions, the problem gets more complicated. Whenever we are managing varied types of variables such as integer, binary, real, etc. it lies in the category of mixed-type optimization.

In multi-objective optimization, we require a method to compare two candidate solutions because the objectives are in conflict and the comparison is not straightforward as the single objective minimization/maximization. Thus, in EMO we have a mathematical term known as partial ordering which is utilized to define the optimal solutions. It is referred to as domination which defines among all the solutions that we have got from the optimizer which ones are better solutions. In other words, it separates the non-dominated solutions into various PFs, and the optimal set of solutions are in the first front that is returned by our upper-level optimizer. Because the first front contains those candidate solutions that are not dominated by any other solution in terms of the objective functions that have been optimized and are a better approximation of the optimal solutions. Besides, these non-dominated solutions on the first front are also called non-dominated front. The

domination rule that a solution $X^{(1)}$ dominates another solution $X^{(2)}$ are simply described by two conditions as follows [29][30]:

1. In all objectives " i " that we have, the candidate solution $X_i^{(1)}$ is no worse than $X_i^{(2)}$.

2. In at least one of the objectives " i' ", the candidate solution $X_{i'}^{(1)}$ is strictly better than $X_{i'}^{(2)}$.

If both of these conditions are true, then the domination rule is applied to split all the candidate solutions into various fronts. Solutions in each front do not dominate each other but they are dominated by the solutions in their previous front(s), and they dominate solutions in the front(s) after them [26]. More detail is provided in Figure 1.2. It is worth mentioning that the computational complexity of selecting the non-dominated front from a set of N candidate solutions is shown in Eq. (1.4):

$$\begin{cases} O(N \text{ Log } N) & \text{for 2 and 3 objectives} \\ O(N \text{ Log}^{M-2} N) & \text{for } M > 3 \text{ objectives} \end{cases} \tag{1.4}$$

In NSGA-II, after applying evolutionary operators and evaluating the newly generated populations, we merge it with the old population and the domination rule is applied to all candidate solutions and groups them into different fronts as mentioned above. Now, for creating the population for the next generation we start selecting from the first front and then second front and so on, till we reach the maximum size of the population (i.e., Npop). When we select a front we select all of the candidate solutions of that front and accommodate it in the new population slots,

but what if the size of a front is larger than our population size limit and we cannot accommodate all of its candidate solutions? In such a scenario, we need to take another measure into account called crowding distance which is calculated for every candidate solution in that front, and we sort every candidate solution of that front based on their crowding distance value in a descending order. We can say that the crowding distance of a candidate solution is a measure of the objective space around it which is filled by none of the other candidate solution in the population [29]. It is calculated based on Eq. (1.5).

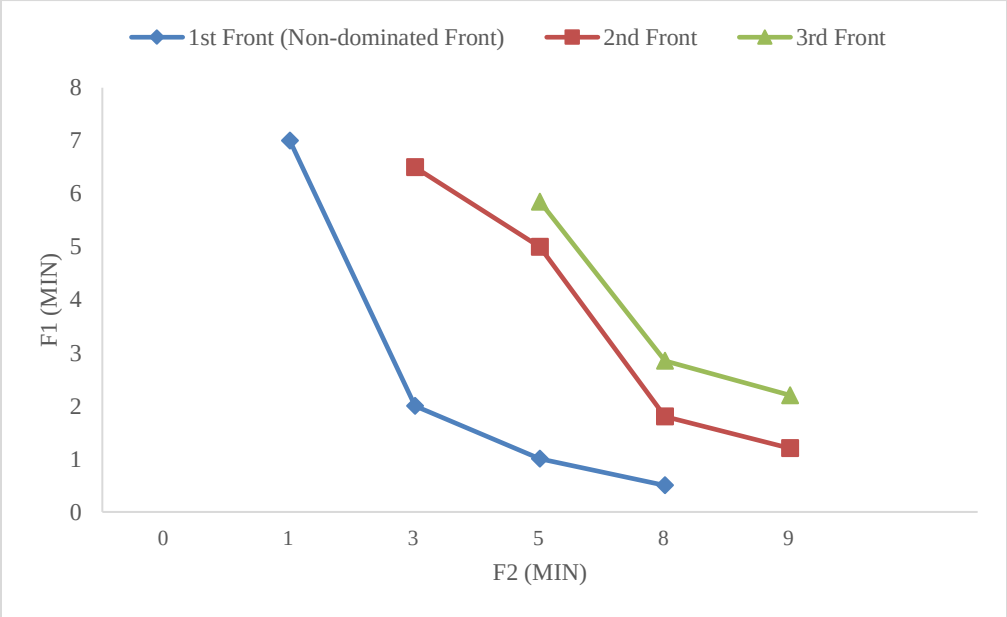

Figure 1.2: Pareto Fronts of a sample population. The first front contains solutions closer to the optimal solutions since they provide a better estimation of the optimal solutions and are not dominated by any other solution.

So, for the front that exceeds the size of the population we only select those candidate solutions with the highest crowding distance score because those candidate solutions

are located in a sparser region of the PF and can help with the diversity of the proposed solutions since similar solutions are close to each other.

$$d_i^{F_j} = \frac{|F_j^{i+1} - F_j^{i-1}|}{F_j^{max} - F_j^{min}}$$
$$d_i = \sum_j d_i^{F_j} \quad (1.5)$$

Where $d_i^{F_j}$ represents the crowding distance of the "i"$_{th}$ candidate solutions w.r.t the "j"$_{th}$ objective function. F_j^{max} and F_j^{min} refers to the max and min objective value of the objective F_j . We need to calculate $d_i^{F_j}$ for all "j" objectives and then take the sum of them as the final crowding distance value for the "i"th candidate solution. The overall schematic framework of NSGA-II is illustrated in Figure 1.3, in which P_t is the old population and Q_t is the newly generated population. More implementational detail is available in [26].

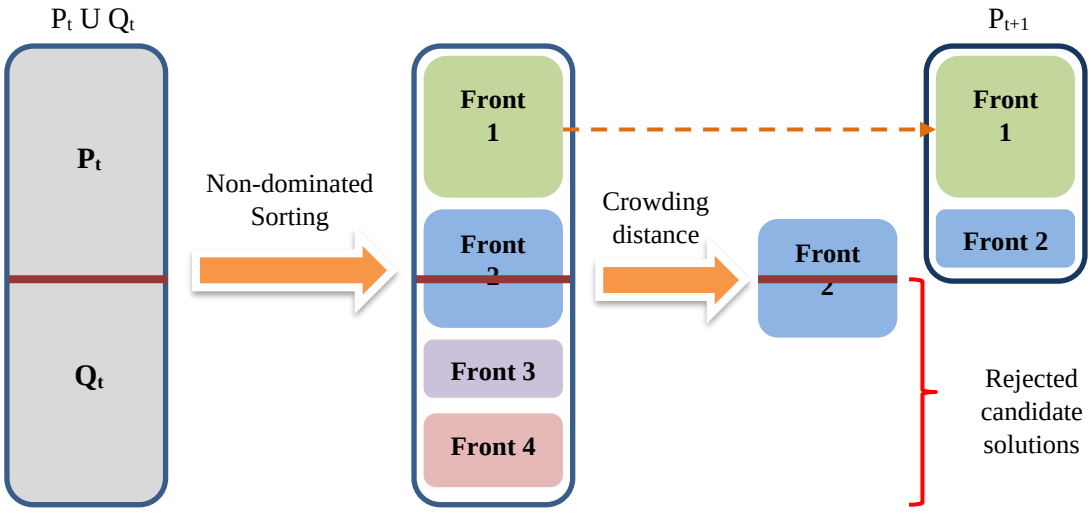

Figure 1.3: Framework of NSGA-II. All solutions are first divided into various fronts based on their domination rank and if any front size exceeds the population size then top solutions with high crowding distance will be passed to the next population.

1.3 Autoencoders (AEs)

Recent years have witnessed the proliferation of machine learning (ML) algorithms being used for various medical and non-medical tasks from classification to detection, clustering and etc. In traditional ML algorithms the human expert needed to select or adjust the input features in order to build a strong model that does not suffer form underfitting and overfitting as the best scenario. Furthermore, for those ML algorithms having large dataset was not helpful and most algorithms could not enhance the overall performance beyond a certain chunk of data and their performance get plateaus. However, among various ML algorithms, neural networks (NNs) have shown a tremendous success in achieving more promising results by feeding them more data to train on. NNs are some how inspired by the connectionist model of the brain and are consisted of some number of layers each of which include some number of neurons that are connected to other neurons in their adjacent layers by a weight connection parameter that can be learned. By having more hidden layers in a NN we make it deeper and also called deep neural networks (DNNs) and can automatically extract the useful features without human intervention and find out what features are most important. As a result, they easily surpass most of the ML algorithms in various real-world tasks and have become a great competitor to traditional ML algorithms. We can group ML algorithms into three class of supervised learning, unsupervised learning, and reinforcement learning (RL) algorithms which are proposed for different purpose. In supervised learning our dataset is labeled and the label acts as a supervisor that trains the model and guides

the model through out the training process; however, in unsupervised learning we do not have any labeled data and the learner has to figure out the similarities and differences between the data samples all by itself without any supervision. Finally, in RL we have no dataset to train on and it is a reward base system in which we have an agent that tries to learn an optimal policy by interacting with the environment with the goal of maximizing the future rewards that it gets form the environment.

Obviously, providing label for data samples is a very expensive and tedious task. Therefore, it makes it really hard to train a very powerful model with only a few numbers of labeled data and that is one of the reasons why unsupervised learning techniques have gained more prosperity over the decades and have become a great tool in machine learning. Autoencoder (AE) [31][32], as an unsupervised learning approach, utilizes neural networks with the same number of neurons at both input and output layers for the purpose of representation learning by trying to reconstruct the original input at the output layer [33]. So, by applying a bottleneck layer (i.e., code layer) we are forcing the network to extract the most impactful features and learn a compressed knowledge representation of the original input.

Since the proposal of AE, a lot of versions of AE have been proposed that has managed various challenging problems and have projected themselves as useful tools into challenging real-world scenarios. Here we are trying to cover some of AE's types and showed some of its astonishing applications. Starting with types of AEs, first, we have undercomplete AEs in which hidden layer is constrained to be smaller in order to limit the amount of information that goes through the networks. And the

code layer will be the smallest layer; therefore, the code layer learns the latent features of the input data. Interestingly, this type of AE does not possess any regularization term to penalize the model from overfitting and the reason is that we have sufficiently restricted the number of neurons in the hidden layers so that it prevents the model from memorizing the input data. Thus, learning happens only by forcing the network toward having lower reconstruction error. Also, Vanilla AE, which is the simplest version of undercomplete AE has only an input, hidden, and output layer while the hidden layer has smaller number of neurons compared to the input and output layers. Next, we have over complete AE, in which the size of the hidden layer is larger than the input and output layers and to avoid learning identity function it must be regularized because not only we have no compression, but each hidden unit can copy its corresponding input feature and pass it to the next layer as well and as a result there is no guarantee that the hidden neurons will extract beneficial information. We can categorize them further into 4 types of shallow undercomplete, deep undercomplete, shallow overcomplete, and deep overcomplete [34] as it is illustrated in Figure 1.4. It should be pointed out that undercomplete AEs have shown a tremendous impact on dimensionality reduction, visualization, feature extraction, etc.

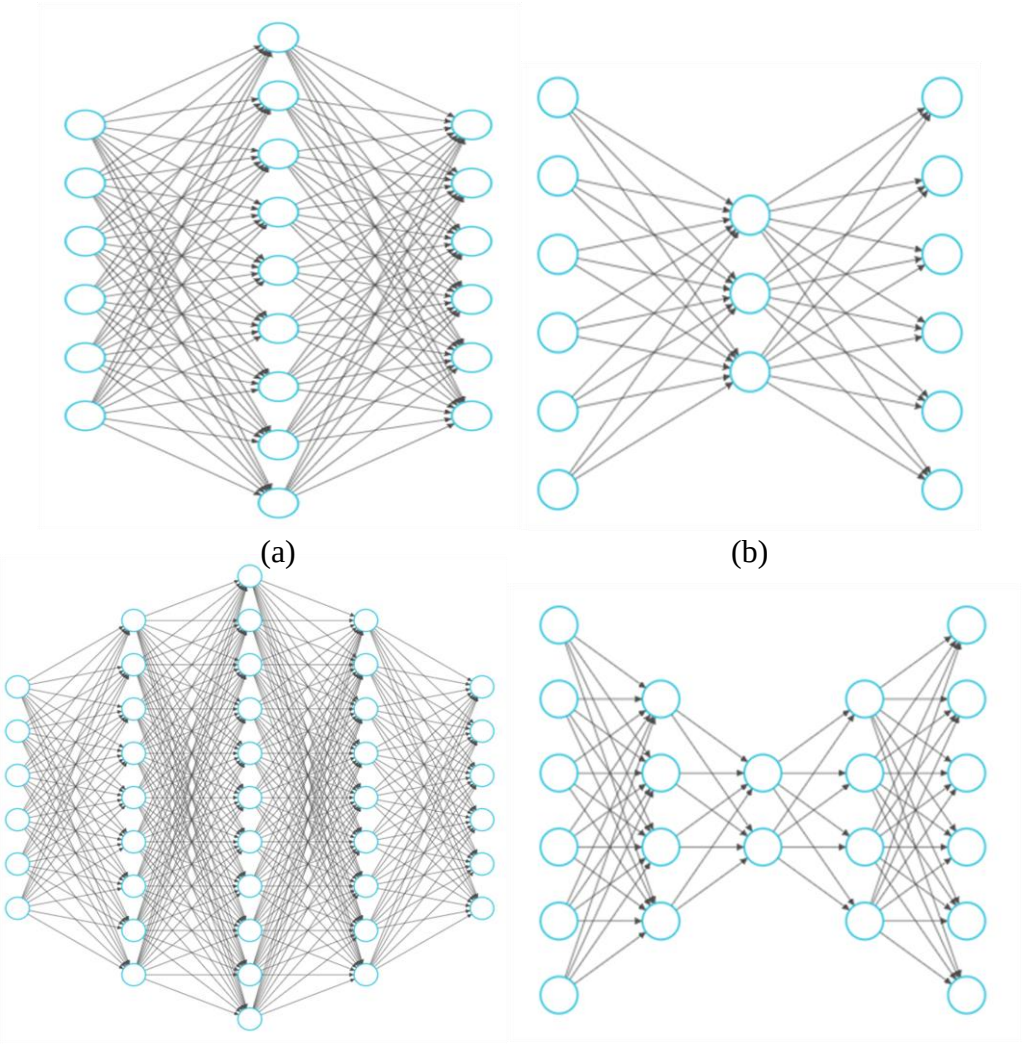

Figure 1.4: General categories of AEs: (a) shallow overcomplete, (b) shallow undercomplete, (c) deep overcomplete, (d) deep undercomplete

As we want the models to uncover latent features within the data and ensure that the model is not memorizing the data, we need to leverage regularization techniques in AEs and prevent overfitting for cases such as overcomplete AE. We can mention Denoising AEs [35], Sparse AEs [36], and Contractive AEs [37]. In denoising AE,

instead of feeding the original input data to the AE we add noise to it and feed the corrupted input to the network and want out model to reconstruct the uncorrupted original data. This way we ensure that our model is not suffering from overfitting, so it is not memorizing the training data since our input data and target output are different. Therefore, it gives the model more generalizability.

Another type is sparse AE shown in Figure 1.5, the idea behind this approach is that you do not need to constrain your hidden layers' size to be smaller to avoid overfitting; so, you can have large number of neurons in your model's hidden layers and still avoid overfitting by defining a loss function which penalizes activations within a layer. Therefore, the sparsity constraint is applied to the hidden layers to prevent them from copying input data to output. And this sparsity penalty (i.e., $\Omega(h)$) is chosen to be close to zero and is added to the reconstruction loss. Basically, it picks the highest activation values in each hidden layer and zero out the rest of the neurons and this forces the AE to only rely on small number of neurons, in other words, it utilizes smaller number of features to prevent having a complex model and reduces overfitting. It is worth mentioning that this is a distinct method of regularization, since we usually regularize the weights of the network, not the activations.

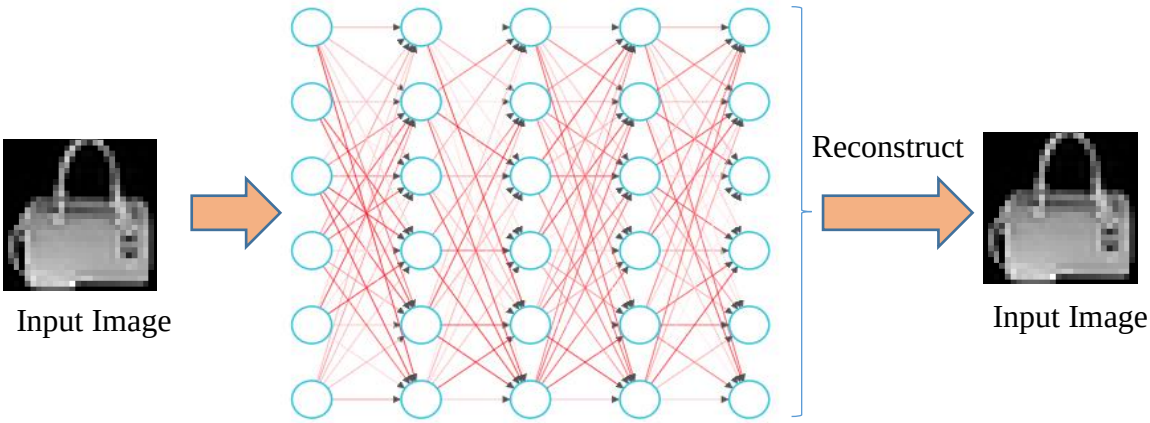

Figure 1.5: Sparse AE

As another type of sparse AE, we can mention contractive AE, which applies a penalty function of Frobenius norm of the Jacobian matrix (it is a matrix of size, number of data samples × total parameters count, consisted of first partial derivatives per each data sample separately and each row represents the full gradient per individual input data sample) to the reconstruction loss. Precisely, it penalizes large gradients of the hidden layers' activations regarding the input data sample. This way it raises the generalizability of the model by making the embedding variables more robust to small changes in the input data or pixel values of the input images. Thus, it encourages the AE to contract a neighbourhood of input data to a tinier output in a neighborhood by learning an encoding that similar input data samples have similar encodings.

Finally, we have another type of AEs for content generation which is also know as generative model called variational AE (VAE) and it can be utilized for generating instances that closely resemble the input data but are totally generated from the code layer's distribution [38]. Thus, the AE is learning the parameters of a probability distribution that models the input data instead of learning the embedding

representation. By doing so, you can sample from the latent probability distribution and generate new input data. Precisely, instead of mapping a data sample to vector of the embedding representation, it first maps it into two vectors of mean and standard deviation. Then for each latent feature "$\boldsymbol{F_i}$" it samples a value from a normal probability distribution with ($\boldsymbol{\mu_i}$,$\boldsymbol{\sigma_i}$). Next, it puts all sampled feature "$\boldsymbol{F_i}$" in a vector, called sampled latent vector, and feed it to the decoder for reconstruction purpose. The loss function is composed of two parts: a reconstruction loss, which matches the decoded sample with original input, and KL divergence loss, that minimizes the difference between the latent distribution and standard normal to optimize $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ parameters of the latent probability distribution.

1.4 Neural Architecture Search (NAS)

In this section details of Neural Architecture Search (NAS) methods will be discussed and how they can be useful. Designing an architecture for neural network models is a very tedious and time-consuming task for human experts and it requires a lot of hyperparameter tuning such as number of layers, number of neurons per layer etc. in order to make it work well on the target dataset. At the end, when you need to use it on another dataset you may need to start all over again because that model was tailored specifically for that dataset, and it might not generalize well to another data. Besides, we cannot say that having only one neural network architecture can perform equally well on every dataset and simply use it for every problem, because they are task-specific and data-oriented, and as clearly stated in [39], the optimal solution to a problem is embodied in the optimal design of the ANN architecture and it is not all about the weights of the model. Due to the mentioned challenges, there has been an

ever-growing interest in automating this process by utilizing automated neural architecture search methods. Moreover, they have shown a great success in beating human designed models for various tasks including, but not limited to, image classification [40][41], object detection [40], and more.

Generally, NAS belongs to a type of DL methods referred to as meta learning [42][43], in which it uses an auxiliary search method such as random search, grid search, evolutionary search or reinforcement learning to design the characteristics of the NN to tackle the problem at hand. In [44], they have categorized NAS techniques based on three aspects of search space, search strategy, and performance estimation strategy. First, the algorithm searches in a search space of different parameters/hyperparameters such as number of layers, types of layers such as convolution, pooling, dilated convolution etc., number of filters, stride and filter size, type of activation function and so on. And through the search strategy approach it opts an architecture, then it evaluates its performance by training the model on training data and validating it on validation and returns its performance result to the search strategy to help exploring the search space better and find an optimal architecture. It should be pointed out that the performance estimation process can be extremely expensive as all models have to be trained for some number of epochs. And this is one of the interesting research directions that has got attention of researchers, as well. Another factor that makes a great impact on the difficulty level of the optimization problem is the search space. Strictly, researchers have followed two different architecture design approaches known as micro-architecture and macro-architecture optimization in which the first one is about designing cells of the

neural network and then stack them together to build a macro architecture, while the later term is concerned about the layout of the stacked cells and how these micro architectures should be connected. But where was this idea of micro and macro architecture initiated from? Basically, there are various ways suggested to parameterize the search space from designing simple models to very complex ones such as having skip connections. The parameters usually include the maximum number of layers, number of neurons in fully connected networks of dense layers, the type of operations such as convolution, pooling, dilated convolution [45], number of filters and their stride in convolutional models [46][47][48]. Therefore, in [40][49][50], rather than designing the whole network they focus on a block level design such that first different blocks are designed and then these blocks are stacked on top of each other to form the final model. This block-type design has some advantage compared to from-scratch design for CNNs, such as shrinkage of the search space since block are smaller models with less layers and also a smaller parametrized search space, they can be easily tuned for other problems by changing the number of blocks or the kernels inside each block [40]. However, you need to think about the number of blocks and how they should be connected to build the final model. In this approach, instead of just optimizing the micro blocks of the model, the best scenario is to optimize both blocks (i.e., the micro-structures) and final model (i.e., the macro-structure consisted of stacked micro-structures) jointly in order to not be biased by designing the final model manually and let your optimizer to explore new designs that might not been designed by humans before. Now, we discuss a bit more about the performance evaluation approaches that are commonly utilized by

researchers. As we briefly mentioned earlier, this step is computationally expensive as you need to train each model and evaluate its quality based on your validation data. In some works, including but not limited to [51][52], they have taken weight inheritance into account and tried to transfer the weights of the base model that is fully trained so that later on the other generated models fine tune these weights. In addition, another approach known as one-shot models, first builds a base model of large size is designed, probably by human expert, and fully train it and call it one-shot model; then, the other generated models are subgraphs of this one-shot model and use its weights; however, the downside of this approach is that the one shot model is defined first and this limits its generated sub models in terms of number of layers and neurons per layer because they cannot exceed the one-shot model [53][54][55]. Moreover, we have the option of extrapolating the learning curve by various means such as eliminating the candidates with poor quality to search swiftly, training a surrogate model [56]. Besides, we can evaluate the performance of each model in a simpler way by training each model for fewer number of epochs, using smaller subset of data, having less filters per layer and reducing the number of blocks which are categorized under the group of lower fidelity approximation methods [57][58], and for more detail refer to [44]. The general framework of NAS is shown in Figure 1.6.

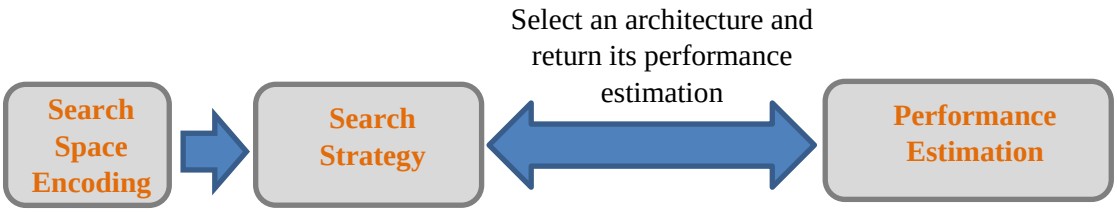

Figure 1.6: NAS Framework

Some of the key challenges of early NAS can be managing an extremely large search space in order to search all the necessary components of ANN and the larger the search space the more expensive to run the search. Because it is almost unfeasible to know the optimal architecture of the ANN for the given dataset without conducting an extensive search for both the architecture and its hyperparameters. Moreover, some NAS methods start building the ANN from scratch with a few neurons while gradually increasing the complexity of the ANN; so, they might ignore the common design choice of human-expert while those ANNs could be used as the starting point of the NAS to let it take those manually designed architectures into account and try to improve them. In addition, since each designed model needs to be trained in order to make a decision about it, we cannot fully train each model due to the computational time limit and, depending on the problem at hand, training for a few numbers of epochs may not show the true quality of that ANN.

Now, some of the studies that have been done by NAS and are more related to this thesis will be discussed. In [39], the authors proposed an interesting approach towards the design of NN which follows an opposing direction to NN pruning. They start the process of NAS by creating very simple NNs that only have a few

connections from input layer to output layer and a single value is used as the weight for every connection within the NN and they, gradually, increase the complexity of the model by adding more neurons between connections, inserting new connection, modifying the activation function of the existing NN. Particularly, they first initialize a population of sparse models and then evaluate each of them based on various single value weight for all connections and then rank the networks based on their mean performance on all the weight values, the maximum performance by the best single weight, and the number of connections which indicates the complexity of the model. Therefore, they rank the models based on the mentioned criterions and generate a new population of NN by adjusting the top ranked NN. Finally, they go through the same process of evaluation, ranking, and modification until the stopping condition is met. They experimented their approach for various tasks in RL and image classification on MNIST digits in which they could achieve accuracy of 92%. The important take away from their works is that the optimal solution for a problem is hidden in the optimal structure designed specifically for that problem and its dataset and weights are not that important in finding the optimal architecture of the NN. Interestingly enough, in [64] they have emphasized more on NAS without training in which they have used evolutionary search for the NN and then try to assign a score to each NN at the initialization step to avoid training by passing a minibatch of data points through the NN and calculating the similarity of each datapoint to each other by extracting and comparing their binary code and store it in a matrix. Finally, they calculate the score of that NN by taking the logarithm of the norm of that matrix and this score represents the quality of that NN without any training. In another work

[65], they are trying to design AEs with goal of having the most impactful features by compressing the data. Therefore, they are taking an evolutionary algorithm into account. They have used three objectives, maximization of the classification accuracy based on the smallest hidden layer (i.e., chokepoint or code layer), minimization of the code layer, and the number of layers in the decoder part of the AEs and the reason is that minimization of the code layer will lead to achieving a compact representation of the input image while minimization of decoder makes the decoding process easier and helps the optimizer to focus on good encoders to compress the data. However, their approach is based on the weighted summation of the objectives, and they solve the problem as a single objective. It is worth mentioning that they let their optimizer hands open to design asymmetric topologies with having a maximum of 6 hidden layers and at least 2 hidden layer and each layer having neurons in range of 1 to 784. The best structure that they suggested was asymmetric which shows that there is no rule to always follow the common symmetric design of AEs. Besides, instead of concentrating on the input image reconstruction, they focused on the reconstruction of the mean image of each class of MNIST digit – for each class of digit an image is created in which each pixel represents the mean of all the images in that specific class for that pixel– by doing this the AEs are guided toward learning a compressed version of the features for each class and also it increases the learning pace. One of the disadvantages of this work is that it cannot discover the concave parts of the pareto front solutions, and it requires tuning a lot of hyperparameters. Additionally, crossover operator should be done carefully since the chromosome have variable length. In [66], it is clearly speculated

that AEs can have variable number of hidden layers each of which having different number of neurons. The suitable architecture is mostly dependable on the complexity of the data in terms of the patterns to be reconstructed and also the limitations imposed by the code layer. Therefore, they are using an evolutionary approach to design symmetric AEs but not forced in a decreasing order and configure each of them into a chromosome of size 14 containing the type of the AE which is a number from 1 to 6 for Basic, Denoising, Contractive, Robust, Sparse, and Variational, respectively. Afterwards, the number of hidden layers in range of 1 to 3 in both encoder and decoder, number of neurons per each layer in range of 1 to the number of features in the dataset, type of the activation function to be utilized in each layer for both encoder and decoder which is a number in the boundary of 1 to 8 per each layer indicating whether it is linear, sigmoid, tanh, relu, selu, elu, softplus, and softsign, respectively. Finally, a number specifying the loss function in range of 1 to 5 to choose form Mean Squared Error, Mean Absolute Error, Mean Absolute Percentage Error, Binary Cross entropy, and Cosine Proximity, respectively. They used GA and Evolutionary Search (ES) separately to tackle the problem, but they follow a single-objective approach and calculate the weighted summation of reconstruction loss of the train data, the number of hidden layers, and the size of the code layer. In addition, they leave the weight optimization for the backpropagation algorithm. They conducted their experiments on two datasets Sonar with dimension of 60 and MNIST with 784. They summarized the results that exhaustive search cannot obtain the reconstruction error of GA and ES, GA performed better on the Sonar while on MNIST dataset ES was better in terms of the AE configurations, and

ES is faster than GA. A disadvantage of this work is that it cannot discover the concave parts of the pareto front solutions, and it requires tuning lots of hyperparameters. In [67], the idea is that there are a vast number of metrics that determine the performance of a NN such as structure, the type and order of the hidden layer, the choice of the activation functions, loss functions and the presentation of data to the network. These decisions should be made within an iterative loop of design, train, and evaluation over a long period of time. Therefore, their aim is to automate this process by utilizing neuro-evolutionary attitude to design convolutional NN (CNN) and symmetric Convolutional AE (CAE) through their modified version of NSGA-II algorithm in multiple worker processes in parallelized approach. In addition, they have tried to reduce the heavy computation time of this evolutionary process by reducing the training input data size used for training classification networks (i.e., using a reduced dimensional representation of the input to each CNN before training for classification) and this dimension reduction is done by CAE. By doing so, the training time of each network structure (i.e., each individual in the population) is reduced because of less values to be processed and also, we can have shallower networks designed. It is worth mentioning that they use evolutionary process twice, first to design a CAE to reduce the input dimension, second, to design a CNN for classification. Therefore, they evolve a population of CAEs based on the reconstruction error of the decoder and then select the best one and encode the whole input data set. Afterwards, a population of CNNs is evolved for classifying the input, each of which trained based on the encoded input from the previous step and classification error. Finally, to get the best final network they select

the CNN with the highest accuracy rate and attach it to the encoder which used to encode the input data. Since they used NSGA-II for their designs, their objectives for the CAEs were reconstruction error and compression ratio and also defined their own mutations for the encoder part since all structures were symmetric and were forced by having the encoder output size to be smaller than its input size, to focus on compressing input data. While for CNNs they followed the baseline framework that they used in their work [68]. For their experiments they used CIFAR-10 dataset and the training of the networks was taken care of by gradient descent. In [69], they have used evolutionary search to design deep convolutional variational AEs by using 4 types of blocks, h-block, μ-block, σ-block and t-block each of which consisted of some specified operations, in the final design of the VAEs to tackle the common challenges of manual design of VAEs and their symmetric limitations by having a flexible design of their architecture and using variable length gene encoding technique. They benchmarked their proposed method against other types of AEs on MNIST, SVHN and CIFAR-10 datasets for the purpose of classification. Although, this approach has outperformed its competitors there are lots of constraints applied to the evolutionary optimizer to not to generate invalid or incorrect architectures; for example, the optimizer should be careful about how to select operations in the decoder to make the output dimension the same as input dimension, or the dense layers are only allowed in the end of the encoder and beginning of the decoder, or deconvolutional layer must be used only in the decoder. Therefore, these type of complex models and designs require careful attention and may also be biased by the way that each block is ruled by human expert.

1.5 Digital Pathology

It is generally accepted that Artificial intelligence (AI) innovations can cope with the enormous amount of data generated during the course of the patient care lifecycle to enhance pathologic diagnosis, classification, and prognostication of diseases. Deep learning, as one of AI's sub-categories, has achieved tremendous success in digital pathology to diminish errors in diagnosis and image classification of different cancer types including, but not limited to, brain, endocrine, liver, prostate, etc. However, they suffer from the crucial problem of high dimensionality which makes it intractable for the models to learn the patterns inside the images and it destroys the classification accuracy, as well [59]. The replacement of microscope with a digital scanner and computer monitor has gained a lot of attraction, recently, and digitization of whole slide images (WSIs) has provided us with numerous benefits such as advanced and efficient workflow and collaboration among pathologists and by utilizing image processing and machine learning algorithms it has brought novel biological insights into the area of histopathology [60][61]. In digital pathology, the usage of WSI has gained attraction among pathologists for diagnostic and research goals like image search or image retrieval. Basically, each WSI comprises of two procedures. First, it utilizes specialized hardware such as a scanner, in order to digitize glass slides, which produces a large representative digital image known as "digital slide" which can be perceived on a computer screen or mobile device. Second, a virtual slide viewer is utilized to view and/or analyze these massive digital files at various magnifications. One of the mostly used datasets in digital pathology is The Cancer Genome Atlas (TCGA) which contains images of 32 cancer types for

25 anatomic sites and comprises 30.072 WSIs. Since we are using a pretrained DenseNet, called KimiaNet, for the medical application in our case study and this dataset was adjusted based on their study, more details regarding the datasets used in this thesis can be found in [60].

Despite the great achievements of digital pathology, we still face some challenges when applying AI techniques to digital pathology. Here we are covering some of these challenges and also mentioning some opportunities in this area as described in [61] and explain briefly how it is addressed in our framework. One of the main challenges in medical image analysis, especially histopathology, is not having enough labeled data because the pathologists need to go through all images and find those regions that have potential sign of the tumor and then label it. So, this is a very time consuming and tedious task that should be done by experts to reduce any mistake while labeling. And this gets out of hands when they manage noisy, blurry, or low-quality images. Therefore, performing classification task on such a small set of labeled data can be really harsh and we cannot train a good model to give a high accuracy. Thus, in our work, we are utilizing AE to train a good model in an unsupervised fashion while extracting the most impactful features from the input data. When the AE is fully trained we use the trained weights for the purpose of classification to help us getting promising results. Another challenge is curse of dimensionality, since each WSI is a gigapixel digital image over 100,000 by 100,000 pixels, while the common data size for DNN is around 300 by 300 pixels. Wherefore, we need to reduce this high dimensional input by applying patching technique, which divides each WSI into many small blocks, and the feed the patches as input to the

NN model. You many also think why not using downsampling methods to reduce the size of each WSI, by doing that our models will suffer from loss of data and it is not in our benefit. Of highest importance is the challenge of computational power and hardware costs, as you might know DNNs are heavily relying on Graphical Processing Units (GPUs) as they can parallelize the process and have shown better performance compared to Central Processing Units (CPUs) [62][63]. Despite using GPUs, we still see that training DNN takes some days to be done and this will constraint them from being easily used in lower computational power devices like smart phones. Moreover, applying NAS to DNNs and, especially, CNNs is extremely expensive and time consuming and might require lots of limitations and constraints and more GPU days in their design process by NAS algorithms. Therefore, motivated by these challenges, in our framework we have designed an evolving AE that uses fully connected layers without any complex operations, and it can be easily added to every DNN to reduce the dimension of the output and increase the classification F1-score at the same time.

1.6 Motivations and Contributions

The lack of large, labeled dataset per tumor type and harshness caused by curse of dimensionality has made a crucial problem in medical applications to build robust and accurate models to be applied separately to each tumor type with only a few numbers of samples. And as a result, the classification accuracy is destroyed. In addition to that, the gap of having a compact representation of each gigapixel WSI is of high importance in digital pathology. Because it will occupy a smaller portion of the storage, and it boosts the classification accuracy of each tumor type by allowing

the model to learn discriminative and non-redundant features from each WSI. Moreover, patch based mechanism will generate many feature vectors per each WSI which is a huge amount of data for digital pathology tasks such as image retrieval or image search. Besides, DNNs are not optimally designed and applying NAS to DNNs will require a lot of computation power and it is not possible to design end-to-end models due to the mentioned reasons. Therefore, having a postprocessing optimizer that fills these gaps and can help reducing the dimension of the classification problem and computational complexity while increasing its accuracy, simultaneously, would be a huge advantage. Precisely, AE is a powerful nonlinear dimensionality reduction method and the goal of having AEs with compact code layer is to reduce the computational complexity of digital pathology tasks such as image retrieval or image search. The optimizing structure of AE can be defined as a black-box multi-objective discrete optimization problem which can be solved by evolutionary algorithms. It is discrete in the sense that we optimize the number of neurons in each layer. It is multi-objective because we have several objectives to design the architecture of each AE. It is black-box as we do not know the quality of each AE model mathematically, in advance. Since evolutionary NAS (ENAS) has become more popular and been used frequently among researchers and practitioners in industry, we focus on ENAS to design various AEs to extract the most significant features in an unsupervised fashion to take the most out of the few labeled dataset that we have per tumor type and increase the classification, as well. It should be pointed out that, each patch is presented by 1024 (extracted from a DNN) and each WSI consists of 135 patches on average, so the number of features is 135×1024. One of the main contribution of

the proposed framework, EAE, is its capability of reducing the high dimensional space of size 1024 to around 3 and increase the overall classification accuracy by around 8% while reducing the complexity of the representation code over 46,000 times (i.e., $\frac{135 \times 1024}{3}$). Besides, it can provide a list of AEs designed, specifically, for each tumor type in around 45 minutes on a core i5 system with 8 GB RAM. Moreover, unlike manually designed models by human expert, our framework integrates various minimization objective such as reconstruction error, FLOPs, code size (i.e., the number of neurons in the code layer), classification error in the design of each model to make them as compact and accurate as possible.

1.7 Thesis Outline and Organization

In Chapter 2, we review the details of our proposed method to resolve the problem of curse of dimensionality and classification accuracy by evolving various AEs. Then, in Chapter 3, we conduct various experiments to investigate different settings and evolutionary design patterns for a benchmark dataset like Fashion MNIST to find a promising scheme to be applied to histopathology images. In Chapter 4, the experiments are based on the histopathology features for medical application in order to tune the best scheme from the Fashion MNIST for the medical domain. Afterwards, we compare our proposed method, EAE, against some of the commonly suggested AE models and also PCA based on the histopathology features (since it was the main real-world application of this thesis) to show the power, robustness, and efficiency of our approach. Finally, we conclude the remarks of this work in Chapter 5.

Chapter 2. Proposed Evolving Autoencoder

In this chapter, we explain our proposed framework, Evolving Autoencoder (EAE), in detail and break it down into small parts like what are the optimization objectives and how each objective function is evaluated. Then provide more information regarding the way of encoding the search space and show how each AE is modeled as a chromosome to be evolved and trained. Finally, the whole framework is elucidated.

2.1 Optimization Objectives and Enhancements

First, all the objectives are described and explain how each are calculated. The objectives are considered for the purpose of each AE's design are Reconstruction Mean Squared Error (RMSE), FLOPs which represents the floating point operations per second and measures the complexity of each model, Classification Error (i.e., 1 – 'F1-score'), Code Size, which is the number of neurons in middle layer of each AE and in cases where middle layer is removed we simply select one of its adjacent layers with higher 'F1-score' value. The RMSE objective simply measures the error between the original input data and its reconstructed version from the output layer and in our experiments we almost always use it to optimize the weight of the AEs and in some schemes it is also one of the objectives that determine whether an AE should be kept among the non-dominated PF solutions. One of the objectives that is sometimes used is code size which refers to the number of neurons in the middle layer or its best adjacent layer if middle layer is removed and its index is called code layer. So, when the code size is an objective we force the optimizer to minimize it along the other objectives and it is important to have small code size because it provides us with compressed representation of the input image which helps with

faster image search and retrieval. The next objective is classification error in which the 'F1-score' is calculated only on the code layer, and it is 1–'F1-score' from K-NN classifier with K = 3. Finally, we have FLOPs that points to the complexity of the designed model and has a great impact on the compression of the model. It is calculated by the number of summations and multiplications between each two consecutive layers. For example, if we have two layers, one with 'n' and the other with 'm' number of neurons, then FLOPs is calculated as $2 \times m \times n$. Therefore, we calculate it between all the adjacent layers and sum up the results to get the final value of FLOPs. Since our models have at most 7 hidden layers and one input and output layer, the FLOPs is calculated as: $(2 \times m_1 \times \text{Input layer size}) + (2 \times m_2 \times m_1) + (2 \times m_3 \times m_2) + (2 \times m_4 \times m_3) + (2 \times m_5 \times m_4) + (2 \times m_6 \times m_5) + (2 \times m_7 \times m_6) + (2 \times \text{Output layer size} \times m_7)$, where m_i represents the number of neurons in layer 'i' and the first 1024 is the input size and the later one is the output layer size. All the mentioned objectives are explained mathematically in Eq. (2.1).

$$
\begin{aligned}
&\text{RMSE:} && \frac{1}{n}\sum_{k=1}^{n}(Y_k - \widehat{Y_k})^2 \\
&\text{FLOPs:} && \sum_{i=1}^{h}(2 \times H_i \times H_{i-1}) \\
&\text{F1-score:} && 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \\
&\text{Precision:} && \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}} \\
&\text{Recall:} && \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}
\end{aligned}
\tag{2.1}
$$

Where "n" is the size of the input data that we have as the input and output layers size. Also "h" is the number of hidden layers plus the input and output layer and "H_i"

is the size of the "i"th hidden layer. We consider "H_0" and "H_h" as the size of input and output layers, correspondingly.

2.2 Encoding of Search Space

Now that we know the optimization objectives, we can discuss the process of encoding the search space of the problem and explain how each AE with the maximum of 'H' = 7 hidden layers is modeled as a chromosome with seven genes and each gene represents a different hidden layer size as shown in Figure 2.1, where #NH1 stands for number of neurons in hidden layer 1 and so on. In our framework, we have defined two types of chromosomes as the encoding of the problem's search space. In the first scheme we map each AE into a chromosome of length 'H' each of which containing an integer value in a specified range for that experiment because in those experiments we let the optimizer hands open to design asymmetric AEs. Thus, when mutation operator is applied to each chromosome, we need to round the result to convert it to integer and be able to consider it as the number of neurons for the corresponding hidden layer. A sample example of this type of encoding is visualized in Figure 2.2.

Gene1: #NH1	Gene2: #NH2	Gene3: #NH3	Gene4: #NH4	Gene5: #NH5	Gene6: #NH6	Gene7: #NH7

Figure 2.1: A visual illustration of the 7-dimensional chromosome encoding. Each gene value represents its corresponding hidden layer size

Gene1:	Gene2:	Gene3:	Gene4:	Gene5:	Gene6:	Gene7:
120	33	276	17	0	52	142

Figure 2.2: A sample example of the 7-dimensional chromosome encoding. Each gene value represents its corresponding hidden layer size

In the second type of chromosome encoding, a set of 4 genes containing real values in range 0 to 1 are forming a chromosome as shown in Figure 2.3. Each gene represents a hidden layer for each AE model from hidden layer 1 to the code layer, and then the structure is mirrored to create a symmetric AE. In other words, each gene represents the ratio of the previous layer size to be considered as the number of neurons for the current layer. Now that each gene is a real value, after applying mutation we do not use any rounding mechanism and only when the gene value is multiplied to the previous layer size, which is an integer, then we round the result, as formulate in Eq. (2.2), in which "P_j" is the real value of gene "i" and "G" is the length of the chromosome. To clarify, based on the example provided in Figure 2.4, the size of the first hidden layer H_1 would be the Round of (0.56 × input size) which would result in an integer value. Then for H_2 we have the Round of (0.33 × H_1) and so on till we have calculated the fourth hidden layer. Since the fourth layer is the middle layer (i.e., the code layer) we mirror the structure from H_3 to H_1 as the hidden layer of H_5 to H_7. This way we have a symmetric AE with a small dimensional encoding search space that helps facilitating the computation time as well.

$$\begin{aligned} H_i &= \text{Round}\,(P_j \times H_{i-1}) \qquad i = 1, 2, 3, \ldots, h; \\ P_j &\in [0, 1) \qquad\qquad\qquad\quad j = 1, 2, 3, \ldots, G. \end{aligned} \tag{2.2}$$

Gene1:	Gene2:	Gene3:	Gene4:
#NH1	#NH2	#NH3	#NH4

Figure 2.3: A visual illustration of the 4-dimensional chromosome encoding. Each gene represents the ratio of the previous layer size to be considered as the number of neurons for the current layer and then it is mirrored from gene 3 to gene 1 to constitute the 7 hidden layers.

Gene1:	Gene2:	Gene3:	Gene4:
0.56	0.33	0.92	0.01

Figure 2.4: A sample example of the 4-dimensional chromosome encoding. Each gene value is multiplied to its previous layers size to build the number of neurons for its corresponding hidden layer and then it is mirrored from gene 3 to gene 1 to constitute the 7 hidden layers.

The reason that we have two different definition of chromosome encoding is that when we first started our experiments we had the first gene encoding that has 'H' = 7 integer values. But as we moved forward in the experiments and tried a variety of schemes we proposed the second gene encoding. Based on the results that we got and also the reduced search space from dimension of 7 to 4, we decided to stay with the second gene encoding.

2.3 Proposed Framework

In this section we dive deeply into the details behind the design of our proposed approach and try to cover all the aspects, key elements and operators implanted within it. From the Darwinian model of survival of the fittest we remember that evolution is more of a stochastic process that tyrannizes the essential architecture of a species; while learning is based on the connectionist model of the brain and tries to tune the architecture to advance an individual's adaptability to the environment.

Therefore, the combination of evolution and learning is characterized by the evolving neural networks and in our case we have AEs as the type of the neural network that is used for the goal of dimension reduction to get a compact representation of each input image, which we call it evolving autoencoder (EAE). So, our final design would look like Figure 2.5, in which the evolving AE part is added to the Fully trained DNN and our framework acts as a postprocessing part that is attached to DNN to satisfy the goal of obtaining a more compact and accurate code to represent the input images. So, we do not need to train the DNN anymore and our approach only focuses on the obtained latent representation vectors extracted from the DNN. This has several advantages such as lower computational power that was mentioned earlier and also it is very useful since end-to-end DNN cannot be used in such cases because end-to-end DNN requires large amount of training data, which we do not have. And more importantly, since we have a very small amount of data per tumor type, we cannot use end-to-end approach for tumor-based classification. While on the other hand, our evolving AE designs models per tumor type and despite having a few data samples per each tumor type it designs very accurate and compressed models tailored specifically for that tumor type.

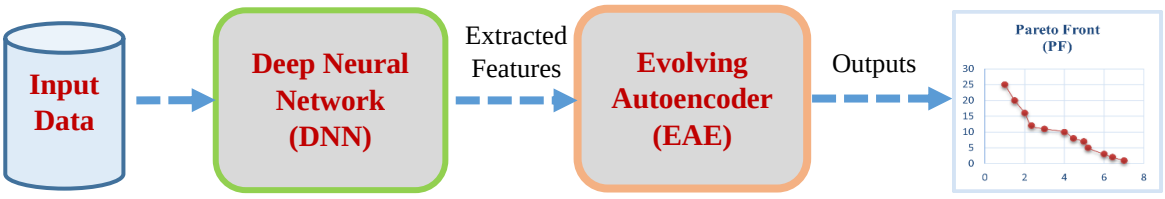

Figure 2.5: Overall schematic of our proposed framework

Despite the great advantages of using AEs for dimension reduction, we face the challenge of their optimal design, and a fixed topology cannot perform well on every

problem. So, we need to optimize their architecture by defining a multi-objective optimization problem for NAS. As mentioned earlier, our EAE framework is highly inspired by the ENAS workflow and we are facing a bilevel optimization problem in which the upper-level optimizer designs various AE architectures, and the lower-level optimizer optimizes the weights of these designed models and returns their performance estimation. It is of high importance to remind that applying NAS to DNN is very time consuming and it requires a lot of computational power. Thus, we are proposing a simple, cost-effective, and swift postprocessing optimization method to reduce the high dimensional output and increase the classification accuracy, simultaneously. Besides, our approach is utilized for each tumor type separately and it can provide really promising results even when we have a very small dataset, which is not possible in end-to-end DNNs. Next, we start discussing all the building blocks of our framework in detail.

Initialization:

In the upper-level optimizer, that optimizes the architecture of each AE model, we are using NSGA-II algorithm and we first initialize a population of individuals, and each individual is, in fact, a chromosome that contains a value per gene which represents the number of neurons in each hidden layer as shown in Figures 2.2 and 2.4. So, we can say that each gene represents a hidden layer of the AE and each chromosome (i.e., an array of genes) is an indication of an AE architecture.

Performance Estimation:

To have an estimation of either initialized (if it is the first time) or evolved models we pass this population to the lower-level optimizer to train the weight parameters of each individual by translating their corresponding chromosome into an AE model and applying Adam optimizer [70] to it and passing the extracted feature vectors from the fully trained DNN as our dataset. In our case study since each WSI contains many patches then for each patch a feature vector is extracted from DenseNet and to obtain one feature vector for whole image, first we compute the mean of features on all patches, then in order to compact the code we feed in the mean of these extracted feature vectors of each patch.

Finally, each AE is trained for some epochs based on the training data and evaluated by the validation set and its performance is returned as its fitness value.

Rank-based Selection:

Afterwards, the rank-based selection is applied in which the solutions are ranked by the non-dominated sorting and crowding distance algorithms discussed earlier in NSGA-II algorithm. These algorithms are trying to give us the set of solutions or AE models, among the whole population, that are good estimate of the optimal solution(s) in terms of the optimization objectives and will help the population to be evolved in that direction.

Termination Condition:

Then, the termination condition is checked to see if the algorithm has reached its maximum number of generations or not. Assuming the termination condition is not met yet and we transfer to the next phase. Otherwise, if the

termination criterion is met, the algorithm ceases and returns the non-dominated solutions on the first front (i.e., the non-dominated front) as the set of proposed AE models to be fully trained and evaluated. Finally, a decision making method can be applied to select one of these fully trained models based on their fitness values. In our experiments, we have selected the solution that had the best performance after fully training.

Mating Selection, Crossover, and Mutation:

Now, we need to apply evolutionary operators to evolve the models based on their chromosome encoding. Thus, we utilize tournament selection operator to select two better individuals and then select the best one as the first parent solution and repeat it again to select the other parent for the purpose of crossover operator. In crossover, we use simulated binary crossover (SBX), since it is the default choice in NSGA-II, to recombine both parent solutions to generate new offspring solutions. After, the polynomial mutation (PM) operator is applied, again it is the default operator in NSGA-II, to randomly mutate the genes of the new offspring population. In our experiments we refined the hyperparameter of these operators based on our case and we suggested distribution index of 2 and 5 as the best setting for SBX and PM, respectively. The distribution index in SBX has direct impact on controlling the spread of offspring models and in PM it controls the shape of the probability distribution function in which the smaller it is, the distant is parent solution from the child.

Offspring Population Evaluation:

So far we have created the new population of offspring, now it is time to evaluate them by passing them through the lower-level optimizer to train them and return their performance. We keep repeating the same procedure from Rank-based Selection step to evaluating the offspring population at each generation till the stopping criterion is satisfied. The detail of our proposed method is illustrated in Figure 2.6.

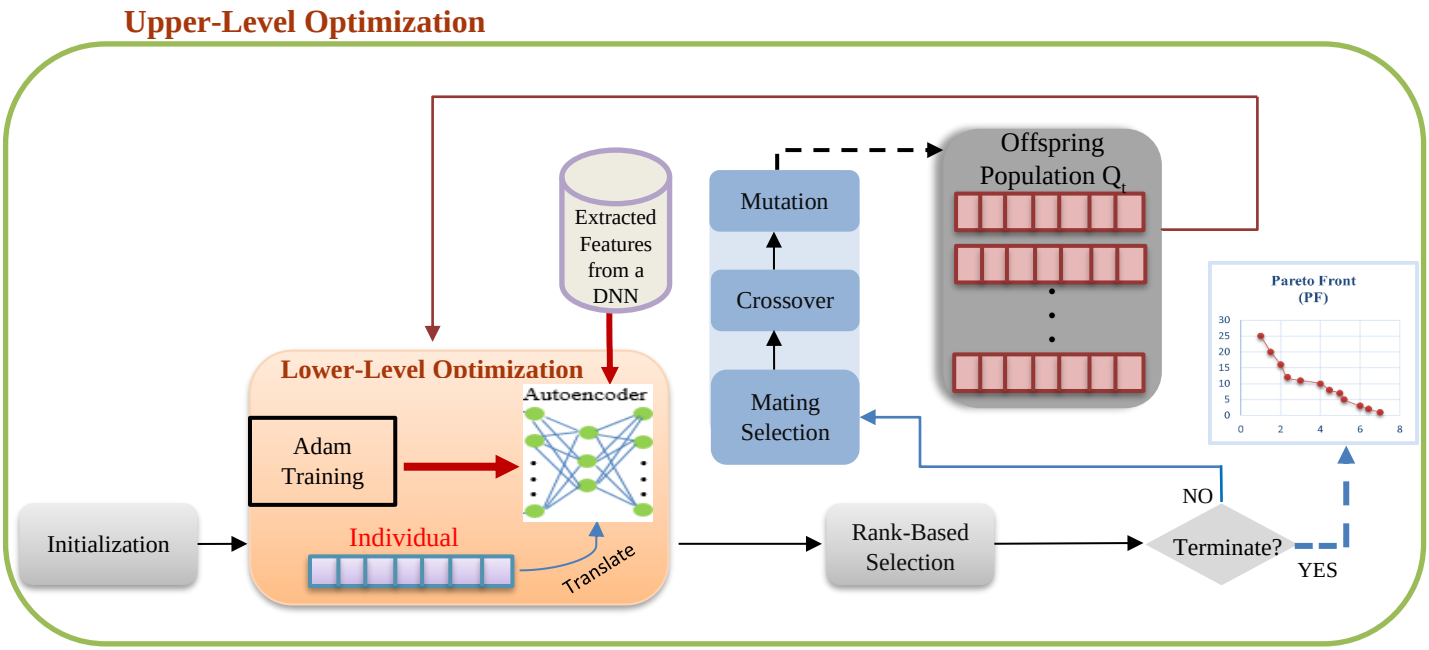

Figure 2.6: Schematic of the proposed method. First, initialized population of AEs are evaluated by the lower-level optimizer and then they will be ranked by non-dominated sorting and crowding distance algorithms into various pareto-fronts (PFs). If the termination condition is not met, then the upper-level optimizer will evolve the designed AEs by applying mating selection, crossover, and mutation operators to generate a new population of AE models. Finally, the new population is evaluated, and the same process is repeated until the stopping condition is met, and the non-dominated PF models are returned to be fully trained later.

Chapter 3. Designing Autoencoders for Fashion MNIST

In this chapter the details of the conducted experiments on Fashion MNIST and our case study, histopathology features that are extracted from KimiaNet, will be described along with the experimental results of each experiment to have a better understanding of the proposed approaches. The reason that we started with a benchmark dataset like Fashion MNIST before the main medical application was to propose an efficient scheme and then enhance it based on the histopathology dataset. We end this chapter with a comprehensive discussion over variety of interesting experiments that help us understand this framework and highlight the differences between each scheme.

We developed our evolving AE framework by starting with a benchmark dataset, Fashion MNIST [71], which has 10 classes of clothes labeled 0 to 9 consisted of 70,000 images of size 28 by 28 (i.e., input and output size of each AE is 784), in which 10,000 were dedicated to test set and the remaining 60,000 were for train set (a sample of 25 images from all class of Fashion MNIST is shown in Figure 3.1). We conducted several experiments consisted of various approaches to design the AE by only considering 2 objectives of RMSE and FLOPs in their design and alter its hyperparameters such as number of layers, number of epochs, population size, data size, etc. while our focus was on RMSE value of each AE based on the validation data. And we will discuss each scheme separately. But why we conducted these experiments? Well, our goal was to automatically design AEs that can discover the potent features of each image from a small sized code layer while having the lowest reconstruction error. For these set of experiments our default choice of hyperparameter settings were as follows: batch size of 4 with 5 training epochs since we had a dataset of size 200 or 500 images per each train, validation, and test sets. Depending

on the experiment we may change these values as it is clearly mentioned in each experiment. The general structure of each AE is to have at maximum 7 hidden layers with all having ReLU activation function as the common choice in most of models. However, it should be mentioned that we tried other activation functions and also other control parameter settings on a few sample architectures, and we realised that these settings are performing the best, so we selected them as our default choice of options. In addition, we investigate on both asymmetric and symmetric design of AEs and see how they perform.

Figure 3.1: A sample of 25 images from all classes of Fashion MNIST dataset

3.1 Investigation on Evolving Autoencoder

Starting with the first experiment where our goal is to design asymmetric AEs with the default setting to see how the EAE performs and designs AEs and whether it finds out any specific design pattern. For the upper-level optimizer setting we had a population size of 20 that went through 100 generations of evolution. It designed AEs with maximum 7 hidden layers and it could automatically remove layers, which is indicated by 0, and we did not force the optimizer in the process of design. So, it resulted asymmetric AEs that each layer had at most 300 neurons per layer. Each individual in the population was a chromosome of length 7 initialized with 7 random integer values in range 0 to 300. For the lower-level optimizer we set the batch size to 4, number of epochs to 5 and randomly selected 50 images per class of cloth and built a dataset of size 500 sampled data. The reason that we reduced the size of the data was two-fold. We did not see much difference compared to when we used the full data, and we wanted to accelerate the computation time, by doing so the fitness evaluation process got 120 times faster while it did not hurt the performance. As Table 3.1 depicts, we realized that as AEs are getting more complex the RMSE value is decreasing and the reconstructed output images are more similar to their original, accordingly. This proves that if you wish to have networks that do well on reducing the reconstruction error then your model should be complex.

Table 3.1: Investigation on Evolving Autoencoder. The PF solutions are consisted of asymmetric AEs that were generated based on the mentioned hyperparameters during evolution. The primary RMSE is the resulted after 5 training epochs with batch size 4 using 500 data samples per train and validation sets during evolution, while fully trained RMSE is resulted after 300 epochs on all the data samples with batch size of 512.

Experiment 1	Npop = 20	# Epochs = 5	
# Neurons per Layer = [0, 300]	# Generations = 100	Dataset Size = 500 × 2	Batch Size = 4
Architecture	Primary RMSE	Fully trained RMSE	FLOPS
784 ,[3, 98, 0, 13, 5, 127, 4], 784	0.08	0.04	1.65E+04
784 ,[10, 106, 11, 32, 12, 149, 2], 784	0.07	0.04	2.89E+04
784 ,[4, 50, 0, 27, 6, 190, 9], 784	0.07	0.03	2.95E+04
784 ,[11, 53, 68, 44, 21, 45, 10], 784	0.05	0.02	5.19E+04
784 ,[8, 0, 35, 73, 52, 30, 15], 784	0.05	0.02	5.33E+04
784 ,[9, 61, 47, 53, 16, 45, 27], 784	0.05	0.02	7.38E+04
784 ,[24, 12, 20, 89, 74, 64, 14], 784	0.04	0.02	8.86E+04
784 ,[17, 47, 37, 47, 22, 153, 30], 784	0.04	0.02	1.00E+05
784 ,[38, 59, 21, 87, 45, 64, 23], 784	0.04	0.01	1.23E+05
784 ,[13, 61, 34, 102, 47, 62, 49], 784	0.04	0.01	1.31E+05
784 ,[0, 35, 141, 73, 21, 33, 45], 784	0.04	0.01	1.63E+05
784 ,[0, 39, 95, 73, 86, 0, 40], 784	0.04	0.01	1.65E+05
784 ,[22, 43, 209, 38, 50, 66, 0], 784	0.04	0.01	1.84E+05
784 ,[41, 55, 49, 0, 19, 183, 79], 784	0.04	0.01	2.36E+05
784 ,[27, 83, 149, 83, 0, 105, 88], 784	0.04	0.01	2.70E+05
784 ,[45, 76, 95, 0, 0, 162, 80], 784	0.03	0.01	2.74E+05
784 ,[0, 79, 91, 81, 84, 0, 78], 784	0.03	0.01	3.02E+05
784 ,[35, 88, 117, 0, 0, 186, 94], 784	0.03	0.01	3.08E+05
784 ,[0, 75, 93, 53, 89, 0, 115], 784	0.03	0.01	3.52E+05
784 ,[0, 72, 102, 0, 0, 298, 105], 784	0.03	0.01	4.16E+05

3.2 Sensitivity Analysis of Hyperparameters

Here we are trying to tune some of the hyperparameters to see how they affect the evolution of AEs in terms of RMSE and FLOPs. Therefore, a series of experiments including population size which controls the number of AEs to be evolved at each generation, Dataset size, number of neurons per layer, number of training epochs, etc. are provided.

3.2.1 Npop, Data Size, Neurons

For the second experiment, we conducted the same approach in the previous experiment but with lower settings to accelerate the computation time even more. We halved the population size to 10 and the dataset size to 200, 20 images from each class, and reduced the number of neurons to range of 0 to 200 per layer while batch size and number of epochs remained the same 4 and 5, respectively. In Table 3.2, we see an increasing pattern in the first and last non-zero hidden layers of all the AEs designed by the optimizer, and it seems that the optimizer has understood that by having larger layers at these layers it can reduce RMSE better and avoid information loss. Now, we have 2.5 smaller data set compared to the previous scheme and 600 times faster computation (2 times for population size and 2.5 times for dataset size which in total is 5 times 120). Surprisingly, after we fully trained each AE for 300 epochs on all the data samples with batch size of 512 (this setting is used for fully training of all the experiments since we tried out various settings for these hyperparameters and this setting was the best) we were able to achieve comparable results for RMSE compared to the previous scheme. So, this scheme performed really well, and we try to adjust its setting and see if we can improve it in the future experiments.

Table 3.2: Investigation on Reduced Npop, Data Size, Neurons. Here we reduced the computation budget to accelerate the process and the PF solutions are consisted of asymmetric AEs. The primary RMSE is resulted after 5 training epochs with batch size 4 using 200 data samples per train and validation sets during evolution, while fully trained RMSE is resulted after 300 epochs on all the data samples with batch size of 512.

Experiment 2	Npop = 10	# Epochs = 5	
# Neurons per Layer = [0, 200]	# Generations = 100	Dataset Size = 200 × 2	Batch Size = 4
Architecture	Primary RMSE	Fully trained RMSE	FLOPS
784 ,[1, 122, 11, 87, 31, 68, 2], 784	0.15	0.07	1.94E+04
784 ,[5, 200, 9, 47, 36, 64, 13], 784	0.08	0.02	4.43E+04
784 ,[4, 162, 49, 59, 35, 61, 11], 784	0.06	0.02	5.62E+04
784 ,[10, 28, 75, 0, 109, 38, 12], 784	0.06	0.02	6.48E+04
784 ,[12, 0, 135, 0, 133, 59, 22], 784	0.06	0.01	1.11E+05
784 ,[10, 200, 16, 98, 0, 57, 0], 784	0.06	0.01	1.30E+05
784 ,[23, 0, 156, 0, 88, 120, 51], 784	0.05	0.01	1.84E+05
784 ,[26, 0, 125, 0, 74, 130, 83], 784	0.05	0.01	2.37E+05
784 ,[51, 165, 0, 100, 0, 83, 0], 784	0.05	0.01	2.77E+05
784 ,[69, 152, 0, 101, 0, 124, 0], 784	0.05	0.01	3.79E+05

3.2.2 Number of Generations

In the third experiment, we kept the second experiments setting except for the number of generations that we increased it from 100 to 500 to see how the upper-level optimizer will evolve the AEs' structures if it was given higher budget. As Table 3.3 shows, it neither reduced the complexity of the models that much nor removed a lot of layers as we expected to happen. Therefore, having a greater number of generations is not beneficial because the lower-level optimizer cannot reduce the RMSE value a lot by using a limited number of epochs and we cannot have lots of training epochs as it hurts the performance evaluation time. Thus, based on Table 3.3, it is better to stay with 100 generations and think of other ways to spend more budget. By the way, we tried smaller number of generations but since 100 was better we stayed with it and set it as a default value.

Table 3.3: Investigation on Number of Generations. Here we kept the reduced computation budget but increased the #Generations and the PF solutions are consisted of asymmetric AEs. The primary RMSE is resulted after 5 training epochs with batch size 4 using 200 data samples per train and validation sets during evolution, while fully trained RMSE is resulted after 300 epochs on all the data samples with batch size of 512.

Experiment 3	Npop = 10	# Epochs = 5	
# Neurons per Layer = [0, 200]	# Generations = 500	Dataset Size = 200 × 2	Batch Size = 4
Architecture	Primary RMSE	Fully trained RMSE	FLOPS
784 ,[1, 52, 3, 4, 31, 7, 1], 784	0.15	0.07	4.27E+03
784 ,[4, 35, 3, 13, 26, 29, 4], 784	0.08	0.05	1.55E+04
784 ,[3, 40, 0, 0, 0, 8, 0], 784	0.08	0.03	1.81E+04
784 ,[5, 51, 0, 20, 45, 16, 12], 784	0.08	0.03	3.28E+04
784 ,[3, 40, 0, 38, 43, 37, 12], 784	0.08	0.03	3.41E+04
784 ,[0, 30, 0, 19, 0, 9, 0], 784	0.06	0.02	6.26E+04
784 ,[11, 72, 37, 82, 117, 41, 0], 784	0.06	0.01	1.23E+05
784 ,[0, 48, 109, 38, 25, 88, 0], 784	0.05	0.01	2.38E+05
784 ,[0, 65, 96, 97, 97, 0, 96], 784	0.05	0.01	3.21E+05
784 ,[0, 46, 91, 106, 124, 0, 168], 784	0.05	0.01	4.31E+05

3.2.3 Box-constraint for Increasing Number of Neurons

In our fourth experiment, we again take the same setting of experiment 2 but we increase the maximum number of neurons per layer from 200 to 700 to see how it impacts the architecture of each AE and its RMSE. In other words, we are allowing the upper-level optimizer to search for the number of neurons per each layer in a larger box-constraint interval. Obviously, as shown in Table 3.4, the complexity of the models has increased a lot, almost doubled, and most of the models had all of their hidden layers occupied and we did not see the optimizer eliminating many layers and the reason is that now the range of neurons is between 0 to 700 so it is harder to find more compressed networks. In addition, we did not get any improvement in terms of RMSE before and after fully training.

Table 3.4: Investigation on Box-constraint by Increasing Number of Neurons. Here we kept the reduced computation budget but increased the #Neurons and the PF solutions are consisted of asymmetric AEs. The primary RMSE is resulted after 5 training epochs with batch size 4 using 200 data samples per train and validation sets during evolution, while fully trained RMSE is resulted after 300 epochs on all the data samples with batch size of 512.

Experiment 4	Npop = 10	# Epochs = 5	
# Neurons per Layer = [0, 700]	# Generations = 100	Dataset Size = 200 × 2	Batch Size = 4
Architecture	Primary RMSE	Fully trained RMSE	FLOPS
784 ,[4, 174, 25, 112, 76, 109, 7], 784	0.08	0.03	6.81E+04
784 ,[29, 42, 21, 60, 17, 84, 14], 784	0.08	0.02	8.14E+04
784 ,[42, 72, 17, 138, 0, 95, 12], 784	0.06	0.02	1.26E+05
784 ,[40, 0, 76, 73, 98, 223, 29], 784	0.06	0.01	1.96E+05
784 ,[0, 86, 72, 0, 107, 167, 40], 784	0.05	0.01	2.74E+05
784 ,[19, 252, 23, 83, 63, 108, 118], 784	0.05	0.01	2.89E+05
784 ,[71, 386, 61, 80, 234, 199, 0], 784	0.05	0.01	6.66E+05
784 ,[45, 299, 67, 224, 414, 99, 197], 784	0.05	0.01	7.83E+05
784 ,[72, 36, 28, 295, 547, 206, 143], 784	0.05	0.01	9.68E+05

3.2.4 Training Epochs

For the fifth experiment, we got the base settings from the second experiment again, but we increased the number of training epochs in the lower-level optimizer during evolution with the hope of having smaller value of RMSE. As result show in Table 3.5, it did not make any difference in the values of RMSE compared to the time that we had 5 epochs and the downside of this scheme compared to the second scheme was that we slowed down to computation time 4 times as we increased the number of epochs and we realised that even by having 20 epochs during evolution we cannot find AEs with smaller RMSE and smaller values of RMSE will show up after 100 epochs which is the case for fully training and is not applicable to be done during evolution. So far, we have realized that increasing the number of epochs, number of generations, or number of neurons per layer have not helped a lot in designing very impressive AEs.

Table 3.5: Investigation on Training Epochs. Here we kept the reduced computation budget but increased the #Epochs during evolution and the PF solutions are consisted of asymmetric AEs. The primary RMSE is resulted after 20 training epochs with batch size 4 using 200 data samples per train and validation sets during evolution, while fully trained RMSE is resulted after 300 epochs on all the data samples with batch size of 512.

Experiment 5	Npop = 10	# Epochs = 20	
# Neurons per Layer = [0, 200]	# Generations = 100	Dataset Size = 200 × 2	Batch Size = 4
Architecture	Primary RMSE	Fully trained RMSE	FLOPS
784 ,[1, 87, 18, 20, 25, 34, 3], 784	0.08	0.05	1.32E+04
784 ,[3, 100, 12, 51, 25, 38, 3], 784	0.08	0.04	1.83E+04
784 ,[4, 88, 21, 16, 24, 44, 5], 784	0.08	0.03	2.25E+04
784 ,[7, 73, 9, 88, 14, 36, 13], 784	0.08	0.02	3.97E+04
784 ,[7, 32, 13, 84, 52, 37, 15], 784	0.08	0.02	5.17E+04
784 ,[28, 145, 18, 53, 33, 50, 15], 784	0.08	0.02	9.10E+04
784 ,[24, 144, 69, 53, 31, 0, 51], 784	0.05	0.01	1.58E+05
784 ,[53, 0, 44, 113, 171, 57, 0], 784	0.05	0.01	2.45E+05
784 ,[57, 0, 32, 102, 200, 93, 0], 784	0.05	0.01	3.23E+05
784 ,[62, 35, 0, 160, 154, 151, 0], 784	0.05	0.01	4.45E+05

3.2.5 Number of Layers

In the sixth experiment we wanted to try having very deep symmetric AEs of 21 hidden layers with at most 500 nodes per each while having the code layer (here we mean the middle layer) of size 1 to see if we can get any interesting design. Thus, we doubled the batch size and number of epochs to 8 and 10, respectively, since we have deeper networks and wish to give them a better chance to be evaluated fairly. Also, we are trying to not to make it computationally harsh that is why we selected small values. As we are designing symmetric AEs in this scheme and the encoder and decoder are having the same number of layers and neurons per each layer, we have a 11-dimensional chromosome representation per AE filled with random integer numbers in range 0 to 500 while the last gene is set to 1 as it shows the code layer. Then the first 10 genes are used to construct the decoder as mirror of the encoder part. We increased the dataset size from 200 to 1000 by sampling 100 images from

each class. As expected, the complexity of the designed models was higher while it did not help with the RMSE that much and it took about a week to finish the evaluations of all the AEs generated through the 100 generations. Table 3.6 shows that for getting smaller RMSE you do not need very deep AEs as the resulted RMSE values were worse than the previous schemes.

Table 3.6: Investigation on Deep Autoencoders. Here we wanted to have deep symmetric AEs, so we adjusted the hyperparameters. The primary RMSE is the result after 10 training epochs with batch size 8 using 1000 data samples per train and validation sets during evolution, while fully trained RMSE is resulted after 300 epochs on all the data samples with batch size of 512.

Experiment 6	Npop = 20	# Epochs = 10	
# Neurons per Layer = [0, 500]	# Generations = 100	Dataset Size = 1000 × 2	Batch Size = 8
Architecture	Primary RMSE	Fully trained RMSE	FLOPS
784 ,[2, 163, 22, 218, 10, 84, 1, 361, 12, 247, 1, 247, 12, 361, 1, 84, 10, 218, 22, 163, 2], 784	0.15	0.09	8.51E+04
784 ,[1, 197, 21, 52, 40, 9, 53, 28, 72, 197, 1, 197, 72, 28, 53, 9, 40, 52, 21, 197, 1], 784	0.15	0.09	1.08E+05
784 ,[1, 212, 3, 48, 43, 6, 33, 23, 98, 230, 1, 230, 98, 23, 33, 6, 43, 48, 3, 212, 1], 784	0.15	0.06	1.20E+05
784 ,[16, 143, 16, 110, 19, 19, 0, 349, 6, 81, 1, 81, 6, 349, 0, 19, 19, 110, 16, 143, 16], 784	0.09	0.09	1.22E+05
784 ,[28, 34, 15, 217, 23, 29, 6, 285, 8, 88 , 1, 88, 8, 285, 6,29, 23, 217, 15, 34, 28], 784	0.09	0.04	1.49E+05
784 ,[29, 64, 18, 229, 16, 19, 6, 234, 8, 82, 1, 82, 8, 234, 6, 19, 16, 229, 18, 64, 29], 784	0.08	0.09	1.52E+05
784 ,[31, 115, 29, 37, 7, 17, 127, 10, 100, 175, 1, 175, 100, 10, 127,17, 7, 37, 29,115, 31], 784	0.08	0.04	2.19E+05
784 ,[41, 120, 30, 61, 29, 15, 62, 19, 56, 187, 1, 187, 56, 19, 62, 15, 29, 61, 30, 120, 41], 784	0.08	0.04	2.34E+05
784 ,[39, 116, 23, 53, 61, 15, 85, 48, 151, 202, 1, 202, 151, 48, 85, 15, 61, 53, 23, 116, 39], 784	0.08	0.04	3.46E+05
784 ,[69, 37, 127, 133, 24, 29, 0, 346, 10, 85, 1, 85, 10, 346, 0, 29,24, 133, 127, 37, 69], 784	0.08	0.04	3.86E+05
784 ,[71, 24, 115, 137, 22, 24, 0, 358, 22, 155, 1, 155, 22, 358, 0, 24, 22, 137, 115, 24, 71], 784	0.08	0.05	3.98E+05
784 ,[28, 33, 77, 236, 21, 31, 19, 268, 114, 146 , 1, 146, 114, 268, 19, 31, 21, 236, 77, 33, 28], 784	0.08	0.04	4.09E+05
784 ,[0, 148, 17, 73, 28, 14, 50, 46, 69, 154, 1,154, 69, 46, 50, 14, 28, 73, 17, 148, 0], 784	0.08	0.04	5.57E+05
784 ,[101, 31, 25, 183, 73, 139, 140, 231, 0, 134, 1, 134, 0, 231, 140,139, 73, 183, 25, 31, 101], 784	0.08	0.04	7.76E+05
784 ,[150, 111, 68, 187, 31, 175, 74, 162, 0, 127, 1, 127, 0, 162, 74, 175, 31, 187, 68, 111, 150], 784	0.07	0.04	8.46E+05
784 ,[159, 77, 0, 170, 69, 49, 133, 231, 72, 434, 1, 434, 72, 231, 133, 49, 69, 170, 0, 77, 159], 784	0.07	0.04	1.00E+06
784 ,[0, 137, 120, 148, 144, 0, 238, 92, 295, 37, 1, 37, 295, 92, 238, 0, 144, 148, 120, 137,0], 784	0.06	0.04	1.03E+06

3.3 Designing Undercomplete Autoencoders

In our coming experiment, we decided to make more accurate AEs with smaller code layer (i.e., middle layer). Thus, we need to restrict the optimizer design choice by forcing it to follow undercomplete design pattern of AE. Therefore, from now on we only focus on designing symmetric undercomplete AEs with maximum of 7 layers that the upper-level optimizer can remove layers from it and adjust the number of neurons at each layer, dynamically, by changing the chromosome representation from random integer numbers to random continuous values. So, the second chromosome encoding as illustrated in Figure 2.3 will be used. Thus, each chromosome is of length 4 consisted of first, second, third hidden layers, and code layer and each gene of the chromosome is a real value that is multiplied by the number of neurons in the previous layer[79]. For instance, the first hidden layer is formulated as the first gene multiplied to 784, the second hidden layer is the second gene multiplied to the size of the first hidden layer and so on. Now having built hidden layer 1 up to the code layer we simply mirror the structure from hidden layer 3 to 1 to build an AE with 7 hidden layers. So, we use the same setting from the second experiment but with population size of 20 instead of 10 to have more proposed options on the final PF. Thus, in the seventh experiment we applied an initialization constraint to force the upper-level optimizer not to design sparse models at the first generation and gradually decrease their complexity. Table 3.7 shows very interesting results, we realized that the optimizer had found a way to have sparse models that most hidden layers were removed (i.e., the hidden layer's size was 0)

while having really good RMSE values. It had learned to zero out almost all the hidden layers except for the first one that the gene value is multiplied to 784 (i.e., input layer) and put a lot of neurons in that layer which is then mirrored for the decoder part. By having models like 784, 406, 0, 0, 0, 0, 0, 406, 784 we only have 2 hidden layers of size 406 so the data loss is super low and the RMSE value is close to 0, consequently. But that is not what we aim for, because we need to have AEs with smallest code layers that can extract the most impactful features, not having too many neurons in a few layers.

Table 3.7: Investigation on Undercomplete Autoencoders. Here we kept the reduced computation budget, and the PF solutions are consisted of undercomplete AEs. The primary RMSE is the result after 5 training epochs with batch size 4 using 200 data samples per train and validation sets during evolution, while fully trained RMSE is resulted after 300 epochs on all the data samples with batch size of 512.

Experiment 7	Npop = 20	# Epochs = 5	
# Neurons per Layer = [0, 1), Initial = [0.5, 1)	# Generations = 100	Dataset Size = 200 × 2	Batch Size = 4
Architecture	Primary RMSE	Fully trained RMSE	FLOPS
784 ,[1, 0, 0, 0, 0, 0, 1], 784	0.15	0.06	3.14E+03
784 ,[3, 1, 0, 0, 0, 1, 3], 784	0.14	0.06	9.42E+03
784 ,[3, 0, 0, 0, 0, 0, 3], 784	0.09	0.06	9.43E+03
784 ,[4, 0, 0, 0, 0, 0, 4], 784	0.08	0.03	1.26E+04
784 ,[8, 0, 0, 0, 0, 0, 8], 784	0.07	0.03	2.52E+04
784 ,[20, 0, 0, 0, 0, 0, 20], 784	0.06	0.02	6.35E+04
784 ,[30, 0, 0, 0, 0, 0, 30], 784	0.05	0.01	9.59E+04
784 ,[39, 0, 0, 0, 0, 0, 39], 784	0.05	0.01	1.25E+05
784 ,[57, 0, 0, 0, 0, 0, 57], 784	0.05	0.01	1.85E+05
784 ,[65, 0, 0, 0, 0, 0, 65], 784	0.05	0.01	2.12E+05
784 ,[83, 0, 0, 0, 0, 0, 83], 784	0.04	0.01	2.74E+05
784 ,[96, 0, 0, 0, 0, 0, 96], 784	0.04	0.01	3.19E+05
784 ,[111, 0, 0, 0, 0, 0, 111], 784	0.04	0.01	3.73E+05
784 ,[154, 0, 0, 0, 0, 0, 154], 784	0.04	0.005	5.30E+05
784 ,[171, 0, 0, 0, 0, 0, 171], 784	0.04	0.005	5.95E+05
784 ,[217, 0, 0, 0, 0, 0, 217], 784	0.04	0.004	7.75E+05
784 ,[229, 0, 0, 0, 0, 0, 229], 784	0.04	0.004	8.23E+05
784 ,[279, 0, 0, 0, 0, 0, 279], 784	0.04	0.005	1.03E+06
784 ,[280, 0, 0, 0, 0, 0, 280], 784	0.04	0.005	1.03E+06
784 ,[406, 0, 0, 0, 0, 0, 406], 784	0.03	0.004	1.60E+06

3.4 Box Constraint Handling

The reason behind these 0 (i.e., removed layers) in the previous experiment was that we multiplied the gene value by the size of the previous hidden layer and if we had a hidden layer removed (i.e., 0) then the whole hidden layers after that would be removed (since their gene value would be multiplied to 0). So, in the eighth experiment, we rectify that problem by considering the multiplication of each gene value to the last non-zero hidden layer (i.e., the hidden layer that we actually have). Also, we made another enhancement to our mutation operator to clip those gene values that get out of the boundary of [0, 1) to be in range of [unviolated boundary, gene value of the best individual] instead of being set to either 0 or 1. For instance, if the mutated gene value is smaller than 0, then we sample a new real value from the range of [gene value of the best individual, 1); and, if the mutated gene value is larger than 1, we sample a new real value from the range of [0, gene value of the best individual). The reason for using such condition is to avoid having a lot of duplicated layers with same number of neurons caused by their clipped values to be 0 or 1 and instead we use a random value to let it have more exploration capability. The provided result in Table 3.8 shows a great move toward our goal of having compressed models with smaller RMSE values.

Table 3.8: Box Constraint Handling. Here we kept the reduced computation budget, and the PF solutions are consisted of undercomplete AEs designed in a much enhanced way during the mutation operator. The primary RMSE is the result after 5 training epochs with batch size 4 using 200 data samples per train and validation sets during evolution, while fully trained RMSE is resulted after 300 epochs on all the data samples with batch size of 512.

Experiment 8	Npop = 20	# Epochs = 5	
# Neurons per Layer = [0, 1), Initial = [0.5, 1)	# Generations = 100	Dataset Size = 200 × 2	Batch Size = 4
Architecture	Primary RMSE	Fully trained RMSE	FLOPS
784 ,[1, 0, 0, 0, 0, 0, 1], 784	0.15	0.09	3.14E+03
784 ,[2, 0, 1, 1, 1, 0, 2], 784	0.14	0.06	6.28E+03
784 ,[8, 6, 0, 5, 0, 6, 8], 784	0.09	0.03	2.54E+04
784 ,[9, 6, 0, 4, 0, 6, 9], 784	0.09	0.03	2.85E+04
784 ,[17, 12, 0, 9, 0, 12, 17], 784	0.08	0.02	5.46E+04
784 ,[22, 16, 0, 12, 0, 16, 22], 784	0.07	0.02	7.12E+04
784 ,[27, 11, 10, 7, 10, 11, 27], 784	0.07	0.02	8.66E+04
784 ,[33, 27, 0, 15, 0, 27, 33], 784	0.06	0.01	1.09E+05
784 ,[42, 32, 0, 21, 0, 32, 42], 784	0.05	0.01	1.40E+05
784 ,[50, 34, 0, 19, 0, 34, 50], 784	0.05	0.01	1.66E+05
784 ,[71, 65, 29, 24, 29, 65, 71], 784	0.05	0.01	2.51E+05
784 ,[78, 63, 31, 13, 31, 63, 78], 784	0.05	0.01	2.74E+05
784 ,[101, 69, 28, 15, 28, 69, 101], 784	0.05	0.01	3.54E+05
784 ,[109, 104, 59, 29, 59, 104, 109], 784	0.05	0.01	4.19E+05
784 ,[124, 88, 0, 55, 0, 88, 124], 784	0.05	0.01	4.52E+05
784 ,[127, 87, 0, 47, 0, 87, 127], 784	0.05	0.01	4.59E+05
784 ,[147, 126, 0, 89, 0, 126, 147], 784	0.05	0.01	5.80E+05
784 ,[170, 139, 0, 100, 0, 139, 170], 784	0.05	0.01	6.83E+05
784 ,[212, 109, 0, 66, 0, 109, 212], 784	0.05	0.01	7.86E+05
784 ,[231, 139, 0, 87, 0, 139, 231], 784	0.05	0.01	9.01E+05

3.5 Investigation on Reconstruction of Mean Image

In the nineth experiment, we took the idea of using mean image of each class for training of each AE [65]. We take the mean of each pixel among all the training and validation images for each class of cloth and instead of reconstructing the original input image we reconstruct the mean image of that input image class, as shown in Figure 3.2. And the idea behind this approach is to make the AE learn the true pattern of each class and accelerate learning by compressing the models and extracting the significant features. As Table 3.9 shows, we got very compressed models compared to other schemes with comparable RMSE results. However, we realised that this idea is not a good candidate for medical imaging since we cannot have a clear pattern of the tumor by taking pixel-wise mean for the purpose of training our models and it might also guide the optimizer in a wrong direction to not focus on the region of interest where the tumor can be diagnosed.

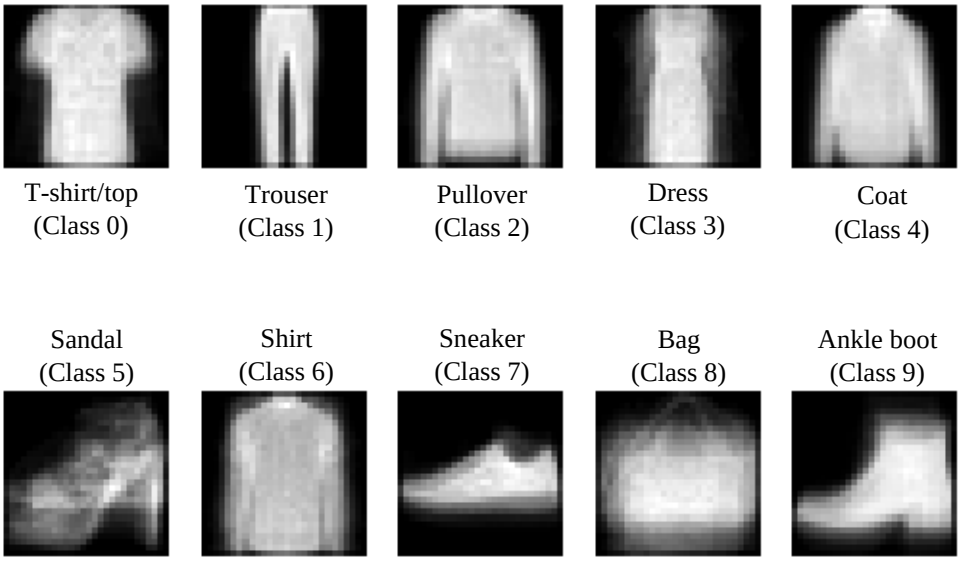

Figure 3.2: Mean Image of each Fashion MNIST Class

Table 3.9: Investigation on Reconstruction of Mean Image. Here we kept the reduced computation budget, and the PF solutions are consisted of undercomplete AEs designed in a much enhanced way during the mutation operator. Only during evolution, we used the mean images to evaluate each AE. The primary RMSE is the result after 5 training epochs with batch size 4 using 200 data samples per train and validation sets during evolution, while fully trained RMSE is resulted after 300 epochs on all the data samples with batch size of 512.

Experiment 9	Npop = 20	# Epochs = 5	
# Neurons per Layer = [0, 1), Initial = [0.5, 1)	# Generations = 100	Dataset Size = 200 × 2	Batch Size = 4
Architecture	Primary RMSE	Fully trained RMSE	FLOPS
784 ,[1, 0, 0, 0, 0, 0, 1], 784	0.14	0.06	3.14E+03
784 ,[5, 5, 3, 2, 3, 5, 5], 784	0.14	0.06	1.59E+04
784 ,[6, 0, 0, 4, 0, 0, 6], 784	0.09	0.04	1.89E+04
784 ,[8, 7, 0, 5, 0, 7, 8], 784	0.09	0.02	2.55E+04
784 ,[9, 8, 0, 5, 0, 8, 9], 784	0.09	0.02	2.87E+04
784 ,[10, 10, 0, 6, 0, 10, 10], 784	0.08	0.02	3.20E+04
784 ,[19, 16, 12, 7, 12, 16, 19], 784	0.08	0.02	6.19E+04
784 ,[19, 17, 13, 7, 13, 17, 19], 784	0.08	0.03	6.21E+04
784 ,[21, 19, 13, 7, 13, 19, 21], 784	0.08	0.02	6.88E+04
784 ,[23, 17, 12, 7, 12, 17, 23], 784	0.06	0.02	7.48E+04
784 ,[23, 22, 15, 8, 15, 22, 23], 784	0.06	0.02	7.60E+04
784 ,[36, 28, 20, 14, 20, 28, 36], 784	0.06	0.02	1.20E+05
784 ,[39, 0, 13, 7, 13, 0, 39], 784	0.06	0.02	1.25E+05
784 ,[43, 31, 28, 17, 28, 31, 43], 784	0.06	0.01	1.46E+05
784 ,[46, 35, 25, 17, 25, 35, 46], 784	0.05	0.01	1.56E+05
784 ,[50, 0, 16, 7, 16, 0, 50], 784	0.05	0.02	1.60E+05
784 ,[0, 140, 43, 32, 43, 140, 0], 784	0.05	0.01	4.69E+05
784 ,[157, 64, 0, 30, 0, 64, 157], 784	0.05	0.01	5.40E+05
784 ,[378, 111, 0, 71, 0, 111, 378], 784	0.04	0.01	1.38E+06

3.6 Investigation on Dataset Size Reduction

For the tenth and eleventh experiments, we concentrated on reducing the dataset size of eighth experiment that was used during evolution from 200 to 50 and 20 by selecting 5 and 2 images per class of cloth, respectively. The provided results in Table 3.10 and 3.11 show an increase over the number of neurons per layer and complexity of the designed models by having most of the hidden layers filled with a lot of neurons, accordingly. Thus, having smaller dataset will definitely reduce the computation time but it did not help improving the performance and design of the

models in our case. So, it is better to keep using the 200 total data samples during evolution while we use the whole dataset for fully training of the final PF solutions, as before.

Table 3.10: Investigation on Dataset Size Reduction. Here we kept the reduced computation budget, and the PF solutions are consisted of undercomplete AEs designed in a much enhanced way during the mutation operator. The primary RMSE is the result after 5 training epochs with batch size 4 using 50 data samples per train and validation sets during evolution, while fully trained RMSE is resulted after 300 epochs on all the data samples with batch size of 512.

Experiment 10	Npop = 20	# Epochs = 5	
# Neurons per Layer = [0, 1), Initial = [0.5, 1)	# Generations = 100	Dataset Size = 50 × 2	Batch Size = 4
Architecture	Primary RMSE	Fully trained RMSE	FLOPS
784 ,[1, 0, 0, 0, 0, 0, 1], 784	0.16	0.09	3.14E+03
784 ,[1, 0, 0, 1, 0, 0, 1], 784	0.16	0.06	3.14E+03
784 ,[4, 1, 0, 1, 0, 1, 4], 784	0.16	0.05	1.26E+04
784 ,[17, 14, 9, 8, 9, 14, 17], 784	0.1	0.02	5.51E+04
784 ,[37, 25, 22, 19, 22, 25, 37], 784	0.09	0.01	1.24E+05
784 ,[39, 29, 24, 20, 24, 29, 39], 784	0.09	0.01	1.32E+05
784 ,[51, 20, 9, 5, 9, 20, 51], 784	0.09	0.02	1.65E+05
784 ,[50, 29, 18, 17, 18, 29, 50], 784	0.09	0.01	1.66E+05
784 ,[74, 54, 41, 34, 41, 54, 74], 784	0.08	0.01	2.62E+05
784 ,[0, 106, 57, 45, 57, 106, 0], 784	0.07	0.01	3.67E+05
784 ,[0, 123, 50, 40, 50, 123, 0], 784	0.07	0.01	4.18E+05
784 ,[0, 138, 71, 61, 71, 138, 0], 784	0.07	0.01	4.89E+05
784 ,[0, 148, 93, 77, 93, 148, 0], 784	0.06	0.01	5.48E+05
784 ,[0, 183, 115, 96, 115, 183, 0], 784	0.06	0.01	7.02E+05
784 ,[0, 243, 94, 82, 94, 243, 0], 784	0.06	0.01	8.84E+05
784 ,[0, 237, 123, 107, 123, 237, 0], 784	0.06	0.01	9.12E+05
784 ,[0, 264, 116, 91, 116, 264, 0], 784	0.06	0.01	9.93E+05
784 ,[0, 371, 172, 95, 172, 371, 0], 784	0.06	0.01	1.48E+06
784 ,[0, 315, 253, 212, 253, 315, 0], 784	0.06	0.004	1.52E+06
784 ,[0, 429, 289, 247, 289, 429, 0], 784	0.06	0.004	2.13E+06

Table 3.11: Investigation on Dataset Size Reduction. Here we kept the reduced computation budget, and the PF solutions are consisted of undercomplete AEs designed in a much enhanced way during the mutation operator. The primary RMSE is the result after 5 training epochs with batch size 4 using 20 data samples per train and validation sets during evolution, while fully trained RMSE is resulted after 300 epochs on all the data samples with batch size of 512.

Experiment 11	Npop = 20	# Epochs = 5	
# Neurons per Layer = [0, 1), Initial = [0.5, 1)	# Generations = 100	Dataset Size = 20 × 2	Batch Size = 4
Architecture	Primary RMSE	Fully trained RMSE	FLOPS
784 ,[1, 0, 0, 0, 0, 0, 1], 784	0.16	0.09	3.14E+03
784 ,[8, 5, 3, 0, 3, 5, 8], 784	0.15	0.03	2.53E+04
784 ,[16, 10, 4, 0, 4, 10, 16], 784	0.15	0.02	5.10E+04
784 ,[19, 13, 5, 1, 5, 13, 19], 784	0.15	0.04	6.09E+04
784 ,[35, 22, 10, 4, 10, 22, 35], 784	0.15	0.02	1.14E+05
784 ,[45, 29, 15, 9, 15, 29, 45], 784	0.12	0.02	1.49E+05
784 ,[51, 27, 21, 10, 21, 27, 51], 784	0.11	0.02	1.69E+05
784 ,[68, 49, 29, 21, 29, 49, 68], 784	0.11	0.01	2.35E+05
784 ,[131, 52, 26, 13, 26, 52, 131], 784	0.11	0.01	4.45E+05
784 ,[147, 102, 56, 43, 56, 102, 147], 784	0.1	0.01	5.53E+05
784 ,[182, 60, 19, 8, 19, 60, 182], 784	0.1	0.01	6.20E+05
784 ,[225, 194, 107, 53, 107, 194, 225], 784	0.09	0.01	9.86E+05
784 ,[290, 176, 79, 63, 79, 176, 290], 784	0.09	0.01	1.19E+06
784 ,[328, 185, 136, 80, 136, 185, 328], 784	0.09	0.01	1.42E+06
784 ,[372, 194, 130, 107, 130, 194, 372], 784	0.09	0.01	1.61E+06
784 ,[471, 194, 92, 65, 92, 194, 471], 784	0.09	0.01	1.94E+06
784 ,[522, 185, 155, 107, 155, 185, 522], 784	0.09	0.01	2.20E+06
784 ,[512, 299, 240, 185, 240, 299, 512], 784	0.09	0.01	2.68E+06
784 ,[523, 323, 213, 155, 213, 323, 523], 784	0.09	0.01	2.72E+06
784 ,[622, 397, 353, 185, 353, 397, 622], 784	0.08	0.004	3.76E+06

3.7 Investigation on One Training Epoch

For the final, twelfth, experiment, we had the idea of having 1 training epoch during the evolution to reduce the expensive performance evaluation phase as explained about NAS methods. Therefore, having 1 epoch is like judging the model based on its architecture rather that its weight values and it is very close to having no training. So, we used 1 epoch instead of 5 and the computation pace got momentum 5 times while we still have the same performance of the models. This was a promising achievement since we can conduct our experiments swiftly and it does not require a

lot of resources. As shown in Table 3.12, the RMSE values during evolution of 1 training epoch were high but when we fully trained them we got really reduced RMSE values and it was even better than experiment 8th, in which we had 5 epochs, and also we got a few AEs with RMSE very close to 0.

Table 3.12: Investigation on One Training Epoch. Here we kept the reduced computation budget, and the PF solutions are consisted of undercomplete AEs designed in a much enhanced way during the mutation operator. The primary RMSE is the result after 1 training epochs with batch size 4 using 200 data samples per train and validation sets during evolution, while fully trained RMSE is resulted after 300 epochs on all the data samples with batch size of 512.

Experiment 12	Npop = 20	# Epochs = 1	
# Neurons per Layer = [0, 1), Initial = [0.5, 1)	# Generations = 100	Dataset Size = 200 × 2	Batch Size = 4
Architecture	Primary RMSE	Fully trained RMSE	FLOPS
784 ,[1, 0, 0, 0, 0, 0, 1], 784	0.16	0.09	3.14E+03
784 ,[2, 1, 0, 0, 0, 1, 2], 784	0.14	0.06	6.28E+03
784 ,[5, 4, 2, 1, 2, 4, 5], 784	0.14	0.05	1.58E+04
784 ,[5, 4, 3, 2, 3, 4, 5], 784	0.12	0.06	1.58E+04
784 ,[17, 8, 1, 0, 1, 8, 17], 784	0.1	0.04	5.39E+04
784 ,[34, 19, 4, 2, 4, 19, 34], 784	0.1	0.04	1.10E+05
784 ,[45, 22, 9, 2, 9, 22, 45], 784	0.1	0.03	1.46E+05
784 ,[58, 33, 26, 24, 26, 33, 58], 784	0.1	0.01	1.95E+05
784 ,[80, 26, 12, 10, 12, 26, 80], 784	0.1	0.02	2.61E+05
784 ,[132, 40, 14, 13, 14, 40, 132], 784	0.1	0.01	4.38E+05
784 ,[149, 65, 58, 48, 58, 65, 149], 784	0.09	0.01	5.32E+05
784 ,[0, 188, 82, 59, 82, 188, 0], 784	0.09	0.01	6.71E+05
784 ,[207, 86, 66, 47, 66, 86, 207], 784	0.07	0.01	7.55E+05
784 ,[0, 249, 117, 93, 117, 249, 0], 784	0.07	0.01	9.41E+05
784 ,[0, 273, 90, 53, 90, 273, 0], 784	0.07	0.01	9.73E+05
784 ,[0, 310, 132, 80, 132, 310, 0], 784	0.07	0.01	1.18E+06
784 ,[0, 313, 199, 166, 199, 313, 0], 784	0.06	0.005	1.36E+06
784 ,[0, 397, 177, 61, 177, 397, 0], 784	0.06	0.01	1.57E+06
784 ,[0, 419, 186, 71, 186, 419, 0], 784	0.06	0.01	1.68E+06
784 ,[0, 469, 224, 126, 224, 469, 0], 784	0.06	0.004	2.00E+06

3.8 Discussion

After conducting a variety of experiment on Fashion MNIST dataset and tuning a lot of hyperparameters we can conclude that based on the real-valued chromosome encoding that we used to design undercomplete AEs, having one

training epoch during evolution is really promising, both in terms of the designed models and computation time. Besides, having smaller dataset size like 400 data samples (i.e., 200 samples per train, and validation) helped a lot to facilitate the evolving process and smaller batch-size did not hurt the final quality of the models. Furthermore, spending more budget on the evolutionary generation is not as beneficial as it seems to be and from a certain point it does not really affect the results a lot.

Chapter 4. Designing Autoencoders for Histopathology Features

In this section we explain all the conducted experiments on KimiaNet [60] extracted features for the purpose of dimensionality reduction and improved classification accuracy in medical imaging. As shown in Figure 4.1, since WSIs are gigapixel images, each WSI is divided to 135 patches on average to be fed to the DNN and each patch is represented by 1024 features. we extract the latent vector from the last pooling layer per each patch and take the mean of all the patches per each WSI to propose a single vector that shows the representation of its corresponding WSI and feed it as input to our framework.

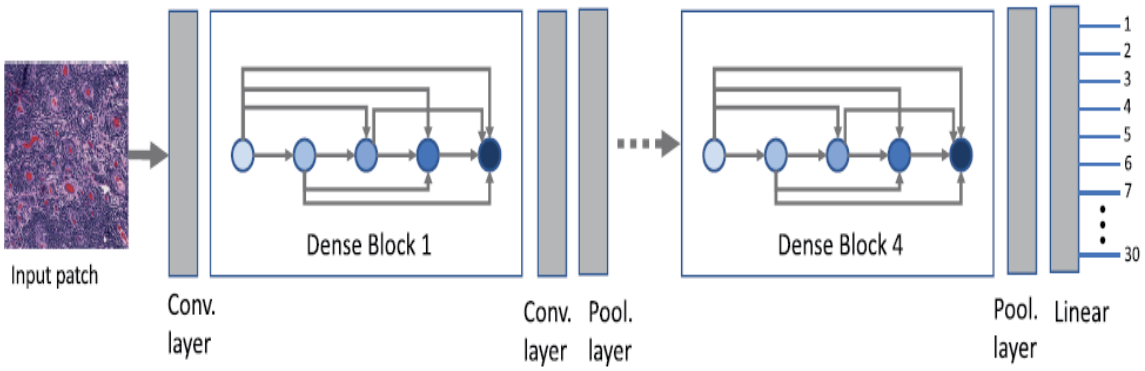

Figure 4.1: DenseNet architecture of KimiaNet

For the purpose of designing KimiaNet, they fine-tuned DenseNet based on the pathology WSI's and performed patch selection on each WSI and represented the input of the networks as a Mosaic of high-cellular patches. And retained it based on 30 class of tumors for 20 epochs. At the end, they present the output of the last pooling layer as the embedded output vector, which is of size 1024, because their goal was representation learning of histopathology WSI's not classification. They have utilized three datasets such as The Cancer Genome Atlas (TCGA), Endometrium, and colorectal cancer datasets for training, validating, and testing of their model. A huge part of TCGA was used for training and

remaining was left for testing but the other two datasets were exclusively used for testing. The datasets that were used in our experiments are from the extracted features from the last pooling layer of KimiaNet. For each tumor type we have train, validation, and test sets in which we use the training set to train each model and validation set to validate the model and, finally, test set to evaluate the model based on unseen data to see the generalization of each model. However, the amount of data per each tumor type is almost less than 1,000 WSIs and this is a critical issue for end-to-end DNNs, but our approach can handle it easily.

Therefore, the input and output layers size are now 1024 instead of 784 and the same evolutionary process is applied again. We use the same scheme from MNIST dataset discussed earlier and consider each chromosome's gene as a real value which is multiplied by its previous non-zero layer to construct the number of neurons in that layer. As depicted in Figure 4.2, the general architecture of each AE follows an undercomplete AE pattern with the inner most layer being the smallest layer (i.e., code layer). Here H1 to H7 represent the hidden layers and since our problem formulation has 4 dimensions from H1 to H4 and then it is mirrored, we get symmetric AEs. As pointed out earlier for MNIST dataset, H1 is constructed by multiplying the first gene of the chromosome to 1024, H2 is the results of the multiplication of the second gene and H1; however, if H1 is zero then it is multiplied to 1024. Following the same procedure, H3 is generated from third gene being multiplied to H2 in which if H2 is zero, it checks H1 and it keeps going back until it finds the first non-zero layer. Besides, the real values inside a chromosome are not sorted and the reason that we get undercomplete models is that each time the gene value is multiplied to a smaller value which leads to smaller hidden layers.

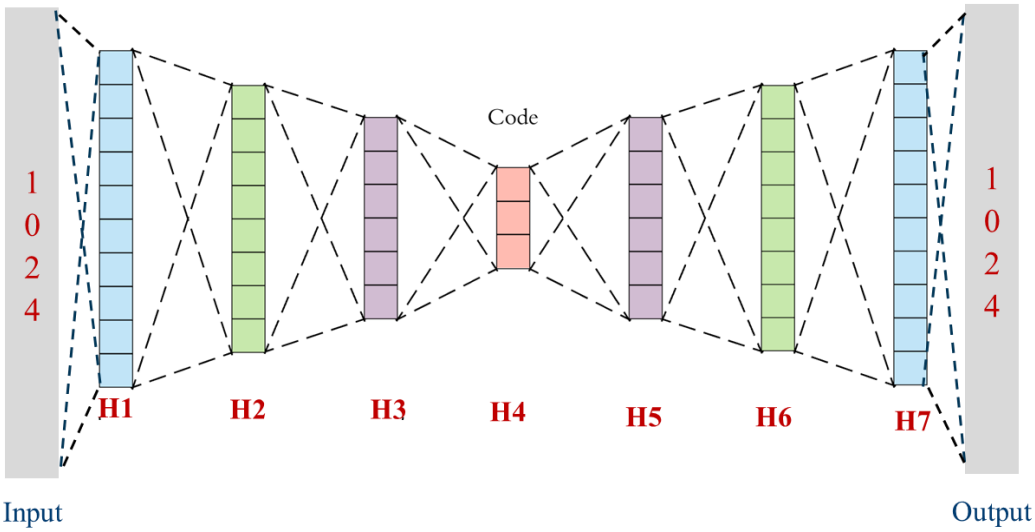

Figure 4.2: The schematic view of the AE architectures utilized in histopathology application. We have input and output layers of size 1024 while other layer have smaller size.

But why did we not consider the existence of a neuron in each layer or connection between layers in your chromosome encoding or applying techniques like dropout instead of defining the number of neurons per layer? Well, if we were going to let the optimizer to have control over the presence of each single neuron or adjust the connections, then the dimensionality of the problem would be much higher than our current approach. Because we would have needed much more parameters in each chromosome (i.e., more genes or variables) so that it could be well understood by the optimizer. While on the other side, in our current approach that each gene only reflects the total number of neurons in that layer util the code layer, the dimensionality of the problem is 4 (comprised of H1 to H4) and that accelerates the computation time to a considerably and reduces the complexity of the optimization problem. Next, proposed scenarios on KimiaNet extracted features per each WSI for the purpose of classification will be explained in detail. Through all the experiments we considered one training epoch and batch size of 4 during evolution where

the lower-level optimizer is evaluating the performance of each designed model by the upper-level optimizer. While after the evolutionary process is done and we have the non-dominated PF of AEs, to fully train each model we consider 200 training epochs and batch size of 64. In addition to that, for the purpose of classification on almost all of the schemes except SAE scheme that we add an extra classification layer, K-NN with K = 3 is used to report the 'F1-score' from all hidden layers and the hidden layer with the highest 'F1-score' is reported as code layer. While in scenarios that classification error (i.e., 1 – 'F1-sore') is one of the optimization objectives, during the evolving process the 'F1-score' is only considered on the middle layer or one of its smaller adjacent layers if middle layer is zero. The reason is that the performance estimation during evolution is really expensive and time killing, and we needed to increase the pace by only checking the middle layer. But during full training we consider 'F1-score' on all hidden layers. It should be pointed out that we tried various values for each hyperparameter such as population-size, #epochs, batch-size, #generations, and K in K-NN and we suggested population-size = 20, #epochs = 1, batch-size = 4, #generations = 100, and K = 3 as the best settings for the evolving framework. An overview of the proposed frameworks is illustrated in Figure 4.3. But why do we use K-NN in our experiments? Because in digital pathology, the purpose is to find the most similar images (rather than classification). Thus, K-NN is utilized to evaluate the process of search.

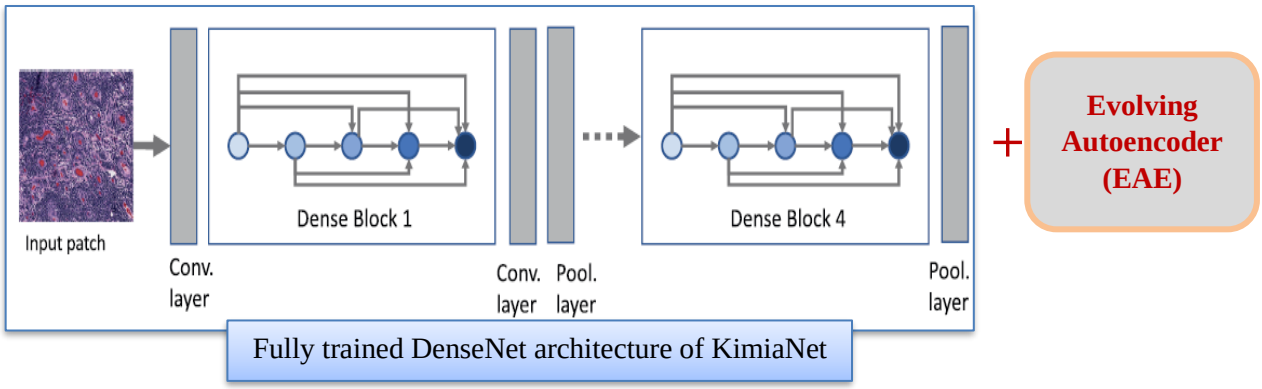

Figure 4.3: Schematic of our proposed framework

In our study, we did not work on denoising AE because we are using the mean features extracted from a DNN and those mean feature vectors do not necessarily represent the input features from each WSIs because each WSI size is reduced to a 1024 latent space. So, unlike WSIs, in the extracted mean feature vectors we cannot certainly say that there is a similar pattern inside each embedded vector for each tumor subtype and that pattern matches the exact pattern inside the large-scale WSI. Thus, adding noise to the extracted mean feature vector of each tumor subtype might completely change the pattern and make the model to mistakenly recognize it as another subtype of the same tumor. I believe that the idea of denoising AE might perform well if we utilize convolutional operations on the corrupted WSIs directly and have a convolutional denoising AE so that we will not be working on the mean of the latent features and then we can ensure that despite adding noise we still have the same pattern inside each WSI.

4.1 Results on Bi-objective Schemes

First, we start our investigation by considering only two objectives in the evolution of AEs. The original input features that we pass as input to our autoencoders during evolution and fully training are real values in range of [a, b], where ‘a’ could be in range 0 to -30 and ‘b’ between 0 to 25. Since we are using sigmoid as activation function of the output layer for each autoencoder, we need to rescale the input features to be in the same range of 0 to 1 as the output layer predictions. In order to have a fair judgment for calculating the Reconstruction Mean Squared Error (RMSE). We dedicated the same budget such as population size, number of generations, training epochs, and batch size to both Brain and Pulmonary tumor types (for now we are just using these two tumor types and once we have found a promising scheme we will take all the other available tumor types into account). Therefore, for all of our experiments scaled input features in both evolution and fully training will be used.

4.1.1 RMSE and FLOPs

In our first experiment we started with RMSE and FLOPs as our two optimization objectives and forced the bilevel optimizer to consider these two objectives in design of each AE. Thus, by comparing different AEs based on these two values and generating new AEs inspired by the promising previous AEs, this scheme provided very promising results on Brain tumor but did a very poor job on Pulmonary as it can be seen in Table 4.1. Full detail regarding all the PF solutions and their performance evaluation both during evolution and after being fully trained

are given in Appendix A1. in Tables 4.12 and 4.13 on Brain and Pulmonary, respectively. Based on Tables 4.12 and 4.13 having complex networks can be useful for lower RMSE values but not in favour of F1-score. It can be seen that on Brain tumor the highest F1-score after full training is resulted from a very compact AE that does not necessarily have a good RMSE value. So, we might be able to get better F1-score from more compact models and we need to tune the objectives.

4.1.2 RMSE and Classification Error

We were really interested to know if we replace the FLOPs objective with Classification Error (i.e., 1 – 'F1-score') how likely it is to make an impact on the results and the F1-score. In the scheme where two objectives are RMSE and Classification Error, since we do not have any control over the complexity of the resulted PF solutions during evolution then it is highly possible to have more complex autoencoders compared to the scheme that we focused on RMSE and Complexity. It can be concluded for the tables that, for RMSE and Classification Error, the complexity of PF solutions is in scope of 1.0E+06 and are more complex with less eliminated layers and almost all layers have some neurons. During the evolution, the values of averaged F1-score were higher (as classification error was lower) than when they were fully trained. And the averaged F1-score was good on both Tumor types, but we had to compensate with much more complex networks. In comparison to the scheme in which we had RMSE and FLOPs, based on the fully training results, the code size has increased a lot and only the F1-score for Pulmonary dataset has increased but the RMSE range has remained the same.

So far, based on Tables 4.1, 4.14, and 4.15 (shown in Appendix A1.) we conclude that replacing FLOPs with classification error as an objective does help increasing the averaged F1-score to small extent after fully training and the scheme with two objectives of RMSE and classification error beats the objectives RMSE and FLOPs in terms of F1-score after fully training because more structures with higher F1-score are in the PF, but we should accept it with the cost of having more complex networks.

4.1.3 Classification Error and FLOPs

It is really intriguing to know what would happen if we had substituted RMSE with Classification Error to see whether having both FLOPs and classification error as our objectives help balancing the gap between complexity and classification accuracy of the networks. Because it is possible that two networks have the same RMSE value but totally different in classification accuracy. Therefore, we contemplate another scheme by 2 objectives of Classification Error and FLOPs to be minimized at the same time as shown in Tables 4.16 and 4.17 in Appendix A1. As Table 4.1 depicts, having only 2 objectives of classification error and FLOPs in which only RMSE is used to optimize the weights during evolution and fully training can outperform the other 2-objectives schemes to high extent both in complexity of networks which results in the size of code layer and also classification accuracy so that we have the highest F1-score of 96% for Brain with a very small code size of 9, which is quite comparable to 8 that was resulted when we had RMSE and FLOPs as objectives. In addition to that, for Pulmonary, we have a very good F1-score of 90% from a very compressed network with code size of 60 which is a great achievement

compared to the other schemes and we have accomplished a network with a code size that is two times smaller. By having a closer look at Tables 4.16 and 4.17 in Appendix A1. we can see that all of the PF models are more compressed than the ones proposed by the last two schemes and, interestingly, on both tumor types the RMSE value has remained almost the same after full training while the F1-score value have had fluctuations or gotten better. Overall, by comparing all these 2-objectives schemes shown in Table 4.1, which are all only optimized by RMSE during evolution and full training, together we clearly get the sense that the one with classification error and FLOPs makes a great balance between classification accuracy and compression of the network and the reason is that RMSE is a not an easy objective for the optimizer to minimize to a great amount, while complexity and classification error are easier objectives. Consequently, when one of our objectives is hard, optimizer tries to minimize the easier objective more and give it more significance in the network's evolution. For instance, in the scheme of RMSE and FLOPs we had more compressed networks, but the classification accuracy was not so promising on Pulmonary. In the scheme of RMSE and classification error, the classification error was easier for the optimizer to reduce so we ended up having good classification accuracy and all networks on the PF had F1-score above 61%, while the complexity of the networks was so large. Nevertheless, when we use both classification error and complexity together, both are kind of an easy objectives to minimize but complexity is a bit easier because by plummeting the size of a layer at a small rate we can get lower number of FLOPs; so, the optimizer gets to minimize both depending on the dataset as much as possible. It is obvious that Brain dataset is easier than Pulmonary because in most

of the schemes, current and later ones, the minimum classification accuracy on Brain was fixated around 93%, while for Pulmonary it was about 90% with a lot of ups and downs. Also, in Pulmonary, good classification accuracies around 89% are often from a gigantic code layer with over 120 neurons. On Brain tumor, no matter how complex the network is we can still get a great classification accuracy, and as a result, we got a great F1-score of 96%. While on the other hand, since Pulmonary is a harder dataset for the networks to learn, so it cannot reduce classification error a lot and instead we got more compressed networks and code layer.

Table 4.1: Investigation on the 2 objectives of RMSE and FLOPs vs. 2 objectives of RMSE and Classification Error vs. 2 objectives of Classification Error and FLOPs

2 Objectives: Classification Error, FLOPs						
Tumor Type	Subtype	# WSI	Results			
			F1-score per class	Averaged F1-score	Code Size	Structure (FLOPs)
Brain	LGG	35	96	96	9	1024, 9, 9, 7, 5, 7, 9, 9, 1024 3.76E+04
	GBM	39	96			
Pulmonary	LUSC	38	92	90	60	1024, 60, 33, 18, 10, 18, 33, 60, 1024 2.57E+05
	LUAD	43	90			
	MESO	5	88			
2 Objectives: RMSE, Classification Error						
Tumor Type	Subtype	# WSI	Results			
			F1-score per class	Averaged F1-score	Code Size	Structure (FLOPs)
Brain	LGG	35	95	95	26	1024, 230, 42, 32, 26, 32, 42, 230, 1024 9.89E+05
	GBM	39	94			
Pulmonary	LUSC	38	92	89	122	1024, 562, 122, 63, 44, 63, 122, 562, 1024 2.62E+06
	LUAD	43	93			
	MESO	5	83			
2 Objectives: RMSE, FLOPs						
Tumor Type	Subtype	# WSI	Results			
			F1-score per class	Averaged F1-score	Code Size	Structure (FLOPs)
Brain	LGG	35	96	96	8	1024, 12, 10, 8, 5, 8, 10, 12, 1024 5.01E+04
	GBM	39	96			
Pulmonary	LUSC	38	90	81	134	1024, 134, 75, 44, 21, 44, 75, 134, 1024 6.06E+05
	LUAD	43	89			
	MESO	5	62			

4.1.4 Sum of losses (Classification Error and RMSE) and FLOPs

As we were stunned by this scheme, we had the idea of utilizing the sum of classification loss and reconstruction loss for evaluating each network during evolution and full training in mind. The underlying reason is based on the idea of multi-task learning in which the reconstruction error is an auxiliary task and acts as a regularizer when it is added to the primary task of classification error. Besides, it can be helpful in preventing overfitting, allowing learning simpler models, and improve the generalization ability of the learned model [72]. Unfortunately, in our case of evolving autoencoder, it did not work in our benefit and the low-level optimizer could not optimize the weights properly and the F1-score on Pulmonary dropped sharply to 68 on a small code layer; while on Brain, F1-score value remained the same, but the code size escalated, drastically, to 245. By taking a deep look at the Table 4.2, when the summation of reconstruction and classification error is utilized during both evolution and full training it has assisted with finding slighter code layer on Pulmonary, but the classification accuracy has slumped to high extent. Interestingly, when only RMSE is used during evolution and full training code size has improved from 245 to 9 on Brain, which is 27 times smaller and have the same F1-score of 96; while on Pulmonary, the F1-score has had a great jump of 22% for classification accuracy on a code layer of 60 by being 10 times bigger than when summation of losses was used. But why this happened? Well, the reason behind it lies in the results obtained after evolution. a possible explanation for this could be that classification error has a zig zagging behavior on different batches of data so when it is added up with reconstruction loss on each batch, it gives an unstable effect

to the final loss value for that single training epoch which affects the classification error since weights cannot be adjusted easily and this have an impact on the quality of solutions found by optimizer, the RMSE column in the tables proves this as you can see some oscillations in RMSE among different PF solutions when RMSE is added to classification error. Another important take away form these results is that, if you take a deep look at the F1-score values of all PF solutions without being fully trained, you can see that when we only had RMSE during evolution on both Tumor types we got totally comparable F1-score in comparison with when the summation of classification and reconstruction losses was used; however, the only difference is that the reconstruction error (i.e., RMSE) has increased when we utilized summation of losses for weight optimization and this clearly indicates that reconstruction error and classification accuracy are not in the same direction and an autoencoder with a high classification accuracy does not necessarily have a good RMSE, and vice versa. That also explains why on most of schemes the fully trained networks on the PF cannot dominate their primary situation before full training and it gets really hard to push the PF to a better position. Later, a semi-supervised scheme will be mentioned that can handle this issue pretty well and enhance the results.

Since the scheme of using summation of losses could help the high-level optimizer to find compressed models on Pulmonary, we wanted to see whether it is possible to get a high F1-score from a compressed model if we only use RMSE during full training of the PF solutions that were found based on summation of losses. In other words, we need to just replace RMSE with RMSE + classification error during full training and see if we can get better classification accuracy from a smaller code layer.

As shown in Table 4.3 you can see that results completely changed and when only RMSE used during full training we it was able to find a smaller code size from a compressed network on Brain Tumor with 1% decrease of F1-score. Conversely, it found more complex model but with 14% higher accuracy form a code layer of size 155. Ultimately, using summation of losses was not in our benefit as it could not provide better results and also it took over 24 hours on a core i5 system to finish it on each Tumor type. The detailed results containing all PF solutions on both tumor types are shown in Tables 4.18 and 4.19 in Appendix A1. For Brain tumor shown in Table 4.18, the performance difference between the best result by summation of losses and single RMSE loss is negligible and that comes from the nature of the data that it is easy to learn no matter if you have a regularized loss function or not. While, on Pulmonary tumor (see Table 4.19), this idea does not work well and there is a considerable difference between summation of losses and single loss final f1-scores after full training.

Table 4.2: Investigation on the single loss vs. multiple summed losses for fully training while RMSE is only used during evolution

Tumor Type	Subtype	# WSI	2 Objectives: Classification Error, FLOPs							
			F1-score per class	Averaged F1-score	Code Size	Structure (FLOPs)	F1-score per class	Averaged F1-score	Code Size	Structure (FLOPs)
Brain	LGG	35	96	96	9	1024, 9, 9, 7, 5, 7, 9, 9, 1024 3.76E+04	96	96	245	1024, 245, 91, 71, 61, 71, 91, 245, 1024 1.14E+06
	GBM	39	96				96			
Pulmonary	LUSC	38	92	90	60	1024, 60, 33, 18, 10, 18, 33, 60, 1024 2.57E+05	73	68	6	1024, 6, 3, 2, 1, 2, 3, 6, 1024 2.47E+04
	LUAD	43	90				78			
	MESO	5	88				53			
			Only RMSE is used for weight optimization during evolution and full training				RMSE + Classification Error is used for weight optimization during evolution and full training			

Table 4.3: Investigation on the single loss vs. multiple summed losses for fully training while summed losses is use during evolution

Tumor Type	Subtype	# WSI	2 Objectives: Classification Error, FLOPs with RMSE + Classification Error During Evolution							
			F1-score per class	Averaged F1-score	Code Size	Structure (FLOPs)	F1-score per class	Averaged F1-score	Code Size	Structure (FLOPs)
Brain	LGG	35	95	95	7	1024, 13, 10, 7, 5, 7, 10, 13, 1024 5.42E+04	96	96	245	1024, 245, 91, 71, 61, 71, 91, 245, 1024 1.14E+06
	GBM	39	94				96			
Pulmonary	LUSC	38	85	82	155	1024, 155, 67, 33, 23, 33, 67, 155, 1024 6.88E+05	73	68	6	1024, 6, 3, 2, 1, 2, 3, 6, 1024 2.47E+04
	LUAD	43	89				78			
	MESO	5	71				53			
			Only RMSE is used for weight optimization during full training				Only RMSE + Classification Error is used for weight optimization during full training			

4.2 Many-objective Schemes

In this scheme, we were so enthused by the previous results and wanted to see whether considering more than two objectives at the same time would guide optimizer toward finding better autoencoders. Consequently, we are embedding four objectives into the evolutionary process of designing the autoencoders comprising RMSE, FLOPs, classification error, and code size, which during the evolution is either the middle layer or one of its adjacent layers in case middle layer is removed, to be minimized. First, we consider only using RMSE during both evolution and full training.

Investigation on 4 objectives with 100 Generations:

As it can be understood from Table 4.4, we had a complexity range from 4.10E+03 to 5.84E+06 all together on both Tumor types and, specifically on Brain, we got a very good F1-score after full training which was 96%. While on Pulmonary was 89% and almost the same as previous schemes' result. However, the only downside is that these high F1-scores after full training were resulted from layers with large number of neurons and kind of complex networks. The reason behind this complexity is that when the number of objectives is more than two, we have more of plane in a high dimensional objective space, so it is highly possible to have complex networks that cannot be dominated by compressed networks because their other objectives may be non-dominated by other objectives of their compressed peers. Therefore, they will remain on the PF and will be passed on to the next generations and finally as the output of the optimizer. Another reason is that we only consider either middle or one

of its adjacent layers whenever middle layer is eliminated, as the code layer during evolution; so, as you can see from the table in most cases the middle layer is selected as the code layer which has the lowest number of neurons compared to the other hidden layers' size. But, during fully training as we consider classification on all hidden layers and in most cases one of the other layers have had the highest F1-score, so the final code size would be a large number. And since the RMSE values are very small and close to each other, it cannot make any huge impact on finding a good network on PF. The details are all included in Tables 4.20 and 4.21 in Appendix A1. in which the PF models have a lot of neurons in the first and last two hidden layers while the third, fourth (i.e., code layer), and fifth hidden layers are smaller. The reason behind this phenomenon is initiating from having both RMSE and code size among the minimization objectives. So, the optimizer tries to put more neurons in the hidden layers to minimize the RMSE value while it is forced to minimize the code size at the same time.

Investigation on 4 objectives with 300 Generations:

As we increase to number of objectives, we did not change the hyperparameter settings and it seemed unfair to spend the same budget on a more complicated optimization problem in a higher dimensional objective space. Therefore, we tripled the number of generations from 100 to 300 in order to give the higher-level optimizer a better chance of evolving structures and discover more from the current solutions and spend the computational budget wisely as shown in Table 4.4. Unfortunately, despite spending more budget we did not get much improvement in terms of FLOPs

and F1-score except for few cases that some layers were removed but still the networks were complex and code sizes were gigantic, accordingly. Tables 4.22 and 4.23 in Appendix A1. can better explain the details. As we investigated more to discover the underlying reason of such a phenomenon, we realized that for getting better RMSE values the optimizer had to sacrifice complexity and opt a huge number of neurons per layer or mainly at the beginning layer which are then mirrored for the decoder part and raise up the complexity of the autoencoders. This important discovery led us to find the most promising scheme which will be discussed later.

Overall, this scheme has helped us to consider 4 objectives in the evolutionary design of autoencoders, simultaneously, which otherwise would be very tricky and challenging for a human expert to intersperse them into a manually designed network. On Pulmonary dataset, among these 4 objective sub schemes, the one where we utilized 100 generations was more promising since it provided a smaller code size. And also, was 3 times faster. However, in terms of complexity and accuracy the final results are not as good as when we only had two objectives of classification error and FLOPs. So, having 4 objectives was not in our benefit for having more compact and accurate models that much and we can still get better results by having half of this objective space.

Table 4.4: Investigation on the 4-objectives schemes

Tumor Type	Subtype	# WSI	Results			
			F1-score per class	Averaged F1-score	Code Size	Structure (FLOPs)
Brain	LGG	35	96	96	119	1024, 119, 70, 35, 1, 35, 70, 119, 1024
	GBM	39	96			5.31E+05
Pulmonary	LUSC	38	89	89	74	1024, 209, 74, 40, 29, 40, 74, 209, 1024
	LUAD	43	90			9.34E+05
	MESO	5	88			
4 Objectives: RMSE, Classification Error, FLOPS, Code Size with 100 Generations						
Tumor Type	Subtype	# WSI	Results			
			F1-score per class	Averaged F1-score	Code Size	Structure (FLOPs)
Brain	LGG	35	95	95	40	1024, 40, 19, 11, 3, 11, 19, 40, 1024
	GBM	39	94			1.68E+05
Pulmonary	LUSC	38	87	89	348	1024, 631, 348, 0, 259, 0, 348, 631, 1024
	LUAD	43	89			3.82E+06
	MESO	5	91			
4 Objectives: RMSE, Classification Error, FLOPS, Code Size with 300 Generations						

4.3 Results on Three-Objectives Schemes

We were really intrigued by the performance of the 2-objectives scheme of classification error and FLOPs. So, we tried to analyze more and see if adding the minimization of code size as the third objective, or you can think of it as removing the RMSE objective from the 4-objective scheme, would help with finding better results in terms of more compressed models with smaller code layer and higher accuracy. In order to have an compact representation of each WSI.

4.3.1 FLOPs, Classification Error, Code Size

As mentioned before, by adding more objectives the objective space's dimension gets larger and more complex networks with medium or small code size showed up among the final PF solutions and by looking at the tables you can clearly see this compared to when we only had two objectives of classification error and FLOPs. Recall that during evolution, the code layer is either the middle layer or one of its adjacent layers (i.e., either its previous or next layer with the smaller size) if middle layer is removed. While during fully training of the PF solutions the layer with highest F1-score among all layers is considered as code layer. In this 3-objectives scheme, where only RMSE is utilized for weight optimization during evolution and full training, we got very promising results on both Brain and Pulmonary Tumor types. On Brain, we reached accuracy of 96% while on Pulmonary we attained accuracy of 90% from code layers of size 29 and 40, respectively. If we benchmark it against the 2-objectives scheme we can see that it has found a smaller code layer for Pulmonary with the same F1-score of 90%; however, on Brain it is a bit larger, but it is not a big problem as it is an easier than Pulmonary. Besides, as shown in Table 4.5, we investigated more and realized that for the winner structure of {1024, 34, 29, 5, 1, 5, 29, 34, 1024} on Brain the variance of F1-score among all layers is quite small and the third hidden layer had F1-score of 95% but since we return the highest F1-score the layer with size of 29 is selected. And this was not the case on Pulmonary because it is a harder dataset and the variance of F1-score between layers was a bit higher. The complete results on Brain and Pulmonary are given in Tables 4.24 and 4.25 in which we have much compact models compared to when we

had 4 objectives because now we do not have RMSE as one of the minimization objectives. Therefore, the optimizer does not need to use large hidden layers and it only focuses on model compression by having smaller FLOPs, code size while having a good F1-score. Despite having smaller code size compared to the 4-objectives scheme after full training, there is still enough room for having much more reduced code size because the K-NN could not provide higher F1-score on the middle layer (i.e., the smallest layer) and the key to get more compact representation layer is to concentrate more on the middle layer which will be discussed next.

4.3.2 Supervised AE

After implementing various schemes and conducting a lot of experiments on Brain and Pulmonary datasets we recommended the 3-objectives scheme that only used RMSE for weight optimization during both evolution and fully training as the best scheme. Next, we wanted to just focus on the code layer with smallest number of neurons and increase its classification F1-score by fully training our PF solutions based on RMSE and then adding a classification layer on the middle layer or one of its adjacent layers, whichever has the highest F1-score when middle layer is removed, and then fine tune its encoder section only for the classification purpose which is known as supervised autoencoder (SAE) [73]. We have provided a schematic view of this scheme in Figure 4.4 to clearly illustrate the underlying idea behind it.

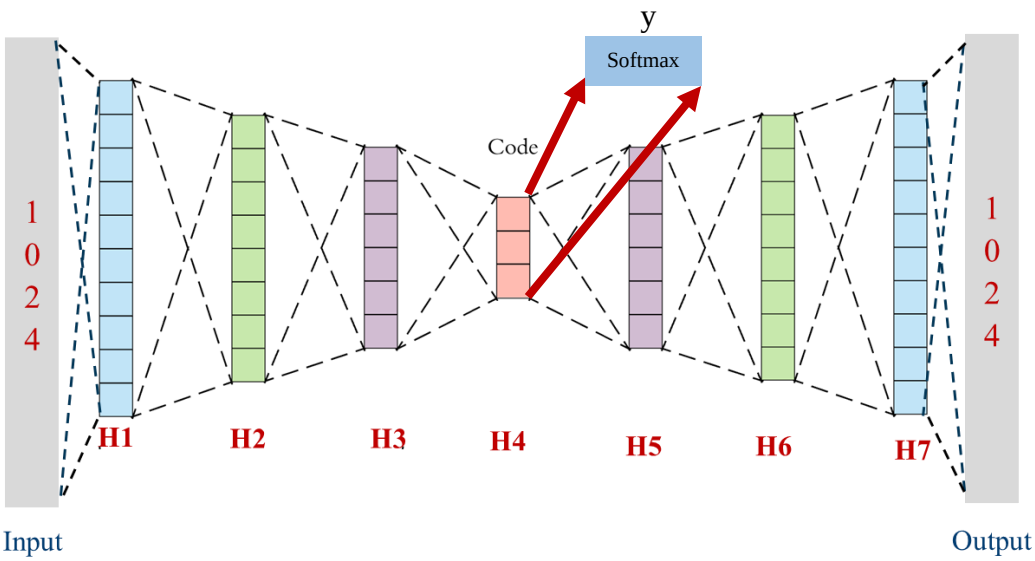

Figure 4.4: The schematic of the SAE scheme utilized in histopathology application. After the whole AE is trained in unsupervised way based on reconstruction error, we add a classification layer (with size of number of tumor subtypes per each class tumor) on the code layer and train the model from Input layer to this classification layer in a supervised way.

As we tested out this scheme on Pulmonary and Brain we got much improved results and it could push the primary PF to a better position in terms of F1-score. We got a sharp increase over the results and in order to avoid confusion by providing a lot of tables separately, we gave it more budget of population size of 100 instead of 20 and run it on all other organs such as Endocrine, Gastro, Gynaeco, Liver, Melanocytic, Prostate, Pulmonary, and Urinary tract to see how useful it would be on these datasets. Unlike previous experiments and schemes, when we wanted to fully train the PF solutions based on the SAE scheme while trying various hyperparameter settings we realized that batch size of 4 is working better that 64. The detailed analysis are provided in Tables 4.26 to 4.34 in Appendix A1. in which we have presented the results of SAE using validation and test sets on the middle layer. In

addition, the last columns show the results of K-NN over test set after fully training for 200 unsupervised epochs under two scenarios: 1) on the middle layer which is the same layer that SAE was performed on. 2) On all hidden layers and then reporting the layer with highest F1-score as the code layer. On almost all the tumor types the best result of SAE outperforms the best result for both scenarios, in terms of F1-score and code size. As we were really curious, we also tested K-NN after the 100 extra supervised training epochs of SAE for a few numbers of structures to see if the extra supervised training will be helpful to k-NN or not; but since we did not see any improvement over the current results, we did not include them to have enough space for our current results.

As Table 4.6 illustrates, we have got very compressed networks with extremely promising F1-scores per class and on average per each tumor type. In all of the datasets the code size that resulted the corresponding F1-score was below 5 which shows how powerful our evolutionary algorithm is for designing such compressed autoencoders that are tailored for that specific dataset. The results of KimiaNet are obtained by using k-NN on the last pooling layer after the network being fully trained. They use k-NN to search for the similar features and they quantified the performance of the search by considering it as a classifier. To be precise, they conducted two kinds of experiments on the test set as: 1) Horizontal search which measures the accuracy of the model in finding the class of tumor across the whole test set. 2) Vertical search in which they seek for the primary diagnosis of a tumor type by searching it subtypes. In other words, in vertical search they wish to measure the accuracy of the model in searching for the correct tumor subtype of each class of tumor.

Now back to our experiment, it is worth mentioning that here we increased the budget by increasing the population size from 20 to 100 with the hope of getting more high-quality networks on the final PF. As earlier we changed the number of generations from 100 to 300 on the 4-objectives scheme and we did not see much improvement in terms of FLOPs and F1-score of the final solutions on the PF, we decided to keep the number of generations fixed on 100 and do not waste time on it. It is so interesting to see that on some of the datasets we had more networks on the final PF but on Prostate we only had two solution that dominated every other solution in terms of all 3 objectives and the top one could almost achieve the F1-score of 100% both during evolution and full training with code size of 1. We can see a similar phenomenon on Melanocytic as well. As we investigated more, we realized that these datasets are way too easy to learn that on most of the model with different structures can be classified with accuracy of 100% and here it is a great superiority of our algorithm to find a structure like {1024, 1, 0, 0, 0, 0, 0, 1, 1024} that is so compressed with only two hidden layers of size 1 which could not be suggested by a human expert otherwise. Such a network can be very good or terrible for classification since it has only two infinitesimal hidden layers and they are not relied on any other hidden layers with good number of neurons to help them during weight optimization. Therefore, if it is not trained well, it will not provide good results. But hopefully, the optimizer could train it well and get the highest accuracy. All in all, these results show a tremendous improvement over KimiaNet results in terms of classification accuracy which we achieved 8% more accuracy, representation compression of each WSI in which it was 135 × 1024 in KimiaNet but 3 on average in our approach (i.e., EAE).

Besides, based on Table 4.5, which compares this SAE scheme with the previous one that only used K-NN for classification, we got smaller code layers with much better F1-scores and the reason is that now we concentrate more on the code layer (i.e., middle layer with smallest size) and try to increase the classification accuracy on that layer specifically.

Table 4.5: Investigation on the 3-objectives schemes

<table>
<tr><td colspan="7">3 Objectives of Classification Error, FLOPs, and Code Size</td></tr>
<tr><td rowspan="2">Tumor Type</td><td rowspan="2">Subtype</td><td rowspan="2"># WSI</td><td colspan="4">Results</td></tr>
<tr><td>F1-score per class</td><td>Averaged F1-score</td><td>Code Size</td><td>Structure (FLOPs)</td></tr>
<tr><td rowspan="2">Brain</td><td>LGG</td><td>35</td><td>96</td><td rowspan="2">96</td><td rowspan="2">29</td><td rowspan="2">1024, 34, 29, 5, 1, 5, 29, 34, 1024
1.44E+05</td></tr>
<tr><td>GBM</td><td>39</td><td>96</td></tr>
<tr><td rowspan="3">Pulmonary</td><td>LUSC</td><td>38</td><td>92</td><td rowspan="3">90</td><td rowspan="3">40</td><td rowspan="2">1024, 477, 167, 92, 40, 92, 167, 477, 1024</td></tr>
<tr><td>LUAD</td><td>43</td><td>89</td></tr>
<tr><td>MESO</td><td>5</td><td>89</td><td>2.35E+06</td></tr>
<tr><td colspan="7">SAE 3 Objectives of Classification Error, FLOPs, and Code Size</td></tr>
<tr><td rowspan="2">Tumor Type</td><td rowspan="2">Subtype</td><td rowspan="2"># WSI</td><td colspan="4">Results</td></tr>
<tr><td>F1-score per class</td><td>Averaged F1-score</td><td>Code Size</td><td>Structure (FLOPs)</td></tr>
<tr><td rowspan="2">Brain</td><td>LGG</td><td>35</td><td>97</td><td rowspan="2">97</td><td rowspan="2">2</td><td rowspan="2">1024, 42, 25, 6, 2, 6, 25, 42, 1024
1.77E+05</td></tr>
<tr><td>GBM</td><td>39</td><td>97</td></tr>
<tr><td rowspan="3">Pulmonary</td><td>LUSC</td><td>38</td><td>94</td><td rowspan="3">97</td><td rowspan="3">5</td><td rowspan="3">1024, 33, 26, 18, 5, 18, 26, 33, 1024
1.41E+05</td></tr>
<tr><td>LUAD</td><td>43</td><td>95</td></tr>
<tr><td>MESO</td><td>5</td><td>100</td></tr>
</table>

Table 4.6: Investigation on the Supervise AE (SAE) scheme where only RMSE is used as the loss function during both evolution and fully training. Results are based on the unseen data of the Test set after full training.

Tumor Type	Subtype	# WSI	KimiaNet		Proposed Method (EAE)			
			F1-score per class	Averaged F1-score	F1-score per class	Averaged F1-score	Code Size	Structure (FLOPs)
Brain	LGG	35	81	81	95	**95**	3	1024, 5, 3, 0, 3, 0, 3, 5, 1024
	GBM	39	81		94			2.06E+04
Endocrine	ACC	6	44	76	83	**92**	3	1024, 7, 6, 0, 3, 0, 6, 7, 1024
	PCPG	15	85		93			2.89E+04
	THCA	51	100		100			
Gastro	COAD	33	76	67	78	**70**	4	1024, 109, 37, 12, 4, 12, 37, 109, 1024
	READ	11	30		35			4.65E+05
	ESCA	14	78		83			
	STAD	30	86		82			
Gynaeco	CESC	17	94	92	94	**94**	3	1024, 104, 33, 10, 3, 10, 33, 104, 1024
	OV	10	95		89			4.41E+05
	UCS	3	86		100			
Liver	CHOL	4	40	71	73	**87**	4	1024, 80, 40, 14, 4, 14, 40, 80, 1024
	LIHC	35	96		96			3.43E+05
	PAAD	12	76		92			
Melanocytic	UVM	4	67	80	75	**85**	1	1024, 1, 0, 0, 0, 0, 0, 1, 1024
	SKCM	24	94		96			4.10E+03
Prostrate	PRAD	40	100	**100**	100	**100**	1	1024, 5, 4, 2, 1, 2, 4, 5, 1024
	TGCT	13	100		100			2.06E+04
Pulmonary	LUSC	38	84	79	88	**92**	5	1024, 33, 26, 18, 5, 18, 26, 33, 1024
	LUAD	43	78		88			1.41E+05
	MESO	5	75		100			
Urinary Tract	BLCA	34	93	92	96	**93**	3	1024, 34, 13, 9, 3, 9, 13, 34, 1024
	KICH	11	86		91			1.427E+05
	KIRC	50	97		97			
	KIRP	28	91		89			
Average of Results				82		**90**	**3**	
SAE Scheme with Three objectives: Classification Error, FLOPs, and Code Size								

4.4 Comprehensive Analysis

Here we wish to provide a comprehensive investigation over the proposed schemes and explain the details behind each scheme and how the optimizer worked. First we have a detailed investigation on the 2-objecrives scheme of RMSE and FLOPs vs. the 4-objectives scheme that had 100 generations of evolution, to understand whether the addition of two other objectives like minimization of Classification Error and Code size to the current objectives of RMSE and FLOPs actually helped with getting more compressed and accurate autoencoders. Since in both schemes RMSE is solely used during both evolution and fully training we wish to see the impact of having more objectives at hand and how beneficial it could be for architecture design of the networks. As tables show and mentioned earlier, when we increase the number of objectives from 2 to 4 the dimension of objective space increases and it would be really hard for compressed networks to dominate complex networks in all dimensions, and it is likely to have more complex networks on final PF. However, the 4 objectives schemes since we have classification error as one of the objectives it is likely that the optimizer finds networks that have good F1-score but very complex on the other side. Therefore, the highest F1-score for Pulmonary is mostly resulted from complex networks with large code size. On the contrary, for Brain dataset we can conclude that compressed networks do as good as complex ones with an averaged F1-score around 92% on both schemes. Therefore, involving more objectives into the evolutionary design process on Brain Tumor type might not be as advantageous as it is for Pulmonary, Tables 4.7 and 4.8.

To conclude, we know that having complexity along with classification error can provide better results in terms of classification accuracy of F1-score both per Tumor type and subtype, but we get much larger code sizes on the other hand when we have other objectives along with them. While having only RMSE and FLOPs as objectives provides compressed networks but less accurate for classification. So, it seems that having classification error (i.e., 1 - 'F1-score') and code size as two additional objectives to RMSE and FLOPs hurts the compression of networks. Thus, it would be better to consider only classification error and FLOPs as objectives and see if it makes a good balance between the compression of networks and classification accuracy, which the full detail is provided in the next section.

Second, we investigate on the 2-objectives scheme of classification error and FLOPs vs. 4-objectives scheme with 100 generations. Here we comprehend whether stacking up two other objectives such as minimization of RMSE and codification layer to the current objectives of classification error and FLOPs helped with getting more compressed and accurate autoencoders. As tables show and was mentioned earlier, when we compared all the 2 objectives schemes together, the one with classification error and FLOPs was the best in terms of classification accuracy and compression of the networks as well as code size. Results clearly point out that the 2 objectives out weight the 4 objectives scheme to great extent and both F1-score and code size are much improved compared to when we had other two extra objectives. As explained earlier, the higher the objective space is, less possible it is for the optimizer to find a very good solution that satisfies all objectives to the best. As Tables 4.7 and 4.8 describe, the 2-objective scheme could find autoencoders at least 3 times more compressed with smaller code sizes and better F1-score values.

Third, we elaborate on the importance of having networks' complexity as an objective. We contrast 2-objectives scheme of RMSE and Classification Error with the 4-objectives first sub scheme and both schemes only use RMSE during evolution and fully training as demonstrated in Tables 4.7 and 4.8. As we compared this 2-objectives scheme (i.e., RMSE and Classification Error) with the one that we had RMSE and Complexity as our 2 objectives, we realized that complexity as an objective plays an important role in the final structure of solutions, and it is an easy objective for the optimizer to minimize it because by decreasing size of a layer a bit the complexity degrades sharply. Now we see the same effect again, as we did not have any control over complexity in this 2-objectives scheme the code size that had the best F1-score was huge which was resulted from a very complex network, accordingly. Despite the challenges that we have in higher dimensional objective space, we can see that in the 4 objective scheme the FLOPs for each Tumor type are approximately half of the ones found in 2 objectives and also the code size on the harder dataset of Pulmonary was much smaller. This affirms the effect of complexity to reduce the code size. Besides, unlike the 2-objectives scheme that all PF solutions had complexity in range of 8.60E+05 to 3.58E+06 among both datasets, in the 4-objective scheme we had compressed networks on PF with good F1-scores having complexity range of 4.10E+03 to 5.84E+06. It is obvious that having complexity along with F1-score in the 4 objectives scheme could provide far better results both for code size and F1-score value on both datasets compared to when we only had RMSE and Classification Error as objectives.

To conclude what we have discussed so far, if we want a superb balance between complexity of networks (i.e., FLOPs), code size, and classification accuracy, then the best scheme among all would be the 2-objectives scheme in which we had classification error

and FLOPs as our objectives. Because it achieved 90% and 96% classification accuracy on Pulmonary and Brain, respectively. It should be pointed out that 90% F1-score on Pulmonary Tumor type is a great achievement and is the top value among all the schemes no matter how many objectives were involved. For this reason, we are very excited to report the 2-objectives scheme of classification error and complexity as the best scheme since it has half of the objective's space of 4-objectives schemes, and it has shown a great potential in finding ideal solutions.

Fourth, we are just elaborating more on the results of the 4-objectives scheme, that went through 300 generations, by comparing it against our best scheme so far, in Tables 4.7 and 4.8, which is the 2-objectives scheme based on classification error and complexity while only RMSE is used during evolution and fully training. And we wish to see which scheme will be the best one. As results show, the 2-objectives scheme outperforms the 4-objectives one to high extent in terms of complexity, code size, and F1-score on the results. The code size is 5 times, and complexity is at least 11 times smaller for the results of the 2-objectives scheme while we have higher F1-score on both Tumor types. Obviously, the 2-objective scheme of classification error and FLOPs is the best so far.

Fifth, we need to have a thorough review over the 2-objective scheme of classification error and FLOPs vs. the 3-objectives scheme of classification error, FLOPs, and code size vs. the 4-oobjectives scheme. As Tables 4.7 and 4.8 summarize, it is difficult to say which scheme is the best since we have the same F1-score on the 2-objectives of classification error and FLOPs and the 3-objectives scheme of classification error, FLOPs, and code size, where both schemes only utilized RMSE during evolution and fully training. The 3-

objectives provided smaller code size on Pulmonary while 2-objectives found smaller code size on Brain; however, this more likely that the optimizer could not find a smaller network and maybe if we run the optimizer again, we may get smaller one. The important that due to the design of our autoencoders (i.e., symmetric structures following a decreasing order of layer size in encoder) it is much better to use the 3-objectives schemes. Because complexity as an objective impacts the code size (i.e., the middle layer or the adjacent layer if middle layer is removed) to some extent and by having a compressed network the code size will be forced to be smaller. So, when we add the code size itself as the third objective it helps with getting much more compressed solutions. More importantly, if you look at the PF solutions resulted from the 2-objectives and 3-objectives schemes, you can see that in 2-objectives there is a very huge decrease in terms of number of neurons from 1024 input dimension on both Tumor types at the first layer and then gradually it gets smaller. But, on 3-objectionves scheme we see larger layer size at the first hidden layer and then, unlike 2-objectives scheme, there is huge decreases between the other hidden layers and as a result the code size will be much smaller. Thus, when we have the code-size as one of the objectives it forces the network to focus more on having a smaller code layer to extract the most significant features while not hurting the data loss much by putting larger layers at first and then drop them dramatically which can help providing more robust solutions. Therefore, we call this 3-objectives scheme as the best scheme and now we wish to enhance it more and try it out on all other Tumor types with more budget which is explained in the next section.

Table 4.7: Investigation on the two, three and four objectives experiments on Brain tumor where only RMSE is used as the loss function during both evolution and fully training

Brain Tumor	Subtype	# WSI	Proposed method (EAE)			
			F1-score per class	Averaged F1-score	Code Size	Structure
Experiment						(FLOPs)
2 Objectives: Classification Error, FLOPs	LGG	35	96	96	9	1024, 9, 9, 7, 5, 7, 9, 9, 1024
	GBM	39	96			3.76E+04
2 Objectives: RMSE, Classification Error	LGG	35	95	95	26	1024, 230, 42, 32, 26, 32, 42, 230, 1024
	GBM	39	94			9.89E+05
2 Objectives: RMSE, FLOPs	LGG	35	96	96	8	1024, 12, 10, 8, 5, 8, 10, 12, 1024
	GBM	39	96			5.01E+04
3 Objectives of Classification Error, FLOPs, and Code Size	LGG	35	96	96	29	1024, 34, 29, 5, 1, 5, 29, 34, 1024
	GBM	39	96			1.44E+05
SAE scheme for the 3 Objectives of Classification Error, FLOPs, and Code Size	LGG	35	97	97	2	1024 [42, 25, 6, 2, 6, 25, 42] 1024
	GBM	39	97			1.77E+05
4 Objectives: RMSE, Classification Error, FLOPS, Code Size with 100 Generations	LGG	35	96	96	119	1024, 119, 70, 35, 1, 35, 70, 119, 1024
	GBM	39	96			5.31E+05
4 Objectives: RMSE, Classification Error, FLOPS, Code Size with 300 Generations	LGG	35	95	95	40	1024, 40, 19, 11, 3, 11, 19, 40, 1024
	GBM	39	94			1.68E+05

Table 4.8: Investigation on the two, three and four objectives experiments on Pulmonary tumor where only RMSE is used as the loss function during both evolution and fully training

Pulmonary Tumor	Subtype	# WSI	Proposed method (EAE)			
			F1-score per class	Averaged F1-score	Code Size	Structure
Experiment						(FLOPs)
2 Objectives: Classification Error, FLOPs	LUSC	38	92	90	60	1024, 60, 33, 18, 10, 18, 33, 60, 1024
	LUAD	43	90			
	MESO	5	88			2.57E+05
2 Objectives: RMSE, Classification Error	LUSC	38	92	89	122	1024, 562, 122, 63, 44, 63, 122, 562, 1024
	LUAD	43	93			
	MESO	5	83			2.62E+06
2 Objectives: RMSE, FLOPs	LUSC	38	90	81	134	1024, 134, 75, 44, 21, 44, 75, 134, 1024
	LUAD	43	89			
	MESO	5	62			6.06E+05
3 Objectives of Classification Error, FLOPs, and Code Size	LUSC	38	92	90	40	1024, 477, 167, 92, 40, 92, 167, 477, 1024
	LUAD	43	89			
	MESO	5	89			2.35E+06
SAE scheme for the 3 Objectives of Classification Error, FLOPs, and Code Size	LUSC	38	94	97	5	1024 [33, 26, 18, 5, 18, 26, 33] 1024
	LUAD	43	95			
	MESO	5	100			1.41E+05
4 Objectives: RMSE, Classification Error, FLOPS, Code Size with 100 Generations	LUSC	38	89	89	74	1024, 209, 74, 40, 29, 40, 74, 209, 1024
	LUAD	43	90			
	MESO	5	88			9.34E+05
4 Objectives: RMSE, Classification Error, FLOPS, Code Size with 300 Generations	LUSC	38	87	89	348	1024, 631, 348, 0, 259, 0, 348, 631, 1024
	LUAD	43	89			
	MESO	5	91			3.82E+06

Now, we investigate more to see why we got much better results when we switched form 4 objectives to 3 and removed RMSE from our list of objectives? What was the effect of RMSE as an objective? It is an undeniable fact that reconstructing the original input of a

network is a much harder and complicated task compared to classification error and when the network wants to reconstruct the input it requires more neurons per layer to prevent the loss of data as much as possible which leads to networks with high complexity and this results in larger code layer, accordingly. Thus, when we removed the RMSE from the 4-objectives scheme and just concentrated on classification error, FLOPs, and code size to be minimized, in a way, we forced the optimizer to not to care about reconstruction error (i.e., RMSE) as a key factor to decide about which networks to be on PF and just use RMSE for training them with 1 epoch to get a sense of other three objectives. Consequently, this guides the optimizer toward designing compressed networks and smaller code layer, accordingly, and increasing F1-score simultaneously.

In order to make the comparison among the schemes more tangible and clearer, we have provided several scenarios to visually illustrate the performance difference among the schemes based on certain aspects. At first we need to visually illustrate how the PF of each scheme after being fully trained would look like and compare them based on some specific objectives. First, we take the fully trained solutions of each scheme and perform nondominated sorting on them to only select the first front of nondominated solutions and then extract the required objectives from them and compare them against other schemes. For convenience, we have provided a detailed description of each scheme in form of Table 4.9.

Table 4.9: Detailed description of all the proposed schemes labeled from 1 to 9 for ease of reference in the comparison plots

Scheme	Description
1	2 optimization objectives: RMSE and FLOPs where only RMSE is used during evolution and fully training.
2	2 optimization objectives: RMSE and Classification Error where only RMSE is used during evolution and fully training.
3	2 optimization objectives: Classification Error and FLOPs where only RMSE is used during evolution and fully training.
4	2 optimization objectives: Classification Error and FLOPs where only RMSE + classification error is used during evolution and fully training.
5	2 optimization objectives: Classification Error and FLOPs where only RMSE + classification error is used during evolution but RMSE during fully training.
6	4 optimization objectives: RMSE, FLOPs, Classification Error, and Code Layer where only RMSE is used during 100 generations of evolution and fully training.
7	4 optimization objectives: RMSE, FLOPs, Classification Error, and Code Layer where only RMSE is used during 300 generations of evolution and fully training.
8	3 optimization objectives: Classification Error, FLOPs, and Code Size where only RMSE is used during evolution and full training.
9	3 optimization objectives: Classification Error, FLOPs, and Code Layer where RMSE is used during evolution and full training then fine tuned by SAE on codification layer for 100 epochs.

First, as plotted in Figures 4.5 and 4.6, we plot results of all those schemes which had both RMSE and FLOPs as optimization objectives. We plotted the results from scheme 1, 6, and 7 which had both of these objectives. As you can see from the plots on both datasets and based on what we explained earlier, the first scheme that only had RMSE and complexity has most of its solutions in better position on the PF and can dominate results of other schemes based on only these 2 objectives. And the reason is that its objective space has 2 dimensions while schemes 6 and 7 have 4 dimensions and it is harder for the optimizer to find results that satisfy all 4 objectives considerably.

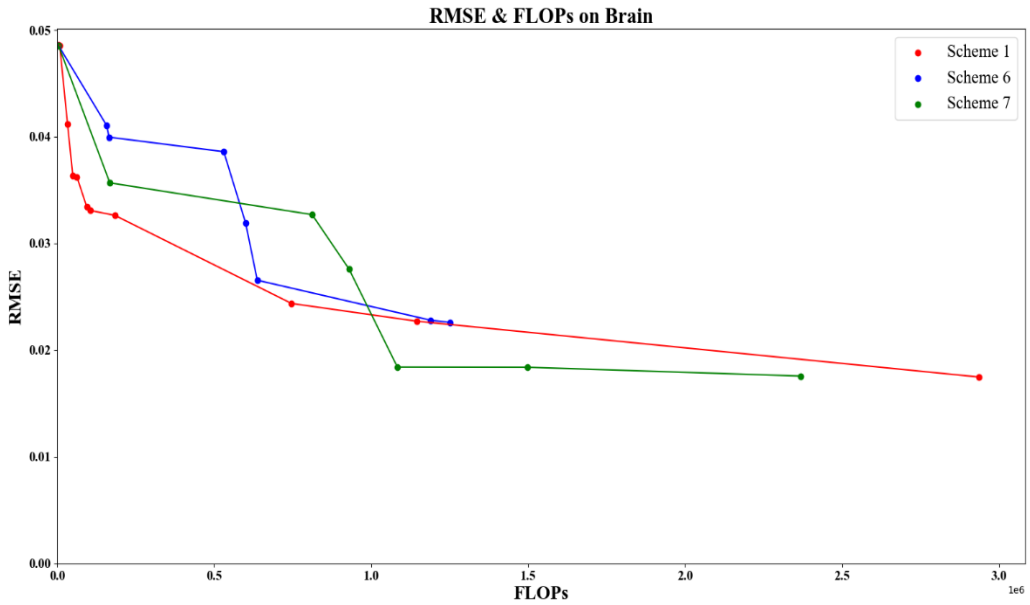

Figure 4.5: Two objectives of RMSE and FLOPs on Brain

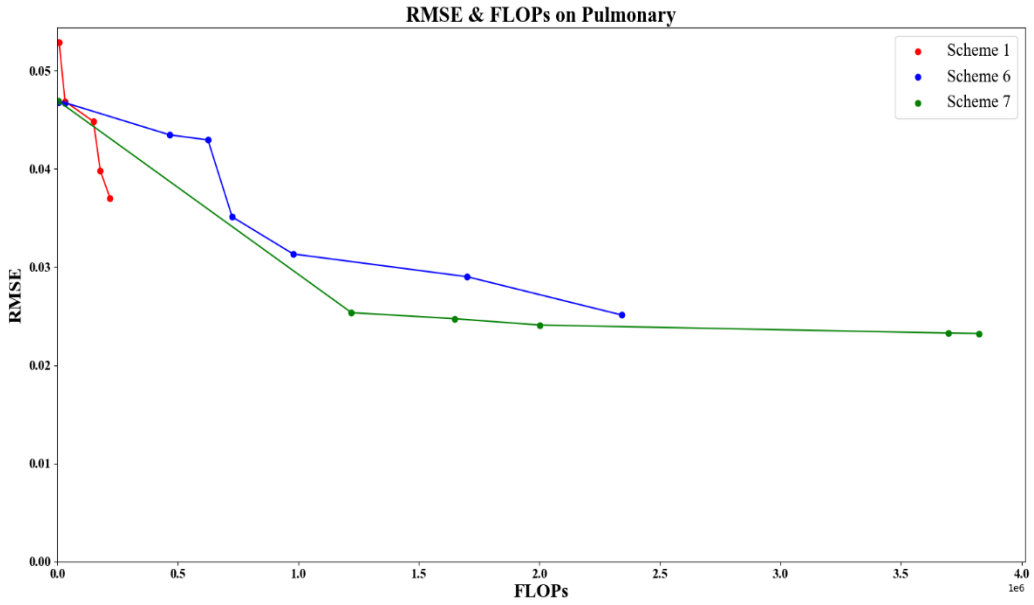

Figure 4.6: Two objectives of RMSE and FLOPs on Pulmonary

Next, we wish to illustrate, in more details, why RMSE does not help with getting higher F1-score based on the primary results after evolution in Figures 4.7 and 4.8. Since F1-score

is a maximization goal and RMSE is minimization goal, then the PF starts from top right a should go towards bottom left of the plot. As mentioned earlier and clearly shown in the plots we cannot say that if we have a good RMSE then our F1-score is going to be good, too. Because here we have F1-score of 99% on brain from two different schemes with different RMSE values. Therefore, having RMSE as an objective is not going to make a huge difference on the F1-score of the models but will increase the complexity of the models as it can be seen in the tables of the 4-objectives or the 2-objectives of RMSE and Classification error schemes.

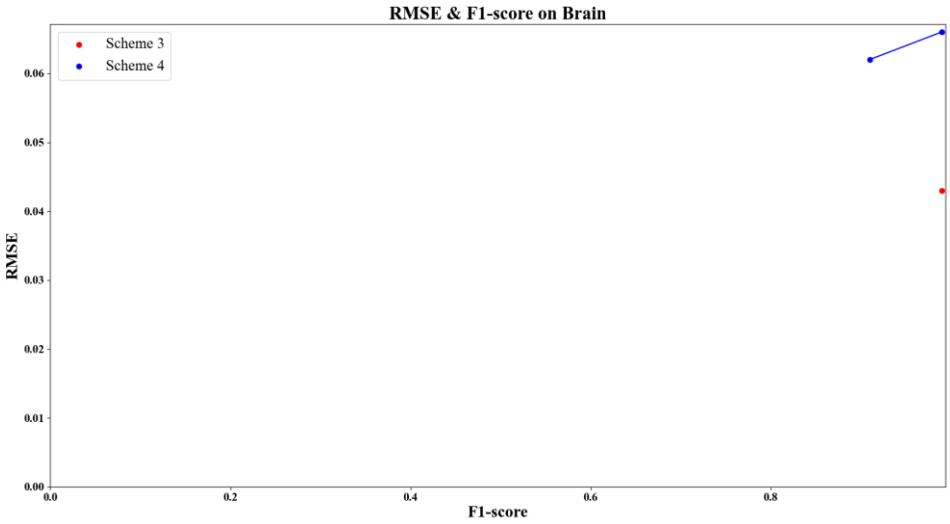

Figure 4.7: Two objectives of RMSE and F1-score on Brain

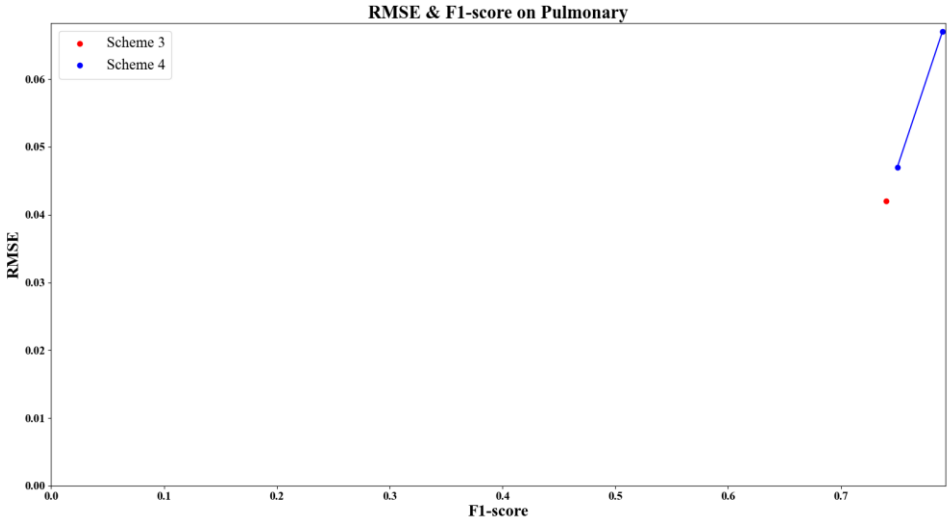

Figure 4.8: Two objectives of RMSE and F1-score on Pulmonary

Now, in Figures 4.9 and 4.10 we clarified all the schemes that had classification error and FLOPs as their objectives, the nineth scheme that was based on SAE had the best results and made a great balance between these 2 objectives when all the AEs were fully trained. As the plots show, it dominates almost all other schemes and it fully trains the compressed models that are provided by the EAE framework to results in higher F1-score (i.e., lower classification error) while other methods could not reach.

Furthermore, we can illustrate the comparison between the 3 and 4-objectives schemes based on the three objectives of classification error, FLOPs, and code size on both Brain and Pulmonary datasets in Figures 4.11 and 4.12, respectively. As it can be seen from the plots the nineth scheme (i.e., the SAE scheme) has shown a great achievement and dominates almost all of the other AEs designed by the other schemes and it has provided very compressed and accurate models when fully trained. Based on the details given earlier,

since in the SAE scheme we first have unsupervised training of all layers on each AE and then remove the decoder part and fine tune the encoder section solely for the classification

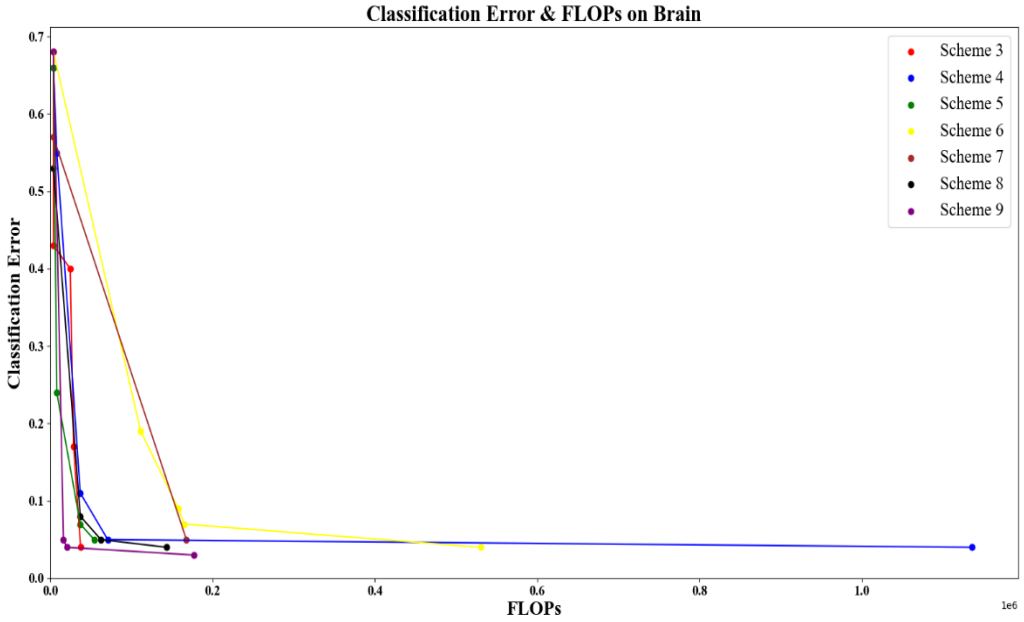

Figure 4.9: Two objectives of Classification Error and FLOPs on Brain

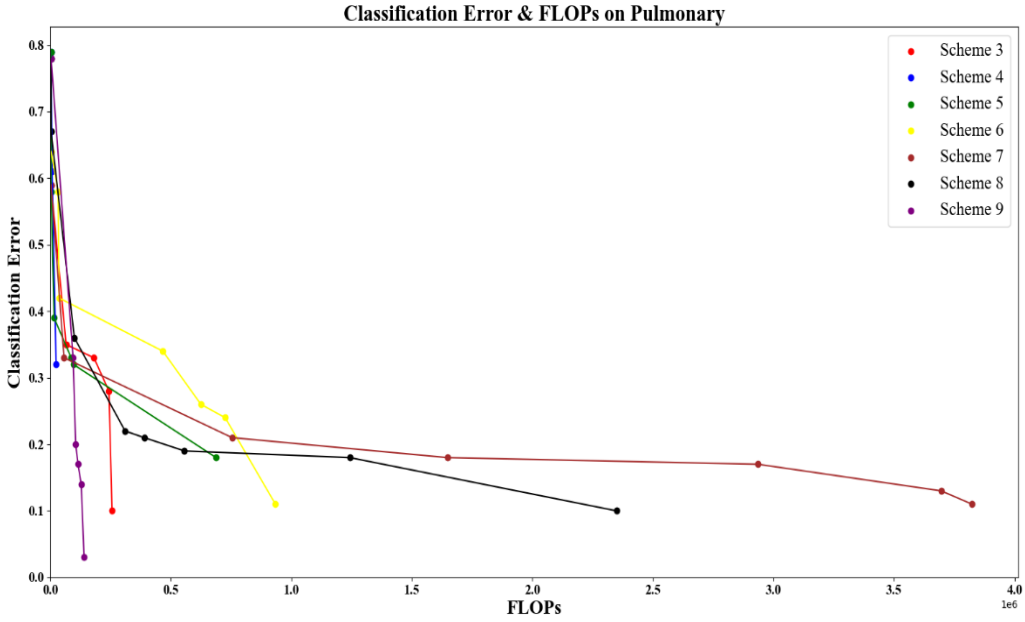

Figure 4.10: Two objectives of Classification Error and FLOPs on Pulmonary

purpose by stacking a new layer on top of the code (i.e., middle layer), the accuracy of the model has got momentum, and this benefit is coming form the effect of fine tuning.

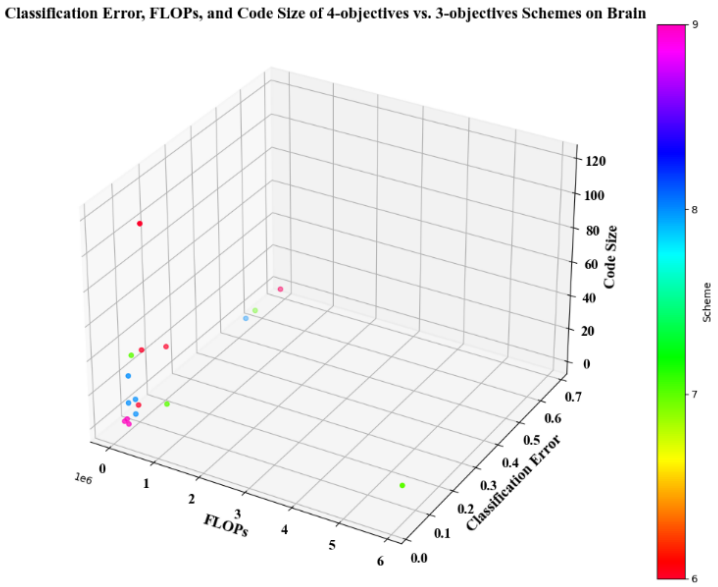

Figure 4.11: Three objectives of Classification Error, FLOPs, and Code Size on Brain

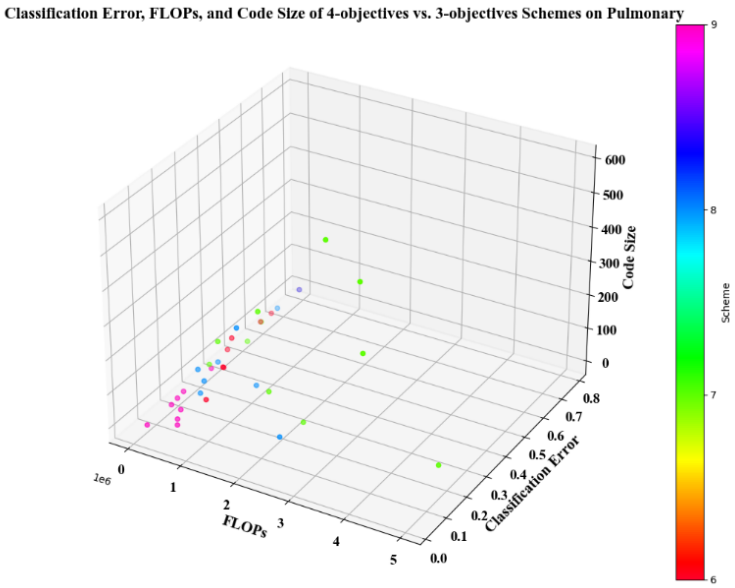

Figure 4.12: Three objectives of Classification Error, FLOPs, and Code Size on Pulmonary

It is really exciting to see among all the proposed schemes on the KimiaNet datasets which scheme did a better job at reducing the dimension of the problem and resulted smaller classification error at the same time. As you can see from plots in Figures 4.13 and 4.14, the SAE scheme, the nineth scheme, easily dominates most of the other designed models by other schemes and has done a phenomenal job at extracting the most significant features and accurately classifying them. It should be pointed out that the structures were designed almost the same way by the bilevel optimizer (i.e., EAE). But the discriminating characteristic of SAE compared to other schemes is that it can perfectly perform on each model and provide higher classification accuracy from the code layer and reduce the high dimensional input space to a high extent.

Finally, based on all the conducted experiments and the analysis that explained earlier, we can conclude that the scheme9 that had 3 optimization objectives of Classification Error, FLOPs, and Code Layer in which the PF solutions were first trained in an unsupervised fashion for 200 epochs and then the encoder part was fine tuned by SAE on codification layer for extra 100 epochs, is the best scheme among all of our proposed schemes. And the detailed results of this scheme on all tumor types (i.e., from Tables 4.26 to 4.34) is an indication of this inference.

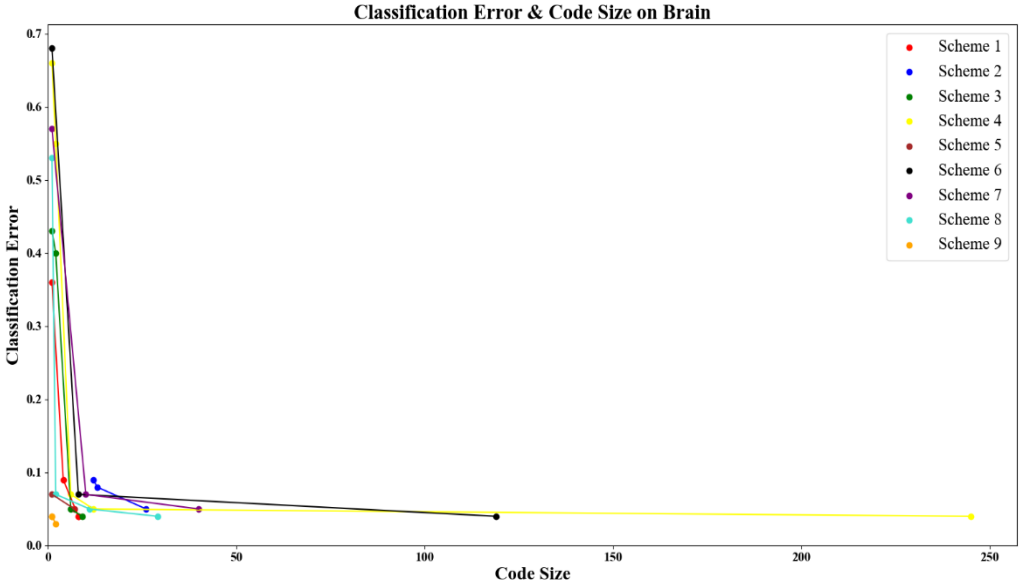

Figure 4.13: Two objectives of Classification Error and Code Size on Brain

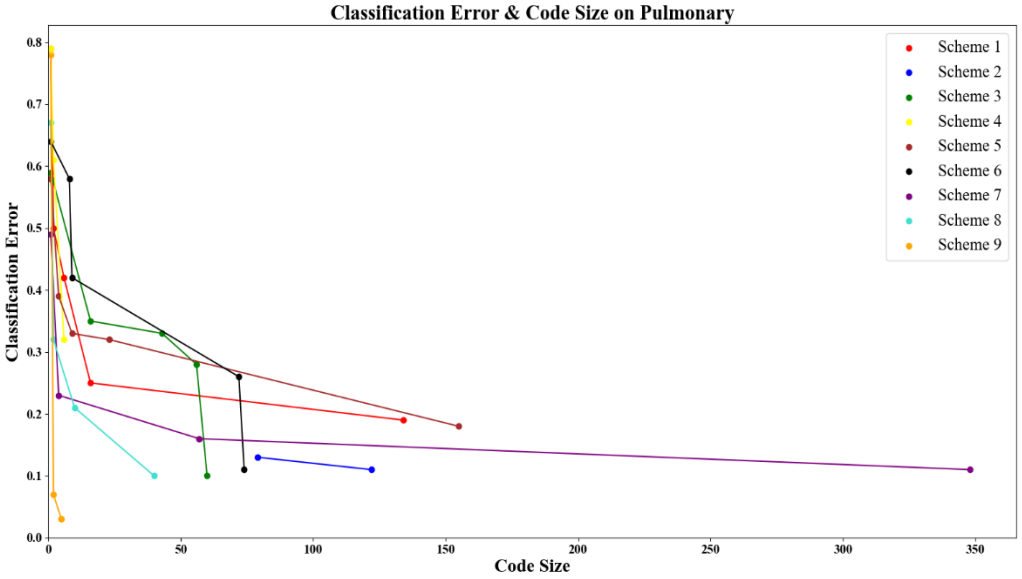

Figure 4.14: Two objectives of Classification Error and Code Size on Pulmonary

4.5 Comparative Studies

Now, we are having a comparison based on the histopathology features between EAE and some of the human-designed AEs as well as PCA which is the most common choice for dimensionality reduction. The provided result of each set of experiments speculates the incredible compactness and accuracy of EAE compared to the others.

4.5.1 Evolving Autoencoders Versus Human-designed models

Here we are comparing our approach, EAE, with some of the common human-designed AEs on all tumor types. For the human-designed AEs we presented various design both manually and some that were suggested by others. In scheme 2, we have set each hidden layer's size to half of its previous layer, and we gradually reduce the number of hidden layer to make a new model based on it. For scheme 3 each hidden layer is round of 2/3 of its previous layer's size. Thus, we kept building new AEs of this type by removing the middle layer and make its previous layer the code layer. Finally, schemes 4 and 5 are selected form [65], [78] which were suggested models. For example, in scheme 4 that AE was the best model that was suggested in their work while in scheme 5 they proposed a formula to design AEs based on the input layer's size and we then took all of their suggested latent representation layer's size and set it as the code size. It should be pointed out that all of the human-designed models were both complex and less accurate than EAE. As results show in Figures 4.15 through 4.23, our approach shown by purple color in the plots as scheme 1, is the best compared to those AEs designed manually. For our approach we have only shown the non-dominated AEs of the final PF after fully trained and as you can see

there are some AE structures that have been proposed by the optimizer that are in all tumor types but with very different classification error values and that is another proof that a fixed architecture cannot be used for any dataset and the optimal solution is hidden in the optimal architecture design that is tailored for that specific dataset. All experimental results including their visual illustration are show in Figures 4.15 to 4.23 on each tumor type; However, the detailed results in form of Tables 4.35 to 4.43 are included in Appendix A2. Based on the results in Table 4.10, it is obvious that the human-designed models are more complex and since they remain the same for every tumor type, we can clearly see how results of a specific AE fluctuates among different tumor types. This is another indication on what we discussed earlier that DNNs are task-specific and data-oriented and the optimal solution for a problem is hidden inside the optimal architecture designed, explicitly, for that problem. This can be inferred from the EAE results as well. For example, the {1024, 1, 0, 0, 0, 0, 0, 1, 1024} is among the final PF models for all the tumor types but it has shown a totally different performance on each of them. Interestingly, most of the proposed AEs design patterns by EAE are far different from the common design choice by human expert. As in most the EAE models there is a very sharp decrease from the input size to the hidden layers' size while in most manually designed models the size of the hidden layers decrease gradually in order to avoid information loss. The reason is that humans cannot discover the statistical patterns of each dataset quickly like ANNs and this could be more challenging in BBO problems, while ANNs, in our case AEs, can easily unravel the hidden pattern and decide how many and which features are more important to solve the problem.

Table 4.10: Human-designed vs. EAE for the Supervised AE (SAE) scheme on All Tumors. For the purpose of a quick comparison ,the Best results from EAE is selected for each class of Tumor

Approach	Structures	FLOPs	Tumor Type								
			Classification Error (Test Set)								
			Brain	Endocrine	Gastro	Gynaeco	Liver	Melanocytic	Prostate	Pulmonary	Urinary Tract
Scheme 1: Best of EAE with SAE on the middle layer in terms of FLOPS and Classification Error per Tumor	1024, [5, 3, 0, 3, 0, 3, 5], 1024 : For Brain	2.06E+04	0.05								
	1024, [7, 6, 0, 3, 0, 6, 7], 1024 : For Endocrine	2.89E+04		0.08							
	1024, [109, 37, 12, 4, 12, 37, 109], 1024 : For Gastro	4.65E+05			0.3						
	1024, [23, 20, 12, 5, 12, 20, 23], 1024 : For Gynaeco	9.72E+04				0.06					
	1024, [80, 40, 14, 4, 14, 40, 80], 1024 : For Liver	3.43E+05					0.13				
	1024, [1, 0, 0, 0, 0, 0, 1], 1024 : For Melanocytic	4.10E+03						0.15			
	1024, [5, 4, 2, 1, 2, 4, 5], 1024 : For Prostate	2.06E+04							0		
	1024, [33, 26, 18, 5, 18, 26, 33], 1024 : For Pulmonary	1.41E+05								0.08	
	1024, [53, 29, 21, 16, 21, 29, 53], 1024 : For Urinary Tract	2.27E+05									0.06
Scheme 2: Division by 2	1024,[512, 256, 128, 64, 32, 10, 32, 64, 128, 256, 512],1024	2.79E+06	0.15	0.18	0.31	0.24	0.17	0.33	0	0.14	0.08
	1024,[512, 256, 128, 64, 32, 64, 128, 256, 512],1024	2.79E+06	0.14	0.12	0.39	0.21	0.29	0.24	0	0.21	0.08
	1024,[512, 256, 128, 64 ,128 ,256, 512],1024	2.79E+06	0.15	0.13	0.32	0.17	0.15	0.35	0	0.1	0.09
	1024,[512, 128, 32, 8 ,32 ,128, 512],1024	2.38E+06	0.12	0.22	0.37	0.18	0.14	0.15	0	0.17	0.14
	1024,[128, 32, 16 ,8 , 16, 32, 128],1024	5.43E+05	0.11	0.15	0.38	0.24	0.15	0.24	0.07	0.21	0.08
Scheme 3: 2/3 of the previous layer	1024,[683,455,303,202,135,90,60,40,27,18,27,40,60,90,135,202,303,455,683],1024	5.03E+06	0.14	0.08	0.42	0.14	0.25	0.2	0.17	0.15	0.11
	1024,[683,455,303,202,135,90,60,40,27,40,60,90,135,202,303,455,683],1024	5.03E+06	0.11	0.17	0.4	0.27	0.3	0.42	0.09	0.08	0.07
	1024,[683,455,303,202,135,90,60,40,60,90,135,202,303,455,683],1024	5.03E+06	0.14	0.21	0.34	0.2	0.11	0.15	0	0.09	0.1
	1024,[683,455,303,202,135,90,60,90,135,202,303,455,683],1024	5.02E+06	0.15	0.13	0.43	0.13	0.17	0.15	0	0.09	0.07
	1024,[683,455,303,202,135,90,135,202,303,455,683],1024	4.99E+06	0.09	0.16	0.41	0.09	0.4	0.15	0.05	0.12	0.09
	1024,[683,455,303,202,135,202,303,455,683],1024	4.95E+06	0.16	0.14	0.33	0.25	0.21	0.19	0.17	0.18	0.07
Scheme 4: [65]	1024,[651, 688, 91, 32, 46],1024	2.46E+06	0.14	0.17	0.35	0.18	0.28	0.29	0.02	0.16	0.09
Scheme 5: [78]	1024,[1244, 522, 261, 37, 261, 522, 1244],1024	8.28E+06	0.15	0.17	0.35	0.16	0.28	0.2	0.02	0.09	0.07
	1024,[1244, 522, 261, 44, 261, 522, 1244],1024	8.28E+06	0.16	0.11	0.37	0.08	0.21	0.32	0	0.16	0.11
	1024,[1244, 522, 261, 17, 261, 522, 1244],1024	8.26E+06	0.15	0.08	0.41	0.21	0.25	0.24	0	0.13	0.09
	1024,[1244, 522, 261, 9, 261, 522, 1244],1024	8.25E+06	0.11	0.16	0.42	0.32	0.11	0.24	0	0.17	0.1
	1024,[1244, 522, 261, 45, 261, 522, 1244],1024	8.28E+06	0.12	0.17	0.31	0.18	0.17	0.24	0	0.14	0.08
	1024,[1244, 522, 261, 47, 261, 522, 1244],1024	8.29E+06	0.15	0.09	0.34	0.22	0.41	0.32	0	0.11	0.07
	1024,[1244, 522, 261, 61, 261, 522, 1244],1024	8.30E+06	0.18	0.08	0.32	0.17	0.18	0.2	0	0.18	0.11
	1024,[1244, 522, 261, 36, 261, 522, 1244],1024	8.28E+06	0.18	0.13	0.34	0.21	0.17	0.2	0	0.11	0.11
	1024,[1244, 522, 261, 33, 261, 522, 1244],1024	8.27E+06	0.15	0.09	0.36	0.18	0.13	0.2	0.07	0.15	0.08
	1024,[1244, 522, 261, 26, 261, 522, 1244],1024	8.27E+06	0.16	0.08	0.3	0.21	0.26	0.15	0.05	0.09	0.08
	1024,[1244, 522, 261, 21, 261, 522, 1244],1024	8.26E+06	0.18	0.09	0.34	0.07	0.16	0.2	0	0.13	0.07

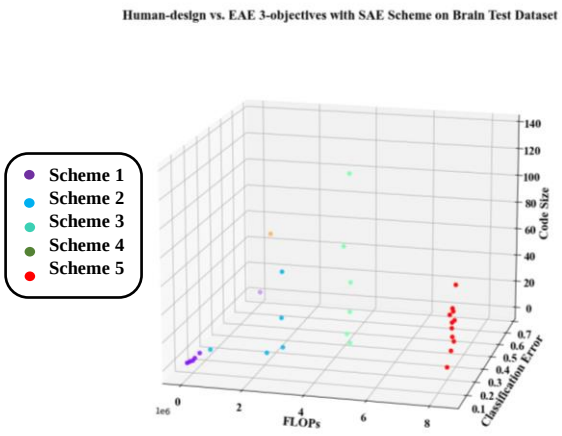

Figure 4.15 Human-designed vs. EAE for the Supervised AE (SAE) scheme on Brain

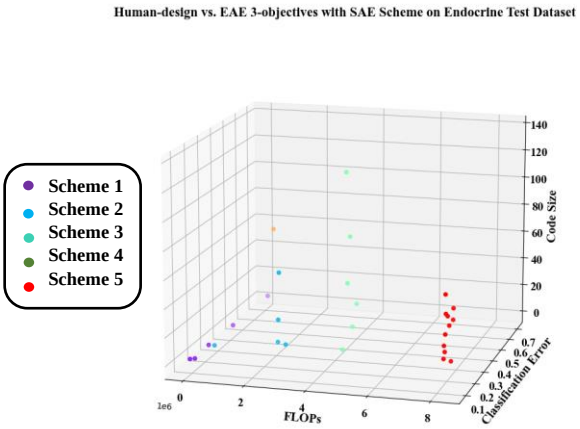

Figure 4.16 Human-designed vs. EAE for the Supervised AE (SAE) scheme on Endocrine

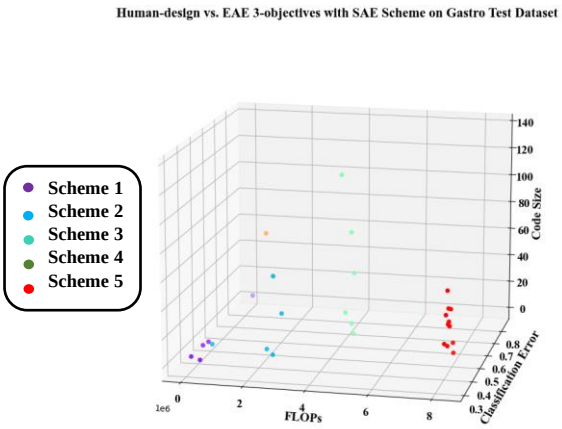

Figure 4.17 Human-designed vs. EAE for the Supervised AE (SAE) scheme on Gastro

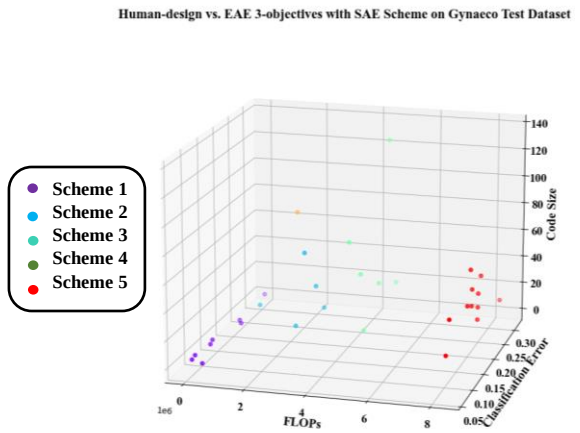

Figure 4.18 Human-designed vs. EAE for the Supervised AE (SAE) scheme on Gynaeco

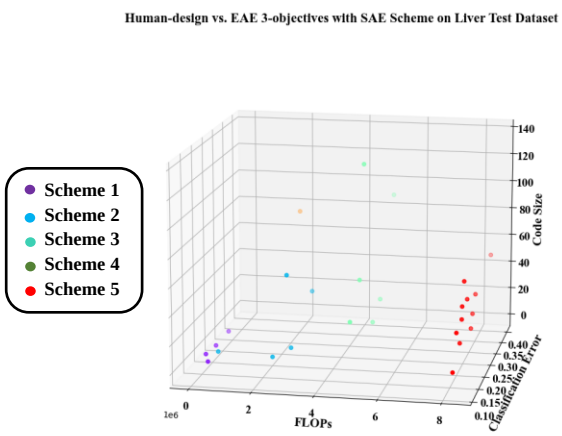

Figure 4.19 Human-designed vs. EAE for the Supervised AE (SAE) scheme on Liver

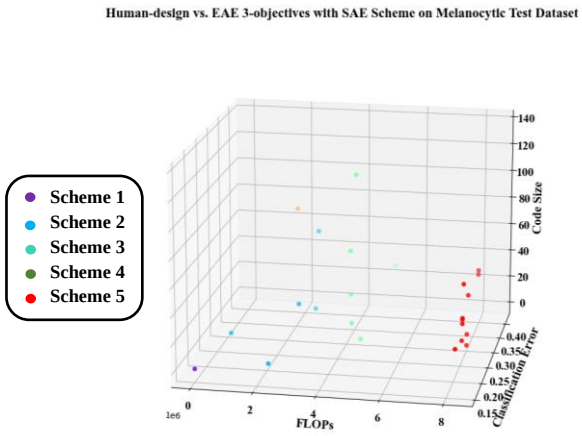

Figure 4.20 Human-designed vs. EAE for the Supervised AE (SAE) scheme on Melanocytic

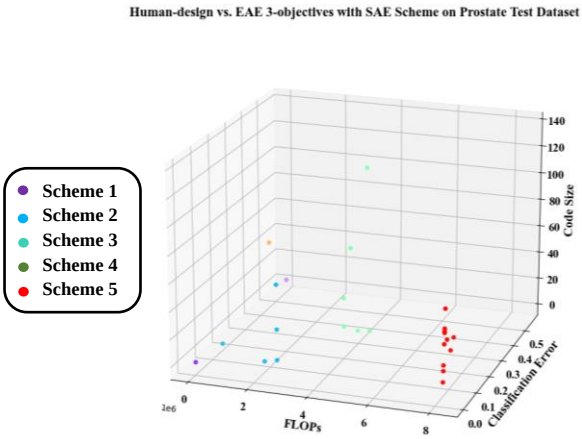

Figure 4.21 Human-designed vs. EAE for the Supervised AE (SAE) scheme on Prostate

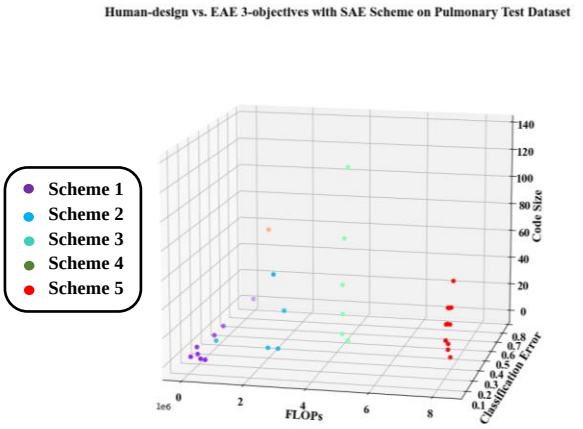

Figure 4.22 Human-designed vs. EAE for the Supervised AE (SAE) scheme on Pulmonary

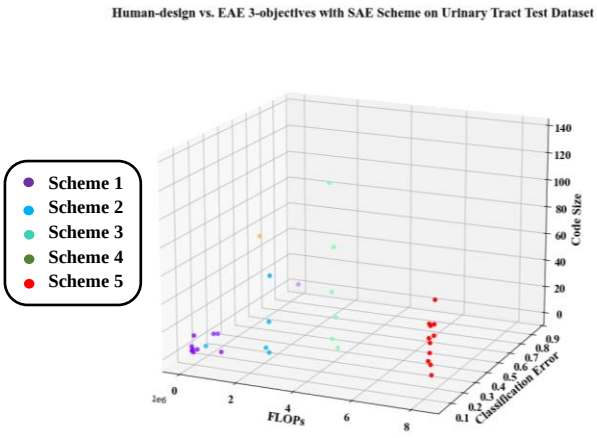

Figure 4.23 Human-designed vs. EAE for the Supervised AE (SAE) scheme on Urinary Tract

4.5.2 Evolving Autoencoders Versus PCA

Here we compare the best results of our SAE scheme against Principal Component Analysis (PCA) algorithm [74]. PCA is a technique that reduces the high dimensionality of the dataset by using linear projection and orthogonality defined by the principal components. In particular, PCA is known as linear dimension reduction method that requires the user to define the number of components manually and this algorithm is popular in machine learning applications. Despite its great success it suffers from some problem such as the defining the number of latent variables (i.e., components) as input to the algorithm which is really challenging and tricky. Besides that, when we have limited number of data samples and noisy data it will be harsher to opt the number of latent variables properly and you need to try out a lot of values to find out which one is better. More importantly, PCA is a linear dimension

reduction algorithm in which it maps the high-dimensional data to a low-dimensional linear manifold and each principal component can be defined as a linear combination of variables, while most of the real-world datasets lay on a low-dimensional non-linear manifold [75]. Moreover, classification of data and dimension reduction are two different phases and cannot happen at the same time in PCA. On the other hand, we have non-linear dimension reduction methods such as AEs that we used in our framework. Generally, AEs as subcategory of neural networks have seemed to be strong enough against noisy data and can perform better in real-world application. Thus, as a comparison between our approach, EAE, and PCA we can speculate that our framework is relying on non-linear dimension reduction technique which is perfectly suitable for any real-world application and can be easily attached to any fully trained DNN to reduce its dimension and increase its classification accuracy, simultaneously. Which it is not the case in PCA. In our approach, user is not required to input the number of latent variables and EAE will automatically figure it out based on its input data and it does not require any manual design and tuning. Finally. As we mentioned earlier, our main goal is to reduce the high dimensional input data and increase classification accuracy at the same time, while this could not be possible by PCA.

For the purpose of the experiments shown in Table 4.11, to have a fair comparison of PCA to our approach we set the number of components to the values that we got as code size for each dataset, and we used the same scaled data that we had in EAE. Obviously, EAE is superior to PCA and results in 10% more classification accuracy on average among all Tumor types and also beats PCA to every single one of them.

Table 4.11: PCA vs. EAE for the Supervised AE (SAE) scheme on All Tumors

Tumor Type	Subtype	# WSI	KimiaNet		Proposed Method(EAE)				PCA		
			F1-score per class	Averaged F1-score	F1-score per class	Averaged F1-score	Code Size	Structure (FLOPs)	F1-score per class	Averaged F1-score	# Components
Brain	LGG	35	81	81	95	**95**	3	1024, 5, 3, 0, 3, 0, 3, 5, 1024 2.06E+04	83	82	3
	GBM	39	81		94				82		
Endocrine	ACC	6	44	76	83	**92**	3	1024, 7, 6, 0, 3, 0, 6, 7, 1024 2.89E+04	71	86	3
	PCPG	15	85		93				86		
	THCA	51	100		100				100		
Gastro	COAD	33	76	67	78	**70**	4	1024, 109, 37, 12, 4, 12, 37, 109, 1024 4.65E+05	75	68	4
	READ	11	30		35				29		
	ESCA	14	78		83				87		
	STAD	30	86		82				83		
Gynaeco	CESC	17	94	92	94	**94**	3	1024, 104, 33, 10, 3, 10, 33, 104, 1024 4.41E+05	90	92	3
	OV	10	95		89				86		
	UCS	3	86		100				100		
Liver	CHOL	4	40	71	73	**87**	4	1024, 80, 40, 14, 4, 14, 40, 80, 1024 3.43E+05	25	69	4
	LIHC	35	96		96				96		
	PAAD	12	76		92				87		
Melanocytic	UVM	4	67	80	75	**85**	1	1024, 1, 0, 0, 0, 0, 0, 1, 1024 4.10E+03	0	45	1
	SKCM	24	94		96				90		
Prostrate	PRAD	40	100	100	100	**100**	1	1024, 5, 4, 2, 1, 2, 4, 5, 1024 2.06E+04	100	100	1
	TGCT	13	100		100				100		
Pulmonary	LUSC	38	84	79	88	**92**	5	1024, 33, 26, 18, 5, 18, 26, 33, 1024 1.41E+05	85	91	5
	LUAD	43	78		88				87		
	MESO	5	75		100				100		
Urinary tract	BLCA	34	93	92	96	**93**	3	1024, 34, 13, 9, 3, 9, 13, 34, 1024 1.42E+05	93	91	3
	KICH	11	86		91				86		
	KIRC	50	97		97				96		
	KIRP	28	91		89				88		
Average of Results				82		**90**	**3**			80	

Chapter 5. Conclusion Remarks and Future Works

In this work we designed a fast, cost-effective, generalizable framework that can be easily employed for various classification and feature extraction scenarios based on different real-world datasets. It acts as a postprocessing optimizer that adjust various AE models to the given dataset and proposes optimized, compressed, and accurate models swiftly. The main idea of this framework was inspired from ENAS framework, which combines evolutionary algorithms with neural architecture search method to evolve different AE architectures and optimize their weight parameters in form of a bilevel optimization problem. Our approach provided 8% increase in terms of classification accuracy compared to the original 1024 dimensional space and reduced complexity of the original model, KimiaNet, more than 46,000 times. All in all, as the results show this framework not only beats most common human-designed AEs in terms of classification accuracy, complexity, and the number of significant extracted features but it outperforms other popular methods like PCA, as well. In addition, since our approach is relying on multi-objective optimization we have trade-off solutions on Pareto-Front which expert users will have a set of choices to select from. This is also an important capability of our proposed method to offer such choices.

Conclusion:

To summarize, the proposed framework evolves AEs for the purpose of dimension reduction and classification, simultaneously. And it convolves several optimization objectives to ensure that the designed models are compact, efficient, and accurate to be implemented in real-world scenarios for various applications. As mentioned earlier, NAS is an expensive method for automatic design of DNNs since it requires training a wide

number of designed models and its cost can rise up depending on the type of DNN that it is designing. In most case the type of model is CNN which are run on GPUs and powerful systems. But, neither most people cannot access powerful systems to apply NAS to their problem nor us, and this is one of the challenged that is commonly encountered by researchers and practitioners. Therefore, having a lightweight framework that can be a handy in such situations is of high importance which our proposed method can do that. In addition, in most real-world challenges the amount of labeled data for training DNNs is very small especially in medical applications where we do not have enough data per tumor types and labeling data by human expert is a tedious and expensive task. So, our framework can easily soldier through this by using unsupervised learning (i.e., AE), to train excellent models for the goal of classification and extraction of impactful features at the same time.

Future Works:

For future works, one direction could be adjusting the current framework to design convolutional, generative, and other types of the AEs by adjusting the chromosome encoding of each AE and investigate on a new way to translate it to an actual model. In addition, we can further facilitate the performance estimation process in the lower-level optimizer by replacing the 1 training epoch with no training setting and just rely on the initial weight parameters to quantify the quality of each evolved model. Or we can utilize gradient free techniques such as coordinated descent/search [76][77], methods to sample the weights of each layer without calculating the gradient.

References

[1] Niclas Andr´easson, Anton Evgrafov, and Michael Patriksson. An introduction to optimization: Foundations and fundamental algorithms. *Chalmers University of Technology Press: Gothenburg, Sweden*, 1:1–205, 2005.

[2] Yurii Nesterov and Arkadii Nemirovskii. *Interior-point polynomial algorithms in convex programming*. SIAM, 1994.

[3] John A Nelder and Roger Mead. A simplex method for function minimization. *The computer journal*, 7(4):308–313, 1965.

[4] Eugene L Lawler and David E Wood. Branch-and-bound methods: A survey. Operations research, 14(4):699–719, 1966.

[5] Vladimir I Norkin, Georg Ch Pflug, and Andrzej Ruszczy´nski. A branch and bound method for stochastic global optimization. Mathematical programming, 83(1):425–450, 1998.Richard E Bellman and Stuart E Dreyfus. *Applied dynamic programming*, volume 2050. Princeton university press, 2015.

[6] Richard E Bellman and Stuart E Dreyfus. Applied dynamic programming, volume 2050. Princeton university press, 2015.

[7] Frederick J Beutler. Dynamic programming: Deterministic and stochastic models (dimitri p. bertsekas). SIAM Review, 31(1):132, 1989.

[8] Heiner M¨uller-Merbach. Heuristics and their design: a survey. European Journal of Operational Research, 8(1):1–23, 1981.

[9] Tansel Dokeroglu, Ender Sevinc, Tayfun Kucukyilmaz, and Ahmet Cosar. A survey on new generation metaheuristic algorithms. Computers & Industrial Engineering, 137:106040, 2019.

[10] Zelda B Zabinsky. Random search algorithms. Wiley encyclopedia of operations research and management science, 2010.

[11] Fred Glover and Manuel Laguna. Tabu search. In Handbook of combinatorial optimization, pages 2093–2229. Springer, 1998.

[12] Peter JM Van Laarhoven and Emile HL Aarts. Simulated annealing. In Simulated annealing: Theory and applications, pages 7–15. Springer, 1987.

[13] David E Goldberg. Genetic algorithms. Pearson Education India, 2006.

[14] Rainer Storn and Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. Journal of global optimization, 11(4):341–359, 1997.

[15] David B Fogel and Lawrence J Fogel. An introduction to evolutionary programming. In European conference on artificial evolution, pages 21–33. Springer, 1995.

[16] Thomas B¨ack, G¨unter Rudolph, and Hans-Paul Schwefel. Evolutionary programming and evolution strategies: Similarities and differences. In In Proceedings of the Second Annual Conference on Evolutionary Programming. Citeseer, 1993.

[17] 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.

[18] Marco Dorigo, Mauro Birattari, and Thomas Stutzle. Ant colony optimization. IEEE computational intelligence magazine, 1(4):28–39, 2006.

[19] Dervis Karaboga. An idea based on honey bee swarm for numerical optimization. Technical report, Citeseer, 2005.

[20] Xin-She Yang. A new metaheuristic bat-inspired algorithm. In Nature inspired cooperative strategies for optimization (NICSO 2010), pages 65– 74. Springer, 2010.

[21] Gerald Whittaker, Rolf F¨are, Shawna Grosskopf, Bradley Barnhart, Moriah Bostian, George Mueller-Warrant, and Stephen Griffith. Spatial targeting of agri-environmental policy using bilevel evolutionary optimization. Omega, 66:15–27, 2017.

[22] Ankur Sinha, Pekka Malo, and Kalyanmoy Deb. A review on bilevel optimization: from classical to evolutionary approaches and applications. IEEE Transactions on Evolutionary Computation, 22(2):276–295, 2017.

[23] Jerome Bracken and James T McGill. Mathematical programs with optimization problems in the constraints. Operations Research, 21(1):37–44, 1973.

[24] Ankur Sinha, Pekka Malo, Kalyanmoy Deb, Pekka Korhonen, and Jyrki Wallenius. Solving bilevel multicriterion optimization problems with lower level decision uncertainty. IEEE Transactions on Evolutionary Computation, 20(2):199–217, 2015.

[25] Kalyanmoy Deb and Ankur Sinha. An efficient and accurate solution methodology for bilevel multi-objective programming problems using a hybrid evolutionary-local-search algorithm. Evolutionary computation, 18(3):403–449, 2010.

[26] 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.

[27] Kalyanmoy Deb. Multi-objective optimization. In Search methodologies, pages 403–449. Springer, 2014.

[28] Melanie Mitchell. An introduction to genetic algorithms. MIT press, 1998.

[29] Kalyanmoy Deb. Multi-objective optimisation using evolutionary algorithms: an introduction. In Multi-objective evolutionary optimisation for product design and manufacturing, pages 3–34. Springer, 2011.

[30] Kaisa Miettinen. Nonlinear multiobjective optimization, volume 12. Springer Science & Business Media, 2012.

[31] Hugo Larochelle, Yoshua Bengio, J´erˆome Louradour, and Pascal Lamblin. Exploring strategies for training deep neural networks. Journal of machine learning research, 10(1), 2009.

[32] Yoshua Bengio, Pascal Lamblin, Dan Popovici, Hugo Larochelle, et al. Greedy layer-wise training of deep networks. Advances in neural information processing systems, 19:153, 2007.

[33] Marc Ranzato, Christopher Poultney, Sumit Chopra, Yann LeCun, et al. Efficient learning of sparse representations with an energy-based model. Advances in neural information processing systems, 19:1137, 2007.

[34] David Charte, Francisco Charte, Salvador Garc´ıa, Mar´ıa J del Jesus, and Francisco Herrera. A practical tutorial on autoencoders for nonlinear feature fusion: Taxonomy, models, software and guidelines. Information Fusion, 44:78–96, 2018.

[35] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, PierreAntoine Manzagol, and L´eon Bottou. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. Journal of machine learning research, 11(12), 2010.

[36] Andrew Ng et al. Sparse autoencoder. CS294A Lecture notes, 72(2011):1– 19, 2011.

[37] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In Icml, 2011.

[38] Carl Doersch. Tutorial on variational autoencoders. arXiv preprint arXiv:1606.05908, 2016.

[39] Adam Gaier and David Ha. Weight agnostic neural networks. arXiv preprint arXiv:1906.04358, 2019.

[40] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 8697–8710, 2018.

[41] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In Proceedings of the aaai conference on artificial intelligence, volume 33, pages 4780–4789, 2019.

[42] Chris Zhang, Mengye Ren, and Raquel Urtasun. Graph hypernetworks for neural architecture search. arXiv preprint arXiv:1810.05749, 2018.

[43] Sebastian Thrun and Lorien Pratt. Learning to learn. Springer Science & Business Media, 2012.

[44] Thomas Elsken, Jan Hendrik Metzen, Frank Hutter, et al. Neural architecture search: A survey. J. Mach. Learn. Res., 20(55):1–21, 2019.

[45] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. arXiv preprint arXiv:1511.07122, 2015.

[46] 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.

[47] Hector Mendoza, Aaron Klein, Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. Towards automatically-tuned neural networks. In Workshop on Automatic Machine Learning, pages 58–65. PMLR, 2016.

[48] 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.

[49] 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.

[50] 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.

[51] 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.

[52] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Efficient multiobjective neural architecture search via lamarckian evolution. arXiv preprint arXiv:1804.09081, 2018.

[53] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In International Conference on Machine Learning, pages 4095–4104. PMLR, 2018.

[54] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. arXiv preprint arXiv:1806.09055, 2018.

[55] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. Snas: stochastic neural architecture search. arXiv preprint arXiv:1812.09926, 2018.

[56] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In Proceedings of the European conference on computer vision (ECCV), pages 19–34, 2018.

[57] Frederic Runge, Danny Stoll, Stefan Falkner, and Frank Hutter. Learning to design rna. arXiv preprint arXiv:1812.11951, 2018.

[58] Arber Zela, Aaron Klein, Stefan Falkner, and Frank Hutter. Towards automated deep learning: Efficient joint neural architecture and hyperparameter search. arXiv preprint arXiv:1807.06906, 2018.

[59] MN Gurcan, LE Boucheron, A Can, A Madabhushi, NM Rajpoot, and B Yener. Histopathological image analysis: a review. ieee rev biomed eng 2: 147–171, 2009.

[60] Abtin Riasatian, Morteza Babaie, Danial Maleki, Shivam Kalra, Mojtaba Valipour, Sobhan Hemati, Manit Zaveri, Amir Safarpoor, Sobhan Shafiei, Mehdi Afshari, et al. Fine-tuning and training of densenet for histopathology image representation using tcga diagnostic slides. Medical Image Analysis, 70:102032, 2021.

[61] Hamid Reza Tizhoosh and Liron Pantanowitz. Artificial intelligence and digital pathology: challenges and opportunities. Journal of pathology informatics, 9, 2018.

[62] VíctorCampos, Francesc Sastre, Maurici Yagües, Míriam Bellver, Xavier Giró-i-Nieto, and Jordi Torres. Distributed training strategies for a computer vision deep learning algorithm on a distributed gpu cluster. Procedia Computer Science, 108:315–324, 2017.

[63] Guilherme Andrade, Renato Ferreira, George Teodoro, Leonardo Rocha, Joel H Saltz, and Tahsin Kurc. Efficient execution of microscopy image analysis on cpu, gpu, and mic equipped cluster systems. In 2014 IEEE 26th International Symposium on Computer Architecture and High Performance Computing, pages 89–96. IEEE, 2014.

[64] Joseph Mellor, Jack Turner, Amos Storkey, and Elliot J Crowley. Neural architecture search without training. arXiv preprint arXiv:2006.04647, 2020.

[65] Filipe Assuncao, David Sereno, Nuno Lourenco, Penousal Machado, and Bernardete Ribeiro. Automatic evolution of autoencoders for compressed representations. In 2018 IEEE Congress on Evolutionary Computation (CEC), pages 1–8. IEEE, 2018.

[66] Francisco Charte, Antonio J Rivera, Francisco Mart´ınez, and Mar´ıa J del Jesus. Automating autoencoder architecture configuration: An evolutionary approach. In International Work-Conference on the Interplay Between Natural and Artificial Computation, pages 339–349. Springer, 2019.

[67] Marijn van Knippenberg, Vlado Menkovski, and Sergio Consoli. Evolutionary construction of convolutional neural networks. In International Conference on Machine Learning, Optimization, and Data Science, pages 293–304. Springer, 2018.

[68] Marc Ranzato, Christopher Poultney, Sumit Chopra, Yann LeCun, et al. Efficient learning of sparse representations with an energy-based model. Advances in neural information processing systems, 19:1137, 2007.

[69] Xiangru Chen, Yanan Sun, Mengjie Zhang, and Dezhong Peng. Evolving deep convolutional variational autoencoders for image classification. IEEE Transactions on Evolutionary Computation, 2020.

[70] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

[71] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:1708.07747, 2017.

[72] Mohri, Mehryar, Afshin Rostamizadeh, and Ameet Talwalkar. Foundations of machine learning. MIT press, 2018.

[73] Le, Lei, Andrew Patterson, and Martha White. Supervised autoencoders: Improving generalization performance with unsupervised regularizers. Advances in neural information processing systems 31, pages 107-117, 2018.

[74] Wold, Svante, Kim Esbensen, and Paul Geladi. "Principal component analysis." *Chemometrics and intelligent laboratory systems* 2.1-3 (1987): 37-52.

[75] Bahadur, Nitish, and Randy Paffenroth. Dimension estimation using autoencoders. arXiv preprint arXiv:1909.10702 (2019).

[76] Farsa, Davood Zaman, and Shahryar Rahnamayan. Discrete Coordinate Descent (DCD). 2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC). IEEE, 2020.

[77] Rahnamayan, Shahryar, and Seyed Jalaleddin Mousavirad. Towards Solving Large-scale Expensive Optimization Problems Efficiently Using Coordinate Descent Algorithm. *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2020.

[78] Taghanaki, Saeid Asgari, Jeremy Kawahara, Brandon Miles, and Ghassan Hamarneh. Pareto-optimal multi-objective dimensionality reduction deep auto-encoder for mammography classification. Computer methods and programs in biomedicine 145, pages 85-93, 2017.

[79] Gandomi, Amir H., et al. "Using semi-independent variables to enhance optimization search." *Expert Systems with Applications* 120 (2019): 279-297.

Appendices

Appendix A.

A.1 Detailed PF solutions for all Histopathology Schemes

To clarify, in all the tables here that we have shown the full detail of the PF like Tables 4.12, 4.13, etc. the top two rows under "Optimization" column are the experimental settings during evolution that yielded the "Primary results". These rows include the Npop, #Epochs, Batch Size, #Generations, Data Size, and Neuron range that were used to generate the PF solution and their primary results. Then, to fully train these PF solutions we used #Epochs = 200, Batch Size = 64, and the same Data Size but, as we do not have any evolutionary process during full training we do not have Npop, #Generations, and Neuron range anymore. This situation is the same for all other experiments. And for the primary results during Evolution, the code layer is the middle layer, while after full training the code layer is the hidden layer that has the highest F1-score among all hidden layers.

Table 4.12: Investigation on the 2 objectives of RMSE and FLOPs on Brain Tumor. The mentioned hyperparameters were only used during evolution. Each AE was fully trained for 200 epochs and batch size of 64 with the same amount of data. The F1-score of each AE is resulted form the K-NN classifier with k = 3 on the corresponding code layer with #Neurons = Code size. For the primary results during evolution, the code layer is the middle layer, while after full training the code layer is the hidden layer that has the highest F1-score among all hidden layers.

Optimization						
Dataset	Npop	#Epochs	Batch size	#Generations	Data Size (train, validation, test)	Neuron range
Brain	20	1	4	100	(1324, 76, 74)	[0,1) , Initial =[0.5,1)

	Primary results		**K-NN over validation set after Full Training**				
Structures	RMSE	FLOPs	RMSE	Averaged F1-score	F1-score per class	Code Layer	Code Size
1024 [1, 0, 0, 0, 0, 0, 1] 1024	0.05	4.10E+03	0.05	0.34	[0.69, 0.0]	1	1
1024 [1, 1, 0, 0, 0, 1, 1] 1024	0.05	4.10E+03	0.05	0.34	[0.69, 0.0]	2	1
1024 [2, 0, 1, 0, 1, 0, 2] 1024	0.05	8.20E+03	0.05	0.64	[0.67, 0.61]	2	1
1024 [8, 6, 4, 1, 4, 6, 8] 1024	0.04	3.31E+04	0.04	0.95	[0.95, 0.94]	1	8
1024 [12, 10, 8, 4, 8, 10, 12] 1024	0.04	5.01E+04	0.04	0.95	[0.95, 0.94]	1	12
1024 [12, 10, 8, 5, 8, 10, 12] 1024	0.04	5.01E+04	0.04	0.96	[0.96, 0.96]	5	8
1024 [15, 13, 11, 8, 11, 13, 15] 1024	0.04	6.31E+04	0.04	0.96	[0.96, 0.96]	4	8
1024 [17, 15, 9, 4, 9, 15, 17] 1024	0.04	7.13E+04	0.04	0.91	[0.91, 0.9]	4	4
1024 [22, 21, 19, 13, 19, 21, 22] 1024	0.04	9.45E+04	0.03	0.91	[0.92, 0.9]	3	19
1024 [23, 21, 19, 12, 19, 21, 23] 1024	0.04	9.86E+04	0.03	0.95	[0.95, 0.94]	4	12
1024 [24, 19, 14, 9, 14, 19, 24] 1024	0.04	1.02E+05	0.03	0.95	[0.95, 0.94]	6	19
1024 [25, 22, 14, 13, 14, 22, 25] 1024	0.04	1.07E+05	0.03	0.95	[0.95, 0.94]	5	14
1024 [31, 24, 17, 12, 17, 24, 31] 1024	0.04	1.32E+05	0.03	0.92	[0.93, 0.91]	5	17
1024 [43, 27, 18, 13, 18, 27, 43] 1024	0.04	1.84E+05	0.03	0.93	[0.94, 0.93]	6	27
1024 [69, 31, 18, 5, 18, 31, 69] 1024	0.04	2.94E+05	0.03	0.95	[0.95, 0.94]	6	31
1024 [149, 125, 87, 53, 87, 125, 149] 1024	0.04	7.47E+05	0.02	0.93	[0.94, 0.93]	2	125
1024 [0, 249, 107, 46, 107, 249, 0] 1024	0.04	1.15E+06	0.02	0.91	[0.92, 0.9]	4	107
1024 [0, 561, 227, 142, 227, 561, 0] 1024	0.04	2.94E+06	0.02	0.93	[0.94, 0.93]	2	227

Table 4.13: Investigation on the 2 objectives of RMSE and FLOPs on Pulmonary Tumor. The mentioned hyperparameters were only used during evolution. Each AE was fully trained for 200 epochs and batch size of 64 with the same amount of data. The F1-score of each AE is resulted form the K-NN classifier with k = 3 on the corresponding code layer with #Neurons = Code size. For the primary results during evolution, the code layer is the middle layer, while after full training the code layer is the hidden layer that has the highest F1-score among all hidden layers.

Optimization						
Dataset	Npop	#Epochs	Batch size	#Generations	Data Size (train, validation, test)	Neuron range
Pulmonary	20	1	4	100	(719, 86, 84)	[0,1) , Initial =[0.5,1)

	Primary results		**K-NN over validation set after Full Training**				
Structures	RMSE	FLOPs	RMSE	Averaged F1-score	F1-score per class	Code Layer	Code Size
1024 [1, 0, 0, 0, 0, 0, 1] 1024	0.05	4.10E+03	0.05	0.21	[0.62, 0.0, 0.0]	1	1
1024 [2, 1, 0, 0, 0, 1, 2] 1024	0.05	8.20E+03	0.05	0.29	[0.4, 0.48, 0.0]	1	2
1024 [6, 2, 1, 0, 1, 2, 6] 1024	0.05	2.46E+04	0.05	0.5	[0.42, 0.58, 0.5]	2	2
1024 [6, 3, 2, 1, 2, 3, 6] 1024	0.05	2.47E+04	0.05	0.58	[0.75, 0.62, 0.38]	1	6
1024 [8, 3, 1, 0, 1, 3, 8] 1024	0.05	3.29E+04	0.05	0.43	[0.47, 0.49, 0.31]	1	8
1024 [32, 27, 19, 9, 19, 27, 32] 1024	0.05	1.37E+05	0.05	0.59	[0.77, 0.69, 0.29]	1	32
1024 [36, 17, 10, 5, 10, 17, 36] 1024	0.05	1.51E+05	0.04	0.57	[0.69, 0.67, 0.34]	1	36
1024 [41, 21, 11, 6, 11, 21, 41] 1024	0.05	1.73E+05	0.05	0.64	[0.87, 0.73, 0.32]	2	21
1024 [42, 26, 10, 6, 10, 26, 42] 1024	0.05	1.78E+05	0.05	0.62	[0.81, 0.75, 0.31]	1	42
1024 [42, 32, 12, 6, 12, 32, 42] 1024	0.05	1.79E+05	0.04	0.74	[0.85, 0.81, 0.56]	7	42
1024 [46, 18, 10, 6, 10, 18, 46] 1024	0.05	1.93E+05	0.05	0.7	[0.77, 0.84, 0.5]	2	18
1024 [50, 39, 31, 18, 31, 39, 50] 1024	0.05	2.20E+05	0.04	0.75	[0.87, 0.85, 0.53]	6	39
1024 [55, 39, 24, 14, 24, 39, 55] 1024	0.04	2.39E+05	0.04	0.71	[0.85, 0.86, 0.43]	7	55
1024 [57, 47, 34, 19, 34, 47, 57] 1024	0.04	2.53E+05	0.04	0.63	[0.84, 0.72, 0.33]	6	47
1024 [134, 75, 44, 21, 44, 75, 134] 1024	0.04	6.06E+05	0.04	0.81	[0.9, 0.89, 0.62]	7	134
1024 [140, 82, 31, 16, 31, 82, 140] 1024	0.04	6.32E+05	0.04	0.75	[0.85, 0.9, 0.5]	4	16

Table 4.14: Investigation on the 2 objectives of RMSE and Classification Error on Brain Tumor. The F1-score of each AE is resulted form the K-NN classifier with k = 3 on the corresponding code layer with #Neurons = Code size. For the primary results during evolution, the code layer is the middle layer, while after full training the code layer is the hidden layer that has the highest F1-score among all hidden layers.

Optimization							Full Training				
Dataset	Npop	#Epochs	Batch size	#Gener-ations	Data Size (train, validation, test)	Neuron range	K	#Epochs	Batch size		
Brain	20	1	4	100	(1324, 76, 74)	[0,1) , Initial =[0.5,1)	3	200	64		
	Primary results only by RMSE						**K-NN over validation set after Full Training**				
Structures	RMSE	Classification Error	Averaged F1-score	F1-score per class	FLOPs	Code Size	Averaged F1-score	F1-score per class	Code Layer	Code Size	RMSE
1024 [354, 38, 14, 12, 14, 38, 354] 1024	0.04	0.01	0.99	[0.99, 0.99]	1.51E+06	12	0.91	[0.92, 0.9]	4	12	0.03
1024 [291, 60, 17, 13, 17, 60, 291] 1024	0.04	0.03	0.97	[0.97, 0.97]	1.27E+06	13	0.92	[0.93, 0.91]	4	13	0.03
1024 [192, 73, 41, 34, 41, 73, 192] 1024	0.04	0.04	0.96	[0.96, 0.96]	8.60E+05	34	0.92	[0.93, 0.91]	4	34	0.03
1024 [230, 42, 32, 26, 32, 42, 230] 1024	0.04	0.05	0.95	[0.95, 0.94]	9.89E+05	26	0.95	[0.95, 0.94]	4	26	0.03
1024 [729, 170, 109, 66, 109, 170, 729] 1024	0.04	0.07	0.93	[0.94, 0.93]	3.58E+06	66	0.93	[0.94, 0.93]	3	109	0.03
1024 [363, 41, 32, 26, 32, 41, 363] 1024	0.04	0.08	0.92	[0.93, 0.91]	1.55E+06	26	0.92	[0.93, 0.91]	4	26	0.03

Table 4.15: Investigation on the 2 objectives of RMSE and Classification Error on Pulmonary Tumor. The F1-score of each AE is resulted form the K-NN classifier with k = 3 on the corresponding code layer with #Neurons = Code size. For the primary results during evolution, the code layer is the middle layer, while after full training the code layer is the hidden layer that has the highest F1-score among all hidden layers.

Optimization							Full Training				
Dataset	Npop	#Epochs	Batch size	#Gener-ations	Data Size (train, validation, test)	Neuron range	K	#Epochs	Batch size		
Pulmonary	20	1	4	100	(719, 86, 84)	[0,1) , Initial =[0.5,1)	3	200	64		
	Primary results only by RMSE						**K-NN over validation set after Full Training**				
Structures	RMSE	Classification Error	Averaged F1-score	F1-score per class	FLOPs	Code Size	Averaged F1-score	F1-score per class	Code Layer	Code Size	RMSE
1024 [493, 210, 115, 54, 115, 210, 493] 1024	0.04	0.27	0.73	[0.89, 0.82, 0.5]	2.55E+06	54	0.86	[0.9, 0.9, 0.77]	7	493	0.03
1024 [527, 140, 101, 44, 101, 140, 527] 1024	0.04	0.28	0.72	[0.89, 0.82, 0.43]	2.53E+06	44	0.8	[0.87, 0.89, 0.62]	6	140	0.04
1024 [321, 204, 141, 79, 141, 204, 321] 1024	0.04	0.3	0.7	[0.85, 0.79, 0.45]	1.74E+06	79	0.87	[0.87, 0.9, 0.83]	4	79	0.03
1024 [637, 187, 130, 84, 130, 187, 637] 1024	0.04	0.32	0.68	[0.85, 0.79, 0.42]	3.23E+06	84	0.86	[0.92, 0.95, 0.71]	5	130	0.03
1024 [478, 254, 139, 93, 139, 254, 478] 1024	0.04	0.32	0.68	[0.9, 0.79, 0.36]	2.64E+06	93	0.82	[0.89, 0.92, 0.67]	3	139	0.03
1024 [446, 344, 266, 194, 266, 344, 446] 1024	0.04	0.32	0.68	[0.82, 0.79, 0.42]	3.01E+06	194	0.82	[0.82, 0.87, 0.77]	7	446	0.03
1024 [562, 122, 63, 44, 63, 122, 562] 1024	0.04	0.32	0.68	[0.83, 0.78, 0.42]	2.62E+06	44	0.89	[0.92, 0.93, 0.83]	6	122	0.04
1024 [593, 209, 155, 78, 155, 209, 593] 1024	0.04	0.32	0.68	[0.8, 0.81, 0.42]	3.10E+06	78	0.82	[0.85, 0.89, 0.71]	7	593	0.03
1024 [469, 280, 214, 160, 214, 280, 469] 1024	0.04	0.33	0.67	[0.78, 0.85, 0.38]	2.82E+06	160	0.88	[0.86, 0.87, 0.91]	7	469	0.03
1024 [388, 104, 33, 20, 33, 104, 388] 1024	0.04	0.36	0.64	[0.79, 0.77, 0.37]	1.77E+06	20	0.82	[0.86, 0.88, 0.71]	7	388	0.04
1024 [230, 45, 28, 12, 28, 45, 230] 1024	0.04	0.36	0.64	[0.81, 0.75, 0.37]	9.90E+05	12	0.67	[0.84, 0.8, 0.38]	1	230	0.04
1024 [576, 190, 143, 93, 143, 190, 576] 1024	0.04	0.36	0.64	[0.8, 0.77, 0.36]	2.96E+06	93	0.81	[0.87, 0.88, 0.67]	7	576	0.03
1024 [550, 174, 142, 67, 142, 174, 550] 1024	0.04	0.38	0.62	[0.77, 0.73, 0.37]	2.77E+06	67	0.83	[0.84, 0.89, 0.77]	7	550	0.03
1024 [518, 225, 153, 85, 153, 225, 518] 1024	0.04	0.39	0.61	[0.75, 0.74, 0.36]	2.78E+06	85	0.87	[0.87, 0.89, 0.83]	7	518	0.03

Table 4.16: Investigation on the 2 objectives of Classification Error and FLOPs on Brain Tumor. The F1-score of each AE is resulted form the K-NN classifier with k = 3 on the corresponding code layer with #Neurons = Code size. For the primary results during evolution, the code layer is the middle layer, while after full training the code layer is the hidden layer that has the highest F1-score among all hidden layers.

Optimization							Full Training				
Dataset	Npop	#Epochs	Batch size	#Generations	Data Size (train, validation, test)	Neuron range	K	#Epochs	Batch size		
Brain	20	1	4	100	(1324, 76, 74)	[0,1) , Initial =[0.5,1)	3	200	64		
	Primary results only by RMSE						**K-NN over validation set after Full Training**				
Structures	RMSE	Classification Error	Averaged F1-score	F1-score per class	FLOPs	Code Size	Averaged F1-score	F1-score per class	Code Layer	Code Size	RMSE
1024 [21, 18, 12, 5, 12, 18, 21] 1024	0.04	0.01	0.99	[0.99, 0.99]	8.86E+04	5	0.95	[0.95, 0.94]	5	12	0.04
1024 [15, 12, 10, 5, 10, 12, 15] 1024	0.04	0.03	0.97	[0.97, 0.97]	6.28E+04	5	0.93	[0.94, 0.93]	2	12	0.04
1024 [11, 10, 7, 6, 7, 10, 11] 1024	0.04	0.03	0.97	[0.97, 0.97]	4.59E+04	6	0.95	[0.95, 0.94]	4	6	0.04
1024 [9, 9, 7, 5, 7, 9, 9] 1024	0.04	0.04	0.96	[0.96, 0.96]	3.76E+04	5	0.96	[0.96, 0.96]	7	9	0.04
1024 [7, 6, 4, 2, 4, 6, 7] 1024	0.05	0.05	0.95	[0.95, 0.95]	2.90E+04	2	0.83	[0.87, 0.8]	1	7	0.05
1024 [6, 5, 3, 2, 3, 5, 6] 1024	0.05	0.07	0.93	[0.94, 0.93]	2.48E+04	2	0.6	[0.58, 0.62]	4	2	0.05
1024 [1, 0, 1, 1, 1, 0, 1] 1024	0.05	0.08	0.92	[0.93, 0.92]	4.10E+03	1	0.48	[0.72, 0.24]	1	1	0.05
1024 [1, 0, 0, 1, 0, 0, 1] 1024	0.05	0.09	0.91	[0.91, 0.91]	4.10E+03	1	0.57	[0.52, 0.63]	1	1	0.05
1024 [1, 0, 0, 0, 0, 0, 1] 1024	0.05	0.17	0.83	[0.84, 0.82]	4.10E+03	1	0.32	[0.0, 0.64]	1	1	0.05

Table 4.17: Investigation on the 2 objectives of Classification Error and FLOPs on Pulmonary Tumor. The F1-score of each AE is resulted form the K-NN classifier with k = 3 on the corresponding code layer with #Neurons = Code size. For the primary results during evolution, the code layer is the middle layer, while after full training the code layer is the hidden layer that has the highest F1-score among all hidden layers.

Optimization							Full Training				
Dataset	Npop	#Epochs	Batch size	#Generati-ons	Data Size (train, validation, test)	Neuron range	K	#Epochs	Batch size		
Pulmonary	20	1	4	100	(719, 86, 84)	[0,1) , Initial =[0.5,1)	3	200	64		
	Primary results only by RMSE						**K-NN over validation set after Full Training**				
Structures	RMSE	Classification Error	AF1	F1-score per class	FLOPs	Code Size	AF1	F1-score per class	Code Layer	Code Size	RMSE
1024 [165, 82, 46, 39, 46, 82, 165] 1024	0.04	0.26	0.74	[0.9, 0.85, 0.45]	7.52E+05	39	0.74	[0.87, 0.86, 0.5]	1	165	0.04
1024 [129, 95, 50, 28, 50, 95, 129] 1024	0.05	0.28	0.72	[0.86, 0.84, 0.45]	6.02E+05	28	0.72	[0.81, 0.84, 0.5]	1	129	0.04
1024 [94, 58, 27, 12, 27, 58, 94] 1024	0.05	0.3	0.7	[0.87, 0.84, 0.4]	4.14E+05	12	0.69	[0.84, 0.83, 0.42]	2	58	0.04
1024 [83, 53, 30, 24, 30, 53, 83] 1024	0.05	0.33	0.67	[0.83, 0.84, 0.34]	3.67E+05	24	0.66	[0.83, 0.8, 0.36]	1	83	0.04
1024 [83, 51, 22, 10, 22, 51, 83] 1024	0.05	0.34	0.66	[0.84, 0.81, 0.34]	3.62E+05	10	0.65	[0.78, 0.81, 0.37]	3	22	0.04
1024 [60, 33, 18, 10, 18, 33, 60] 1024	0.04	0.34	0.66	[0.8, 0.77, 0.4]	2.57E+05	10	0.9	[0.92, 0.90, 0.88]	7	60	0.04
1024 [56, 42, 23, 14, 23, 42, 56] 1024	0.05	0.35	0.65	[0.85, 0.74, 0.37]	2.44E+05	14	0.72	[0.84, 0.85, 0.48]	1	56	0.04
1024 [43, 23, 14, 5, 14, 23, 43] 1024	0.05	0.36	0.64	[0.82, 0.76, 0.36]	1.82E+05	5	0.67	[0.85, 0.79, 0.38]	7	43	0.05
1024 [35, 23, 13, 7, 13, 23, 35] 1024	0.05	0.37	0.63	[0.72, 0.71, 0.45]	1.48E+05	7	0.65	[0.82, 0.79, 0.34]	1	35	0.05
1024 [17, 9, 3, 0, 3, 9, 17] 1024	0.05	0.44	0.56	[0.75, 0.71, 0.22]	7.04E+04	3	0.65	[0.68, 0.77, 0.5]	1	17	0.05
1024 [16, 9, 6, 2, 6, 9, 16] 1024	0.05	0.49	0.51	[0.61, 0.56, 0.37]	6.64E+04	2	0.65	[0.8, 0.74, 0.4]	1	16	0.05
1024 [1, 1, 0, 0, 0, 1, 1] 1024	0.05	0.49	0.51	[0.52, 0.55, 0.45]	4.10E+03	1	0.33	[0.29, 0.57, 0.13]	1	1	0.05
1024 [1, 0, 0, 0, 0, 0, 1] 1024	0.05	0.51	0.49	[0.7, 0.76, 0.0]	4.10E+03	1	0.41	[0.47, 0.44, 0.31]	1	1	0.05

Table 4.18: Investigation on the 2 objectives of Classification Error and FLOPs with Summation of Losses vs. single RMSE Loss on Brain Tumor. The F1-score of each AE is resulted form the K-NN classifier with k = 3 on the corresponding code layer with #Neurons = Code size. For the primary results during evolution, the code layer is the middle layer, while after full training the code layer is the hidden layer that has the highest F1-score among all hidden layers.

Optimization							Full Training by Sum of Losses					Full Training by RMSE				
Dataset	Npop	#Epoch	Batch size	#Ge-nerat-ions	Data Size (train, validation, test)	Neuron range	K	#Epoch	Batch size			K	#Epoch	Batch size		
Brain	20	1	4	100	(1324, 76, 74)	[0,1) , Initial =[0.5,1)	3	200	64			3	200	64		
	Primary results only by Sum of Losses						**K-NN over validation set after Full Training**					**K-NN over validation set after Full Training**				
Structures	RMSE	Classif-ication Error	AF1	F1-score per class	FLOPs	Code Size	AF1	F1-score per class	Code Layer	Code Size	RMSE	AF1	F1-score per class	Code Layer	Code Size	RMSE
1024 [418, 174, 103, 68, 103, 174, 418] 1024	0.07	0.01	0.99	[0.99, 0.99]	2.10E+06	68	0.91	[0.92, 0.9]	1	418	0.28	0.95	[0.95, 0.94]	7	418	0.03
1024 [245, 91, 71, 61, 71, 91, 245] 1024	0.12	0.03	0.97	[0.97, 0.97]	1.14E+06	61	0.96	[0.96, 0.96]	1	245	0.06	0.91	[0.92, 0.9]	1	245	0.02
1024 [99, 23, 6, 6, 6, 23, 99] 1024	0.12	0.03	0.97	[0.98, 0.97]	4.15E+05	6	0.93	[0.94, 0.93]	3	6	0.06	0.93	[0.94, 0.93]	3	6	0.03
1024 [34, 18, 5, 2, 5, 18, 34] 1024	0.11	0.04	0.96	[0.96, 0.96]	1.42E+05	2	0.78	[0.79, 0.76]	1	34	0.05	0.95	[0.95, 0.95]	2	18	0.04
1024 [26, 14, 12, 7, 12, 14, 26] 1024	0.12	0.04	0.96	[0.96, 0.96]	1.09E+05	7	0.43	[0.24, 0.63]	6	14	0.07	0.92	[0.92, 0.92]	1	26	0.04
1024 [17, 12, 8, 4, 8, 12, 17] 1024	0.13	0.05	0.95	[0.95, 0.94]	7.10E+04	4	0.95	[0.95, 0.95]	2	12	0.07	0.93	[0.94, 0.93]	1	17	0.04
1024 [15, 6, 5, 3, 5, 6, 15] 1024	0.12	0.07	0.93	[0.94, 0.93]	6.20E+04	3	0.46	[0.67, 0.26]	7	15	0.08	0.76	[0.79, 0.74]	1	15	0.05
1024 [13, 10, 7, 5, 7, 10, 13] 1024	0.09	0.08	0.92	[0.93, 0.92]	5.42E+04	5	0.4	[0.2, 0.59]	6	10	0.08	0.95	[0.95, 0.94]	3	7	0.04
1024 [9, 6, 4, 1, 4, 6, 9] 1024	0.11	0.08	0.92	[0.93, 0.91]	3.72E+04	1	0.89	[0.89, 0.89]	1	9	0.08	0.93	[0.94, 0.93]	4	1	0.04
1024 [2, 2, 1, 0, 1, 2, 2] 1024	0.06	0.09	0.91	[0.91, 0.9]	8.22E+03	1	0.45	[0.7, 0.2]	1	2	0.05	0.76	[0.78, 0.74]	1	2	0.05
1024 [1, 0, 0, 0, 0, 0, 0, 1] 1024	0.07	0.11	0.89	[0.9, 0.89]	4.10E+03	1	0.34	[0.69, 0.0]	1	1	0.05	0.34	[0.69, 0.0]	1	1	0.05

Table 4.19: Investigation on the 2 objectives of Classification Error and FLOPs with Summation of Losses vs. single RMSE Loss on Pulmonary Tumor. The F1-score of each AE is resulted form the K-NN classifier with k = 3 on the corresponding code layer with #Neurons = Code size. For the primary results during evolution, the code layer is the middle layer, while after full training the code layer is the hidden layer that has the highest F1-score among all hidden layers.

Optimization							Full Training by Sum of Losses			Full Training by RMSE		
Dataset	Npop	#Epoch	Batch size	#Gene-rations	Data Size (train, validation, test)	Neuron range	K	#Epoch	Batch size	K	#Epoch	Batch size
Pulmonary	20	1	4	100	(719, 86, 84)	[0,1) , Initial =[0.5,1)	3	200	64	3	200	64

	Primary results only by Sum of Losses						**K-NN over validation set after Full Training**					**K-NN over validation set after Full Training**				
Structures	RMSE	Classif-ication Error	AF1	F1-score per class	FLOPs	Code Size	AF1	F1-score per class	Code Layer	Code Size	RMSE	AF1	F1-score per class	Code Layer	Code Size	RMSE
1024 [155, 67, 33, 23, 33, 67, 155] 1024	0.07	0.21	0.79	[0.74, 0.81, 0.83]	6.88E+05	23	0.58	[0.63, 0.65, 0.48]	1	155	0.07	0.82	[0.85, 0.89, 0.71]	7	155	0.04
1024 [33, 6, 3, 2, 3, 6, 33] 1024	0.05	0.25	0.75	[0.7, 0.71, 0.83]	1.36E+05	2	0.39	[0.38, 0.37, 0.42]	4	2	0.07	0.52	[0.67, 0.57, 0.32]	1	33	0.05
1024 [28, 14, 12, 6, 12, 14, 28] 1024	0.07	0.3	0.7	[0.76, 0.75, 0.59]	1.17E+05	6	0.47	[0.45, 0.53, 0.42]	4	6	0.07	0.62	[0.81, 0.72, 0.31]	1	28	0.05
1024 [23, 15, 11, 5, 11, 15, 23] 1024	0.12	0.32	0.68	[0.72, 0.8, 0.5]	9.65E+04	5	0.52	[0.62, 0.59, 0.36]	1	23	0.07	0.68	[0.85, 0.75, 0.43]	1	23	0.05
1024 [21, 11, 9, 4, 9, 11, 21] 1024	0.08	0.39	0.61	[0.58, 0.69, 0.56]	8.75E+04	4	0.55	[0.69, 0.62, 0.34]	1	21	0.07	0.67	[0.83, 0.77, 0.4]	3	9	0.04
1024 [16, 9, 6, 4, 6, 9, 16] 1024	0.07	0.39	0.61	[0.6, 0.67, 0.56]	6.64E+04	4	0.51	[0.7, 0.51, 0.31]	1	16	0.06	0.59	[0.73, 0.58, 0.48]	5	6	0.05
1024 [9, 7, 4, 3, 4, 7, 9] 1024	0.07	0.41	0.59	[0.6, 0.63, 0.53]	3.73E+04	3	0.04	[0.0, 0.0, 0.11]	4	3	0.10	0.46	[0.7, 0.43, 0.25]	1	9	0.05
1024 [6, 3, 2, 1, 2, 3, 6] 1024	0.08	0.42	0.58	[0.46, 0.65, 0.62]	2.47E+04	1	0.68	[0.73, 0.78, 0.53]	1	6	0.05	0.39	[0.42, 0.37, 0.38]	1	6	0.05
1024 [4, 0, 3, 2, 3, 0, 4] 1024	0.07	0.44	0.56	[0.68, 0.64, 0.36]	1.65E+04	2	0.04	[0.0, 0.0, 0.12]	3	2	0.15	0.61	[0.54, 0.63, 0.67]	1	4	0.05
1024 [2, 1, 0, 0, 0, 1, 2] 1024	0.05	0.44	0.56	[0.48, 0.61, 0.59]	8.20E+03	1	0.39	[0.64, 0.53, 0.0]	1	2	0.05	0.42	[0.58, 0.68, 0.0]	2	1	0.05
1024 [1, 0, 1, 0, 1, 0, 1] 1024	0.05	0.51	0.49	[0.61, 0.62, 0.25]	4.10E+03	1	0.21	[0.59, 0.04, 0.0]	1	1	0.05	0.42	[0.47, 0.63, 0.15]	1	1	0.05
1024 [1, 0, 0, 0, 0, 0, 1] 1024	0.05	0.71	0.29	[0.45, 0.42, 0.0]	4.10E+03	1	0.21	[0.62, 0.0, 0.0]	1	1	0.05	0.21	[0.62, 0.0, 0.0]	1	1	0.05

Table 4.20: Investigation on the 4 objectives of RMSE, Classification Error, Code Size and FLOPs on Brain Tumor with 100 Generations. The F1-score of each AE is resulted form the K-NN classifier with k = 3 on the corresponding code layer with #Neurons = Code size. For the primary results during evolution, the code layer is the middle layer, while after full training the code layer is the hidden layer that has the highest F1-score among all hidden layers.

Optimization							Full training via RMSE				
Dataset	Npop	#Epochs	Batch size	#Generations	Data Size (train, validation, test)	Neuron range	K	#Epochs	Batch size		
Brain	20	1	4	100	(1324, 76, 74)	[0,1) , Initial=[0.5,1)	3	200	64		
	Primary results only by RMSE						**K-NN over validation set after Full Training**				
Structures	RMSE	Classification Error	Averaged F1-score	F1-score per class	FLOPs	Code Size	Averaged F1-score	F1-score per class	Code Layer	Code Size	RMSE
1024 [167, 97, 45, 3, 45, 97, 167] 1024	0.04	0.03	0.97	[0.97, 0.97]	7.67E+05	3	0.96	[0.96, 0.96]	1	167	0.05
1024 [575, 196, 108, 71, 108, 196, 575] 1024	0.04	0.03	0.97	[0.97, 0.97]	2.92E+06	71	0.93	[0.94, 0.93]	5	108	0.03
1024 [119, 70, 35, 1, 35, 70, 119] 1024	0.04	0.03	0.97	[0.97, 0.97]	5.31E+05	1	0.96	[0.96, 0.96]	1	119	0.04
1024 [262, 138, 52, 0, 52, 138, 262] 1024	0.04	0.04	0.96	[0.96, 0.96]	1.25E+06	52	0.95	[0.95, 0.94]	5	138	0.02
1024 [38, 14, 3, 2, 3, 14, 38] 1024	0.04	0.05	0.95	[0.95, 0.95]	1.58E+05	2	0.91	[0.91, 0.9]	1	38	0.04
1024 [244, 150, 61, 0, 61, 150, 244] 1024	0.04	0.05	0.95	[0.95, 0.94]	1.19E+06	61	0.93	[0.94, 0.93]	4	61	0.02
1024 [431, 116, 40, 27, 40, 116, 431] 1024	0.04	0.05	0.95	[0.95, 0.94]	1.99E+06	27	0.92	[0.93, 0.91]	3	40	0.03
1024 [447, 235, 25, 7, 25, 235, 447] 1024	0.04	0.05	0.95	[0.95, 0.94]	2.28E+06	7	0.93	[0.94, 0.93]	3	25	0.03
1024 [230, 126, 54, 17, 54, 126, 230] 1024	0.04	0.07	0.93	[0.94, 0.93]	1.09E+06	17	0.92	[0.93, 0.91]	3	54	0.03
1024 [137, 93, 52, 29, 52, 93, 137] 1024	0.04	0.07	0.93	[0.94, 0.93]	6.37E+05	29	0.92	[0.93, 0.91]	1	137	0.03
1024 [338, 104, 25, 6, 25, 104, 338] 1024	0.04	0.08	0.92	[0.93, 0.91]	1.54E+06	6	0.93	[0.94, 0.93]	7	338	0.03
1024 [131, 95, 33, 9, 33, 95, 131] 1024	0.04	0.09	0.91	[0.91, 0.9]	6.00E+05	9	0.93	[0.94, 0.93]	4	9	0.03
1024 [40, 9, 3, 1, 3, 9, 40] 1024	0.05	0.12	0.88	[0.89, 0.87]	1.65E+05	1	0.93	[0.94, 0.93]	1	40	0.04
1024 [40, 8, 3, 1, 3, 8, 40] 1024	0.05	0.16	0.84	[0.85, 0.83]	1.65E+05	1	0.93	[0.94, 0.93]	2	8	0.04
1024 [27, 9, 2, 1, 2, 9, 27] 1024	0.05	0.23	0.77	[0.8, 0.75]	1.12E+05	1	0.81	[0.83, 0.8]	1	27	0.05
1024 [1, 0, 0, 0, 0, 0, 1] 1024	0.05	0.68	0.32	[0.0, 0.64]	4.10E+03	1	0.32	[0.0, 0.64]	1	1	0.05

Table 4.21: Investigation on the 4 objectives of RMSE, Classification Error, Code Size and FLOPs on Pulmonary Tumor with 100 Generations. The F1-score of each AE is resulted form the K-NN classifier with k = 3 on the corresponding code layer with #Neurons = Code size. For the primary results during evolution, the code layer is the middle layer, while after full training the code layer is the hidden layer that has the highest F1-score among all hidden layers.

Optimization							Full training via RMSE				
Dataset	Npop	#Epochs	Batch size	#Generations	Data Size (train, validation, test)	Neuron range	K	#Epochs	Batch size		
Pulmonary	20	1	4	100	(719, 86, 84)	[0,1) , Initial=[0.5,1)	3	200	64		
	Primary results only by RMSE						**K-NN over validation set after Full Training**				
Structures	RMSE	Classification Error	Averaged F1-score	F1-score per class	FLOPs	Code Size	Averaged F1-score	F1-score per class	Code Layer	Code Size	RMSE
1024 [690, 475, 394, 211, 394, 475, 690] 1024	0.05	0.25	0.75	[0.85, 0.82, 0.59]	5.22E+06	211	0.87	[0.89, 0.88, 0.83]	6	475	0.03
1024 [315, 209, 133, 68, 133, 209, 315] 1024	0.04	0.28	0.72	[0.86, 0.88, 0.43]	1.70E+06	68	0.8	[0.88, 0.88, 0.62]	1	315	0.03
1024 [209, 74, 40, 29, 40, 74, 209] 1024	0.04	0.29	0.71	[0.89, 0.79, 0.43]	9.34E+05	29	0.89	[0.89, 0.9, 0.88]	2	74	0.04
1024 [0, 363, 83, 47, 83, 363, 0] 1024	0.04	0.31	0.69	[0.86, 0.83, 0.38]	1.62E+06	47	0.84	[0.86, 0.89, 0.77]	2	83	0.03
1024 [949, 187, 111, 18, 111, 187, 949] 1024	0.04	0.33	0.67	[0.86, 0.74, 0.42]	4.69E+06	18	0.84	[0.92, 0.9, 0.71]	7	949	0.04
1024 [351, 46, 21, 13, 21, 46, 351] 1024	0.05	0.34	0.66	[0.84, 0.74, 0.4]	1.51E+06	13	0.74	[0.85, 0.88, 0.5]	1	351	0.04
1024 [0, 466, 187, 112, 187, 466, 0] 1024	0.04	0.34	0.66	[0.81, 0.76, 0.4]	2.34E+06	112	0.84	[0.9, 0.92, 0.71]	4	187	0.03
1024 [211, 95, 66, 35, 66, 95, 211] 1024	0.04	0.36	0.64	[0.84, 0.75, 0.33]	9.79E+05	35	0.81	[0.87, 0.88, 0.67]	7	211	0.03
1024 [465, 134, 46, 27, 46, 134, 465] 1024	0.04	0.37	0.63	[0.82, 0.74, 0.34]	2.18E+06	27	0.79	[0.87, 0.89, 0.62]	7	465	0.03
1024 [158, 87, 48, 37, 48, 87, 158] 1024	0.04	0.4	0.6	[0.77, 0.68, 0.36]	7.26E+05	37	0.76	[0.85, 0.88, 0.56]	2	87	0.04
1024 [242, 87, 35, 16, 35, 87, 242] 1024	0.04	0.43	0.57	[0.77, 0.62, 0.32]	1.09E+06	16	0.85	[0.93, 0.92, 0.71]	7	242	0.04
1024 [9, 4, 2, 1, 2, 4, 9] 1024	0.05	0.46	0.54	[0.7, 0.75, 0.15]	3.70E+04	1	0.58	[0.67, 0.41, 0.67]	1	9	0.05
1024 [956, 485, 33, 14, 33, 485, 956] 1024	0.04	0.47	0.53	[0.68, 0.62, 0.28]	5.84E+06	14	0.79	[0.85, 0.92, 0.59]	2	485	0.04
1024 [904, 217, 63, 7, 63, 217, 904] 1024	0.05	0.55	0.45	[0.55, 0.44, 0.34]	4.54E+06	7	0.78	[0.86, 0.89, 0.59]	7	904	0.04
1024 [954, 109, 30, 3, 30, 109, 954] 1024	0.05	0.59	0.41	[0.33, 0.53, 0.37]	4.34E+06	3	0.75	[0.88, 0.89, 0.48]	1	954	0.05
1024 [8, 4, 3, 1, 3, 4, 8] 1024	0.05	0.62	0.38	[0.28, 0.59, 0.29]	3.30E+04	1	0.42	[0.43, 0.52, 0.31]	1	8	0.05
1024 [139, 72, 53, 4, 53, 72, 139] 1024	0.05	0.64	0.36	[0.32, 0.42, 0.33]	6.25E+05	4	0.74	[0.83, 0.86, 0.53]	2	72	0.04
1024 [108, 49, 17, 8, 17, 49, 108] 1024	0.05	0.68	0.32	[0.41, 0.27, 0.28]	4.67E+05	8	0.66	[0.84, 0.73, 0.42]	7	108	0.04
1024 [1, 0, 0, 0, 0, 0, 1] 1024	0.05	0.71	0.29	[0.64, 0.24, 0.0]	4.10E+03	1	0.36	[0.37, 0.38, 0.33]	1	1	0.05
1024 [7, 4, 3, 1, 3, 4, 7] 1024	0.05	0.79	0.21	[0.62, 0.0, 0.0]	2.88E+04	1	0.29	[0.2, 0.51, 0.16]	1	7	0.05

Table 4.22: Investigation on the 4 objectives of RMSE, Classification Error, Code Size and FLOPs on Brain Tumor with 300 Generations. The F1-score of each AE is resulted form the K-NN classifier with k = 3 on the corresponding code layer with #Neurons = Code size. For the primary results during evolution, the code layer is the middle layer, while after full training the code layer is the hidden layer that has the highest F1-score among all hidden layers.

Optimization							Full training via RMSE				
Dataset	Npop	#Epochs	Batch size	#Generations	Data Size (train, validation, test)	Neuron range	K	#Epochs	Batch size		
Brain	20	1	4	300	(1324, 76, 74)	[0,1) , Initial= [0.5,1)	3	200	64		
	Primary results only by RMSE						**K-NN over validation set after Full Training**				
Structures	RMSE	Classification Error	AF1	F1-score per class	FLOPs	Code Size	Averaged F1-score	F1-score per class	Code Layer	Code Size	RMSE
1024 [40, 19, 11, 3, 11, 19, 40] 1024	0.043	0.01	0.99	[0.99, 0.99]	1.68E+05	3	0.95	[0.95, 0.94]	7	40	0.04
1024 [105, 34, 9, 1, 9, 34, 105] 1024	0.044	0.01	0.99	[0.99, 0.99]	4.46E+05	1	0.93	[0.94, 0.93]	1	105	0.05
1024 [187, 57, 14, 4, 14, 57, 187] 1024	0.043	0.03	0.97	[0.97, 0.97]	8.12E+05	4	0.93	[0.94, 0.93]	5	14	0.03
1024 [917, 342, 72, 19, 72, 342, 917] 1024	0.041	0.04	0.96	[0.96, 0.96]	5.11E+06	19	0.93	[0.94, 0.93]	2	342	0.03
1024 [0, 439, 250, 131, 250, 439, 0] 1024	0.039	0.04	0.96	[0.96, 0.96]	2.37E+06	131	0.92	[0.93, 0.91]	3	131	0.02
1024 [899, 298, 48, 12, 48, 298, 899] 1024	0.042	0.05	0.95	[0.95, 0.95]	4.81E+06	12	0.89	[0.9, 0.88]	6	298	0.03
1024 [804, 157, 126, 23, 126, 157, 804] 1024	0.041	0.05	0.95	[0.95, 0.95]	3.89E+06	23	0.89	[0.9, 0.88]	5	126	0.03
1024 [0, 223, 133, 97, 133, 223, 0] 1024	0.038	0.05	0.95	[0.95, 0.94]	1.08E+06	97	0.92	[0.93, 0.91]	1	223	0.02
1024 [728, 255, 193, 38, 193, 255, 728] 1024	0.04	0.05	0.95	[0.95, 0.94]	3.95E+06	38	0.91	[0.92, 0.9]	5	193	0.03
1024 [299, 176, 150, 64, 150, 176, 299] 1024	0.041	0.05	0.95	[0.95, 0.94]	1.58E+06	64	0.92	[0.93, 0.91]	1	299	0.02
1024 [0, 240, 123, 56, 123, 240, 0] 1024	0.039	0.07	0.93	[0.94, 0.93]	1.13E+06	56	0.91	[0.92, 0.9]	3	56	0.02
1024 [206, 81, 48, 17, 48, 81, 206] 1024	0.042	0.07	0.93	[0.94, 0.93]	9.29E+05	17	0.93	[0.94, 0.93]	4	17	0.03
1024 [0, 464, 111, 40, 111, 464, 0] 1024	0.038	0.07	0.93	[0.94, 0.93]	2.12E+06	40	0.93	[0.94, 0.93]	1	464	0.02
1024 [345, 101, 78, 37, 78, 101, 345] 1024	0.04	0.07	0.93	[0.94, 0.93]	1.60E+06	37	0.92	[0.93, 0.91]	4	37	0.03
1024 [965, 467, 59, 10, 59, 467, 965] 1024	0.04	0.08	0.92	[0.93, 0.92]	5.87E+06	10	0.93	[0.94, 0.93]	4	10	0.03
1024 [0, 614, 76, 27, 76, 614, 0] 1024	0.039	0.08	0.92	[0.93, 0.91]	2.71E+06	27	0.92	[0.93, 0.92]	4	76	0.03
1024 [0, 304, 166, 76, 166, 304, 0] 1024	0.038	0.08	0.92	[0.93, 0.91]	1.50E+06	76	0.91	[0.92, 0.9]	4	166	0.02
1024 [1, 0, 0, 0, 0, 0, 1] 1024	0.046	0.11	0.89	[0.9, 0.89]	4.10E+03	1	0.43	[0.46, 0.41]	1	1	0.05
1024 [465, 119, 99, 8, 99, 119, 465] 1024	0.039	0.12	0.88	[0.89, 0.87]	2.18E+06	8	0.91	[0.92, 0.9]	1	465	0.03

Table 4.23: Investigation on the 4 objectives of RMSE, Classification Error, Code Size and FLOPs on Pulmonary Tumor with 300 Generations. The F1-score of each AE is resulted form the K-NN classifier with k = 3 on the corresponding code layer with #Neurons = Code size. For the primary results during evolution, the code layer is the middle layer, while after full training the code layer is the hidden layer that has the highest F1-score among all hidden layers.

Optimization							Full training via RMSE				
Dataset	Npop	#Epochs	Batch size	#Generations	Data Size (train, validation, test)	Neuron range	K	#Epochs	Batch size		
Pulmonary	20	1	4	300	(719, 86, 84)	[0,1) , Initial=[0.5,1)	3	200	64		
	Primary results only by RMSE						**K-NN over validation set after Full Training**				
Structures	RMSE	Classification Error	Averaged F1-score	F1-score per class	FLOPs	Code Size	Averaged F1-score	F1-score per class	Code Layer	Code Size	RMSE
1024 [531, 445, 0, 323, 0, 445, 531] 1024	0.043	0.22	0.78	[0.89, 0.89, 0.56]	3.70E+06	323	0.87	[0.89, 0.88, 0.83]	5	531	0.02
1024 [631, 348, 0, 259, 0, 348, 631] 1024	0.044	0.26	0.74	[0.84, 0.85, 0.53]	3.82E+06	259	0.89	[0.87, 0.89, 0.91]	4	348	0.02
1024 [996, 627, 134, 96, 134, 627, 996] 1024	0.044	0.29	0.71	[0.85, 0.82, 0.45]	6.97E+06	96	0.88	[0.85, 0.88, 0.91]	7	996	0.03
1024 [240, 166, 0, 118, 0, 166, 240] 1024	0.042	0.31	0.69	[0.85, 0.85, 0.38]	1.22E+06	118	0.79	[0.87, 0.89, 0.62]	1	240	0.03
1024 [736, 291, 0, 168, 0, 291, 736] 1024	0.04	0.32	0.68	[0.85, 0.82, 0.38]	4.07E+06	168	0.88	[0.84, 0.89, 0.91]	5	736	0.02
1024 [354, 255, 0, 187, 0, 255, 354] 1024	0.041	0.34	0.66	[0.85, 0.75, 0.38]	2.00E+06	187	0.81	[0.85, 0.87, 0.71]	5	354	0.02
1024 [592, 184, 0, 102, 0, 184, 592] 1024	0.041	0.35	0.65	[0.82, 0.78, 0.36]	2.94E+06	102	0.83	[0.85, 0.88, 0.77]	5	592	0.03
1024 [327, 160, 0, 154, 0, 160, 327] 1024	0.042	0.37	0.63	[0.8, 0.74, 0.36]	1.65E+06	154	0.82	[0.85, 0.89, 0.71]	5	327	0.02
1024 [441, 91, 45, 20, 45, 91, 441] 1024	0.049	0.37	0.63	[0.82, 0.75, 0.32]	1.99E+06	20	0.75	[0.87, 0.89, 0.5]	1	441	0.04
1024 [320, 254, 87, 67, 87, 254, 320] 1024	0.043	0.38	0.62	[0.85, 0.7, 0.32]	1.75E+06	67	0.8	[0.87, 0.89, 0.62]	5	87	0.03
1024 [2, 0, 1, 0, 1, 0, 2] 1024	0.053	0.42	0.58	[0.53, 0.6, 0.62]	8.20E+03	1	0.33	[0.42, 0.34, 0.22]	1	2	0.05
1024 [905, 331, 57, 39, 57, 331, 905] 1024	0.042	0.43	0.57	[0.74, 0.67, 0.31]	4.99E+06	39	0.84	[0.9, 0.89, 0.71]	3	57	0.04
1024 [1, 1, 0, 0, 0, 1, 1] 1024	0.054	0.48	0.52	[0.39, 0.58, 0.59]	4.10E+03	1	0.22	[0.05, 0.62, 0.0]	1	1	0.05
1024 [182, 12, 10, 1, 10, 12, 182] 1024	0.048	0.53	0.47	[0.5, 0.55, 0.37]	7.55E+05	1	0.79	[0.84, 0.86, 0.67]	1	182	0.05
1024 [14, 6, 3, 1, 3, 6, 14] 1024	0.051	0.54	0.46	[0.52, 0.51, 0.36]	5.78E+04	1	0.67	[0.69, 0.82, 0.5]	1	14	0.05
1024 [1, 0, 0, 0, 0, 0, 1] 1024	0.046	0.55	0.45	[0.48, 0.57, 0.29]	4.10E+03	1	0.41	[0.48, 0.39, 0.34]	1	1	0.05
1024 [478, 364, 5, 3, 5, 364, 478] 1024	0.046	0.61	0.39	[0.46, 0.41, 0.29]	2.66E+06	3	0.67	[0.64, 0.73, 0.62]	6	364	0.05
1024 [419, 314, 17, 13, 17, 314, 419] 1024	0.045	0.66	0.34	[0.17, 0.52, 0.32]	2.26E+06	13	0.72	[0.84, 0.85, 0.48]	7	419	0.04
1024 [439, 264, 4, 2, 4, 264, 439] 1024	0.045	0.71	0.29	[0.12, 0.49, 0.26]	2.27E+06	2	0.77	[0.78, 0.82, 0.71]	3	4	0.05
1024 [34, 7, 4, 1, 4, 7, 34] 1024	0.046	0.79	0.21	[0.62, 0.0, 0.0]	1.40E+05	1	0.51	[0.59, 0.57, 0.37]	4	1	0.05

Table 4.24: Investigation on the 3 objectives of Classification Error, Code Size and FLOPs on Brain Tumor. The F1-score of each AE is resulted form the K-NN classifier with k = 3 on the corresponding code layer with #Neurons = Code size. For the primary results during evolution, the code layer is the middle layer, while after full training the code layer is the hidden layer that has the highest F1-score among all hidden layers.

Optimization							Full training via RMSE				
Dataset	Npop	#Epochs	Batch size	#Generation	Data Size (train, validation, test)	Neuron range	K	#Epochs	Batch size		
Brain	20	1	4	100	(1324, 76, 74)	[0,1) , Initial= [0.5,1)	3	200	64		
	Primary results only by RMSE						**K-NN over validation set after Full Training**				
Structures	RMSE	Classification Error	Averaged F1-score	F1-score per class	FLOPs	Code Size	Averaged F1-score	F1-score per class	Code Layer	Code Size	RMSE
1024 [49, 18, 9, 5, 9, 18, 49] 1024	0.04	0.01	0.99	[0.99, 0.99]	2.05E+05	5	0.92	[0.93, 0.92]	5	9	0.03
1024 [80, 10, 5, 3, 5, 10, 80] 1024	0.04	0.03	0.97	[0.97, 0.97]	3.31E+05	3	0.92	[0.92, 0.92]	7	80	0.04
1024 [25, 17, 12, 6, 12, 17, 25] 1024	0.04	0.03	0.97	[0.97, 0.97]	1.05E+05	6	0.93	[0.94, 0.93]	4	6	0.04
1024 [104, 11, 5, 1, 5, 11, 104] 1024	0.04	0.03	0.97	[0.97, 0.97]	4.31E+05	1	0.92	[0.93, 0.91]	1	104	0.05
1024 [92, 14, 6, 2, 6, 14, 92] 1024	0.04	0.03	0.97	[0.97, 0.97]	3.82E+05	2	0.91	[0.91, 0.91]	3	6	0.04
1024 [41, 7, 3, 0, 3, 7, 41] 1024	0.05	0.03	0.97	[0.97, 0.97]	1.69E+05	3	0.77	[0.8, 0.75]	1	41	0.05
1024 [34, 29, 5, 1, 5, 29, 34] 1024	0.04	0.04	0.96	[0.96, 0.96]	1.44E+05	1	0.96	[0.96, 0.96]	2	29	0.04
1024 [15, 11, 8, 4, 8, 11, 15] 1024	0.05	0.04	0.96	[0.96, 0.96]	6.26E+04	4	0.95	[0.95, 0.94]	6	11	0.04
1024 [17, 15, 13, 3, 13, 15, 17] 1024	0.04	0.04	0.96	[0.96, 0.96]	7.16E+04	3	0.85	[0.84, 0.87]	2	15	0.04
1024 [23, 7, 3, 2, 3, 7, 23] 1024	0.05	0.05	0.95	[0.95, 0.95]	9.50E+04	2	0.93	[0.94, 0.93]	4	2	0.04
1024 [9, 7, 4, 1, 4, 7, 9] 1024	0.05	0.07	0.93	[0.94, 0.93]	3.72E+04	1	0.92	[0.93, 0.91]	1	9	0.04
1024 [1, 0, 0, 0, 0, 0, 1] 1024	0.05	0.09	0.91	[0.91, 0.9]	4.10E+03	1	0.47	[0.6, 0.34]	1	1	0.05

Table 4.25: Investigation on the 3 objectives of Classification Error, Code Size and FLOPs on Pulmonary Tumor. The F1-score of each AE is resulted form the K-NN classifier with k = 3 on the corresponding code layer with #Neurons = Code size. For the primary results during evolution, the code layer is the middle layer, while after full training the code layer is the hidden layer that has the highest F1-score among all hidden layers.

Optimization							Full training via RMSE				
Dataset	Npop	#Epochs	Batch size	#Generations	Data Size (train, validation, test)	Neuron range	K	#Epochs	Batch size		
Pulmonary	20	1	4	100	(719, 86, 84)	[0,1) , Initial=[0.5,1)	3	200	64		
	Primary results only by RMSE						**K-NN over validation set after Full Training**				
Structures	RMSE	Classification Error	Averaged F1-score	F1-score per class	FLOPs	Code Size	Averaged F1-score	F1-score per class	Code Layer	Code Size	RMSE
1024 [477, 167, 92, 40, 92, 167, 477] 1024	0.05	0.25	0.75	[0.9, 0.82, 0.53]	2.35E+06	40	0.9	[0.92, 0.89, 0.89]	4	40	0.03
1024 [261, 136, 49, 28, 49, 136, 261] 1024	0.04	0.27	0.73	[0.93, 0.83, 0.43]	1.24E+06	28	0.82	[0.9, 0.9, 0.67]	1	261	0.04
1024 [125, 66, 34, 23, 34, 66, 125] 1024	0.04	0.27	0.73	[0.91, 0.82, 0.45]	5.57E+05	23	0.81	[0.9, 0.89, 0.62]	6	66	0.04
1024 [96, 69, 37, 22, 37, 69, 96] 1024	0.04	0.29	0.71	[0.88, 0.83, 0.43]	4.33E+05	22	0.78	[0.82, 0.88, 0.62]	1	96	0.04
1024 [164, 103, 34, 12, 34, 103, 164] 1024	0.04	0.29	0.71	[0.88, 0.79, 0.45]	7.55E+05	12	0.72	[0.78, 0.85, 0.53]	7	164	0.04
1024 [204, 63, 34, 6, 34, 63, 204] 1024	0.05	0.31	0.69	[0.87, 0.82, 0.38]	8.96E+05	6	0.71	[0.78, 0.84, 0.5]	1	204	0.04
1024 [158, 50, 8, 6, 8, 50, 158] 1024	0.04	0.32	0.68	[0.85, 0.81, 0.4]	6.81E+05	6	0.73	[0.81, 0.86, 0.53]	7	158	0.04
1024 [115, 75, 29, 9, 29, 75, 115] 1024	0.05	0.32	0.68	[0.87, 0.79, 0.38]	5.15E+05	9	0.65	[0.79, 0.8, 0.37]	7	115	0.04
1024 [118, 59, 23, 7, 23, 59, 118] 1024	0.04	0.33	0.67	[0.86, 0.77, 0.38]	5.17E+05	7	0.7	[0.85, 0.82, 0.42]	2	59	0.04
1024 [283, 82, 9, 3, 9, 82, 283] 1024	0.04	0.35	0.65	[0.79, 0.81, 0.36]	1.26E+06	3	0.71	[0.85, 0.84, 0.43]	1	283	0.05
1024 [89, 58, 18, 10, 18, 58, 89] 1024	0.05	0.35	0.65	[0.84, 0.78, 0.33]	3.90E+05	10	0.79	[0.86, 0.88, 0.62]	4	10	0.04
1024 [80, 47, 23, 16, 23, 47, 80] 1024	0.05	0.35	0.65	[0.87, 0.76, 0.32]	3.49E+05	16	0.67	[0.82, 0.8, 0.4]	7	80	0.04
1024 [75, 47, 22, 13, 22, 47, 75] 1024	0.05	0.35	0.65	[0.82, 0.75, 0.37]	3.27E+05	13	0.73	[0.84, 0.86, 0.5]	1	75	0.04
1024 [233, 59, 11, 2, 11, 59, 233] 1024	0.05	0.36	0.64	[0.81, 0.76, 0.36]	1.01E+06	2	0.68	[0.68, 0.76, 0.59]	4	2	0.05
1024 [72, 43, 17, 10, 17, 43, 72] 1024	0.04	0.36	0.64	[0.81, 0.78, 0.33]	3.11E+05	10	0.78	[0.83, 0.84, 0.67]	7	72	0.04
1024 [40, 30, 9, 5, 9, 30, 40] 1024	0.05	0.36	0.64	[0.79, 0.73, 0.38]	1.70E+05	5	0.61	[0.79, 0.72, 0.3]	1	40	0.05
1024 [46, 18, 9, 2, 9, 18, 46] 1024	0.05	0.38	0.62	[0.71, 0.73, 0.42]	1.92E+05	2	0.64	[0.81, 0.78, 0.33]	1	46	0.05
1024 [24, 18, 7, 4, 7, 18, 24] 1024	0.05	0.41	0.59	[0.77, 0.62, 0.38]	1.01E+05	4	0.64	[0.81, 0.74, 0.36]	5	7	0.04
1024 [1, 1, 0, 0, 0, 1, 1] 1024	0.05	0.47	0.53	[0.53, 0.55, 0.5]	4.10E+03	1	0.33	[0.54, 0.46, 0.0]	1	1	0.05
1024 [1, 0, 0, 0, 0, 0, 1] 1024	0.05	0.57	0.43	[0.54, 0.39, 0.36]	4.10E+03	1	0.22	[0.0, 0.66, 0.0]	1	1	0.05

Table 4.26: Investigation on the 3 objectives of Classification Error, Code Size and FLOPs with the SAE Method on Brain Tumor

Optimization							SAE Full Training on Middle Layer								K-NN Over Test Set Searched on Test					
Dataset	Npop	#Epochs	Batch size	#Generations	Data Size (train, validation, test)	Neuron range	K	Unsupervise #Epochs		SAE #Epoch			Batch size		K = 3					
Brain	100	1	4	100	(1324, 76, 74)	[0,1) , Initial= [0.5,1)	3	200		100			4							
	Primary results only by RMSE						On Validation Set			On Test Set			Middle Layer		On Middle Layer		Best Among All Layers			
Structures	RMSE	Classification Error	AF1	F1-score per class	FLOPs	Code Size	AF1	F1-score per class	RMSE	AF1	F1-score per class	RMSE	Code Layer	Code Size	AF1	F1-score per class	AF1	F1-score per class	Code Layer	Code Size
1024 [45, 24, 9, 3, 9, 24, 45] 1024	0.05	0.01	0.99	[0.99, 0.99]	1.90E+05	3	0.93	[0.94, 0.93]	0.04	0.88	[0.88, 0.87]	0.04	4	3	0.78	[0.78, 0.78]	0.85	[0.85, 0.85]	7	45
1024 [146, 46, 6, 1, 6, 46, 146] 1024	0.04	0.01	0.99	[0.99, 0.99]	6.26E+05	1	0.95	[0.95, 0.94]	0.04	0.88	[0.88, 0.88]	0.04	4	1	0.73	[0.74, 0.71]	0.86	[0.87, 0.86]	1	146
1024 [13, 10, 8, 5, 8, 10, 13] 1024	0.05	0.03	0.97	[0.97, 0.97]	5.42E+04	5	0.95	[0.95, 0.95]	0.04	0.89	[0.9, 0.89]	0.04	4	5	0.76	[0.77, 0.74]	0.82	[0.83, 0.82]	7	13
1024 [16, 15, 12, 4, 12, 15, 16] 1024	0.04	0.03	0.97	[0.97, 0.97]	6.74E+04	4	0.92	[0.93, 0.92]	0.04	0.86	[0.86, 0.86]	0.04	4	4	0.85	[0.87, 0.83]	0.86	[0.88, 0.85]	1	16
1024 [22, 11, 6, 3, 6, 11, 22] 1024	0.04	0.03	0.97	[0.97, 0.97]	9.14E+04	3	0.93	[0.94, 0.93]	0.04	0.9	[0.91, 0.9]	0.04	4	3	0.82	[0.84, 0.81]	0.86	[0.87, 0.86]	6	11
1024 [57, 31, 7, 1, 7, 31, 57] 1024	0.04	0.03	0.97	[0.97, 0.97]	2.41E+05	1	0.32	[0.0, 0.64]	0.05	0.32	[0.0, 0.64]	0.05	4	1	0.32	[0.0, 0.64]	0.78	[0.77, 0.79]	1	57
1024 [42, 25, 6, 2, 6, 25, 42] 1024	0.04	0.03	0.97	[0.97, 0.97]	1.77E+05	2	0.97	[0.97, 0.97]	0.04	0.89	[0.89, 0.89]	0.04	4	2	0.77	[0.78, 0.75]	0.88	[0.88, 0.87]	1	42
1024 [11, 7, 0, 4, 0, 7, 11] 1024	0.04	0.03	0.97	[0.97, 0.97]	4.55E+04	4	0.95	[0.95, 0.94]	0.04	0.92	[0.92, 0.92]	0.04	3	4	0.86	[0.87, 0.86]	0.9	[0.91, 0.9]	1	11
1024 [23, 10, 5, 2, 5, 10, 23] 1024	0.04	0.03	0.97	[0.97, 0.97]	9.54E+04	2	0.32	[0.0, 0.64]	0.05	0.32	[0.0, 0.64]	0.05	4	2	0.32	[0.0, 0.64]	0.72	[0.77, 0.68]	2	10
1024 [9, 4, 0, 0, 0, 4, 9] 1024	0.05	0.04	0.96	[0.96, 0.96]	3.70E+04	4	0.32	[0.0, 0.64]	0.04	0.31	[0.0, 0.62]	0.04	2	4	0.88	[0.89, 0.87]	0.89	[0.89, 0.89]	1	9
1024 [7, 5, 4, 0, 4, 5, 7] 1024	0.04	0.04	0.96	[0.96, 0.96]	2.89E+04	4	0.95	[0.95, 0.94]	0.05	0.92	[0.92, 0.92]	0.05	3	4	0.67	[0.72, 0.61]	0.78	[0.8, 0.76]	2	5
1024 [50, 22, 4, 1, 4, 22, 50] 1024	0.04	0.04	0.96	[0.96, 0.96]	2.10E+05	1	0.32	[0.0, 0.64]	0.05	0.32	[0.0, 0.64]	0.05	4	1	0.32	[0.0, 0.64]	0.81	[0.81, 0.81]	1	50
1024 [6, 5, 5, 0, 5, 5, 6] 1024	0.04	0.04	0.96	[0.96, 0.96]	2.48E+04	5	0.95	[0.95, 0.94]	0.05	0.88	[0.88, 0.88]	0.05	3	5	0.54	[0.56, 0.51]	0.59	[0.61, 0.58]	1	6
1024 [10, 9, 4, 2, 4, 9, 10] 1024	0.04	0.04	0.96	[0.96, 0.96]	4.15E+04	2	0.93	[0.93, 0.94]	0.04	0.86	[0.85, 0.88]	0.04	4	2	0.89	[0.9, 0.89]	0.89	[0.9, 0.89]	4	2
1024 [43, 26, 6, 1, 6, 26, 43] 1024	0.04	0.05	0.95	[0.95, 0.95]	1.81E+05	1	0.96	[0.96, 0.96]	0.04	0.86	[0.86, 0.87]	0.04	4	1	0.89	[0.89, 0.89]	0.89	[0.89, 0.89]	3	6
1024 [7, 2, 0, 0, 0, 2, 7] 1024	0.04	0.05	0.95	[0.95, 0.94]	2.87E+04	2	0.32	[0.0, 0.64]	0.05	0.32	[0.0, 0.64]	0.05	2	2	0.37	[0.45, 0.28]	0.6	[0.66, 0.54]	1	7
1024 [23, 18, 3, 1, 3, 18, 23] 1024	0.04	0.05	0.95	[0.95, 0.94]	9.61E+04	1	0.32	[0.0, 0.64]	0.05	0.32	[0.0, 0.64]	0.05	4	1	0.32	[0.0, 0.64]	0.78	[0.81, 0.74]	2	18
1024 [9, 6, 3, 1, 3, 6, 9] 1024	0.05	0.07	0.93	[0.94, 0.93]	3.72E+04	1	0.92	[0.93, 0.92]	0.04	0.91	[0.91, 0.9]	0.04	4	1	0.91	[0.91, 0.9]	0.91	[0.91, 0.9]	4	1
1024 [6, 4, 3, 1, 3, 4, 6] 1024	0.04	0.08	0.92	[0.92, 0.92]	2.47E+04	1	0.32	[0.0, 0.64]	0.05	0.32	[0.0, 0.64]	0.05	4	1	0.32	[0.0, 0.64]	0.65	[0.63, 0.67]	2	4
1024 [5, 3, 0, 3, 0, 3, 5] 1024	0.05	0.08	0.92	[0.93, 0.91]	2.06E+04	3	0.95	[0.95, 0.94]	0.04	0.95	[0.95, 0.94]	0.04	3	3	0.93	[0.94, 0.93]	0.93	[0.94, 0.93]	2	3
1024 [5, 3, 1, 0, 1, 3, 5] 1024	0.05	0.09	0.91	[0.91, 0.91]	2.06E+04	1	0.96	[0.96, 0.96]	0.04	0.89	[0.89, 0.89]	0.04	3	1	0.93	[0.93, 0.93]	0.93	[0.93, 0.93]	3	1
1024 [4, 2, 0, 0, 0, 2, 4] 1024	0.05	0.09	0.91	[0.92, 0.9]	1.64E+04	2	0.93	[0.93, 0.94]	0.04	0.93	[0.94, 0.93]	0.04	2	2	0.93	[0.94, 0.93]	0.93	[0.94, 0.93]	2	2
1024 [4, 1, 1, 0, 1, 1, 4] 1024	0.05	0.11	0.89	[0.9, 0.89]	1.64E+04	1	0.95	[0.95, 0.94]	0.05	0.85	[0.85, 0.85]	0.05	2	1	0.48	[0.33, 0.62]	0.63	[0.65, 0.62]	1	4
1024 [1, 0, 0, 0, 0, 0, 1] 1024	0.05	0.12	0.88	[0.88, 0.88]	4.10E+03	1	0.32	[0.0, 0.64]	0.05	0.32	[0.0, 0.64]	0.05	1	1	0.32	[0.05, 0.6]	0.32	[0.0, 0.64]	2	1

Table 4.27: Investigation on the 3 objectives of Classification Error, Code Size and FLOPs with the SAE Method on Endocrine Tumor

Optimization							SAE Full Training on Middle Layer								K-NN Over Test Set Searched on Test					
Dataset	Npop	#Epochs	Batch size	#Generations	Data Size (train, validation, test)	Neuron range	K	Unsupervise #Epochs		SAE #Epoch			Batch size		K = 3					
Endocrine	100	1	4	100	(746, 71, 72)	[0,1) , Initial= [0.5,1)	3	200		100			4							
	Primary results only by RMSE						On Validation Set			On Test Set			Middle Layer		On Middle Layer		Best Among All Layers			
Structures	RMSE	Classification Error	AF1	F1-score per class	FLOPs	Code Size	AF1	F1-score per class	RMSE	AF1	F1-score per class	RMSE	Code Layer	Code Size	AF1	F1-score per class	AF1	F1-score per class	Code Layer	Code Size
1024 [8, 7, 5, 2, 5, 7, 8] 1024	0.04	0	1	[1.0, 1.0, 1.0]	3.32E+04	2	0.89	[0.71, 1.0, 0.96]	0.04	0.78	[0.46, 0.91, 0.96]	0.04	4	2	0.82	[0.67, 0.81, 0.98]	0.84	[0.67, 0.87, 1.0]	1	8
1024 [7, 6, 0, 3, 0, 6, 7] 1024	0.04	0	1	[1.0, 1.0, 1.0]	2.89E+04	3	0.97	[0.91, 1.0, 0.99]	0.04	0.92	[0.83, 0.93, 1.0]	0.04	3	3	0.84	[0.67, 0.87, 1.0]	0.88	[0.73, 0.9, 1.0]	4	6
1024 [13, 4, 2, 1, 2, 4, 13] 1024	0.05	0.03	0.97	[0.91, 1.0, 0.99]	5.35E+04	1	0.96	[0.91, 0.97, 1.0]	0.05	0.89	[0.77, 0.9, 1.0]	0.05	4	1	0.5	[0.0, 0.64, 0.86]	0.86	[0.73, 0.87, 0.99]	1	13
1024 [7, 5, 0, 3, 0, 5, 7] 1024	0.05	0.05	0.95	[0.89, 0.97, 1.0]	2.89E+04	3	0.28	[0.0, 0.0, 0.84]	0.05	0.28	[0.0, 0.0, 0.83]	0.05	3	3	0	[0.0, 0.0, 0.0]	0.44	[0.0, 0.48, 0.85]	1	7
1024 [4, 3, 0, 2, 0, 3, 4] 1024	0.05	0.06	0.94	[0.83, 1.0, 0.98]	1.65E+04	2	0.28	[0.0, 0.0, 0.84]	0.05	0.28	[0.0, 0.0, 0.83]	0.05	3	2	0	[0.0, 0.0, 0.0]	0.46	[0.46, 0.09, 0.83]	1	4
1024 [3, 2, 0, 2, 0, 2, 3] 1024	0.05	0.06	0.94	[0.91, 0.93, 0.97]	1.23E+04	2	0.28	[0.0, 0.0, 0.84]	0.05	0.28	[0.0, 0.0, 0.83]	0.05	3	2	0	[0.0, 0.0, 0.0]	0.33	[0.29, 0.0, 0.72]	1	3
1024 [3, 2, 2, 1, 2, 2, 3] 1024	0.05	0.1	0.9	[0.75, 0.94, 1.0]	1.23E+04	1	0.28	[0.0, 0.0, 0.84]	0.05	0.28	[0.0, 0.0, 0.83]	0.05	4	1	0	[0.0, 0.0, 0.0]	0	[0.0, 0.0, 0.0]	4	1
1024 [3, 2, 1, 1, 1, 2, 3] 1024	0.05	0.16	0.84	[0.67, 0.86, 1.0]	1.23E+04	1	0.28	[0.0, 0.0, 0.84]	0.05	0.28	[0.0, 0.0, 0.83]	0.05	4	1	0	[0.0, 0.0, 0.0]	0.22	[0.0, 0.0, 0.67]	2	2
1024 [2, 1, 1, 0, 1, 1, 2] 1024	0.05	0.18	0.82	[0.62, 0.87, 0.97]	8.21E+03	1	0.67	[0.5, 0.5, 1.0]	0.05	0.58	[0.43, 0.32, 1.0]	0.05	2	1	0.29	[0.0, 0.1, 0.79]	0.29	[0.0, 0.1, 0.79]	2	1
1024 [2, 1, 0, 0, 0, 1, 2] 1024	0.05	0.25	0.75	[0.42, 0.93, 0.88]	8.20E+03	1	0.28	[0.0, 0.0, 0.84]	0.05	0.28	[0.0, 0.0, 0.83]	0.05	2	1	0	[0.0, 0.0, 0.0]	0.49	[0.0, 0.55, 0.93]	1	2
1024 [1, 1, 0, 1, 0, 1, 1] 1024	0.05	0.26	0.74	[0.33, 0.88, 1.0]	4.10E+03	1	0.28	[0.0, 0.0, 0.84]	0.05	0.28	[0.0, 0.0, 0.83]	0.05	3	1	0	[0.0, 0.0, 0.0]	0	[0.0, 0.0, 0.0]	1	1
1024 [1, 0, 0, 0, 0, 0, 1] 1024	0.04	0.38	0.62	[0.0, 0.86, 1.0]	4.10E+03	1	0.28	[0.0, 0.0, 0.84]	0.05	0.28	[0.0, 0.0, 0.83]	0.05	2	1	0	[0.0, 0.0, 0.0]	0.16	[0.0, 0.0, 0.48]	1	1

Table 4.28: Investigation on the 3 objectives of Classification Error, Code Size and FLOPs with the SAE Method on Gastro Tumor

Optimization							SAE Full Training on Middle Layer								K-NN Over Test Set Searched on Test					
Dataset	Npop	#Epochs	Batch size	#Generations	Data Size (train, validation, test)	Neuron range	K	Unsupervise #Epochs		SAE #Epoch			Batch size		K = 3					
Gastro	100	1	4	100	(620, 81, 88)	[0,1) , Initial= [0.5,1)	3	200		100			4							
	Primary results only by RMSE						On Validation Set			On Test Set			Middle Layer		On Middle Layer		Best Among All Layers			
Structures	RMSE	Classification Error	AF1	F1-score per class	FLOPs	Code Size	AF1	F1-score per class	RMSE	AF1	F1-score per class	RMSE	Code Layer	Code Size	AF1	F1-score per class	AF1	F1-score per class	Code Layer	Code Size
1024 [202, 89, 30, 14, 30, 89, 202] 1024	0.04	0.24	0.76	[0.81, 0.47, 0.93, 0.82]	9.12E+05	14	0.65	[0.85, 0.0, 0.9, 0.86]	0.03	0.64	[0.71, 0.35, 0.72, 0.79]	0.03	4	14	0.53	[0.6, 0.26, 0.58, 0.69]	0.63	[0.76, 0.36, 0.59, 0.79]	1	202
1024 [200, 102, 21, 6, 21, 102, 200] 1024	0.04	0.25	0.75	[0.83, 0.56, 0.83, 0.77]	9.10E+05	6	0.76	[0.86, 0.53, 0.8, 0.85]	0.03	0.61	[0.72, 0.12, 0.76, 0.82]	0.03	4	6	0.57	[0.66, 0.35, 0.61, 0.68]	0.69	[0.72, 0.64, 0.64, 0.74]	7	200
1024 [113, 87, 57, 36, 57, 87, 113] 1024	0.04	0.26	0.74	[0.77, 0.43, 0.93, 0.84]	5.30E+05	36	0.63	[0.84, 0.0, 0.85, 0.85]	0.03	0.63	[0.77, 0.14, 0.79, 0.81]	0.03	4	36	0.6	[0.65, 0.38, 0.64, 0.75]	0.68	[0.75, 0.48, 0.7, 0.8]	7	113
1024 [109, 37, 12, 4, 12, 37, 109] 1024	0.04	0.26	0.74	[0.81, 0.48, 0.88, 0.8]	4.65E+05	4	0.68	[0.85, 0.18, 0.86, 0.84]	0.03	0.7	[0.78, 0.35, 0.83, 0.82]	0.03	4	4	0.67	[0.74, 0.36, 0.85, 0.73]	0.69	[0.76, 0.36, 0.9, 0.74]	3	12
1024 [80, 60, 36, 20, 36, 60, 80] 1024	0.04	0.26	0.74	[0.82, 0.53, 0.81, 0.8]	3.58E+05	20	0.7	[0.83, 0.31, 0.88, 0.81]	0.03	0.6	[0.75, 0.13, 0.75, 0.78]	0.03	4	20	0.69	[0.72, 0.4, 0.89, 0.74]	0.69	[0.71, 0.4, 0.93, 0.73]	6	60
1024 [69, 44, 33, 28, 33, 44, 69] 1024	0.04	0.26	0.74	[0.86, 0.44, 0.85, 0.79]	3.04E+05	28	0.73	[0.89, 0.29, 0.89, 0.86]	0.03	0.66	[0.75, 0.33, 0.77, 0.79]	0.03	4	28	0.61	[0.59, 0.33, 0.76, 0.77]	0.68	[0.68, 0.4, 0.81, 0.82]	2	44
1024 [91, 69, 35, 17, 35, 69, 91] 1024	0.04	0.27	0.73	[0.8, 0.5, 0.83, 0.79]	4.10E+05	17	0.69	[0.83, 0.15, 0.9, 0.88]	0.03	0.67	[0.78, 0.22, 0.84, 0.84]	0.03	4	17	0.65	[0.66, 0.42, 0.7, 0.83]	0.68	[0.72, 0.48, 0.74, 0.79]	6	69
1024 [32, 24, 18, 5, 18, 24, 32] 1024	0.04	0.27	0.73	[0.87, 0.5, 0.79, 0.75]	1.36E+05	5	0.76	[0.91, 0.46, 0.81, 0.86]	0.03	0.69	[0.82, 0.27, 0.83, 0.85]	0.03	4	5	0.59	[0.68, 0.32, 0.69, 0.65]	0.73	[0.81, 0.64, 0.71, 0.77]	6	24
1024 [11, 6, 5, 1, 5, 6, 11] 1024	0.05	0.29	0.71	[0.83, 0.5, 0.74, 0.79]	4.55E+04	1	0.62	[0.82, 0.0, 0.83, 0.83]	0.04	0.53	[0.71, 0.0, 0.64, 0.76]	0.04	4	1	0.56	[0.64, 0.0, 0.86, 0.75]	0.63	[0.69, 0.18, 0.83, 0.83]	1	11
1024 [6, 3, 0, 2, 0, 3, 6] 1024	0.05	0.3	0.7	[0.86, 0.5, 0.71, 0.72]	2.47E+04	2	0.63	[0.82, 0.0, 0.81, 0.88]	0.04	0.57	[0.76, 0.0, 0.72, 0.81]	0.04	3	2	0.59	[0.68, 0.33, 0.69, 0.64]	0.7	[0.74, 0.35, 0.86, 0.85]	1	6
1024 [3, 2, 2, 2, 2, 2, 3] 1024	0.05	0.31	0.69	[0.92, 0.53, 0.69, 0.63]	1.23E+04	2	0.14	[0.55, 0.0, 0.0, 0.0]	0.05	0.13	[0.53, 0.0, 0.0, 0.0]	0.05	4	2	0.16	[0.29, 0.0, 0.0, 0.33]	0.29	[0.5, 0.06, 0.19, 0.42]	1	3
1024 [4, 3, 2, 1, 2, 3, 4] 1024	0.05	0.31	0.69	[0.79, 0.5, 0.74, 0.72]	1.65E+04	1	0.14	[0.55, 0.0, 0.0, 0.0]	0.05	0.13	[0.53, 0.0, 0.0, 0.0]	0.05	4	1	0.13	[0.12, 0.22, 0.11, 0.07]	0.22	[0.46, 0.15, 0.11, 0.16]	2	3
1024 [4, 1, 0, 0, 0, 1, 4] 1024	0.05	0.34	0.66	[0.9, 0.4, 0.62, 0.71]	1.64E+04	1	0.14	[0.55, 0.0, 0.0, 0.0]	0.05	0.13	[0.53, 0.0, 0.0, 0.0]	0.05	2	1	0.17	[0.54, 0.0, 0.0, 0.12]	0.29	[0.46, 0.21, 0.4, 0.11]	1	4
1024 [2, 1, 1, 1, 1, 1, 2] 1024	0.05	0.36	0.64	[0.87, 0.31, 0.67, 0.71]	8.21E+03	1	0.14	[0.55, 0.0, 0.0, 0.0]	0.05	0.13	[0.53, 0.0, 0.0, 0.0]	0.05	4	1	0.13	[0.53, 0.0, 0.0, 0.0]	0.3	[0.56, 0.0, 0.0, 0.66]	1	2
1024 [2, 1, 0, 1, 0, 1, 2] 1024	0.05	0.39	0.61	[0.8, 0.29, 0.69, 0.65]	8.20E+03	1	0.14	[0.55, 0.0, 0.0, 0.0]	0.05	0.13	[0.53, 0.0, 0.0, 0.0]	0.05	3	1	0.13	[0.53, 0.0, 0.0, 0.0]	0.13	[0.53, 0.0, 0.0, 0.0]	2	1
1024 [1, 0, 1, 0, 1, 0, 1] 1024	0.05	0.42	0.58	[0.84, 0.18, 0.67, 0.64]	4.10E+03	1	0.14	[0.55, 0.0, 0.0, 0.0]	0.05	0.13	[0.53, 0.0, 0.0, 0.0]	0.05	1	1	0.19	[0.57, 0.0, 0.21, 0.0]	0.19	[0.57, 0.0, 0.21, 0.0]	1	1
1024 [1, 0, 0, 0, 0, 0, 1] 1024	0.05	0.45	0.55	[0.75, 0.29, 0.59, 0.56]	4.10E+03	1	0.14	[0.55, 0.0, 0.0, 0.0]	0.05	0.13	[0.53, 0.0, 0.0, 0.0]	0.05	1	1	0.13	[0.53, 0.0, 0.0, 0.0]	0.13	[0.53, 0.0, 0.0, 0.0]	1	1

Table 4.29: Investigation on the 3 objectives of Classification Error, Code Size and FLOPs with the SAE Method on Gynaeco Tumor

Optimization							SAE Full Training on Middle Layer								K-NN Over Test Set Searched on Test					
Dataset	Npop	#Epochs	Batch size	#Generation	Data Size (train, validation , test)	Neuron range	K	Unsupervise #Epochs		SAE #Epoch			Batch size		K = 3					
Gynaeco	100	1	4	100	(277, 35,30)	[0,1) , Initial= [0.5,1)	3	200		100			4							
	Primary results only by RMSE						On Validation Set			On Test Set			Middle Layer		On Middle Layer		Best Among All Layers			
Structures	RMSE	Classification Error	AF1	F1-score per class	FLOPs	Code Size	AF1	F1-score per class	RMSE	AF1	F1-score per class	RMSE	Code Layer	Code Size	AF1	F1-score per class	AF1	F1-score per class	Code Layer	Code Size
1024 [19, 18, 14, 8, 14, 18, 19] 1024	0.06	0	1	[1.0, 1.0, 1.0]	8.06E+04	8	1	[1.0, 1.0, 1.0]	0.05	0.81	[0.86, 0.82, 0.75]	0.06	4	8	0.64	[0.86, 0.74, 0.33]	0.8	[0.88, 0.78, 0.75]	7	19
1024 [23, 20, 12, 5, 12, 20, 23] 1024	0.07	0	1	[1.0, 1.0, 1.0]	9.72E+04	5	1	[1.0, 1.0, 1.0]	0.05	0.94	[0.94, 0.89, 1.0]	0.06	4	5	0.78	[0.88, 0.8, 0.67]	0.92	[0.91, 0.86, 1.0]	2	20
1024 [41, 22, 8, 4, 8, 22, 41] 1024	0.06	0	1	[1.0, 1.0, 1.0]	1.72E+05	4	0.94	[0.98, 1.0, 0.86]	0.05	0.82	[0.91, 0.89, 0.67]	0.06	4	4	0.79	[0.92, 0.95, 0.5]	0.9	[0.94, 0.95, 0.8]	7	41
1024 [121, 52, 22, 2, 22, 52, 121] 1024	0.06	0	1	[1.0, 1.0, 1.0]	5.26E+05	2	0.26	[0.77, 0.0, 0.0]	0.05	0.24	[0.72, 0.0, 0.0]	0.06	4	2	0.84	[0.89, 0.84, 0.8]	0.84	[0.89, 0.84, 0.8]	4	2
1024 [110, 79, 36, 2, 36, 79, 110] 1024	0.06	0.02	0.98	[0.98, 0.95, 1.0]	4.97E+05	2	0.62	[1.0, 0.87, 0.0]	0.05	0.58	[0.91, 0.82, 0.0]	0.06	4	2	0.46	[0.85, 0.55, 0.0]	0.88	[0.94, 0.82, 0.86]	7	110
1024 [104, 33, 10, 3, 10, 33, 104] 1024	0.06	0.02	0.98	[0.98, 0.95, 1.0]	4.41E+05	3	1	[1.0, 1.0, 1.0]	0.05	0.94	[0.94, 0.89, 1.0]	0.06	4	3	0.78	[0.84, 0.71, 0.8]	0.84	[0.89, 0.82, 0.8]	1	104
1024 [22, 20, 13, 7, 13, 20, 22] 1024	0.07	0.06	0.94	[0.98, 1.0, 0.86]	9.33E+04	7	1	[1.0, 1.0, 1.0]	0.05	0.93	[1.0, 0.95, 0.86]	0.06	4	7	0.8	[0.83, 0.71, 0.86]	0.83	[0.86, 0.78, 0.86]	3	13
1024 [86, 44, 3, 0, 3, 44, 86] 1024	0.06	0.06	0.94	[0.98, 1.0, 0.86]	3.68E+05	3	0.93	[1.0, 0.95, 0.86]	0.05	0.74	[0.92, 0.89, 0.4]	0.07	4	3	0.44	[0.76, 0.56, 0.0]	0.58	[0.72, 0.63, 0.4]	2	44
1024 [65, 42, 25, 2, 25, 42, 65] 1024	0.07	0.07	0.93	[1.0, 0.95, 0.86]	2.82E+05	2	0.94	[0.98, 1.0, 0.86]	0.05	0.8	[0.97, 0.75, 0.67]	0.06	4	2	0.81	[0.82, 0.74, 0.86]	0.86	[0.89, 0.82, 0.86]	5	25
1024 [18, 14, 11, 7, 11, 14, 18] 1024	0.07	0.08	0.92	[0.95, 0.95, 0.86]	7.57E+04	7	1	[1.0, 1.0, 1.0]	0.05	0.83	[0.86, 0.78, 0.86]	0.06	4	7	0.81	[0.82, 0.74, 0.86]	0.83	[0.86, 0.78, 0.86]	3	11
1024 [32, 14, 6, 4, 6, 14, 32] 1024	0.07	0.08	0.92	[0.95, 0.95, 0.86]	1.33E+05	4	0.62	[0.92, 0.95, 0.0]	0.05	0.53	[0.83, 0.75, 0.0]	0.07	4	4	0.88	[0.89, 0.75, 1.0]	0.91	[0.92, 0.82, 1.0]	1	32
1024 [17, 14, 11, 7, 11, 14, 17] 1024	0.07	0.08	0.92	[0.95, 0.95, 0.86]	7.15E+04	7	0.93	[1.0, 0.95, 0.86]	0.06	0.69	[0.92, 0.82, 0.33]	0.06	4	7	0.44	[0.67, 0.67, 0.0]	0.76	[0.81, 0.67, 0.8]	2	14
1024 [22, 13, 6, 0, 6, 13, 22] 1024	0.07	0.08	0.92	[0.95, 0.95, 0.86]	9.16E+04	6	1	[1.0, 1.0, 1.0]	0.05	0.93	[1.0, 0.95, 0.86]	0.06	3	6	0.73	[0.86, 0.75, 0.57]	0.77	[0.82, 0.63, 0.86]	1	22
1024 [23, 7, 4, 0, 4, 7, 23] 1024	0.07	0.08	0.92	[0.95, 0.95, 0.86]	9.50E+04	4	0.94	[0.98, 1.0, 0.86]	0.06	0.76	[0.83, 0.95, 0.5]	0.07	3	4	0.55	[0.82, 0.82, 0.0]	0.55	[0.81, 0.84, 0.0]	1	23
1024 [25, 19, 10, 3, 10, 19, 25] 1024	0.07	0.08	0.92	[0.95, 0.95, 0.86]	1.05E+05	3	1	[1.0, 1.0, 1.0]	0.05	0.71	[0.89, 0.82, 0.4]	0.06	4	3	0.76	[0.81, 0.67, 0.8]	0.9	[0.94, 0.95, 0.8]	1	25
1024 [31, 20, 15, 2, 15, 20, 31] 1024	0.07	0.09	0.91	[0.98, 0.89, 0.86]	1.31E+05	2	1	[1.0, 1.0, 1.0]	0.05	0.79	[0.92, 0.95, 0.5]	0.06	4	2	0.45	[0.8, 0.55, 0.0]	0.86	[0.86, 0.74, 1.0]	1	31
1024 [16, 12, 6, 3, 6, 12, 16] 1024	0.07	0.12	0.88	[0.88, 0.76, 1.0]	6.67E+04	3	0.94	[0.98, 1.0, 0.86]	0.05	0.88	[0.94, 0.84, 0.86]	0.06	4	3	0.73	[0.8, 0.6, 0.8]	0.82	[0.86, 0.8, 0.8]	1	16
1024 [10, 3, 0, 2, 0, 3, 10] 1024	0.07	0.13	0.87	[0.91, 0.84, 0.86]	4.11E+04	2	0.26	[0.77, 0.0, 0.0]	0.06	0.23	[0.7, 0.0, 0.0]	0.06	3	2	0.83	[0.83, 0.67, 1.0]	0.86	[0.86, 0.74, 1.0]	1	10
1024 [7, 6, 5, 4, 5, 6, 7] 1024	0.07	0.17	0.83	[0.84, 0.8, 0.86]	2.90E+04	4	0.81	[0.98, 0.78, 0.67]	0.06	0.87	[0.85, 0.76, 1.0]	0.07	4	4	0.45	[0.8, 0.55, 0.0]	0.48	[0.82, 0.61, 0.0]	3	5
1024 [13, 3, 1, 0, 1, 3, 13] 1024	0.06	0.21	0.79	[0.86, 0.84, 0.67]	5.34E+04	1	0.26	[0.77, 0.0, 0.0]	0.07	0.23	[0.7, 0.0, 0.0]	0.07	3	1	0.23	[0.7, 0.0, 0.0]	0.81	[0.8, 0.63, 1.0]	1	13
1024 [7, 5, 3, 1, 3, 5, 7] 1024	0.06	0.24	0.76	[0.91, 0.7, 0.67]	2.89E+04	1	0.6	[0.98, 0.83, 0.0]	0.06	0.58	[0.94, 0.8, 0.0]	0.06	4	1	0.87	[0.85, 0.76, 1.0]	0.89	[0.89, 0.78, 1.0]	2	5
1024 [3, 3, 2, 1, 2, 3, 3] 1024	0.06	0.25	0.75	[0.81, 0.7, 0.75]	1.24E+04	1	0.26	[0.77, 0.0, 0.0]	0.07	0.24	[0.72, 0.0, 0.0]	0.07	4	1	0.24	[0.72, 0.0, 0.0]	0.24	[0.72, 0.0, 0.0]	4	1
1024 [2, 0, 1, 1, 1, 0, 2] 1024	0.07	0.27	0.73	[0.95, 0.62, 0.6]	8.20E+03	1	0.26	[0.77, 0.0, 0.0]	0.07	0.24	[0.72, 0.0, 0.0]	0.07	3	1	0.24	[0.72, 0.0, 0.0]	0.48	[0.5, 0.38, 0.57]	1	2
1024 [1, 0, 0, 0, 0, 0, 1] 1024	0.07	0.29	0.71	[0.77, 0.89, 0.46]	4.10E+03	1	0.86	[1.0, 0.9, 0.67]	0.07	0.69	[0.92, 0.82, 0.33]	0.07	1	1	0.22	[0.67, 0.0, 0.0]	0.24	[0.72, 0.0, 0.0]	2	1

Table 4.30: Investigation on the 3 objectives of Classification Error, Code Size and FLOPs with the SAE Method on Liver Tumor

Optimization							SAE Full Training on Middle Layer								K-NN Over Test Set Searched on Test					
Dataset	Npop	#Epochs	Batch size	#Generations	Data Size (train, validation, test)	Neuron range	K	Unsupervised #Epochs		SAE #Epochs			Batch size		K = 3					
Liver	100	1	4	100	(416, 51, 51)	[0,1) , Initial=[0.5,1)	3	200		100			4							
	Primary results only by RMSE						On Validation Set			On Test Set			Middle Layer		On Middle Layer		Best Among All Layers			
Structures	RMSE	Classification Error	AF1	F1-score per class	FLOPs	Code Size	AF1	F1-score per class	RMSE	AF1	F1-score per class	RMSE	Code Layer	Code Size	AF1	F1-score per class	AF1	F1-score per class	Code Layer	Code Size
1024 [11, 8, 6, 5, 6, 8, 11] 1024	0.05	0.07	0.93	[0.86, 0.97, 0.96]	4.57E+04	5	0.83	[0.67, 0.93, 0.89]	0.03	0.61	[0.0, 0.97, 0.86]	0.04	4	5	0.68	[0.29, 0.94, 0.8]	0.69	[0.33, 0.93, 0.8]	2	8
1024 [181, 28, 15, 3, 15, 28, 181] 1024	0.05	0.07	0.93	[0.86, 0.97, 0.96]	7.64E+05	3	0.27	[0.0, 0.81, 0.0]	0.05	0.27	[0.0, 0.81, 0.0]	0.05	4	3	0.05	[0.15, 0.0, 0.0]	0.7	[0.33, 0.94, 0.83]	2	28
1024 [88, 51, 39, 2, 39, 51, 88] 1024	0.05	0.09	0.91	[0.86, 0.96, 0.92]	3.87E+05	2	0.27	[0.0, 0.81, 0.0]	0.05	0.27	[0.0, 0.81, 0.0]	0.05	4	2	0.05	[0.15, 0.0, 0.0]	0.76	[0.67, 0.93, 0.67]	2	51
1024 [80, 40, 14, 4, 14, 40, 80] 1024	0.05	0.09	0.91	[0.86, 0.96, 0.92]	3.43E+05	4	0.81	[0.57, 0.93, 0.92]	0.03	0.87	[0.73, 0.96, 0.92]	0.04	4	4	0.52	[0.0, 0.93, 0.64]	0.6	[0.0, 0.97, 0.83]	7	80
1024 [54, 21, 7, 3, 7, 21, 54] 1024	0.05	0.1	0.9	[0.86, 0.94, 0.89]	2.26E+05	3	0.91	[0.75, 0.97, 1.0]	0.04	0.74	[0.44, 0.96, 0.83]	0.04	4	3	0.7	[0.36, 0.97, 0.76]	0.8	[0.6, 0.97, 0.82]	3	7
1024 [91, 31, 2, 0, 2, 31, 91] 1024	0.04	0.1	0.9	[0.86, 0.94, 0.89]	3.84E+05	2	0.93	[0.86, 0.97, 0.96]	0.04	0.6	[0.0, 0.96, 0.85]	0.04	3	2	0.6	[0.0, 0.96, 0.85]	0.64	[0.22, 0.96, 0.75]	1	91
1024 [16, 8, 5, 4, 5, 8, 16] 1024	0.05	0.11	0.89	[0.75, 0.96, 0.96]	6.63E+04	4	0.86	[0.67, 0.94, 0.96]	0.03	0.84	[0.62, 0.94, 0.96]	0.04	4	4	0.59	[0.0, 0.96, 0.8]	0.62	[0.0, 0.96, 0.89]	1	16
1024 [25, 8, 5, 3, 5, 8, 25] 1024	0.05	0.11	0.89	[0.75, 0.96, 0.96]	1.03E+05	3	0.93	[0.86, 0.97, 0.96]	0.04	0.61	[0.0, 0.96, 0.88]	0.04	4	3	0.61	[0.0, 0.94, 0.89]	0.66	[0.25, 0.97, 0.75]	1	25
1024 [13, 11, 6, 2, 6, 11, 13] 1024	0.05	0.13	0.87	[0.67, 0.96, 1.0]	5.41E+04	2	0.86	[0.67, 0.96, 0.96]	0.04	0.79	[0.5, 0.99, 0.87]	0.04	4	2	0.56	[0.0, 0.91, 0.76]	0.74	[0.5, 0.96, 0.76]	1	13
1024 [7, 0, 5, 3, 5, 0, 7] 1024	0.05	0.15	0.85	[0.67, 0.97, 0.92]	2.89E+04	3	0.82	[0.67, 0.94, 0.86]	0.03	0.51	[0.17, 0.94, 0.42]	0.04	3	3	0.57	[0.0, 0.96, 0.75]	0.59	[0.0, 0.96, 0.8]	4	5
1024 [11, 8, 2, 2, 2, 8, 11] 1024	0.05	0.15	0.85	[0.67, 0.97, 0.92]	4.55E+04	2	0.27	[0.0, 0.81, 0.0]	0.05	0.27	[0.0, 0.81, 0.0]	0.05	4	2	0.43	[0.55, 0.73, 0.0]	0.43	[0.55, 0.73, 0.0]	3	2
1024 [48, 19, 1, 0, 1, 19, 48] 1024	0.05	0.16	0.84	[0.67, 0.96, 0.89]	2.00E+05	1	0.27	[0.0, 0.81, 0.0]	0.05	0.27	[0.0, 0.81, 0.0]	0.05	3	1	0.05	[0.15, 0.0, 0.0]	0.61	[0.2, 0.94, 0.7]	1	48
1024 [11, 6, 2, 0, 2, 6, 11] 1024	0.05	0.16	0.84	[0.67, 0.96, 0.89]	4.54E+04	2	0.87	[0.67, 0.96, 1.0]	0.04	0.63	[0.0, 0.96, 0.92]	0.04	3	2	0.62	[0.0, 0.96, 0.89]	0.62	[0.0, 0.96, 0.89]	3	2
1024 [18, 9, 2, 1, 2, 9, 18] 1024	0.05	0.18	0.82	[0.57, 0.96, 0.92]	7.45E+04	1	0.27	[0.0, 0.81, 0.0]	0.05	0.27	[0.0, 0.81, 0.0]	0.05	4	1	0.05	[0.15, 0.0, 0.0]	0.55	[0.2, 0.93, 0.53]	1	18
1024 [5, 4, 3, 2, 3, 4, 5] 1024	0.05	0.19	0.81	[0.5, 0.96, 0.96]	2.06E+04	2	0.27	[0.0, 0.81, 0.0]	0.05	0.27	[0.0, 0.81, 0.0]	0.05	4	2	0.05	[0.15, 0.0, 0.0]	0.43	[0.0, 0.77, 0.52]	3	3
1024 [10, 7, 3, 1, 3, 7, 10] 1024	0.05	0.21	0.79	[0.5, 0.94, 0.92]	4.13E+04	1	0.27	[0.0, 0.81, 0.0]	0.05	0.27	[0.0, 0.81, 0.0]	0.05	4	1	0.05	[0.15, 0.0, 0.0]	0.55	[0.0, 0.86, 0.78]	1	10
1024 [3, 1, 0, 0, 0, 1, 3] 1024	0.05	0.24	0.76	[0.44, 0.93, 0.92]	1.23E+04	1	0.27	[0.0, 0.81, 0.0]	0.05	0.27	[0.0, 0.81, 0.0]	0.05	2	1	0.05	[0.15, 0.0, 0.0]	0.05	[0.15, 0.0, 0.0]	2	1
1024 [2, 2, 1, 0, 1, 2, 2] 1024	0.05	0.3	0.7	[0.25, 0.94, 0.92]	8.22E+03	1	0.27	[0.0, 0.81, 0.0]	0.05	0.27	[0.0, 0.81, 0.0]	0.05	3	1	0.35	[0.33, 0.72, 0.0]	0.35	[0.33, 0.72, 0.0]	3	1
1024 [1, 0, 0, 1, 0, 0, 1] 1024	0.05	0.3	0.7	[0.29, 0.96, 0.85]	4.10E+03	1	0.65	[0.0, 0.95, 1.0]	0.04	0.63	[0.0, 0.97, 0.92]	0.04	2	1	0.62	[0.0, 0.96, 0.89]	0.62	[0.0, 0.96, 0.89]	1	1
1024 [1, 0, 0, 0, 0, 0, 1] 1024	0.05	0.31	0.69	[0.25, 0.93, 0.88]	4.10E+03	1	0.86	[0.67, 0.97, 0.96]	0.05	0.72	[0.29, 0.96, 0.92]	0.05	1	1	0.33	[0.22, 0.75, 0.0]	0.33	[0.22, 0.75, 0.0]	1	1

Table 4.31: Investigation on the 3 objectives of Classification Error, Code Size and FLOPs with the SAE Method on Melanocytic Tumor

Optimization							SAE Full Training on Middle Layer								K-NN Over Test Set Searched on Test					
Dataset	Npop	#Epochs	Batch size	#Generations	Data Size (train, validation, test)	Neuron range	K	Unsupervised #Epochs		SAE #Epochs			Batch size		K = 3					
Melanocytic	100	1	4	100	(230, 29, 28)	[0,1) , Initial=[0.5,1)	3	200		100			4							
	Primary results only by RMSE						On Validation Set			On Test Set			Middle Layer		On Middle Layer		Best Among All Layers			
Structures	RMSE	Classification Error	AF1	F1-score per class	FLOPs	Code Size	AF1	F1-score per class	RMSE	AF1	F1-score per class	RMSE	Code Layer	Code Size	AF1	F1-score per class	AF1	F1-score per class	Code Layer	Code Size
1024 [1, 0, 0, 0, 0, 0, 1] 1024	0.07	0	1	[1.0, 1.0]	4.10E+03	1	1	[1.0, 1.0]	0.06	0.85	[0.75, 0.96]	0.06	1	1	0.45	[0.0, 0.9]	0.45	[0.0, 0.9]	1	1

Table 4.32: Investigation on the 3 objectives of Classification Error, Code Size and FLOPs with the SAE Method on Prostate Tumor

Optimization							SAE Full Training on Middle Layer								K-NN Over Test Set Searched on Test					
Dataset	Npop	#Epochs	Batch size	#Generations	Data Size (train, validation, test)	Neuron range	K	Unsupervised #Epochs		SAE #Epochs			Batch size		K = 3					
Prostate	100	1	4	100	(517, 51, 53)	[0,1) , Initial=[0.5,1)	3	200		100			4							
	Primary results only by RMSE						On Validation Set			On Test Set			Middle Layer		On Middle Layer		Best Among All Layers			
Structures	RMSE	Classification Error	AF1	F1-score per class	FLOPs	Code Size	AF1	F1-score per class	RMSE	AF1	F1-score per class	RMSE	Code Layer	Code Size	AF1	F1-score per class	AF1	F1-score per class	Code Layer	Code Size
1024 [5, 4, 2, 1, 2, 4, 5] 1024	0.06	0	1	[1.0, 1.0]	2.06E+04	1	0.97	[0.99, 0.96]	0.06	1	[1.0, 1.0]	0.06	4	1	0.43	[0.86, 0.0]	0.61	[0.81, 0.4]	1	5
1024 [1, 0, 0, 0, 0, 0, 1] 1024	0.05	0.03	0.97	[0.99, 0.96]	4.10E+03	1	0.43	[0.85, 0.0]	0.06	0.43	[0.86, 0.0]	0.06	1	1	0.43	[0.86, 0.0]	0.43	[0.86, 0.0]	1	1

Table 4.33: Investigation on the 3 objectives of Classification Error, Code Size and FLOPs with the SAE Method on Pulmonary Tumor

Optimization							SAE Full Training on Middle Layer								K-NN Over Test Set Searched on Test					
Dataset	Npop	#Epochs	Batch size	#Generations	Data Size (train, validation, test)	Neuron range	K	Unsupervise #Epochs		SAE #Epoch			Batch size		K = 3					
Pulmonary	100	1	4	100	(719, 86, 84)	[0,1) , Initial= [0.5,1)	3	200		100			4							
	Primary results only by RMSE						On Validation Set			On Test Set			Middle Layer		On Middle Layer		Best Among All Layers			
Structures	RMSE	Classification Error	AF1	F1-score per class	FLOPs	Code Size	AF1	F1-score per class	RMSE	AF1	F1-score per class	RMSE	Code Layer	Code Size	AF1	F1-score per class	AF1	F1-score per class	Code Layer	Code Size
1024 [249, 91, 32, 24, 32, 91, 249] 1024	0.04	0.27	0.73	[0.86, 0.84, 0.5]	1.13E+06	24	0.95	[0.92, 0.93, 1.0]	0.04	0.88	[0.85, 0.89, 0.89]	0.03	4	24	0.64	[0.79, 0.8, 0.33]	0.75	[0.83, 0.87, 0.57]	1	249
1024 [126, 82, 38, 17, 38, 82, 126] 1024	0.04	0.28	0.72	[0.84, 0.8, 0.53]	5.72E+05	17	0.95	[0.91, 0.93, 1.0]	0.04	0.9	[0.84, 0.86, 1.0]	0.03	4	17	0.80	[0.81, 0.84, 0.75]	0.86	[0.83, 0.86, 0.89]	5	38
1024 [181, 103, 40, 14, 40, 103, 181] 1024	0.05	0.28	0.72	[0.85, 0.84, 0.48]	8.35E+05	14	0.84	[0.93, 0.92, 0.67]	0.04	0.82	[0.88, 0.87, 0.71]	0.03	4	14	0.72	[0.76, 0.84, 0.57]	0.75	[0.79, 0.87, 0.57]	1	181
1024 [108, 51, 17, 4, 17, 51, 108] 1024	0.05	0.28	0.72	[0.9, 0.81, 0.45]	4.68E+05	4	0.81	[0.94, 0.91, 0.59]	0.04	0.71	[0.82, 0.81, 0.5]	0.04	4	4	0.69	[0.75, 0.83, 0.5]	0.75	[0.81, 0.86, 0.57]	1	108
1024 [85, 48, 31, 19, 31, 48, 85] 1024	0.04	0.29	0.71	[0.92, 0.82, 0.4]	3.73E+05	19	0.9	[0.93, 0.94, 0.83]	0.04	0.88	[0.84, 0.89, 0.91]	0.03	4	19	0.50	[0.7, 0.79, 0.0]	0.74	[0.81, 0.84, 0.57]	1	85
1024 [64, 53, 35, 19, 35, 53, 64] 1024	0.05	0.3	0.7	[0.85, 0.79, 0.45]	2.86E+05	19	0.91	[0.92, 0.91, 0.91]	0.04	0.9	[0.86, 0.84, 1.0]	0.04	4	19	0.64	[0.79, 0.83, 0.29]	0.85	[0.82, 0.84, 0.89]	6	53
1024 [102, 47, 30, 5, 30, 47, 102] 1024	0.05	0.3	0.7	[0.8, 0.84, 0.45]	4.43E+05	5	0.97	[0.94, 0.95, 1.0]	0.04	0.87	[0.83, 0.88, 0.91]	0.03	4	5	0.67	[0.82, 0.87, 0.33]	0.80	[0.81, 0.85, 0.75]	5	30
1024 [99, 37, 11, 8, 11, 37, 99] 1024	0.04	0.32	0.68	[0.82, 0.76, 0.48]	4.22E+05	8	0.85	[0.94, 0.94, 0.67]	0.04	0.8	[0.84, 0.83, 0.71]	0.03	4	8	0.67	[0.69, 0.73, 0.57]	0.80	[0.82, 0.84, 0.75]	1	99
1024 [91, 40, 16, 7, 16, 40, 91] 1024	0.05	0.32	0.68	[0.87, 0.79, 0.38]	3.90E+05	7	0.84	[0.91, 0.94, 0.67]	0.04	0.86	[0.8, 0.86, 0.91]	0.04	4	7	0.49	[0.72, 0.76, 0.0]	0.71	[0.76, 0.79, 0.57]	3	16
1024 [107, 46, 15, 4, 15, 46, 107] 1024	0.04	0.32	0.68	[0.82, 0.78, 0.45]	4.61E+05	4	0.91	[0.96, 0.94, 0.83]	0.04	0.92	[0.88, 0.88, 1.0]	0.04	4	4	0.60	[0.69, 0.77, 0.33]	0.79	[0.79, 0.81, 0.75]	1	107
1024 [83, 30, 14, 4, 14, 30, 83] 1024	0.04	0.32	0.68	[0.84, 0.85, 0.36]	3.52E+05	4	0.87	[0.89, 0.88, 0.83]	0.05	0.83	[0.82, 0.85, 0.8]	0.04	4	4	0.63	[0.77, 0.79, 0.33]	0.89	[0.89, 0.9, 0.89]	1	83
1024 [69, 60, 10, 8, 10, 60, 69] 1024	0.05	0.32	0.68	[0.89, 0.78, 0.37]	3.02E+05	8	0.85	[0.94, 0.94, 0.67]	0.04	0.89	[0.84, 0.84, 1.0]	0.03	4	8	0.74	[0.72, 0.76, 0.75]	0.79	[0.82, 0.81, 0.75]	7	69
1024 [53, 29, 26, 11, 26, 29, 53] 1024	0.05	0.34	0.66	[0.81, 0.81, 0.37]	2.27E+05	11	0.8	[0.88, 0.9, 0.62]	0.04	0.77	[0.84, 0.85, 0.62]	0.03	4	11	0.86	[0.89, 0.89, 0.8]	0.87	[0.78, 0.82, 1.0]	1	53
1024 [69, 34, 12, 8, 12, 34, 69] 1024	0.04	0.34	0.66	[0.82, 0.8, 0.37]	2.94E+05	8	0.91	[0.89, 0.92, 0.91]	0.04	0.85	[0.87, 0.88, 0.8]	0.04	4	8	0.59	[0.76, 0.78, 0.25]	0.84	[0.81, 0.81, 0.89]	1	69
1024 [54, 33, 16, 5, 16, 33, 54] 1024	0.05	0.34	0.66	[0.86, 0.77, 0.36]	2.31E+05	5	0.91	[0.91, 0.92, 0.91]	0.04	0.87	[0.87, 0.84, 0.91]	0.04	4	5	0.49	[0.76, 0.73, 0.0]	0.82	[0.84, 0.86, 0.75]	1	54
1024 [156, 14, 8, 3, 8, 14, 156] 1024	0.05	0.34	0.66	[0.84, 0.75, 0.38]	6.48E+05	3	0.61	[0.89, 0.95, 0.0]	0.05	0.56	[0.81, 0.87, 0.0]	0.05	4	3	0.43	[0.59, 0.72, 0.0]	0.88	[0.8, 0.85, 1.0]	1	156
1024 [33, 26, 18, 5, 18, 26, 33] 1024	0.05	0.35	0.65	[0.82, 0.77, 0.37]	1.41E+05	5	0.97	[0.94, 0.95, 1.0]	0.04	0.92	[0.88, 0.88, 1.0]	0.04	4	5	0.61	[0.74, 0.76, 0.33]	0.74	[0.82, 0.83, 0.57]	2	26
1024 [28, 8, 0, 4, 0, 8, 28] 1024	0.05	0.35	0.65	[0.84, 0.79, 0.33]	1.16E+05	4	0.83	[0.94, 0.89, 0.67]	0.04	0.68	[0.84, 0.74, 0.47]	0.04	3	4	0.67	[0.85, 0.84, 0.33]	0.69	[0.87, 0.87, 0.33]	4	8
1024 [128, 35, 14, 2, 14, 35, 128] 1024	0.05	0.35	0.65	[0.84, 0.79, 0.33]	5.44E+05	2	0.93	[0.89, 0.9, 1.0]	0.04	0.9	[0.85, 0.86, 1.0]	0.04	4	2	0.75	[0.66, 0.71, 0.89]	0.91	[0.85, 0.87, 1.0]	1	128
1024 [68, 41, 8, 3, 8, 41, 68] 1024	0.05	0.36	0.64	[0.81, 0.75, 0.37]	2.91E+05	3	0.83	[0.8, 0.7, 1.0]	0.05	0.67	[0.72, 0.54, 0.75]	0.04	4	3	0.85	[0.76, 0.8, 1.0]	0.93	[0.89, 0.91, 1.0]	1	68
1024 [51, 26, 5, 2, 5, 26, 51] 1024	0.05	0.38	0.62	[0.79, 0.74, 0.33]	2.15E+05	2	0.78	[0.9, 0.85, 0.59]	0.05	0.75	[0.82, 0.85, 0.59]	0.05	4	2	0.58	[0.77, 0.75, 0.22]	0.89	[0.83, 0.85, 1.0]	1	51
1024 [31, 12, 8, 2, 8, 12, 31] 1024	0.05	0.39	0.61	[0.75, 0.79, 0.3]	1.29E+05	2	0.86	[0.89, 0.92, 0.77]	0.06	0.85	[0.84, 0.88, 0.83]	0.05	4	2	0.84	[0.75, 0.78, 1.0]	0.86	[0.82, 0.84, 0.91]	1	31
1024 [25, 21, 13, 7, 13, 21, 25] 1024	0.05	0.4	0.6	[0.78, 0.65, 0.37]	1.06E+05	7	0.8	[0.89, 0.91, 0.59]	0.04	0.85	[0.82, 0.81, 0.91]	0.04	4	7	0.66	[0.81, 0.85, 0.33]	0.82	[0.84, 0.86, 0.75]	2	21
1024 [23, 10, 4, 0, 4, 10, 23] 1024	0.05	0.43	0.57	[0.7, 0.7, 0.32]	9.53E+04	4	0.67	[0.45, 0.73, 0.83]	0.05	0.58	[0.42, 0.72, 0.6]	0.04	3	4	0.69	[0.76, 0.79, 0.5]	0.80	[0.83, 0.83, 0.75]	1	23
1024 [1, 0, 0, 0, 0, 0, 1] 1024	0.05	0.44	0.56	[0.62, 0.62, 0.44]	4.10E+03	1	0.22	[0.0, 0.66, 0.0]	0.05	0.22	[0.0, 0.67, 0.0]	0.04	1	1	0.80	[0.71, 0.69, 1.0]	0.80	[0.71, 0.69, 1.0]	1	1

Table 4.34: Investigation on the 3 objectives of Classification Error, Code Size and FLOPs with the SAE Method on Urinary Tract Tumor

Optimization							SAE Full Training on Middle Layer								K-NN Over Test Set Searched on Test					
Dataset	Npop	#Epochs	Batch size	#Generations	Data Size (train, validation, test)	Neuron range	K	Unsupervised #Epochs		SAE #Epochs			Batch size		K = 3					
Urinary Tract	100	1	4	100	(1048, 120, 123)	[0,1) , Initial= [0.5,1)	3	200		100			4							
	Primary results only by RMSE						On Validation Set			On Test Set			Middle Layer		On Middle Layer		Best Among All Layers			
Structures	RMSE	Classification Error	Averaged F1-score	F1-score per class	FLOPs	Code Size	Averaged F1-score	F1-score per class	RMSE	Averaged F1-score	F1-score per class	RMSE	Code Layer	Code Size	Averaged F1-score	F1-score per class	Averaged F1-score	F1-score per class	Code Layer	Code Size
1024 [69, 42, 35, 17, 35, 42, 69] 1024	0.03	0.02	0.98	[1.0, 0.96, 0.98, 0.98]	3.02E+05	17	0.94	[1.0, 0.91, 0.94, 0.92]	0.02	0.91	[0.93, 0.84, 0.96, 0.89]	0.03	4	17	0.89	[0.93, 0.9, 0.91, 0.81]	0.91	[0.92, 0.9, 0.95, 0.87]	2	42
1024 [378, 10, 8, 4, 8, 10, 378] 1024	0.04	0.02	0.98	[1.0, 0.96, 0.98, 0.98]	1.56E+06	4	0.15	[0.0, 0.0, 0.59, 0.0]	0.04	0.14	[0.0, 0.0, 0.58, 0.0]	0.05	4	4	0	[0.02, 0.0, 0.0, 0.0]	0.89	[0.94, 0.8, 0.96, 0.85]	1	378
1024 [116, 29, 15, 10, 15, 29, 116] 1024	0.04	0.03	0.97	[1.0, 0.96, 0.97, 0.96]	4.91E+05	10	0.94	[1.0, 0.91, 0.94, 0.92]	0.02	0.93	[0.94, 0.9, 0.95, 0.91]	0.03	4	10	0.86	[0.86, 0.84, 0.95, 0.78]	0.89	[0.9, 0.84, 0.96, 0.85]	7	116
1024 [53, 29, 21, 16, 21, 29, 53] 1024	0.04	0.03	0.97	[1.0, 0.96, 0.97, 0.96]	2.27E+05	16	0.96	[1.0, 0.91, 0.96, 0.96]	0.02	0.94	[0.96, 0.95, 0.95, 0.91]	0.03	4	16	0.92	[0.91, 0.9, 0.97, 0.88]	0.93	[0.94, 0.9, 0.97, 0.91]	1	53
1024 [110, 50, 27, 11, 27, 50, 110] 1024	0.03	0.03	0.97	[1.0, 0.95, 0.97, 0.96]	4.79E+05	11	0.95	[0.99, 0.96, 0.96, 0.92]	0.02	0.9	[0.88, 0.95, 0.96, 0.8]	0.03	4	11	0.88	[0.93, 0.84, 0.93, 0.84]	0.92	[0.94, 0.86, 0.96, 0.91]	1	110
1024 [20, 18, 12, 6, 12, 18, 20] 1024	0.04	0.04	0.96	[1.0, 0.96, 0.96, 0.94]	8.45E+04	6	0.96	[1.0, 0.95, 0.96, 0.94]	0.03	0.9	[0.93, 0.84, 0.95, 0.87]	0.03	4	6	0.9	[0.93, 0.84, 0.96, 0.86]	0.92	[0.93, 0.9, 0.96, 0.91]	6	18
1024 [41, 30, 23, 5, 23, 30, 41] 1024	0.04	0.04	0.96	[1.0, 0.91, 0.96, 0.96]	1.76E+05	5	0.95	[1.0, 0.87, 0.97, 0.96]	0.02	0.89	[0.96, 0.78, 0.95, 0.87]	0.03	4	5	0.86	[0.92, 0.84, 0.9, 0.79]	0.9	[0.93, 0.9, 0.94, 0.84]	6	30
1024 [140, 75, 36, 4, 36, 75, 140] 1024	0.04	0.05	0.95	[1.0, 0.91, 0.95, 0.94]	6.27E+05	4	0.97	[1.0, 0.95, 0.97, 0.96]	0.03	0.9	[0.92, 0.84, 0.97, 0.89]	0.03	4	4	0.86	[0.9, 0.7, 0.96, 0.88]	0.92	[0.94, 0.86, 0.97, 0.93]	6	75
1024 [84, 36, 17, 4, 17, 36, 84] 1024	0.04	0.06	0.94	[1.0, 0.91, 0.94, 0.92]	3.59E+05	4	0.95	[0.99, 0.95, 0.94, 0.92]	0.03	0.91	[0.93, 0.9, 0.94, 0.86]	0.03	4	4	0.84	[0.83, 0.73, 0.97, 0.83]	0.91	[0.9, 0.9, 0.96, 0.87]	2	36
1024 [51, 37, 29, 3, 29, 37, 51] 1024	0.04	0.06	0.94	[1.0, 0.86, 0.95, 0.96]	2.21E+05	3	0.98	[1.0, 0.95, 0.98, 0.98]	0.03	0.91	[0.93, 0.84, 0.95, 0.91]	0.03	4	3	0.86	[0.92, 0.7, 0.97, 0.85]	0.89	[0.94, 0.76, 0.97, 0.89]	7	51
1024 [279, 132, 69, 2, 69, 132, 279] 1024	0.04	0.06	0.94	[1.0, 0.87, 0.94, 0.94]	1.33E+06	2	0.15	[0.0, 0.0, 0.59, 0.0]	0.04	0.14	[0.0, 0.0, 0.58, 0.0]	0.05	4	2	0	[0.0, 0.0, 0.0, 0.0]	0.88	[0.93, 0.82, 0.97, 0.81]	1	279
1024 [17, 16, 14, 10, 14, 16, 17] 1024	0.04	0.06	0.94	[1.0, 0.91, 0.93, 0.9]	7.22E+04	10	0.95	[0.99, 0.95, 0.95, 0.9]	0.03	0.88	[0.94, 0.8, 0.94, 0.85]	0.03	4	10	0.86	[0.92, 0.76, 0.95, 0.81]	0.88	[0.92, 0.84, 0.95, 0.81]	7	17
1024 [43, 26, 22, 2, 22, 26, 43] 1024	0.04	0.07	0.93	[0.99, 0.87, 0.94, 0.92]	1.83E+05	2	0.79	[0.91, 0.52, 0.89, 0.85]	0.03	0.69	[0.87, 0.29, 0.79, 0.81]	0.03	4	2	0.83	[0.91, 0.7, 0.94, 0.75]	0.87	[0.91, 0.84, 0.95, 0.79]	3	22
1024 [30, 25, 14, 5, 14, 25, 30] 1024	0.04	0.07	0.93	[0.99, 0.87, 0.94, 0.92]	1.28E+05	5	0.97	[1.0, 0.95, 0.97, 0.96]	0.03	0.93	[0.94, 0.9, 0.95, 0.91]	0.03	4	5	0.9	[0.91, 0.9, 0.96, 0.83]	0.92	[0.94, 0.9, 0.95, 0.87]	6	25
1024 [20, 19, 10, 7, 10, 19, 20] 1024	0.04	0.08	0.92	[0.99, 0.87, 0.93, 0.9]	8.45E+04	7	0.96	[0.99, 0.95, 0.97, 0.94]	0.03	0.93	[0.94, 0.9, 0.96, 0.93]	0.03	4	7	0.87	[0.93, 0.8, 0.93, 0.81]	0.9	[0.93, 0.84, 0.95, 0.88]	6	19
1024 [34, 22, 10, 4, 10, 22, 34] 1024	0.04	0.08	0.92	[0.99, 0.82, 0.95, 0.92]	1.43E+05	4	0.96	[1.0, 0.95, 0.96, 0.94]	0.03	0.93	[0.96, 0.9, 0.95, 0.93]	0.03	4	4	0.89	[0.88, 0.84, 0.97, 0.87]	0.91	[0.91, 0.9, 0.96, 0.87]	2	22
1024 [28, 16, 8, 5, 8, 16, 28] 1024	0.04	0.09	0.91	[0.97, 0.87, 0.93, 0.88]	1.17E+05	5	0.97	[0.97, 1.0, 0.97, 0.94]	0.03	0.93	[0.94, 0.91, 0.97, 0.91]	0.03	4	5	0.9	[0.94, 0.86, 0.94, 0.87]	0.92	[0.94, 0.9, 0.96, 0.89]	5	8
1024 [14, 12, 10, 7, 10, 12, 14] 1024	0.04	0.09	0.91	[1.0, 0.83, 0.91, 0.9]	5.88E+04	7	0.95	[0.96, 0.91, 0.96, 0.98]	0.03	0.87	[0.92, 0.76, 0.94, 0.86]	0.03	4	7	0.89	[0.89, 0.84, 0.97, 0.85]	0.92	[0.92, 0.9, 0.96, 0.89]	3	10
1024 [34, 13, 9, 3, 9, 13, 34] 1024	0.04	0.09	0.91	[0.97, 0.8, 0.94, 0.92]	1.42E+05	3	0.96	[0.99, 0.96, 0.96, 0.92]	0.03	0.93	[0.96, 0.91, 0.97, 0.89]	0.03	4	3	0.85	[0.9, 0.82, 0.92, 0.75]	0.9	[0.92, 0.84, 0.97, 0.89]	7	34
1024 [20, 13, 7, 3, 7, 13, 20] 1024	0.04	0.1	0.9	[0.96, 0.73, 0.95, 0.96]	8.34E+04	3	0.96	[1.0, 0.91, 0.96, 0.96]	0.03	0.89	[0.9, 0.9, 0.95, 0.81]	0.03	4	3	0.85	[0.91, 0.7, 0.94, 0.84]	0.88	[0.92, 0.82, 0.96, 0.84]	1	20
1024 [17, 17, 13, 6, 13, 17, 17] 1024	0.04	0.1	0.9	[0.97, 0.76, 0.93, 0.92]	7.20E+04	6	0.96	[0.99, 0.96, 0.97, 0.94]	0.03	0.9	[0.93, 0.84, 0.94, 0.89]	0.03	4	6	0.88	[0.93, 0.9, 0.91, 0.78]	0.9	[0.93, 0.86, 0.95, 0.86]	1	17
1024 [16, 13, 7, 6, 7, 13, 16] 1024	0.04	0.11	0.89	[0.99, 0.76, 0.91, 0.9]	6.69E+04	6	0.96	[1.0, 0.91, 0.96, 0.96]	0.03	0.9	[0.97, 0.8, 0.95, 0.88]	0.03	4	6	0.88	[0.91, 0.84, 0.96, 0.81]	0.9	[0.9, 0.9, 0.95, 0.86]	7	16
1024 [40, 27, 21, 2, 21, 27, 40] 1024	0.04	0.12	0.88	[1.0, 0.73, 0.93, 0.86]	1.71E+05	2	0.15	[0.0, 0.0, 0.59, 0.0]	0.04	0.14	[0.0, 0.0, 0.58, 0.0]	0.05	4	2	0	[0.0, 0.0, 0.0, 0.0]	0.9	[0.97, 0.82, 0.93, 0.87]	1	40
1024 [28, 23, 13, 2, 13, 23, 28] 1024	0.04	0.12	0.88	[0.97, 0.76, 0.93, 0.85]	1.19E+05	2	0.15	[0.0, 0.0, 0.59, 0.0]	0.04	0.14	[0.0, 0.0, 0.58, 0.0]	0.05	4	2	0	[0.0, 0.0, 0.0, 0.0]	0.82	[0.92, 0.67, 0.9, 0.78]	1	28
1024 [16, 9, 5, 3, 5, 9, 16] 1024	0.04	0.12	0.88	[0.93, 0.87, 0.89, 0.82]	6.64E+04	3	0.95	[0.97, 0.91, 0.95, 0.98]	0.03	0.89	[0.89, 0.84, 0.93, 0.89]	0.03	4	3	0.9	[0.96, 0.8, 0.96, 0.88]	0.9	[0.94, 0.84, 0.95, 0.87]	2	9
1024 [22, 12, 5, 2, 5, 12, 22] 1024	0.04	0.13	0.87	[1.0, 0.78, 0.88, 0.82]	9.14E+04	2	0.95	[0.99, 0.9, 0.98, 0.94]	0.03	0.85	[0.93, 0.7, 0.95, 0.81]	0.04	4	2	0.84	[0.9, 0.67, 0.96, 0.82]	0.84	[0.9, 0.67, 0.96, 0.82]	4	2
1024 [14, 11, 6, 4, 6, 11, 14] 1024	0.04	0.13	0.87	[0.93, 0.8, 0.92, 0.82]	5.83E+04	4	0.95	[0.99, 0.96, 0.96, 0.92]	0.03	0.89	[0.89, 0.84, 0.97, 0.85]	0.03	4	4	0.88	[0.89, 0.78, 0.97, 0.89]	0.93	[0.94, 0.9, 0.97, 0.91]	1	14
1024 [12, 9, 6, 4, 6, 9, 12] 1024	0.04	0.16	0.84	[0.97, 0.73, 0.86, 0.8]	4.99E+04	4	0.96	[0.99, 0.95, 0.96, 0.96]	0.03	0.9	[0.94, 0.86, 0.95, 0.85]	0.03	4	4	0.85	[0.89, 0.8, 0.93, 0.78]	0.91	[0.96, 0.8, 0.96, 0.91]	7	12
1024 [9, 7, 5, 3, 5, 7, 9] 1024	0.04	0.19	0.81	[0.97, 0.73, 0.85, 0.69]	3.73E+04	3	0.91	[0.96, 0.78, 0.96, 0.94]	0.03	0.9	[0.94, 0.78, 0.95, 0.92]	0.04	4	3	0.76	[0.94, 0.36, 0.93, 0.81]	0.87	[0.97, 0.73, 0.94, 0.86]	2	7
1024 [16, 10, 4, 2, 4, 10, 16] 1024	0.04	0.2	0.8	[0.91, 0.65, 0.8, 0.86]	6.64E+04	2	0.94	[0.97, 0.91, 0.95, 0.94]	0.03	0.89	[0.91, 0.82, 0.96, 0.86]	0.03	4	2	0.82	[0.96, 0.57, 0.96, 0.78]	0.91	[0.96, 0.8, 0.97, 0.89]	1	16
1024 [8, 5, 4, 2, 4, 5, 8] 1024	0.04	0.2	0.8	[1.0, 0.67, 0.78, 0.75]	3.30E+04	2	0.15	[0.0, 0.0, 0.59, 0.0]	0.04	0.14	[0.0, 0.0, 0.58, 0.0]	0.05	4	2	0	[0.0, 0.0, 0.0, 0.0]	0.7	[0.78, 0.47, 0.86, 0.71]	1	8
1024 [188, 61, 36, 1, 36, 61, 188] 1024	0.04	0.24	0.76	[0.97, 0.52, 0.88, 0.67]	8.25E+05	1	0.87	[1.0, 0.64, 0.93, 0.92]	0.03	0.87	[0.94, 0.7, 0.96, 0.87]	0.03	4	1	0.8	[0.93, 0.55, 0.96, 0.76]	0.83	[0.94, 0.6, 0.96, 0.83]	1	188
1024 [7, 0, 6, 3, 6, 0, 7] 1024	0.04	0.27	0.73	[0.99, 0.32, 0.92, 0.7]	2.89E+04	3	0.92	[0.96, 0.82, 0.96, 0.94]	0.03	0.88	[0.91, 0.72, 0.96, 0.94]	0.03	3	3	0.91	[0.96, 0.8, 0.96, 0.92]	0.91	[0.96, 0.8, 0.96, 0.92]	3	3
1024 [41, 20, 17, 1, 17, 20, 41] 1024	0.04	0.28	0.72	[0.89, 0.38, 0.91, 0.72]	1.73E+05	1	0.15	[0.0, 0.0, 0.59, 0.0]	0.04	0.14	[0.0, 0.0, 0.58, 0.0]	0.05	4	1	0	[0.0, 0.0, 0.0, 0.0]	0.72	[0.92, 0.7, 0.84, 0.43]	1	41
1024 [7, 4, 3, 0, 3, 4, 7] 1024	0.04	0.31	0.69	[0.85, 0.55, 0.76, 0.62]	2.89E+04	3	0.96	[1.0, 0.91, 0.96, 0.96]	0.03	0.92	[0.97, 0.84, 0.95, 0.93]	0.03	4	3	0.92	[0.97, 0.82, 0.97, 0.91]	0.92	[0.97, 0.86, 0.97, 0.89]	1	7
1024 [7, 3, 2, 1, 2, 3, 7] 1024	0.04	0.31	0.69	[0.88, 0.32, 0.87, 0.68]	2.88E+04	1	0.15	[0.0, 0.0, 0.59, 0.0]	0.04	0.14	[0.0, 0.0, 0.58, 0.0]	0.05	4	1	0	[0.0, 0.0, 0.0, 0.0]	0.71	[0.89, 0.44, 0.83, 0.68]	1	7
1024 [5, 5, 5, 0, 5, 5, 5] 1024	0.04	0.31	0.69	[0.72, 0.44, 0.83, 0.75]	2.07E+04	5	0.76	[0.96, 0.37, 0.91, 0.78]	0.03	0.77	[0.97, 0.33, 0.93, 0.85]	0.03	1	5	0.92	[0.96, 0.84, 0.96, 0.93]	0.92	[0.96, 0.84, 0.96, 0.93]	1	5
1024 [5, 3, 2, 1, 2, 3, 5] 1024	0.04	0.38	0.62	[0.87, 0.25, 0.82, 0.54]	2.06E+04	1	0.15	[0.0, 0.0, 0.59, 0.0]	0.04	0.14	[0.0, 0.0, 0.58, 0.0]	0.05	4	1	0	[0.0, 0.0, 0.0, 0.0]	0.31	[0.38, 0.1, 0.48, 0.27]	1	5
1024 [4, 3, 3, 0, 3, 3, 4] 1024	0.04	0.4	0.6	[0.69, 0.42, 0.79, 0.52]	1.65E+04	3	0.76	[0.88, 0.42, 0.96, 0.77]	0.04	0.74	[0.88, 0.33, 0.95, 0.81]	0.04	2	3	0.68	[0.94, 0.11, 0.93, 0.73]	0.68	[0.94, 0.1, 0.93, 0.73]	2	3
1024 [5, 0, 3, 1, 3, 0, 5] 1024	0.04	0.4	0.6	[0.86, 0.21, 0.8, 0.52]	2.06E+04	1	0.15	[0.0, 0.0, 0.59, 0.0]	0.04	0.14	[0.0, 0.0, 0.58, 0.0]	0.05	3	1	0	[0.0, 0.0, 0.0, 0.0]	0.55	[0.71, 0.1, 0.8, 0.58]	1	5
1024 [4, 2, 1, 1, 1, 2, 4] 1024	0.04	0.41	0.59	[0.84, 0.09, 0.86, 0.58]	1.64E+04	1	0.15	[0.0, 0.0, 0.59, 0.0]	0.04	0.14	[0.0, 0.0, 0.58, 0.0]	0.05	4	1	0.09	[0.04, 0.0, 0.33, 0.0]	0.49	[0.67, 0.0, 0.7, 0.6]	1	4
1024 [1, 0, 0, 1, 0, 0, 1] 1024	0.04	0.54	0.46	[0.39, 0.1, 0.53, 0.84]	4.10E+03	1	0.15	[0.0, 0.0, 0.59, 0.0]	0.04	0.14	[0.0, 0.0, 0.58, 0.0]	0.05	2	1	0	[0.0, 0.0, 0.0, 0.0]	0.39	[0.3, 0.0, 0.72, 0.53]	1	1
1024 [1, 0, 0, 0, 0, 0, 1] 1024	0.04	0.56	0.44	[0.39, 0.46, 0.63, 0.27]	4.10E+03	1	0.15	[0.0, 0.0, 0.59, 0.0]	0.04	0.14	[0.0, 0.0, 0.58, 0.0]	0.05	1	1	0	[0.0, 0.0, 0.0, 0.0]	0	[0.0, 0.0, 0.0, 0.0]	1	1

A.2 EAE vs. Human-designed AEs

To have a detailed comparison between EAE and human-designed models, we have shown the non-dominated PF solutions from EAE based on the SAE method and for the human-designed models we applied the same SAE method with the same exact conditions. We have provided the results on both validation set and Test set on the middle layer (code layer is the index of the layer while code size is the number of neurons at the code layer).

Table 4.35 : Human-designed vs. EAE for the Supervised AE (SAE) scheme on Brain

Tumor Type	Brain	FLOPs	Classification Error (Test Set)	SAE Full Training on Middle Layer							
					Unsupervised #Epochs		SAE #Epochs			Batch size	
					200		100			4	
				On Validation Set			On Test Set			Middle Layer	
Approach	Structures			Averaged F1-score	F1-score per class	RMSE	Averaged F1-score	F1-score per class	RMSE	Code Layer	Code Size
Scheme 1: EAE with SAE on the middle layer	1024 [9, 6, 3, 1, 3, 6, 9] 1024	3.72E+04	0.09	0.92	[0.93, 0.92]	0.04	0.91	[0.91, 0.9]	0.04	4	1
	1024 [5, 3, 0, 3, 0, 3, 5] 1024	2.06E+04	0.05	0.95	[0.95, 0.94]	0.04	0.95	[0.95, 0.94]	0.04	3	3
	1024 [5, 3, 1, 0, 1, 3, 5] 1024	2.06E+04	0.11	0.96	[0.96, 0.96]	0.04	0.89	[0.89, 0.89]	0.04	3	1
	1024 [4, 2, 0, 0, 0, 2, 4] 1024	1.64E+04	0.07	0.93	[0.93, 0.94]	0.04	0.93	[0.94, 0.93]	0.04	2	2
	1024 [4, 1, 1, 0, 1, 1, 4] 1024	1.64E+04	0.15	0.95	[0.95, 0.94]	0.05	0.85	[0.85, 0.85]	0.05	2	1
	1024 [1, 0, 0, 0, 0, 0, 1] 1024	4.10E+03	0.68	0.32	[0.0, 0.64]	0.05	0.32	[0.0, 0.64]	0.05	1	1
Scheme 2: Division by 2	1024,[512, 256, 128, 64, 32, 10, 32, 64, 128, 256, 512],1024	2.79E+06	0.15	0.92	[0.93, 0.91]	0.04	0.85	[0.85, 0.86]	0.04	6	10
	1024,[512, 256, 128, 64, 32, 64, 128, 256, 512],1024	2.79E+06	0.14	0.92	[0.93, 0.91]	0.04	0.86	[0.86, 0.87]	0.04	5	32
	1024,[512, 256, 128, 64 ,128 ,256, 512],1024	2.79E+06	0.15	0.95	[0.95, 0.94]	0.03	0.85	[0.85, 0.86]	0.04	4	64
	1024,[512, 128, 32, 8 ,32 ,128, 512],1024	2.38E+06	0.12	0.93	[0.94, 0.93]	0.04	0.88	[0.88, 0.88]	0.04	4	8
	1024,[128, 32, 16 ,8 , 16, 32, 128],1024	5.43E+05	0.11	0.91	[0.92, 0.9]	0.03	0.89	[0.89, 0.9]	0.04	4	8
Scheme 3: 2/3 of the previous layer	1024,[683,455,303,202,135,90,60,40,27,18,27,40,60,90,135,202,303,455,683],1024	5.03E+06	0.14	0.92	[0.93, 0.92]	0.04	0.86	[0.86, 0.86]	0.04	10	18
	1024,[683,455,303,202,135,90,60,40,27,40,60,90,135,202,303,455,683],1024	5.03E+06	0.11	0.93	[0.94, 0.93]	0.04	0.89	[0.89, 0.89]	0.04	9	27
	1024,[683,455,303,202,135,90,60,40,60,90,135,202,303,455,683],1024	5.03E+06	0.14	0.93	[0.93, 0.94]	0.04	0.86	[0.86, 0.87]	0.04	8	40
	1024,[683,455,303,202,135,90,60,90,135,202,303,455,683],1024	5.02E+06	0.15	0.96	[0.96, 0.96]	0.04	0.85	[0.85, 0.86]	0.04	7	60
	1024,[683,455,303,202,135,90,135,202,303,455,683],1024	4.99E+06	0.09	0.96	[0.96, 0.96]	0.04	0.91	[0.91, 0.9]	0.04	6	90
	1024,[683,455,303,202,135,202,303,455,683],1024	4.95E+06	0.16	0.92	[0.93, 0.92]	0.04	0.84	[0.84, 0.83]	0.04	5	135
Scheme 4: [65]	1024,[651, 688, 91, 32, 46],1024	2.46E+06	0.14	0.95	[0.95, 0.94]	0.03	0.86	[0.86, 0.86]	0.03	3	91
Scheme 5: [78]	1024,[1244, 522, 261, 37, 261, 522, 1244],1024	8.28E+06	0.15	0.95	[0.95, 0.94]	0.04	0.85	[0.85, 0.85]	0.04	4	37
	1024,[1244, 522, 261, 44, 261, 522, 1244],1024	8.28E+06	0.16	0.95	[0.95, 0.95]	0.04	0.84	[0.83, 0.84]	0.04	4	44
	1024,[1244, 522, 261, 17, 261, 522, 1244],1024	8.26E+06	0.15	0.93	[0.94, 0.93]	0.04	0.85	[0.85, 0.86]	0.04	4	17
	1024,[1244, 522, 261, 9, 261, 522, 1244],1024	8.25E+06	0.11	0.93	[0.94, 0.93]	0.04	0.89	[0.89, 0.89]	0.04	4	9
	1024,[1244, 522, 261, 45, 261, 522, 1244],1024	8.28E+06	0.12	0.92	[0.93, 0.91]	0.04	0.88	[0.88, 0.88]	0.04	4	45
	1024,[1244, 522, 261, 47, 261, 522, 1244],1024	8.29E+06	0.15	0.92	[0.93, 0.92]	0.04	0.85	[0.85, 0.85]	0.04	4	47
	1024,[1244, 522, 261, 61, 261, 522, 1244],1024	8.30E+06	0.18	0.95	[0.95, 0.95]	0.04	0.82	[0.81, 0.84]	0.04	4	61
	1024,[1244, 522, 261, 36, 261, 522, 1244],1024	8.28E+06	0.18	0.95	[0.95, 0.94]	0.04	0.82	[0.82, 0.83]	0.04	4	36
	1024,[1244, 522, 261, 33, 261, 522, 1244],1024	8.27E+06	0.15	0.95	[0.95, 0.94]	0.04	0.85	[0.84, 0.86]	0.04	4	33
	1024,[1244, 522, 261, 26, 261, 522, 1244],1024	8.27E+06	0.16	0.95	[0.95, 0.94]	0.04	0.84	[0.83, 0.84]	0.04	4	26
	1024,[1244, 522, 261, 21, 261, 522, 1244],1024	8.26E+06	0.18	0.95	[0.95, 0.94]	0.04	0.82	[0.82, 0.83]	0.04	4	21

Table 4.36: Human-designed vs. EAE for the Supervised AE (SAE) scheme on Endocrine

Tumor Type	Endocrine	FLOPs	Classification Error (Test Set)	SAE Full Training on Middle Layer							
					Unsupervised #Epochs		SAE #Epochs			Batch size	
					200		100			4	
				On Validation Set			On Test Set			Middle Layer	
Approach	Structures			Averaged F1-score	F1-score per class	RMSE	Averaged F1-score	F1-score per class	RMSE	Code Layer	Code Size
Scheme 1: EAE with SAE on the middle layer	1024 [8, 7, 5, 2, 5, 7, 8] 1024	3.32E+04	0.22	0.89	[0.71, 1.0, 0.96]	0.04	0.78	[0.46, 0.91, 0.96]	0.04	4	2
	1024 [7, 6, 0, 3, 0, 6, 7] 1024	2.89E+04	0.08	0.97	[0.91, 1.0, 0.99]	0.04	0.92	[0.83, 0.93, 1.0]	0.04	3	3
	1024 [13, 4, 2, 1, 2, 4, 13] 1024	5.35E+04	0.11	0.96	[0.91, 0.97, 1.0]	0.05	0.89	[0.77, 0.9, 1.0]	0.05	4	1
	1024 [2, 1, 1, 0, 1, 1, 2] 1024	8.21E+03	0.42	0.67	[0.5, 0.5, 1.0]	0.05	0.58	[0.43, 0.32, 1.0]	0.05	2	1
	1024 [1, 0, 0, 0, 0, 0, 1] 1024	4.10E+03	0.72	0.28	[0.0, 0.0, 0.84]	0.05	0.28	[0.0, 0.0, 0.83]	0.05	2	1
Scheme 2: Division by 2	1024,[512, 256, 128, 64, 32, 10, 32, 64, 128, 256, 512],1024	2.79E+06	0.18	0.87	[0.77, 0.91, 0.94]	0.04	0.82	[0.6, 0.87, 1.0]	0.04	6	10
	1024,[512, 256, 128, 64, 32, 64, 128, 256, 512],1024	2.79E+06	0.12	1	[1.0, 1.0, 1.0]	0.03	0.88	[0.8, 0.86, 0.99]	0.03	5	32
	1024,[512, 256, 128, 64 ,128 ,256, 512],1024	2.79E+06	0.13	0.9	[0.77, 0.97, 0.98]	0.03	0.87	[0.77, 0.86, 0.99]	0.03	4	64
	1024,[512, 128, 32, 8 ,32 ,128, 512],1024	2.38E+06	0.22	0.97	[0.91, 1.0, 0.99]	0.03	0.78	[0.5, 0.85, 0.99]	0.03	4	8
	1024,[128, 32, 16 ,8 , 16, 32, 128],1024	5.43E+05	0.15	1	[1.0, 1.0, 1.0]	0.03	0.85	[0.67, 0.9, 0.99]	0.03	4	8
Scheme 3: 2/3 of the previous layer	1024,[683,455,303,202,135,90,60,40,27,18,27,40,60,90,135,202,303,455,683],1024	5.03E+06	0.08	0.96	[0.91, 0.97, 1.0]	0.04	0.92	[0.83, 0.93, 1.0]	0.04	10	18
	1024,[683,455,303,202,135,90,60,40,27,40,60,90,135,202,303,455,683],1024	5.03E+06	0.17	0.89	[0.83, 0.87, 0.98]	0.04	0.83	[0.67, 0.81, 1.0]	0.04	9	27
	1024,[683,455,303,202,135,90,60,40,60,90,135,202,303,455,683],1024	5.03E+06	0.21	0.86	[0.71, 0.9, 0.95]	0.04	0.79	[0.55, 0.85, 0.98]	0.04	8	40
	1024,[683,455,303,202,135,90,60,90,135,202,303,455,683],1024	5.02E+06	0.13	0.9	[0.77, 0.97, 0.98]	0.04	0.87	[0.71, 0.9, 0.99]	0.04	7	60
	1024,[683,455,303,202,135,90,135,202,303,455,683],1024	4.99E+06	0.16	0.84	[0.62, 0.93, 0.96]	0.04	0.84	[0.67, 0.86, 0.99]	0.03	6	90
	1024,[683,455,303,202,135,202,303,455,683],1024	4.95E+06	0.14	0.96	[1.0, 0.91, 0.97]	0.04	0.86	[0.75, 0.86, 0.98]	0.04	5	135
Scheme 4: [65]	1024,[651, 688, 91, 32, 46],1024	2.46E+06	0.17	1	[1.0, 1.0, 1.0]	0.03	0.83	[0.67, 0.81, 1.0]	0.03	3	91
Scheme 5: [78]	1024,[1244, 522, 261, 37, 261, 522, 1244],1024	8.28E+06	0.17	0.94	[0.83, 1.0, 0.98]	0.04	0.83	[0.67, 0.86, 0.97]	0.04	4	37
	1024,[1244, 522, 261, 44, 261, 522, 1244],1024	8.28E+06	0.11	0.89	[0.77, 0.93, 0.97]	0.04	0.89	[0.77, 0.9, 1.0]	0.03	4	44
	1024,[1244, 522, 261, 17, 261, 522, 1244],1024	8.26E+06	0.08	1	[1.0, 1.0, 1.0]	0.03	0.92	[0.84, 0.92, 1.0]	0.03	4	17
	1024,[1244, 522, 261, 9, 261, 522, 1244],1024	8.25E+06	0.16	1	[1.0, 1.0, 1.0]	0.04	0.84	[0.71, 0.83, 0.99]	0.04	4	9
	1024,[1244, 522, 261, 45, 261, 522, 1244],1024	8.28E+06	0.17	0.92	[0.83, 0.93, 1.0]	0.04	0.83	[0.71, 0.8, 0.98]	0.04	4	45
	1024,[1244, 522, 261, 47, 261, 522, 1244],1024	8.29E+06	0.09	0.94	[0.83, 1.0, 0.98]	0.03	0.91	[0.83, 0.9, 0.99]	0.04	4	47
	1024,[1244, 522, 261, 61, 261, 522, 1244],1024	8.30E+06	0.08	1	[1.0, 1.0, 1.0]	0.03	0.92	[0.83, 0.93, 1.0]	0.03	4	61
	1024,[1244, 522, 261, 36, 261, 522, 1244],1024	8.28E+06	0.13	0.9	[0.77, 0.97, 0.98]	0.04	0.87	[0.77, 0.86, 0.99]	0.03	4	36
	1024,[1244, 522, 261, 33, 261, 522, 1244],1024	8.27E+06	0.09	0.89	[0.77, 0.93, 0.97]	0.03	0.91	[0.83, 0.9, 0.99]	0.03	4	33
	1024,[1244, 522, 261, 26, 261, 522, 1244],1024	8.27E+06	0.08	0.97	[0.91, 1.0, 0.99]	0.04	0.92	[0.83, 0.93, 1.0]	0.03	4	26
	1024,[1244, 522, 261, 21, 261, 522, 1244],1024	8.26E+06	0.09	0.93	[0.83, 0.97, 0.99]	0.03	0.91	[0.83, 0.9, 0.99]	0.03	4	21

Table 4.37: Human-designed vs. EAE for the Supervised AE (SAE) scheme on Gastro

Tumor Type	Gastro	FLOPs	Classification Error (Test Set)	SAE Full Training on Middle Layer							
					Unsupervised #Epochs		SAE #Epochs			Batch size	
					200		100			4	
				On Validation Set			On Test Set			Middle Layer	
Approach	Structures			Averaged F1-score	F1-score per class	RMSE	Averaged F1-score	F1-score per class	RMSE	Code Layer	Code Size
Scheme 1: EAE with SAE on the middle layer	1024 [109, 37, 12, 4, 12, 37, 109] 1024	4.65E+05	0.3	0.68	[0.85, 0.18, 0.86, 0.84]	0.03	0.7	[0.78, 0.35, 0.83, 0.82]	0.03	4	4
	1024 [32, 24, 18, 5, 18, 24, 32] 1024	1.36E+05	0.31	0.76	[0.91, 0.46, 0.81, 0.86]	0.03	0.69	[0.82, 0.27, 0.83, 0.85]	0.03	4	5
	1024 [11, 6, 5, 1, 5, 6, 11] 1024	4.55E+04	0.47	0.62	[0.82, 0.0, 0.83, 0.83]	0.04	0.53	[0.71, 0.0, 0.64, 0.76]	0.04	4	1
	1024 [6, 3, 0, 2, 0, 3, 6] 1024	2.47E+04	0.43	0.63	[0.82, 0.0, 0.81, 0.88]	0.04	0.57	[0.76, 0.0, 0.72, 0.81]	0.04	3	2
	1024 [1, 0, 0, 0, 0, 0, 1] 1024	4.10E+03	0.87	0.14	[0.55, 0.0, 0.0, 0.0]	0.05	0.13	[0.53, 0.0, 0.0, 0.0]	0.05	1	1
Scheme 2: Division by 2	1024,[512, 256, 128, 64, 32, 10, 32, 64, 128, 256, 512],1024	2.79E+06	0.31	0.62	[0.81, 0.0, 0.86, 0.82]	0.03	0.69	[0.77, 0.33, 0.86, 0.81]	0.03	6	10
	1024,[512, 256, 128, 64, 32, 64, 128, 256, 512],1024	2.79E+06	0.39	0.63	[0.87, 0.0, 0.8, 0.86]	0.03	0.61	[0.79, 0.0, 0.83, 0.83]	0.03	5	32
	1024,[512, 256, 128, 64 ,128 ,256, 512],1024	2.79E+06	0.32	0.68	[0.86, 0.18, 0.81, 0.85]	0.03	0.68	[0.77, 0.32, 0.81, 0.83]	0.03	4	64
	1024,[512, 128, 32, 8 ,32 ,128, 512],1024	2.38E+06	0.37	0.64	[0.85, 0.0, 0.85, 0.87]	0.03	0.63	[0.75, 0.13, 0.81, 0.81]	0.03	4	8
	1024,[128, 32, 16 ,8 , 16, 32, 128],1024	5.43E+05	0.38	0.67	[0.84, 0.15, 0.83, 0.86]	0.03	0.62	[0.78, 0.13, 0.75, 0.83]	0.03	4	8
Scheme 3: 2/3 of the previous layer	1024,[683,455,303,202,135,90,60,40,27,18,27,40,60,90,135,202,303,455,683],1024	5.03E+06	0.42	0.64	[0.83, 0.0, 0.88, 0.87]	0.04	0.58	[0.78, 0.0, 0.75, 0.79]	0.04	10	18
	1024,[683,455,303,202,135,90,60,40,27,40,60,90,135,202,303,455,683],1024	5.03E+06	0.4	0.76	[0.85, 0.5, 0.89, 0.8]	0.04	0.6	[0.67, 0.11, 0.85, 0.79]	0.04	9	27
	1024,[683,455,303,202,135,90,60,40,60,90,135,202,303,455,683],1024	5.03E+06	0.34	0.73	[0.88, 0.31, 0.85, 0.88]	0.04	0.66	[0.77, 0.42, 0.71, 0.75]	0.04	8	40
	1024,[683,455,303,202,135,90,60,90,135,202,303,455,683],1024	5.02E+06	0.43	0.63	[0.87, 0.0, 0.79, 0.86]	0.04	0.57	[0.78, 0.0, 0.74, 0.76]	0.04	7	60
	1024,[683,455,303,202,135,90,135,202,303,455,683],1024	4.99E+06	0.41	0.72	[0.85, 0.31, 0.83, 0.88]	0.04	0.59	[0.74, 0.12, 0.69, 0.81]	0.04	6	90
	1024,[683,455,303,202,135,202,303,455,683],1024	4.95E+06	0.33	0.68	[0.85, 0.17, 0.87, 0.85]	0.03	0.67	[0.82, 0.44, 0.69, 0.73]	0.03	5	135
Scheme 4: [65]	1024,[651, 688, 91, 32, 46],1024	2.46E+06	0.35	0.66	[0.83, 0.0, 0.93, 0.86]	0.02	0.65	[0.76, 0.27, 0.76, 0.82]	0.03	3	91
Scheme 5: [78]	1024,[1244, 522, 261, 37, 261, 522, 1244],1024	8.28E+06	0.35	0.69	[0.84, 0.15, 0.88, 0.88]	0.03	0.65	[0.78, 0.13, 0.89, 0.81]	0.03	4	37
	1024,[1244, 522, 261, 44, 261, 522, 1244],1024	8.28E+06	0.37	0.68	[0.84, 0.15, 0.89, 0.84]	0.03	0.63	[0.75, 0.25, 0.76, 0.75]	0.03	4	44
	1024,[1244, 522, 261, 17, 261, 522, 1244],1024	8.26E+06	0.41	0.74	[0.87, 0.36, 0.86, 0.85]	0.03	0.59	[0.75, 0.13, 0.71, 0.75]	0.03	4	17
	1024,[1244, 522, 261, 9, 261, 522, 1244],1024	8.25E+06	0.42	0.65	[0.84, 0.2, 0.74, 0.83]	0.03	0.58	[0.72, 0.0, 0.77, 0.84]	0.03	4	9
	1024,[1244, 522, 261, 45, 261, 522, 1244],1024	8.28E+06	0.31	0.66	[0.79, 0.0, 0.93, 0.9]	0.03	0.69	[0.78, 0.42, 0.75, 0.82]	0.03	4	45
	1024,[1244, 522, 261, 47, 261, 522, 1244],1024	8.29E+06	0.34	0.63	[0.79, 0.0, 0.86, 0.87]	0.03	0.66	[0.77, 0.4, 0.69, 0.77]	0.03	4	47
	1024,[1244, 522, 261, 61, 261, 522, 1244],1024	8.30E+06	0.32	0.72	[0.83, 0.35, 0.83, 0.86]	0.03	0.68	[0.74, 0.42, 0.76, 0.79]	0.03	4	61
	1024,[1244, 522, 261, 36, 261, 522, 1244],1024	8.28E+06	0.34	0.64	[0.83, 0.0, 0.86, 0.86]	0.03	0.66	[0.8, 0.27, 0.77, 0.8]	0.04	4	36
	1024,[1244, 522, 261, 33, 261, 522, 1244],1024	8.27E+06	0.36	0.77	[0.84, 0.53, 0.86, 0.84]	0.03	0.64	[0.75, 0.24, 0.76, 0.8]	0.03	4	33
	1024,[1244, 522, 261, 26, 261, 522, 1244],1024	8.27E+06	0.3	0.65	[0.81, 0.0, 0.93, 0.87]	0.03	0.7	[0.79, 0.4, 0.79, 0.81]	0.04	4	26
	1024,[1244, 522, 261, 21, 261, 522, 1244],1024	8.26E+06	0.34	0.67	[0.77, 0.24, 0.84, 0.85]	0.03	0.66	[0.72, 0.4, 0.7, 0.84]	0.03	4	21

Table 4.38: Human-designed vs. EAE for the Supervised AE (SAE) scheme on Gynaeco

Tumor Type	Gynaeco	FLOPs	Classification Error (Test Set)	SAE Full Training on Middle Layer								
					Unsupervised #Epochs		SAE #Epochs			Batch size	
					200		100			4	
				On Validation Set			On Test Set			Middle Layer	
Approach	Structures			Averaged F1-score	F1-score per class	RMSE	Averaged F1-score	F1-score per class	RMSE	Code Layer	Code Size
Scheme 1: EAE with SAE on the middle layer	1024 [23, 20, 12, 5, 12, 20, 23] 1024	9.72E+04	0.06	1	[1.0, 1.0, 1.0]	0.05	0.94	[0.94, 0.89, 1.0]	0.06	4	5
	1024 [104, 33, 10, 3, 10, 33, 104] 1024	4.41E+05	0.06	1	[1.0, 1.0, 1.0]	0.05	0.94	[0.94, 0.89, 1.0]	0.06	4	3
	1024 [65, 42, 25, 2, 25, 42, 65] 1024	2.82E+05	0.2	0.94	[0.98, 1.0, 0.86]	0.05	0.8	[0.97, 0.75, 0.67]	0.06	4	2
	1024 [22, 13, 6, 0, 6, 13, 22] 1024	9.16E+04	0.07	1	[1.0, 1.0, 1.0]	0.05	0.93	[1.0, 0.95, 0.86]	0.06	3	6
	1024 [31, 20, 15, 2, 15, 20, 31] 1024	1.31E+05	0.21	1	[1.0, 1.0, 1.0]	0.05	0.79	[0.92, 0.95, 0.5]	0.06	4	2
	1024 [16, 12, 6, 3, 6, 12, 16] 1024	6.67E+04	0.12	0.94	[0.98, 1.0, 0.86]	0.05	0.88	[0.94, 0.84, 0.86]	0.06	4	3
	1024 [7, 6, 5, 4, 5, 6, 7] 1024	2.90E+04	0.13	0.81	[0.98, 0.78, 0.67]	0.06	0.87	[0.85, 0.76, 1.0]	0.07	4	4
	1024 [1, 0, 0, 0, 0, 0, 1] 1024	4.10E+03	0.31	0.86	[1.0, 0.9, 0.67]	0.07	0.69	[0.92, 0.82, 0.33]	0.07	1	1
Scheme 2: Division by 2	1024,[512, 256, 128, 64, 32, 10, 32, 64, 128, 256, 512],1024	2.79E+06	0.24	0.94	[0.98, 1.0, 0.86]	0.05	0.76	[0.8, 0.73, 0.75]	0.06	6	10
	1024,[512, 256, 128, 64, 32, 64, 128, 256, 512],1024	2.79E+06	0.21	1	[1.0, 1.0, 1.0]	0.05	0.79	[0.86, 0.82, 0.67]	0.06	5	32
	1024,[512, 256, 128, 64 ,128 ,256, 512],1024	2.79E+06	0.17	0.94	[0.98, 1.0, 0.86]	0.05	0.83	[0.91, 0.84, 0.75]	0.06	4	64
	1024,[512, 128, 32, 8 ,32 ,128, 512],1024	2.38E+06	0.18	0.98	[0.98, 0.95, 1.0]	0.05	0.82	[0.86, 0.75, 0.86]	0.06	4	8
	1024,[128, 32, 16 ,8 , 16, 32, 128],1024	5.43E+05	0.24	0.89	[0.98, 0.95, 0.75]	0.05	0.76	[0.97, 0.75, 0.55]	0.06	4	8
Scheme 3: 2/3 of the previous layer	1024,[683,455,303,202,135,90,60,40,27,18,27,40,60,90,135,202,303,455,683],1024	5.03E+06	0.14	0.94	[0.98, 1.0, 0.86]	0.05	0.86	[0.89, 0.82, 0.86]	0.06	10	18
	1024,[683,455,303,202,135,90,60,40,27,40,60,90,135,202,303,455,683],1024	5.03E+06	0.27	0.98	[0.98, 0.95, 1.0]	0.05	0.73	[0.87, 0.82, 0.5]	0.06	9	27
	1024,[683,455,303,202,135,90,60,40,60,90,135,202,303,455,683],1024	5.03E+06	0.2	0.94	[0.98, 1.0, 0.86]	0.06	0.8	[0.88, 0.95, 0.57]	0.07	8	40
	1024,[683,455,303,202,135,90,60,90,135,202,303,455,683],1024	5.02E+06	0.13	0.94	[0.98, 1.0, 0.86]	0.05	0.87	[0.92, 0.89, 0.8]	0.06	7	60
	1024,[683,455,303,202,135,90,135,202,303,455,683],1024	4.99E+06	0.09	0.9	[0.96, 0.95, 0.8]	0.05	0.91	[0.92, 0.82, 1.0]	0.06	6	90
	1024,[683,455,303,202,135,202,303,455,683],1024	4.95E+06	0.25	1	[1.0, 1.0, 1.0]	0.05	0.75	[0.91, 0.78, 0.57]	0.06	5	135
Scheme 4: [65]	1024,[651, 688, 91, 32, 46],1024	2.46E+06	0.18	1	[1.0, 1.0, 1.0]	0.04	0.82	[0.91, 0.89, 0.67]	0.05	3	91
Scheme 5: [78]	1024,[1244, 522, 261, 37, 261, 522, 1244],1024	8.28E+06	0.16	1	[1.0, 1.0, 1.0]	0.05	0.84	[0.89, 0.82, 0.8]	0.06	4	37
	1024,[1244, 522, 261, 44, 261, 522, 1244],1024	8.28E+06	0.08	1	[1.0, 1.0, 1.0]	0.05	0.92	[0.91, 0.84, 1.0]	0.06	4	44
	1024,[1244, 522, 261, 17, 261, 522, 1244],1024	8.26E+06	0.21	1	[1.0, 1.0, 1.0]	0.05	0.79	[0.86, 0.82, 0.67]	0.06	4	17
	1024,[1244, 522, 261, 9, 261, 522, 1244],1024	8.25E+06	0.32	0.94	[0.98, 1.0, 0.86]	0.05	0.68	[0.71, 0.89, 0.43]	0.06	4	9
	1024,[1244, 522, 261, 45, 261, 522, 1244],1024	8.28E+06	0.18	0.98	[0.98, 0.95, 1.0]	0.05	0.82	[0.86, 0.75, 0.86]	0.06	4	45
	1024,[1244, 522, 261, 47, 261, 522, 1244],1024	8.29E+06	0.22	0.94	[0.98, 1.0, 0.86]	0.05	0.78	[0.91, 0.78, 0.67]	0.06	4	47
	1024,[1244, 522, 261, 61, 261, 522, 1244],1024	8.30E+06	0.17	0.94	[0.98, 1.0, 0.86]	0.05	0.83	[0.91, 0.84, 0.75]	0.06	4	61
	1024,[1244, 522, 261, 36, 261, 522, 1244],1024	8.28E+06	0.21	1	[1.0, 1.0, 1.0]	0.05	0.79	[0.86, 0.82, 0.67]	0.06	4	36
	1024,[1244, 522, 261, 33, 261, 522, 1244],1024	8.27E+06	0.18	0.92	[0.95, 0.95, 0.86]	0.05	0.82	[0.86, 0.75, 0.86]	0.06	4	33
	1024,[1244, 522, 261, 26, 261, 522, 1244],1024	8.27E+06	0.21	1	[1.0, 1.0, 1.0]	0.05	0.79	[0.86, 0.82, 0.67]	0.06	4	26
	1024,[1244, 522, 261, 21, 261, 522, 1244],1024	8.26E+06	0.07	1	[1.0, 1.0, 1.0]	0.05	0.93	[0.94, 0.86, 1.0]	0.06	4	21

Table 4.39: Human-designed vs. EAE for the Supervised AE (SAE) scheme on Liver

Tumor Type	Liver	FLOPs	Classification Error (Test Set)	SAE Full Training on Middle Layer							
					Unsupervised #Epochs		SAE #Epochs			Batch size	
					200		100			4	
				On Validation Set			On Test Set			Middle Layer	
Approach	Structures			Averaged F1-score	F1-score per class	RMSE	Averaged F1-score	F1-score per class	RMSE	Code Layer	Code Size
Scheme 1: EAE with SAE on the middle layer	1024 [80, 40, 14, 4, 14, 40, 80] 1024	3.43E+05	0.13	0.81	[0.57, 0.93, 0.92]	0.03	0.87	[0.73, 0.96, 0.92]	0.04	4	4
	1024 [16, 8, 5, 4, 5, 8, 16] 1024	6.63E+04	0.16	0.86	[0.67, 0.94, 0.96]	0.03	0.84	[0.62, 0.94, 0.96]	0.04	4	4
	1024 [13, 11, 6, 2, 6, 11, 13] 1024	5.41E+04	0.21	0.86	[0.67, 0.96, 0.96]	0.04	0.79	[0.5, 0.99, 0.87]	0.04	4	2
	1024 [1, 0, 0, 0, 0, 0, 1] 1024	4.10E+03	0.28	0.86	[0.67, 0.97, 0.96]	0.05	0.72	[0.29, 0.96, 0.92]	0.05	1	1
Scheme 2: Division by 2	1024,[512, 256, 128, 64, 32, 10, 32, 64, 128, 256, 512],1024	2.79E+06	0.17	0.8	[0.5, 0.93, 0.96]	0.04	0.83	[0.62, 0.96, 0.91]	0.04	6	10
	1024,[512, 256, 128, 64, 32, 64, 128, 256, 512],1024	2.79E+06	0.29	0.86	[0.67, 0.94, 0.96]	0.03	0.71	[0.43, 0.88, 0.83]	0.04	5	32
	1024,[512, 256, 128, 64 ,128 ,256, 512],1024	2.79E+06	0.15	0.89	[0.75, 0.96, 0.96]	0.03	0.85	[0.67, 0.96, 0.92]	0.03	4	64
	1024,[512, 128, 32, 8 ,32 ,128, 512],1024	2.38E+06	0.14	0.89	[0.75, 0.96, 0.96]	0.03	0.86	[0.67, 0.99, 0.92]	0.04	4	8
	1024,[128, 32, 16 ,8 , 16, 32, 128],1024	5.43E+05	0.15	0.89	[0.75, 0.96, 0.96]	0.03	0.85	[0.67, 0.96, 0.92]	0.03	4	8
Scheme 3: 2/3 of the previous layer	1024,[683,455,303,202,135,90,60,40,27,18,27,40,60,90,135,202,303,455,683],1024	5.03E+06	0.25	0.86	[0.67, 0.96, 0.96]	0.04	0.75	[0.4, 0.97, 0.89]	0.04	10	18
	1024,[683,455,303,202,135,90,60,40,27,40,60,90,135,202,303,455,683],1024	5.03E+06	0.3	0.81	[0.5, 0.94, 1.0]	0.04	0.7	[0.33, 0.94, 0.82]	0.05	9	27
	1024,[683,455,303,202,135,90,60,40,60,90,135,202,303,455,683],1024	5.03E+06	0.11	0.77	[0.57, 0.9, 0.86]	0.04	0.89	[0.73, 0.96, 1.0]	0.04	8	40
	1024,[683,455,303,202,135,90,60,90,135,202,303,455,683],1024	5.02E+06	0.17	0.84	[0.57, 0.96, 1.0]	0.04	0.83	[0.62, 0.96, 0.91]	0.04	7	60
	1024,[683,455,303,202,135,90,135,202,303,455,683],1024	4.99E+06	0.4	0.84	[0.67, 0.94, 0.92]	0.03	0.6	[0.0, 0.99, 0.83]	0.04	6	90
	1024,[683,455,303,202,135,202,303,455,683],1024	4.95E+06	0.21	0.82	[0.57, 0.94, 0.96]	0.03	0.79	[0.6, 0.94, 0.82]	0.04	5	135
Scheme 4: [65]	1024,[651, 688, 91, 32, 46],1024	2.46E+06	0.28	0.83	[0.67, 0.93, 0.89]	0.02	0.72	[0.36, 0.91, 0.88]	0.03	3	91
Scheme 5: [78]	1024,[1244, 522, 261, 37, 261, 522, 1244],1024	8.28E+06	0.28	0.84	[0.67, 0.94, 0.92]	0.03	0.72	[0.33, 0.97, 0.86]	0.03	4	37
	1024,[1244, 522, 261, 44, 261, 522, 1244],1024	8.28E+06	0.21	0.89	[0.75, 0.96, 0.96]	0.03	0.79	[0.55, 0.94, 0.87]	0.04	4	44
	1024,[1244, 522, 261, 17, 261, 522, 1244],1024	8.26E+06	0.25	0.85	[0.6, 0.94, 1.0]	0.03	0.75	[0.5, 0.91, 0.83]	0.04	4	17
	1024,[1244, 522, 261, 9, 261, 522, 1244],1024	8.25E+06	0.11	0.91	[0.75, 0.97, 1.0]	0.03	0.89	[0.73, 0.96, 1.0]	0.04	4	9
	1024,[1244, 522, 261, 45, 261, 522, 1244],1024	8.28E+06	0.17	0.81	[0.57, 0.93, 0.92]	0.03	0.83	[0.67, 0.94, 0.87]	0.04	4	45
	1024,[1244, 522, 261, 47, 261, 522, 1244],1024	8.29E+06	0.41	0.85	[0.67, 0.97, 0.92]	0.03	0.59	[0.0, 0.96, 0.81]	0.03	4	47
	1024,[1244, 522, 261, 61, 261, 522, 1244],1024	8.30E+06	0.18	0.84	[0.57, 0.96, 1.0]	0.03	0.82	[0.67, 0.97, 0.82]	0.03	4	61
	1024,[1244, 522, 261, 36, 261, 522, 1244],1024	8.28E+06	0.17	0.84	[0.57, 0.96, 1.0]	0.03	0.83	[0.67, 0.99, 0.83]	0.03	4	36
	1024,[1244, 522, 261, 33, 261, 522, 1244],1024	8.27E+06	0.13	0.87	[0.75, 0.94, 0.92]	0.03	0.87	[0.73, 0.96, 0.92]	0.04	4	33
	1024,[1244, 522, 261, 26, 261, 522, 1244],1024	8.27E+06	0.26	0.84	[0.57, 0.96, 1.0]	0.03	0.74	[0.44, 0.96, 0.83]	0.04	4	26
	1024,[1244, 522, 261, 21, 261, 522, 1244],1024	8.26E+06	0.16	0.82	[0.57, 0.94, 0.96]	0.03	0.84	[0.67, 0.94, 0.91]	0.03	4	21

Table 4.40: Human-designed vs. EAE for the Supervised AE (SAE) scheme on Melanocytic

Tumor Type	Melanocytic	FLOPs	Classification Error (Test Set)	SAE Full Training on Middle Layer							
					Unsupervised #Epochs		SAE #Epochs			Batch size	
					200		100			4	
				On Validation Set			On Test Set			Middle Layer	
Approach	Structures			Averaged F1-score	F1-score per class	RMSE	Averaged F1-score	F1-score per class	RMSE	Code Layer	Code Size
Scheme 1: **EAE with SAE on the middle layer**	1024 [1, 0, 0, 0, 0, 0, 1] 1024	4.10E+03	0.15	1	[1.0, 1.0]	0.06	0.85	[0.75, 0.96]	0.06	1	1
Scheme 2: **Division by 2**	1024,[512, 256, 128, 64, 32, 10, 32, 64, 128, 256, 512],1024	2.79E+06	0.33	0.92	[0.86, 0.98]	0.06	0.67	[0.44, 0.89]	0.06	6	10
	1024,[512, 256, 128, 64, 32, 64, 128, 256, 512],1024	2.79E+06	0.24	0.92	[0.86, 0.98]	0.06	0.76	[0.6, 0.91]	0.05	5	32
	1024,[512, 256, 128, 64 ,128 ,256, 512],1024	2.79E+06	0.35	1	[1.0, 1.0]	0.05	0.65	[0.46, 0.84]	0.06	4	64
	1024,[512, 128, 32, 8 ,32 ,128, 512],1024	2.38E+06	0.15	1	[1.0, 1.0]	0.06	0.85	[0.75, 0.96]	0.05	4	8
	1024,[128, 32, 16 ,8 , 16, 32, 128],1024	5.43E+05	0.24	1	[1.0, 1.0]	0.06	0.76	[0.6, 0.91]	0.06	4	8
Scheme 3: **2/3 of the previous layer**	1024,[683,455,303,202,135,90,60,40,27,18,27,40,60,90,135,202,303,455,683],1024	5.03E+06	0.2	1	[1.0, 1.0]	0.06	0.8	[0.67, 0.94]	0.06	10	18
	1024,[683,455,303,202,135,90,60,40,27,40,60,90,135,202,303,455,683],1024	5.03E+06	0.42	1	[1.0, 1.0]	0.06	0.58	[0.33, 0.82]	0.06	9	27
	1024,[683,455,303,202,135,90,60,40,60,90,135,202,303,455,683],1024	5.03E+06	0.15	1	[1.0, 1.0]	0.06	0.85	[0.75, 0.96]	0.06	8	40
	1024,[683,455,303,202,135,90,60,90,135,202,303,455,683],1024	5.02E+06	0.15	1	[1.0, 1.0]	0.06	0.85	[0.75, 0.96]	0.06	7	60
	1024,[683,455,303,202,135,90,135,202,303,455,683],1024	4.99E+06	0.15	1	[1.0, 1.0]	0.06	0.85	[0.75, 0.96]	0.06	6	90
	1024,[683,455,303,202,135,202,303,455,683],1024	4.95E+06	0.19	1	[1.0, 1.0]	0.06	0.81	[0.67, 0.96]	0.06	5	135
Scheme 4: **[65]**	1024,[651, 688, 91, 32, 46],1024	2.46E+06	0.29	1	[1.0, 1.0]	0.04	0.71	[0.5, 0.92]	0.05	3	91
Scheme 5: **[78]**	1024,[1244, 522, 261, 37, 261, 522, 1244],1024	8.28E+06	0.2	1	[1.0, 1.0]	0.06	0.8	[0.67, 0.94]	0.05	4	37
	1024,[1244, 522, 261, 44, 261, 522, 1244],1024	8.28E+06	0.32	1	[1.0, 1.0]	0.06	0.68	[0.5, 0.86]	0.05	4	44
	1024,[1244, 522, 261, 17, 261, 522, 1244],1024	8.26E+06	0.24	1	[1.0, 1.0]	0.06	0.76	[0.57, 0.94]	0.06	4	17
	1024,[1244, 522, 261, 9, 261, 522, 1244],1024	8.25E+06	0.24	1	[1.0, 1.0]	0.06	0.76	[0.6, 0.91]	0.06	4	9
	1024,[1244, 522, 261, 45, 261, 522, 1244],1024	8.28E+06	0.24	1	[1.0, 1.0]	0.06	0.76	[0.6, 0.91]	0.05	4	45
	1024,[1244, 522, 261, 47, 261, 522, 1244],1024	8.29E+06	0.32	1	[1.0, 1.0]	0.06	0.68	[0.5, 0.86]	0.06	4	47
	1024,[1244, 522, 261, 61, 261, 522, 1244],1024	8.30E+06	0.2	1	[1.0, 1.0]	0.06	0.8	[0.67, 0.94]	0.05	4	61
	1024,[1244, 522, 261, 36, 261, 522, 1244],1024	8.28E+06	0.2	1	[1.0, 1.0]	0.06	0.8	[0.67, 0.94]	0.05	4	36
	1024,[1244, 522, 261, 33, 261, 522, 1244],1024	8.27E+06	0.2	1	[1.0, 1.0]	0.06	0.8	[0.67, 0.94]	0.05	4	33
	1024,[1244, 522, 261, 26, 261, 522, 1244],1024	8.27E+06	0.15	1	[1.0, 1.0]	0.05	0.85	[0.75, 0.96]	0.06	4	26
	1024,[1244, 522, 261, 21, 261, 522, 1244],1024	8.26E+06	0.2	1	[1.0, 1.0]	0.06	0.8	[0.67, 0.94]	0.06	4	21

Table 4.41: Human-designed vs. EAE for the Supervised AE (SAE) scheme on Prostate

Tumor Type	Prostate	FLOPs	Classification Error (Test Set)	SAE Full Training on Middle Layer								
					Unsupervised #Epochs		SAE #Epochs			Batch size	
					200		100			4	
				On Validation Set			On Test Set			Middle Layer	
Approach	Structures			Averaged F1-score	F1-score per class	RMSE	Averaged F1-score	F1-score per class	RMSE	Code Layer	Code Size
Scheme 1: EAE with SAE on the middle layer	1024 [5, 4, 2, 1, 2, 4, 5] 1024	2.06E+04	0	0.97	[0.99, 0.96]	0.06	1	[1.0, 1.0]	0.06	4	1
	1024 [1, 0, 0, 0, 0, 0, 1] 1024	4.10E+03	0.57	0.43	[0.85, 0.0]	0.06	0.43	[0.86, 0.0]	0.06	1	1
Scheme 2: Division by 2	1024,[512, 256, 128, 64, 32, 10, 32, 64, 128, 256, 512],1024	2.79E+06	0	1	[1.0, 1.0]	0.04	1	[1.0, 1.0]	0.04	6	10
	1024,[512, 256, 128, 64, 32, 64, 128, 256, 512],1024	2.79E+06	0	0.97	[0.99, 0.96]	0.04	1	[1.0, 1.0]	0.04	5	32
	1024,[512, 256, 128, 64 ,128 ,256, 512],1024	2.79E+06	0	1	[1.0, 1.0]	0.04	1	[1.0, 1.0]	0.04	4	64
	1024,[512, 128, 32, 8 ,32 , 128, 512],1024	2.38E+06	0	1	[1.0, 1.0]	0.04	1	[1.0, 1.0]	0.04	4	8
	1024,[128, 32, 16 ,8 , 16, 32, 128],1024	5.43E+05	0.07	0.95	[0.97, 0.93]	0.04	0.93	[0.96, 0.9]	0.04	4	8
Scheme 3: 2/3 of the previous layer	1024,[683,455,303,202,135,90,60,40,27,18,27,40,60,90,135,202,303,455,683],1024	5.03E+06	0.17	0.97	[0.99, 0.96]	0.05	0.83	[0.89, 0.76]	0.05	10	18
	1024,[683,455,303,202,135,90,60,40,27,40,60,90,135,202,303,455,683],1024	5.03E+06	0.09	0.97	[0.99, 0.96]	0.04	0.91	[0.95, 0.87]	0.05	9	27
	1024,[683,455,303,202,135,90,60,40,60,90,135,202,303,455,683],1024	5.03E+06	0	0.95	[0.97, 0.92]	0.05	1	[1.0, 1.0]	0.05	8	40
	1024,[683,455,303,202,135,90,60,90,135,202,303,455,683],1024	5.02E+06	0	0.97	[0.99, 0.96]	0.04	1	[1.0, 1.0]	0.04	7	60
	1024,[683,455,303,202,135,90,135,202,303,455,683],1024	4.99E+06	0.05	0.97	[0.99, 0.96]	0.04	0.95	[0.97, 0.93]	0.04	6	90
	1024,[683,455,303,202,135,202,303,455,683],1024	4.95E+06	0.17	0.95	[0.97, 0.92]	0.04	0.83	[0.89, 0.76]	0.05	5	135
Scheme 4: [65]	1024,[651, 688, 91, 32, 46],1024	2.46E+06	0.02	0.97	[0.99, 0.96]	0.03	0.98	[0.99, 0.96]	0.03	3	91
Scheme 5: [78]	1024,[1244, 522, 261, 37, 261, 522, 1244],1024	8.28E+06	0.02	0.97	[0.99, 0.96]	0.04	0.98	[0.99, 0.96]	0.04	4	37
	1024,[1244, 522, 261, 44, 261, 522, 1244],1024	8.28E+06	0	0.95	[0.97, 0.92]	0.04	1	[1.0, 1.0]	0.04	4	44
	1024,[1244, 522, 261, 17, 261, 522, 1244],1024	8.26E+06	0	1	[1.0, 1.0]	0.04	1	[1.0, 1.0]	0.04	4	17
	1024,[1244, 522, 261, 9, 261, 522, 1244],1024	8.25E+06	0	0.97	[0.99, 0.96]	0.04	1	[1.0, 1.0]	0.05	4	9
	1024,[1244, 522, 261, 45, 261, 522, 1244],1024	8.28E+06	0	0.97	[0.99, 0.96]	0.04	1	[1.0, 1.0]	0.04	4	45
	1024,[1244, 522, 261, 47, 261, 522, 1244],1024	8.29E+06	0	0.95	[0.97, 0.92]	0.04	1	[1.0, 1.0]	0.04	4	47
	1024,[1244, 522, 261, 61, 261, 522, 1244],1024	8.30E+06	0	0.97	[0.99, 0.96]	0.04	1	[1.0, 1.0]	0.04	4	61
	1024,[1244, 522, 261, 36, 261, 522, 1244],1024	8.28E+06	0	1	[1.0, 1.0]	0.04	1	[1.0, 1.0]	0.04	4	36
	1024,[1244, 522, 261, 33, 261, 522, 1244],1024	8.27E+06	0.07	0.95	[0.97, 0.93]	0.04	0.93	[0.96, 0.9]	0.04	4	33
	1024,[1244, 522, 261, 26, 261, 522, 1244],1024	8.27E+06	0.05	0.97	[0.99, 0.96]	0.04	0.95	[0.97, 0.93]	0.04	4	26
	1024,[1244, 522, 261, 21, 261, 522, 1244],1024	8.26E+06	0	1	[1.0, 1.0]	0.04	1	[1.0, 1.0]	0.04	4	21

Table 4.42: Human-designed vs. EAE for the Supervised AE (SAE) scheme on Pulmonary

Tumor Type	Pulmonary	FLOPs	Classification Error (Test Set)	SAE Full Training on Middle Layer								
					Unsupervised #Epochs		SAE #Epochs			Batch size		
					200		100			4		
				On Validation Set			On Test Set			Middle Layer		
Approach	Structures			Averaged F1-score	F1-score per class	RMSE	Averaged F1-score	F1-score per class	RMSE	Code Layer	Code Size
Scheme 1: EAE with SAE on the middle layer	1024 [107, 46, 15, 4, 15, 46, 107] 1024	4.61E+05	0.08	0.91	[0.96, 0.94, 0.83]	0.04	0.92	[0.88, 0.88, 1.0]	0.04	4	4
	1024 [33, 26, 18, 5, 18, 26, 33] 1024	1.41E+05	0.08	0.97	[0.94, 0.95, 1.0]	0.04	0.92	[0.88, 0.88, 1.0]	0.04	4	5
	1024 [28, 8, 0, 4, 0, 8, 28] 1024	1.16E+05	0.32	0.83	[0.94, 0.89, 0.67]	0.04	0.68	[0.84, 0.74, 0.47]	0.04	3	4
	1024 [128, 35, 14, 2, 14, 35, 128] 1024	5.44E+05	0.1	0.93	[0.89, 0.9, 1.0]	0.04	0.9	[0.85, 0.86, 1.0]	0.04	4	2
	1024 [31, 12, 8, 2, 8, 12, 31] 1024	1.29E+05	0.15	0.86	[0.89, 0.92, 0.77]	0.06	0.85	[0.84, 0.88, 0.83]	0.05	4	2
	1024 [25, 21, 13, 7, 13, 21, 25] 1024	1.06E+05	0.15	0.8	[0.89, 0.91, 0.59]	0.04	0.85	[0.82, 0.81, 0.91]	0.04	4	7
	1024 [23, 10, 4, 0, 4, 10, 23] 1024	9.53E+04	0.42	0.67	[0.45, 0.73, 0.83]	0.05	0.58	[0.42, 0.72, 0.6]	0.04	3	4
	1024 [1, 0, 0, 0, 0, 0, 1] 1024	4.10E+03	0.78	0.22	[0.0, 0.66, 0.0]	0.05	0.22	[0.0, 0.67, 0.0]	0.04	1	1
Scheme 2: Division by 2	1024,[512, 256, 128, 64, 32, 10, 32, 64, 128, 256, 512],1024	2.79E+06	0.14	0.91	[0.92, 0.92, 0.91]	0.04	0.86	[0.83, 0.86, 0.89]	0.04	6	10
	1024,[512, 256, 128, 64, 32, 64, 128, 256, 512],1024	2.79E+06	0.21	0.81	[0.92, 0.9, 0.62]	0.04	0.79	[0.84, 0.81, 0.71]	0.04	5	32
	1024,[512, 256, 128, 64 ,128 ,256, 512],1024	2.79E+06	0.1	0.91	[0.93, 0.96, 0.83]	0.04	0.9	[0.87, 0.91, 0.91]	0.03	4	64
	1024,[512, 128, 32, 8 ,32 ,128, 512],1024	2.38E+06	0.17	0.91	[0.94, 0.95, 0.83]	0.04	0.83	[0.83, 0.87, 0.77]	0.03	4	8
	1024,[128, 32, 16 ,8 , 16, 32, 128],1024	5.43E+05	0.21	0.94	[0.96, 0.95, 0.91]	0.04	0.79	[0.78, 0.77, 0.83]	0.04	4	8
Scheme 3: 2/3 of the previous layer	1024,[683,455,303,202,135,90,60,40,27,18,27,40,60,90,135,202,303,455,683],1024	5.03E+06	0.15	0.93	[0.95, 0.94, 0.91]	0.05	0.85	[0.82, 0.84, 0.89]	0.04	10	18
	1024,[683,455,303,202,135,90,60,40,27,40,60,90,135,202,303,455,683],1024	5.03E+06	0.08	0.97	[0.96, 0.96, 1.0]	0.05	0.92	[0.88, 0.88, 1.0]	0.04	9	27
	1024,[683,455,303,202,135,90,60,40,60,90,135,202,303,455,683],1024	5.03E+06	0.09	0.97	[0.95, 0.95, 1.0]	0.05	0.91	[0.86, 0.86, 1.0]	0.04	8	40
	1024,[683,455,303,202,135,90,60,90,135,202,303,455,683],1024	5.02E+06	0.09	0.87	[0.92, 0.91, 0.77]	0.05	0.91	[0.86, 0.86, 1.0]	0.05	7	60
	1024,[683,455,303,202,135,90,135,202,303,455,683],1024	4.99E+06	0.12	0.84	[0.9, 0.89, 0.71]	0.05	0.88	[0.8, 0.85, 1.0]	0.05	6	90
	1024,[683,455,303,202,135,202,303,455,683],1024	4.95E+06	0.18	0.9	[0.95, 0.93, 0.83]	0.05	0.82	[0.83, 0.85, 0.77]	0.04	5	135
Scheme 4: [65]	1024,[651, 688, 91, 32, 46],1024	2.46E+06	0.16	0.97	[0.95, 0.95, 1.0]	0.03	0.84	[0.86, 0.87, 0.77]	0.03	3	91
Scheme 5: [78]	1024,[1244, 522, 261, 37, 261, 522, 1244],1024	8.28E+06	0.09	0.96	[0.93, 0.94, 1.0]	0.04	0.91	[0.86, 0.87, 1.0]	0.03	4	37
	1024,[1244, 522, 261, 44, 261, 522, 1244],1024	8.28E+06	0.16	0.86	[0.93, 0.94, 0.71]	0.04	0.84	[0.85, 0.85, 0.83]	0.04	4	44
	1024,[1244, 522, 261, 17, 261, 522, 1244],1024	8.26E+06	0.13	0.82	[0.93, 0.91, 0.62]	0.04	0.87	[0.84, 0.85, 0.91]	0.03	4	17
	1024,[1244, 522, 261, 9, 261, 522, 1244],1024	8.25E+06	0.17	0.81	[0.92, 0.9, 0.62]	0.04	0.83	[0.86, 0.87, 0.77]	0.04	4	9
	1024,[1244, 522, 261, 45, 261, 522, 1244],1024	8.28E+06	0.14	0.96	[0.93, 0.94, 1.0]	0.04	0.86	[0.83, 0.86, 0.89]	0.03	4	45
	1024,[1244, 522, 261, 47, 261, 522, 1244],1024	8.29E+06	0.11	0.87	[0.96, 0.94, 0.71]	0.04	0.89	[0.84, 0.84, 1.0]	0.04	4	47
	1024,[1244, 522, 261, 61, 261, 522, 1244],1024	8.30E+06	0.18	0.85	[0.91, 0.93, 0.71]	0.05	0.82	[0.82, 0.86, 0.77]	0.04	4	61
	1024,[1244, 522, 261, 36, 261, 522, 1244],1024	8.28E+06	0.11	0.94	[0.96, 0.95, 0.91]	0.04	0.89	[0.82, 0.85, 1.0]	0.04	4	36
	1024,[1244, 522, 261, 33, 261, 522, 1244],1024	8.27E+06	0.15	0.87	[0.92, 0.91, 0.77]	0.04	0.85	[0.82, 0.82, 0.91]	0.03	4	33
	1024,[1244, 522, 261, 26, 261, 522, 1244],1024	8.27E+06	0.09	0.87	[0.88, 0.91, 0.83]	0.04	0.91	[0.84, 0.88, 1.0]	0.04	4	26
	1024,[1244, 522, 261, 21, 261, 522, 1244],1024	8.26E+06	0.13	0.86	[0.94, 0.93, 0.71]	0.04	0.87	[0.9, 0.88, 0.83]	0.04	4	21

Table 4.43: Human-designed vs. EAE for the Supervised AE (SAE) scheme on Urinary Tract

Tumor Type	Urinary Tract	FLOPs	Classification Error (Test Set)	SAE Full Training on Middle Layer								
					Unsupervised #Epochs		SAE #Epochs			Batch size	
					200		100			4	
				On Validation Set			On Test Set			Middle Layer	
Approach	Structures			Averaged F1-score	F1-score per class	RMSE	Averaged F1-score	F1-score per class	RMSE	Code Layer	Code Size
Scheme 1: EAE with SAE on the middle layer	1024 [53, 29, 21, 16, 21, 29, 53] 1024	2.27E+05	0.06	0.96	[1.0, 0.91, 0.96, 0.96]	0.02	0.94	[0.96, 0.95, 0.95, 0.91]	0.03	4	16
	1024 [20, 19, 10, 7, 10, 19, 20] 1024	8.45E+04	0.07	0.96	[0.99, 0.95, 0.97, 0.94]	0.03	0.93	[0.94, 0.9, 0.96, 0.93]	0.03	4	7
	1024 [28, 16, 8, 5, 8, 16, 28] 1024	1.17E+05	0.07	0.97	[0.97, 1.0, 0.97, 0.94]	0.03	0.93	[0.94, 0.91, 0.97, 0.91]	0.03	4	5
	1024 [34, 13, 9, 3, 9, 13, 34] 1024	1.42E+05	0.07	0.96	[0.99, 0.96, 0.96, 0.92]	0.03	0.93	[0.96, 0.91, 0.97, 0.89]	0.03	4	3
	1024 [16, 10, 4, 2, 4, 10, 16] 1024	6.64E+04	0.11	0.94	[0.97, 0.91, 0.95, 0.94]	0.03	0.89	[0.91, 0.82, 0.96, 0.86]	0.03	4	2
	1024 [188, 61, 36, 1, 36, 61, 188] 1024	8.25E+05	0.13	0.87	[1.0, 0.64, 0.93, 0.92]	0.03	0.87	[0.94, 0.7, 0.96, 0.87]	0.03	4	1
	1024 [7, 4, 3, 0, 3, 4, 7] 1024	2.89E+04	0.08	0.96	[1.0, 0.91, 0.96, 0.96]	0.03	0.92	[0.97, 0.84, 0.95, 0.93]	0.03	4	3
	1024 [5, 5, 5, 0, 5, 5, 5] 1024	2.07E+04	0.23	0.76	[0.96, 0.37, 0.91, 0.78]	0.03	0.77	[0.97, 0.33, 0.93, 0.85]	0.03	1	5
	1024 [4, 3, 3, 0, 3, 3, 4] 1024	1.65E+04	0.26	0.76	[0.88, 0.42, 0.96, 0.77]	0.04	0.74	[0.88, 0.33, 0.95, 0.81]	0.04	2	3
	1024 [1, 0, 0, 0, 0, 0, 1] 1024	4.10E+03	0.86	0.15	[0.0, 0.0, 0.59, 0.0]	0.04	0.14	[0.0, 0.0, 0.58, 0.0]	0.05	1	1
Scheme 2: Division by 2	1024,[512, 256, 128, 64, 32, 10, 32, 64, 128, 256, 512],1024	2.79E+06	0.08	0.96	[0.99, 0.95, 0.97, 0.94]	0.03	0.92	[0.92, 0.9, 0.96, 0.88]	0.03	6	10
	1024,[512, 256, 128, 64, 32, 64, 128, 256, 512],1024	2.79E+06	0.08	0.94	[0.99, 0.87, 0.95, 0.94]	0.02	0.92	[0.93, 0.9, 0.96, 0.91]	0.03	5	32
	1024,[512, 256, 128, 64 ,128 ,256, 512],1024	2.79E+06	0.09	0.96	[0.99, 0.95, 0.97, 0.94]	0.02	0.91	[0.9, 0.9, 0.95, 0.89]	0.03	4	64
	1024,[512, 128, 32, 8 ,32 ,128, 512],1024	2.38E+06	0.14	0.94	[0.99, 0.87, 0.97, 0.94]	0.03	0.86	[0.9, 0.82, 0.93, 0.79]	0.03	4	8
	1024,[128, 32, 16 ,8 , 16, 32, 128],1024	5.43E+05	0.08	0.95	[0.99, 0.95, 0.96, 0.92]	0.03	0.92	[0.93, 0.9, 0.96, 0.91]	0.03	4	8
Scheme 3: 2/3 of the previous layer	1024,[683,455,303,202,135,90,60,40,27,18,27,40,60,90,135,202,303,455,683],1024	5.03E+06	0.11	0.97	[0.97, 1.0, 0.97, 0.94]	0.03	0.89	[0.93, 0.82, 0.96, 0.86]	0.03	10	18
	1024,[683,455,303,202,135,90,60,40,27,40,60,90,135,202,303,455,683],1024	5.03E+06	0.07	0.95	[0.97, 0.92, 0.97, 0.94]	0.03	0.93	[0.94, 0.9, 0.97, 0.91]	0.03	9	27
	1024,[683,455,303,202,135,90,60,40,60,90,135,202,303,455,683],1024	5.03E+06	0.1	0.95	[0.97, 0.95, 0.95, 0.92]	0.03	0.9	[0.92, 0.9, 0.94, 0.84]	0.04	8	40
	1024,[683,455,303,202,135,90,60,90,135,202,303,455,683],1024	5.02E+06	0.07	0.97	[0.99, 1.0, 0.97, 0.92]	0.03	0.93	[0.94, 0.9, 0.96, 0.93]	0.03	7	60
	1024,[683,455,303,202,135,90,135,202,303,455,683],1024	4.99E+06	0.09	0.94	[0.96, 0.91, 0.97, 0.92]	0.03	0.91	[0.92, 0.9, 0.97, 0.87]	0.03	6	90
	1024,[683,455,303,202,135,202,303,455,683],1024	4.95E+06	0.07	0.95	[0.99, 0.91, 0.96, 0.94]	0.03	0.93	[0.94, 0.9, 0.97, 0.91]	0.03	5	135
Scheme 4: [65]	1024,[651, 688, 91, 32, 46],1024	2.46E+06	0.09	0.96	[1.0, 0.91, 0.96, 0.96]	0.02	0.91	[0.96, 0.84, 0.95, 0.87]	0.02	3	91
Scheme 5: [78]	1024,[1244, 522, 261, 37, 261, 522, 1244],1024	8.28E+06	0.07	0.94	[0.97, 0.92, 0.96, 0.92]	0.02	0.93	[0.94, 0.9, 0.96, 0.93]	0.03	4	37
	1024,[1244, 522, 261, 44, 261, 522, 1244],1024	8.28E+06	0.11	0.92	[0.96, 0.91, 0.93, 0.88]	0.03	0.89	[0.92, 0.8, 0.94, 0.89]	0.03	4	44
	1024,[1244, 522, 261, 17, 261, 522, 1244],1024	8.26E+06	0.09	0.91	[0.96, 0.82, 0.95, 0.92]	0.03	0.91	[0.92, 0.86, 0.97, 0.89]	0.03	4	17
	1024,[1244, 522, 261, 9, 261, 522, 1244],1024	8.25E+06	0.1	0.95	[0.99, 0.91, 0.96, 0.94]	0.03	0.9	[0.93, 0.84, 0.95, 0.87]	0.03	4	9
	1024,[1244, 522, 261, 45, 261, 522, 1244],1024	8.28E+06	0.08	0.95	[0.97, 0.92, 0.97, 0.94]	0.03	0.92	[0.92, 0.9, 0.97, 0.91]	0.03	4	45
	1024,[1244, 522, 261, 47, 261, 522, 1244],1024	8.29E+06	0.07	0.94	[0.99, 0.91, 0.95, 0.92]	0.03	0.93	[0.94, 0.9, 0.95, 0.91]	0.03	4	47
	1024,[1244, 522, 261, 61, 261, 522, 1244],1024	8.30E+06	0.11	0.96	[1.0, 0.95, 0.96, 0.94]	0.03	0.89	[0.9, 0.84, 0.95, 0.88]	0.03	4	61
	1024,[1244, 522, 261, 36, 261, 522, 1244],1024	8.28E+06	0.11	0.94	[0.99, 0.91, 0.94, 0.94]	0.03	0.89	[0.93, 0.86, 0.94, 0.84]	0.03	4	36
	1024,[1244, 522, 261, 33, 261, 522, 1244],1024	8.27E+06	0.08	0.96	[0.99, 0.95, 0.96, 0.96]	0.03	0.92	[0.93, 0.86, 0.96, 0.93]	0.03	4	33
	1024,[1244, 522, 261, 26, 261, 522, 1244],1024	8.27E+06	0.08	0.94	[0.94, 0.96, 0.95, 0.92]	0.03	0.92	[0.93, 0.9, 0.95, 0.89]	0.03	4	26
	1024,[1244, 522, 261, 21, 261, 522, 1244],1024	8.26E+06	0.07	0.95	[0.99, 0.91, 0.96, 0.94]	0.03	0.93	[0.93, 0.9, 0.97, 0.93]	0.03	4	21